Training Drift Counteraction Optimal Control Policies Using Reinforcement Learning: An Adaptive Cruise Control Example

Zhaojian Li, *Member, IEEE*, Tianshu Chu, *Student Member, IEEE*, Ilya V. Kolmanovsky, *Fellow, IEEE*, and Xiang Yin, *Member, IEEE*

***Abstract*— The objective of drift counteraction optimal control (DCOC) problem is to compute an optimal control law that maximizes the expected time of violating specified system constraints. In this paper, we reformulate the DCOC problem as a reinforcement learning (RL) one, removing the requirements of disturbance measurements and prior knowledge of the disturbance evolution. The optimal control policy for the DCOC is then trained with RL algorithms. As an example, we treat the problem of adaptive cruise control, where the objective is to maintain desired distance headway and time headway from the lead vehicle, while the acceleration and speed of the host vehicle are constrained based on safety, comfort, and fuel economy considerations. An informed approximate Q-learning algorithm is developed with efficient training, fast convergence, and good performance. The control performance is compared with a heuristic driver model in simulation and superior performance is demonstrated.**

***Index Terms*— Adaptive cruise control, approximate Q-learning, drift counteraction control, reinforcement learning**

I. Introduction

THE objective of the drift counteraction optimal control (DCOC) is to find a control law such that the expected time before system constraint violations occur is maximized. Such a control law can be often interpreted as providing drift counteraction which explains its name – drift counteraction optimal control [1]. Numerous DCOC applications have been previously reported, including satellite control [2], glider flight management [3], and hybrid electric vehicle energy management [4].

Variants of value iteration methods have been previously exploited in the literature to solve the DCOC problem [1]–[4], where two prerequisites are needed. Firstly, the disturbance needs to be measured at each time step. Secondly, the disturbance is typically modeled as a Markov chain and the transition probabilities are assumed to be known. Both the assumptions of measured disturbance and known transition probabilities are limiting in view of potential applications of these techniques to practical systems.

To address the above issues, we reformulate the DCOC as a reinforcement learning (RL) problem. RL is a computational approach to get the agent to act so as to maximize its expected cumulative rewards through iterative interaction with an unknown, complex, and uncertain environment. This reformulation provides several advantages. Firstly, the RL does not require the disturbance measurements; only state information is needed. Secondly, RL is able to obtain an optimal policy through trial-and-error learning without prior knowledge of the transition probability of the Markov chain. In addition, the explicit form of the model may also be unknown. Furthermore, a rich set of algorithms have been developed to solve RL problems.

Among numerous RL algorithms developed in the past few decades, Q-learning [5] is arguably the most well-known model-free RL algorithm. It iteratively learns and updates the optimal action-value functions based on received costs. By virtue of guaranteed convergence and ease of implementation, Q-learning has found great success in many applications [6]–[9]. However, it is noteworthy that Q-learning only works for Markov Decision Processes (MDPs) with discrete state space and discrete action space [10]. For systems with large or continuous state space, an action value function estimator needs to be exploited to avoid information loss and curse of dimensionality caused by state/action discretization. In practice, the linear function based approximate Q-learning (AQL) [11] is frequently used due to its simplicity and fast convergence [12]–[14]. See Section II for a more detailed introduction on RL.

As a case study, we apply RL to the development of an Adaptive Cruise Control (ACC) system, which automatically adjusts a car's speed to maintain a safe following distance. Extensive studies on ACC control designs have been reported in the literature to control a car to follow a desired equilibrium speed-gap condition, including Proportional Integral and Derivative control [15], Fuzzy Logic [16], and Model Predictive Control [17], [18]. In this paper, we design the

Manuscript received January 1, 2017; revised June 5, 2017 and August 23, 2017; accepted October 19, 2017. Date of publication November 27, 2017; date of current version September 7, 2018. The Associate Editor for this paper was L. Li. *(Corresponding author: Xiang Yin.)*

Z. Li was with the University of Michigan, Ann Arbor, MI 48109 USA. He is now with the Department of Mechanical Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: lizhaoj1@egr.msu.edu).

T. Chu is with the Department of Civil and Environmental Engineering, Stanford University, CA 94305 USA (e-mail: cts1988@stanford.edu).

I. V. Kolmanovsky is with the Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: ilya@umich.edu).

X. Yin is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: xiangyin@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2017.2767083

ACC control in a DCOC framework [19], that is, we aim to maximize the expected time before system constraints (due to safety, comfort, and fuel economy) are violated. Note although distance to the lead vehicle can be measured at current step, the lead vehicle speed change at next step is unknown, which poses a great challenge to maintain the ego/host vehicle within the desired system constraints. This motivates the use of RL that can iteratively learn the lead vehicle dynamics by interacting with the environment and improve the control policy adaptively.

In this study, we first reformulate the DCOC ACC problem as a RL one. By exploitation of system dynamics, the feature space, stage cost, and action selection are carefully designed and a model-based Informed Approximate Q-learning (IAQL) algorithm is developed with efficient exploitation and fast convergence. Unlike conventional model-free Approximate Q-Learning (AQL), the proposed IAQL exploits the system dynamics in the feature design and control space construction to guide the training. As we show in the paper, the proposed IAQL converges whereas the conventional AQL may not. We train the IAQL algorithm in simulation, and a hybrid Markov process is used to model the lead vehicle dynamics. The RL control performance is then compared with a benchmark and its enhanced variant. Superior performance is demonstrated.

The contributions of this paper include the following. Firstly, we demonstrate how the DCOC problem can be reformulated as a RL problem, removing the requirements of prior knowledge of disturbance measurements and transition probabilities. Secondly, the ACC problem is treated and the IAQL algorithm is developed to train the control law efficiently and effectively by exploitation of system model whereas conventional AQL does not converge in this ACC problem. The control performance is compared with a benchmark and superior performance is demonstrated.

The rest of the paper is organized as follows. Section II presents a background introduction on RL. The reformulation of a DCOC as a RL one and the ACC problem formulation are presented in Section III. In Section IV, the IAQL algorithm is developed to optimize the ACC. In Section V, a heuristic driver model is introduced as a benchmark, with which the RL control performance is compared. Conclusions are drawn in Section VI.

II. Background on Reinforcement Learning

Reinforcement learning (RL) is a trial-and-error learning algorithm that optimizes the agent's actions to minimize accumulated costs (or maximize accumulated rewards) during its interaction with the environment. At each discrete time step t, the agent receives an observation $x_t \in \mathbb{R}^{n_x}$, takes a real-valued action u_t[1] $\in \mathbb{R}^{n_u}$ and receives a scalar cost (or reward) $r_{t+1} \in \mathbb{R}$ at time $t+1$.[2] The entire history of observation and action is an information state, $s_t = (x_1, u_1, \ldots, u_{t-1}, x_t)$. We assume a Markovian environment and redefine information state as $s_t = x_t$.

The stochastic environment E is modeled as a Markov decision process (MDP), $\mathcal{M} = \{\mathcal{S}, \mathcal{U}, \mathcal{R}, \mathcal{P}, \gamma\}$, where $\mathcal{S} \subseteq \mathbb{R}^{n_x}$ represents the state space; $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ is the action space; $\mathcal{R} : (\mathcal{S}, \mathcal{U}) \rightarrow \mathbb{R}$ is the scalar cost function and $\mathcal{R}_s^u = \mathbb{E}[R_{t+1}|S_t = s, U_t = u]$; $\mathcal{P}$ represents the state transition map and $\mathcal{P}_{ss'}^u = \Pr[S_{t+1} = s'|S_t = s, U_t = u]$; $\gamma \in [0, 1)$ is a discounting factor.

A policy, $\pi : \mathcal{S} \rightarrow \Pr(\mathcal{U})$, is a mapping from states to a probability distribution over the actions and it describes the agent's behavior. Note that the probability of a deterministic policy is one for the chosen action and zero for all other actions. Applying a policy π to the MDP defines a Markov chain and we use $\mathbb{E}_\pi[\cdot]$ to denote expectations over this chain. The return G_t is the total discounted cost from time-step t,

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1}$$

The goal of the RL is to learn an optimal policy π^* which minimizes the expected return from the start state, i.e.,

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\ \mathbb{E}_\pi[G_1]. \tag{2}$$

The state value function $\bar{v}(s)$ under policy π is defined as the expected return starting from state s,

$$\bar{v}_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]. \tag{3}$$

Compared to the state value, the action value function is more commonly used in RL. It is defined as the expected return after taking an action u_t at state s_t and following policy π thereafter,

$$Q^\pi(s_t, u_t) = \mathbb{E}_\pi[G_t|S_t = s_t, U_t = u_t]. \tag{4}$$

If we are able to find the optimal action-value function,

$$Q^*(s_t, u_t) = \min_\pi \mathbb{E}_\pi[Q^\pi(s_t, u_t)],$$

then the optimal policy can be obtained by

$$\pi^*(s) = \underset{u \in \mathcal{U}}{\operatorname{argmin}}\ Q^*(s, u). \tag{5}$$

Numerous RL algorithms have been developed in the past few decades. Depending on whether the MDP model is explicitly learnt, RL algorithms fall into two categories: model-based and model-free. Model-based methods first/simultaneoursly learn the MDP and traditional optimization methods such as policy iteration and value iteration can then be applied to solve the MDP efficiently. On the other hand, model-free RL methods directly learn optimal value functions or optimal policy without explicitly learning the MDP, avoiding requirements on disturbance measurements or prior knowledge of model transition probabilities.

Q-learning [5] is arguably the most well-known model-free RL algorithm. It iteratively learns and updates the optimal action-value functions based on received costs. Specifically, at each time step t the agent selects an action u_t based on the state s_t, and receives a cost r_{t+1} and leads to a new state s_{t+1}

[1] In most publications, the action is denoted by a. However, in the adaptive cruise control example, we thus use a to denote the acceleration. To avoid confusions, we use u to represent the action.

[2] In some publications, the reward is assigned at time step t. In this study, we follow the notation convention from [20].

at time $t+1$. Then the value function for $Q(s_t, u_t)$ is updated as

$$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha[r_{t+1} + \gamma \min_u Q(s_{t+1}, u) - Q(s_t, u_t)], \tag{6}$$

where $\alpha \in (0, 1]$ is the learning rate and $\gamma \in [0, 1)$ is the discount factor. The Q-learning is in essence an iterative learning method where the values are updated from the difference between the learnt value $r_{t+1} + \gamma \min_u Q(s_{t+1}, u)$ and the old value $Q(s_t, u_t)$. It is proved in [21] that if the learning rate α satisfies some appropriate conditions, then the action functions converge to the true values.

By virtue of guaranteed convergence and ease of implementation, Q-learning has found great success in numerous applications [6]–[9]. However, it is noteworthy that Q-learning only works for MDPs with discrete state space and discrete action space [10].

For systems with large or continuous state space, value table lookup methods used in Q-learning are not feasible. One can discretize the state space but it leads to information loss and may have curse of dimensionality. An alternative is to exploit a function estimator to estimate the action value function, i.e.,

$$\hat{Q}(S, U, \theta) \approx Q_\pi(S, U), \tag{7}$$

where θ is the vector of parameters to be identified. The optimal parameters θ can be obtained by minimising the mean-squared error between approximated action-value function $\hat{Q}(S, A, \theta)$ and true action-value function $Q_\pi(S, A)$, that is,

$$J(\theta) = \mathbb{E}_\pi\big[(Q_\pi(S, U) - \hat{Q}(S, U, \theta))^2\big]. \tag{8}$$

Stochastic gradient descent can be used to update the parameters to find a local minimum,

$$\begin{aligned} \Delta\theta &= -\frac{1}{2}\alpha \bigtriangledown_\theta J(\theta) \\ &\approx \alpha(Q_\pi(S, U) - \hat{Q}(S, U, \theta)) \bigtriangledown_\theta \hat{Q}(S, U, \theta), \end{aligned} \tag{9}$$

where $\alpha > 0$ is the step size. The true action value function $Q_\pi(S, U)$ can be approximated using Monte Carlo or Temporal Difference method [20], and the function approximation $\hat{Q}(S, U, \theta)$ and the resulting gradient $\bigtriangledown_\theta \hat{Q}(S, U, \theta)$ are evaluated with the parameters θ in the current iteration.

It is a common practice to represent state and action by a feature vector,

$$\phi(S, U) = \begin{bmatrix} \phi_1(S, U) \\ \vdots \\ \phi_n(S, U) \end{bmatrix}. \tag{10}$$

The approximation function can then be described using the feature vector and parameters, $\hat{Q}(S, U, \theta) = f(\phi(S, U), \theta)$. A variety of differentiable functions $f(\cdot)$ can be exploited, from simple linear function [14] to deep neural networks [10].

The linear function based approximate Q-learning (AQL) is frequently used in applications, in which a linear Q-function $Q_\theta(s, u) = \theta^T \phi(s, u)$ is defined, where θ is the *weight vector*, and ϕ is the *feature vector* extracted from the state-control pair as in (10). Given the Q-function, the best control is,

$$u_t \in \operatorname*{argmin}_{u \in \mathcal{U}} Q_\theta(s_t, u). \tag{11}$$

For online AQL update, we estimate the one-step ahead Q-function $\mathcal{T}Q_\theta(s_t, u_t)$ as:

$$\mathcal{T}Q_\theta(s_t, u_t) = r_{t+1} + \gamma \min_{u \in \mathcal{U}} Q_\theta(s_{t+1}, u), \tag{12}$$

where $\mathcal{T}$ is called the *dynamic programming operator*. Then we can update the weight vector using the stochastic gradient descent,

$$\theta \leftarrow \theta - \alpha \Delta Q_{\theta,t} \phi(s_t, u_t), \tag{13}$$

where α is the learning rate that controls the updating step size, and $\Delta Q_{\theta,t}$ is the *temporal difference* [20]:

$$\Delta Q_{\theta,t} := (Q_\theta - \mathcal{T}Q_\theta)(s_t, u_t) \tag{14}$$

Note that the afore-mentioned value-based RL algorithms need to evaluate the best action based on the action-value function as in (5). This evaluation itself can be impractical for systems with large or continuous action space. For these systems, we can instead directly parameterize the policy, i.e.,

$$\pi_\theta = \Pr[u|s, \theta], \tag{15}$$

where θ are some parameters that control the probability distribution of a policy. One example of the policy distribution function is the Gaussian policy. Let $\phi(s)$ be the state features; then we can sample our action from a Gaussian distribution, $u \sim \mathcal{N}(\mu(s), \sigma^2)$, where $\mu(s) = \phi(s)^{\mathrm{T}}\theta$ is a linear combination of the features parameterized by θ. The variance σ^2 can be either fixed or also parameterized.

The goal is to find the optimal parameter θ^* to maximize the average value

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) V^{\pi_\theta}(s), \tag{16}$$

where $d^{\pi_\theta}(s)$ is the stationary distribution of the Markov chain with policy π_θ and $V^{\pi_\theta}(s)$ is the state value function of s with policy π_θ. Stochastic gradient descent can be used to update the parameters. This method is referred to as policy gradient RL [22].

III. From DCOC to RL: An Adaptive Cruise Control (ACC) Example

The ACC system can automatically adjust the ego vehicle speed to maintain a safe distance from the lead vehicle; it has become an increasingly popular feature in modern vehicles. The ACC involves a vehicle following problem illustrated in Figure 1. The relative motion dynamics can be described as:

$$\begin{aligned} \dot{d} &= v_l - v_f, \\ \dot{v}_f &= u, \end{aligned} \tag{17}$$

where d represents the longitudinal distance (often referred to as the range) between the lead vehicle (right) and the ego/host vehicle (left). The variables v_l and v_f are the longitudinal speed of the lead vehicle and the ego vehicle, respectively; The control u is the longitudinal acceleration of the ego vehicle.

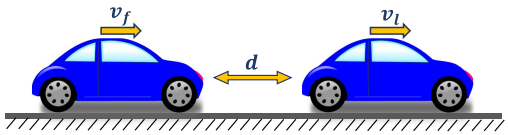


Fig. 1. Schematic diagram of the vehicle-follow dynamics.

The following constraints are considered to ensure comfort, safety, and acceptable fuel economy:

$$u_{\min} \le u \le u_{\max}, \tag{18}$$
$$0 \le v_f \le v_{f,\max}, \tag{19}$$
$$T_{h,\min} \le T_h \le T_{h,\max}, \tag{20}$$
$$d \ge d_{\min}, \tag{21}$$

where $u_{\max}$ and $u_{\min}$ are, respectively, the maximum and minimum acceleration limits for good ride comfort and fuel economy; $v_{f,\max}$ represents the maximum speed due to speed limit; $T_h \triangleq \frac{d}{v_f}$ is referred to as time headway, which is constrained between $T_{h,\min}$ and $T_{h,\max}$ for safety and comfort considerations; $d_{\min}$ designates the minimum distance the vehicle should maintain from the lead vehicle.

With a sampling time Δ, (17) can be discretized as

$$\begin{aligned} d_{t+1} &= d_t + (v_{l,t} - v_{f,t}) \cdot \Delta, \\ v_{f,t+1} &= v_{f,t} + u_t \cdot \Delta. \end{aligned} \tag{22}$$

By defining $x_t = [d_t,\ v_{f,t}]^{\mathrm{T}}$ and $w_t = v_{l,t}$, we have

$$x_{t+1} = Ax_t + Bu_t + B_w w_t, \tag{23}$$

where $A = \begin{bmatrix} 1 & -\Delta \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ \Delta \end{bmatrix}$, and $B_w = \begin{bmatrix} \Delta \\ 0 \end{bmatrix}$.

The DCOC formulation [19] of the problem is to determine a control policy, $u(x, w)$, that maximizes the time duration till any constraint from (18)-(21) is violated, i.e.,

$$\begin{aligned} &\max_{u(\cdot,\cdot)} \ \tau(x_0, w_0, u(\cdot,\cdot)), \\ &\text{s.t.} \ \ u_t \in U,\ x_t \in \mathcal{X},\ \forall t = 1, 2, \cdots, \tau - 1, \\ &\qquad \text{and } x_\tau \notin \mathcal{X}, \end{aligned} \tag{24}$$

where $U = [u_{\min}, u_{\max}]$, and

$$\mathcal{X} = \left\{(d, v_f) | d \in [d_{\min}, T_{h,\max} \cdot v_{f,\max}],\ v_f \in [0, v_{f,\max}]\right\}.$$

The DCOC problem (24) can be solved using a variant of value iteration developed in [1]. However, two items are needed for implementation. Firstly, the lead vehicle speed, v_l, needs to be measured, which requires a radar, a camera or V2V communications, and signal processing. Secondly, the lead vehicle speed v_l is modeled as a Markov chain and the transition probabilities need to be known a priori. While online learning algorithms [23] can be used, value iterations are too complex to be performed onboard; hence the implementation may require a bank of policies computed for several transition probability matrices as well as non-trivial interpolation techniques.

To develop a simpler solution, we reformulate the DCOC problem (24) as a RL one, removing the requirements on the measurement of lead vehicle speed and its transition probabilities. To implement a RL algorithm, we first model the system dynamics as an MDP with

- $\mathcal{S} = \{d,\ v_f\}$: is the set of states including the range and speed of the ego vehicle.
- $\mathcal{U} = \begin{bmatrix} -u_{\max} & u_{\max} \end{bmatrix}$: is the set of permissive accelerations.
- $\mathcal{P}$ is a state transition matrix and $\mathcal{P}^u_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, U_t = u]$. Note that this matrix need not be known a priori for RL implementations.
- $\mathcal{R}$ is a cost/reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, U_t = u]$. We assign a cost of one for each step that constraints (18)-(21) are violated and zero vice versa.
- $\gamma \in (0, 1)$ is a discount factor that differentiates the importance of future costs and present costs. This is the necessary condition for the convergence of RL.

The objective of the RL ACC is to define a policy $\pi^* : \mathcal{S} \to \Pr(\mathcal{U})$, such that

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\ \mathbb{E}_\pi[G_1], \tag{25}$$

where G_1 is the return function defined in (1). We next develop a RL algorithm to efficiently find π^*.

Remark 1: Although the dynamics (22) is a simple linear system, the DCOC ACC problem (24) is still challenging due to the uncertainty of the lead vehicle dynamics. Sudden velocity change of the lead vehicle can quickly lead to violations of system constraints (19)-(21). This motivates the use of RL to iteratively learn the dynamics of the lead vehicle and improves its control policy.

IV. Informed Approximate Q-Learning

In real-world applications, the underlying transition probability $\mathcal{P}$ is most likely unobservable. Fortunately, as long as the underlying dynamics are stationary [24], RL algorithms can be used to learn the Q-value functions from the transition history, $\{s_t, u_t, r_{t+1}, s_{t+1}, u_{t+1}, \ldots\}$. If the state space is continuous or large, function approximations are needed to fit the Q-value function, based on the state-action features. This class of Q-learning algorithms is referred to as approximate Q-learning. However, approximate Q-learning algorithms typically have no guaranteed convergence properties, that is, the approximated Q-value function may not converge to the true value function. As far as the authors are aware, the only known convergence result, proved in [14], is for linear approximation functions under rather restrictive conditions on the sample distribution of the transition data.

Nonetheless, a careful design of stage costs and features can improve the convergence of the RL algorithm. Since the distance and speed states in the ACC problem are continuous, a function approximation is needed to avoid information loss caused by discretization. To deal with the convergence challenges, we develop an informed approximate Q-learning (IAQL) algorithm, with good learning performance and fast convergence for the ACC problem. Specifically, we exploit appropriate stage cost to promote obtaining information useful for learning, design appropriate feature vector to capture the Q-function shape, and make efficient action selections for fast convergence.

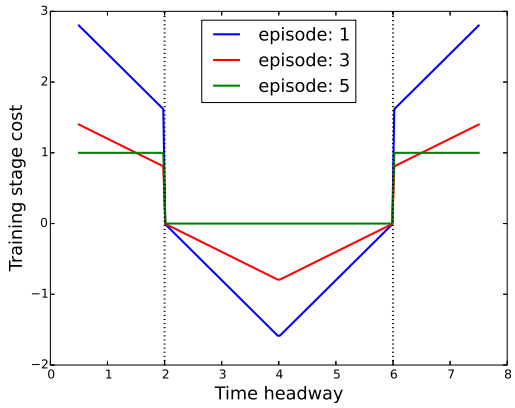


Fig. 2. Informed training stage cost over the first 5 training episodes. The dotted vertical lines indicate $T_{h,\min}$ and $T_{h,\max}$. For the first 4 episodes, a piecewise linear stage cost that captures the closeness to the boundary of time headway constraint is used to facilitate training. From episode 5, the zero/one stage cost is used to ensure the convergence to the true action value.

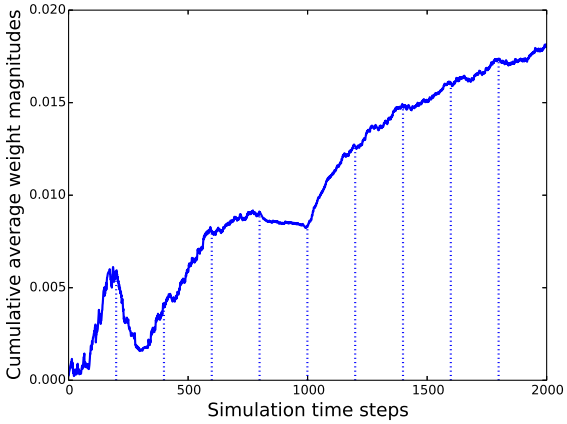


Fig. 3. Cumulative average weight (sum of absolute value of the weights divided by number of weights) over 10 training episodes using preliminary features.

A. Stage Cost Engineering

A natural choice of stage cost for DCOC ACC is a "zero/one" function so that a penalty is charged whenever the time headway constraint is violated

$$r(t) = \begin{cases} 0, & \text{if } T_h(t) \in [T_{h,\min}, T_{h,\max}], \\ 1, & \text{otherwise.} \end{cases} \tag{26}$$

However, this "zero/one" stage cost is sparsely distributed and it is always zero whenever the constraint is satisfied. Therefore the RL agent can hardly obtain useful information from most transition data, resulting in unnecessary exploration and inefficient learning. For example, even if the current headway is close to the lower boundary, the RL agent cannot get this information immediately. Rather, it performs random exploration at the beginning until it violates the constraints. As a result, it is not able to learn to decelerate until the penalty effect is passed along from the boundary state after a considerable number of updating steps.

In order to improve the learning efficiency, we utilize the information feedback given to the RL agent through the stage cost to avoid unnecessary exploration. In particular, instead of using the "zero/one" indicator function, we use a piecewise linear function to highlight the eligibility of $T_h(t)$ in the early training episodes. In the ACC problem, an episode is defined as the time when the constraint is violated or after a fixed duration. To better utilize the information during the exploration stage, for the first five training episodes, we use the following stage cost

$$\tilde{r}(t) = \min\big(5,\ \max(0, (5-k)/5)r'(t) + \min(1, k/5)r(t)\big), \tag{27}$$

for episode $k = 1, 2, \ldots$, where

$$r'(t) = \begin{cases} T_{h,\min} + (T_{h,\min} - T_h(t)) & \text{if } T_h(t) < T_{h,\min}, \\ T_{h,\min} + (T_h(t) - T_{h,\max}) & \text{if } T_h(t) > T_{h,\max}, \\ |T_h(t) - 0.5(T_{h,\min} + T_{h,\max})| & \\ \quad -0.5(T_{h,\max} - T_{h,\min}) & \text{otherwise.} \end{cases} \tag{28}$$

In other words, $r'(t)$ is a penalty reflecting how far the current time headway $T_h(t)$ is away from the constraint boundaries. Note that $\tilde{r}(t)$ uses the combined function of $r'(t)$ and $r(t)$ for the first five episodes to maximize the information feedback, and smoothly transfers to the "zero/one" indication function $r(t)$ afterwards to ensure convergence. Also, $\tilde{r}(t)$ is clipped to 5 to avoid aggressive updating during the early stage exploration.

Figure 2 illustrates the change of the stage cost function, $\tilde{r}(t)$, as a function of episode number. After 5 episodes, the stage cost function becomes identical to the cost function $r(t)$.

B. Feature Engineering

We apply a parameterized learning model to approximate the Q-value function as $Q_\theta(s, u)$, with θ being the parameters to be identified. The typical choices include:

- Linear Regression (LR) model: $Q_\theta(s, u) = \theta^T \phi(s, u)$.
- Artificial Neural Network (ANN) model: $Q_\theta(s, u) = b + w^T \sigma(W\phi(s, u) + a)$, where σ is the activation function, and $\theta = \{W, a, w, b\}$.

In this paper, we choose the LR model due to its simplicity and better convergence properties. Note the design of an appropriate feature mapping function $\phi : \mathcal{S} \times \mathcal{U} \to \mathbb{R}^r$ is critical to the learning performance and the convergence properties. A preliminary choice of the feature is a collection of the states and actions, i.e., $\phi(s, u) = [d, v_f, u]^{\mathrm{T}}$. The training results (see next Section for the simulation setup) with the above features are shown in Figure 3 and Figure 4. It can be seen that the parameters do not converge and the control performance does not improve over time.

The reason is that our stage cost is a non-linear function of states and action. Therefore using the linear features may have the risk of divergence since $Q_\theta(s, u) = \theta^T \phi(s, u)$ can hardly capture the stage cost and the Q-function.

As an alternative, we define the informed feature vector as:

$$\phi(s(t), u(t)) = \begin{bmatrix} \hat{r}(t, 1) \\ \hat{r}(t, 0) \\ \hat{r}(t, -1) \\ T_{h,\min} - \hat{T}_h(t+1, \delta) \\ \hat{T}_h(t+1, \delta) - T_{h,\min} \end{bmatrix}. \tag{29}$$

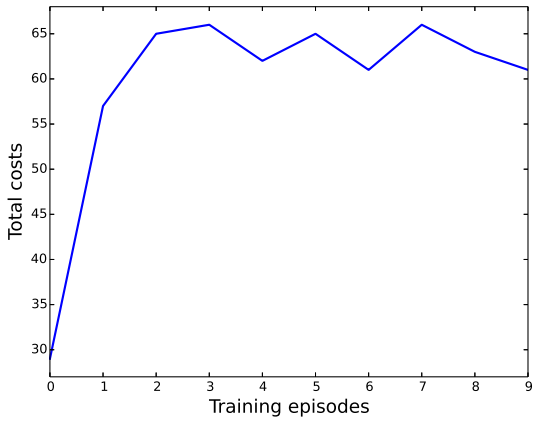


Fig. 4. Total cost vs. training episode during the training procedure of AQL using preliminary features.

The cost $\hat{r}(t,\delta) = |\hat{T}_h(t+1,\delta) - \frac{1}{2}(T_{h,\min} + T_{h,\max})|$ represents the distance between the predicted time headway and center of the time headway constraint range with δ being the assumed lead vehicle speed change. The predicted time headway $\tilde{T}_h(t+1,\delta)$ is computed as follows. First the distance is approximated as

$$\hat{d}(t+1) = d(t) + (v_l(t-1) - v_f(t))\Delta, \tag{30}$$

using the lead vehicle speed at step $t-1$ to approximate the one at step t. Note that the lead vehicle speed at $t-1$ can be computed at step t from (22):

$$v_l(t-1) = v_f(t-1) + \frac{1}{\Delta}\big(d(t) - d(t-1)\big). \tag{31}$$

Since δ represents the assumed speed change, we have

$$\hat{v}_l(t+1) = v_l(t-1) + \delta. \tag{32}$$

Note in (32) we use $v_l(t-1)$ instead of $v_l(t)$ for prediction because at time t, $v_l(t-1)$ can be computed using (31) while the knowledge of $v_l(t)$ would require vehicle-to-vehicle communications. In the implementation, we choose δ to be 1, 0, and -1 since it can represent the increase, no change, and decrease, respectively. Other choices of δ can also be used.

As a result, the predicted time headway at time $t+1$ is

$$\hat{T}_h(t+1,\delta) = \frac{\hat{d}(t+1)}{\hat{v}_l(t+1)} = \frac{d(t) + (v_l(t-1) - v_f(t))\Delta}{v_l(t-1) + \delta}. \tag{33}$$

Note that the selected features reflect the estimated stage cost $\tilde{r}$ in a predicted manner and can thus enhance training efficiency as well as improve the convergence.

C. Action Selection

With the Q-value function weights updating at each step, the agent can pick the best action (acceleration) as

$$u^*(t) \in \underset{u\in\mathcal{U}}{\operatorname{argmin}}\ Q_\theta(s(t), u(t)). \tag{34}$$

However, due to the constraint (19) on the ego vehicle speed, the best control in (34) may end up violating the constraint and unnecessarily end the training. To address this issue, we explicitly consider the constraint (19) in the action selection, i.e., require

$$0 \le v_f(t) + u(t)\Delta \le v_{f,\max}, \tag{35}$$

leading to $u(t) \in \big[-\frac{1}{\Delta}v_f(t), \frac{1}{\Delta}(v_{f,\max} - v_f(t))\big]$. Therefore, we use a time-varying action space for search:

$$\bar{\mathcal{U}}(t) = \big[u_{\min}, u_{\max}\big] \cap \big[-\frac{1}{\Delta}v_f(t), \frac{1}{\Delta}(v_{f,\max} - v_f(t))\big], \tag{36}$$

which guarantees the satisfaction of constraints (18) and (19).

Furthermore, the RL agent needs to consider the *exploration* and *exploitation* dilemma for action selection. *Exploration* finds more information about the environment while *exploitation* exploits known information to minimize costs. It is therefore important to explore as well as to exploit. As such, we discretize the action space $\bar{\mathcal{U}}(t)$ into 100 points and apply the following ϵ-greedy method to pick the action:

$$\Pr(u|s) = \begin{cases} \epsilon_j/n_u + 1 - \epsilon_j & \text{if } \ u^* = \operatorname{argmax}_{u\in\bar{\mathcal{U}}(t)}\ Q(s,u) \\ \epsilon_j/n_u & \text{otherwise} \end{cases} \tag{37}$$

where n_u is the number of actions (100 in our formulation), ϵ_j, $j = 1, 2, \cdots$ is a positive scalar such that the agent takes the greedy action with probability $1-\epsilon_j+\epsilon_j/n_u$ and chooses an action at random with probability ϵ_j/n_u. It is a simple idea to ensure continual exploration. The value ϵ_j typically decays as time and the number of episodes increases to guarantee the exploitation of the optimal policy eventually.

TABLE I
SIMULATION PARAMETERS

$v_{f,max}\ (m/s)$	$u_{\min}\ (N)$	$u_{\max}\ (N)$	$T_{h,\min}\ (s)$	$T_{h,\max}\ (s)$
33	-5	5	2	6
$d_{\min}\ (m)$	γ	α	$\Delta\ (s)$	ϵ_1
5	0.9	5e-6	1	0.9

D. IAQL

With the choices of stage cost, feature representation, and action selection made, we now present our IAQL algorithm for the ACC problem as Algorithm 1. The parameters of Algorithm 1 include the discount factor γ, episode time duration T, number of episodes M, learning rate α, ϵ-greedy action selection parameter $\{\epsilon_j\}_{j=0}^{M-1}$, and system constraint parameters $T_{h,\min}$, $T_{h,\max}$, $v_{f,\max}$, $u_{\min}$, $u_{\max}$, and $d_{\min}$ from constraints (18)-(21). The episode number M can be infinity if the agent continuously interacts with the environment.

The parameter vector θ is set to a zero vector at the beginning. For each episode, the agent takes measurements of the range d, range rate $\dot{d}$ and the speed of the follow vehicle v_f, and the clock is reset to zero as in Lines 3 and 4 of Algorithm 1. The $\epsilon-$greedy strategy (37) is exploited to choose the action u based on the approximated Q-function as shown in Lines 6-11. The agent then applies the control u, observes the new states and cost, and computes the stage cost

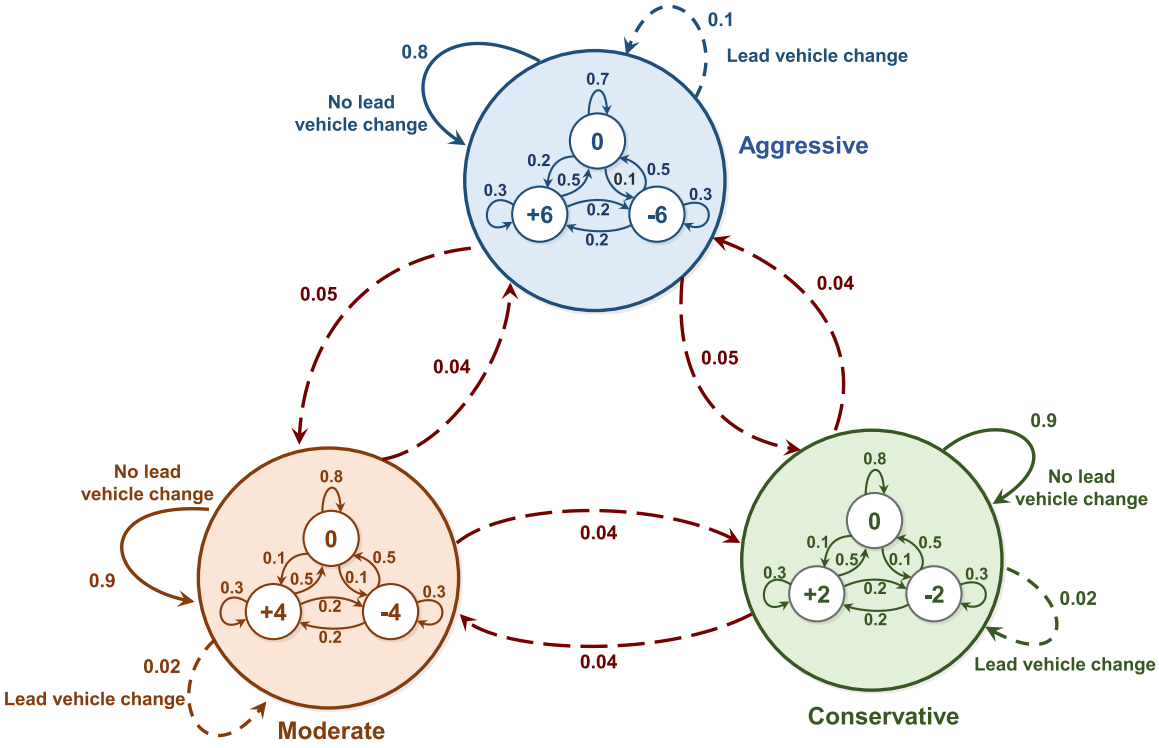


Fig. 5. Lead vehicle speed modeling as a hybrid Markov process. Three outer modes are used to model different driving styles (aggressive, moderate, conservative) and transitions between these modes can capture a lane change or vehicle cut-in. Within each mode, a Markov chain is used to characterize speed change dynamics for the corresponding driving style.

as shown in Line 12. Stochastic Gradient descent is then used to update θ and the clock time is incremented as shown in Lines 13 and 14.

V. Simulation and Performance Comparison

A. Simulation Setup

In this section, we train the IAQL algorithm using simulations. Specifically, we use a hybrid Markov process to model the lead vehicle dynamics as in Figure 5.

The outer three modes represent three different driving styles: aggressive, moderate, and conservative. When there is a lane change or a vehicle cut-in between the lead vehicle and the follow vehicle, the driving style of the lead vehicle may change. For example, from Figure 5, if the current lead vehicle is driving aggressively, there is a probability of 0.8 that the lead vehicle does not change so the driving style is still aggressive; there is a probability of 0.1 that the lead vehicle changes but the new lead vehicle also drives aggressively. In addition, the probability of changing to follow a moderate-driving vehicle and a conservative-driving vehicle are 0.05 and 0.04, respectively.

Inside each driving mode, there are three states representing the possible speed change. For example, inside the conservative driving mode, if there is no speed change from the last step, there is a probability of 0.8 that the speed remains constant while there is a probability of 0.1 that the speed will increase or decrease by 2 m/s. In this study, the transition probabilities in the hybrid model are specified empirically. However, we note that real-world driving data can be used to estimate the transition probabilities.

We run simulations with the parameters specified in Table I for 10 episodes where each episode has a duration of 200 seconds. The initial conditions are set as $a_l(0) = 0$, $v_l(0) = 20$, $v_f(0) = 20$, $d(0) = 75$, and the driving style of the lead vehicle is aggressive, where $a_l(0)$ represents the initial acceleration/speed change of the lead vehicle. The exploration rate ϵ_j decreases from 0.9 to 0.1 linearly over the ten training episodes, and each episode has $T = 200$ steps.

Note that when there is a lead vehicle change according to the Markov chain, we reset $a_l(t^+) = 0$, $v_l(t^+) = v_l(t)$, and $d(t^+) = d(t)$. Whenever there is a constraint violation, we initialize the simulation using the above parameters to continue the training.

With our IAQL algorithm implementation, the training weights and control performance are shown in Figure 6 and Figure 7, respectively. It can be seen that the weights converge and the IAQL algorithm can enable the ACC system with near zero constraint violations.

Remark 2: Unlike conventional model-free AQL, the proposed IAQL exploits the system dynamics in the feature design (29) and action space reconstruction (36) to guide the AQL training. This leads to improved performance, as compared in Figure 4 and Figure 7. Moreover, the proposed IAQL converges while the conventional AQL does not as compared in Figure 3 and Figure 6. We presume this is due to that our feature vector (29) and stage cost (27) are appropriately designed so that the true Q-function (which is a linear

Algorithm 1 Informed Approximated Q-Learning (IAQL)

Parameters: γ, T, M, α, $\{\epsilon_j\}_{j=0}^{M-1}$, $T_{h,\min}$, $T_{h,\max}$, $v_{f,\max}$, $u_{\min}$, $u_{\max}$, $d_{\min}$.
Result: $\hat{\theta}^*$.
initialize $\theta \leftarrow 0$;
for $j = 0 \rightarrow M-1$ **do**
  Measure $v_f(0), d(0), \dot{d}(0)$;
  initialize $t = 0$;
  while $t < T$ **do**
    /* acting step */
    sample random variable $e \sim U((0,1])$;
    if $e \leq \epsilon_j$ **then**
      sample random action $u(t) \sim U(\mathcal{U}(t))$;
    else
      calculate action $u(t) \in \text{argmin}_{u \in \mathcal{U}(t)} Q_\theta(s(t), u)$;
    end
    /* interacting step */
    perform $u(t)$, **observe** $a_f(t+1), v_f(t+1), d(t+1)$, r_{t+1}, and **calculate** $\tilde{r}(t)$ using (27) ;
    /* learning step */
    update θ using Equation (13) ;
    update $t \leftarrow t+1$;
  end
end
return θ

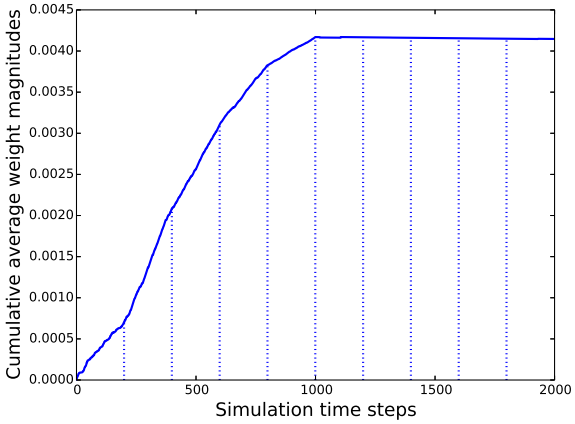


Fig. 6. Cumulative average weight magnitudes (sum of absolute value of the weights divided by the number of weights) over 10 training episodes of IAQL.

combination of zero/one costs) can be actually approximated as a linear combination of given features. As a result, the stationary global optimal policy can be found by using linear regression on Q-values.

Furthermore, our IAQL algorithm also demonstrates a novel sample-efficient RL training strategy. Unlike most existing works that focus on the improvement of exploration strategy and data sampling [25], we develop a surrogate training cost function to guide the gradient updating step size instead. This approach is efficient and does not have side effects

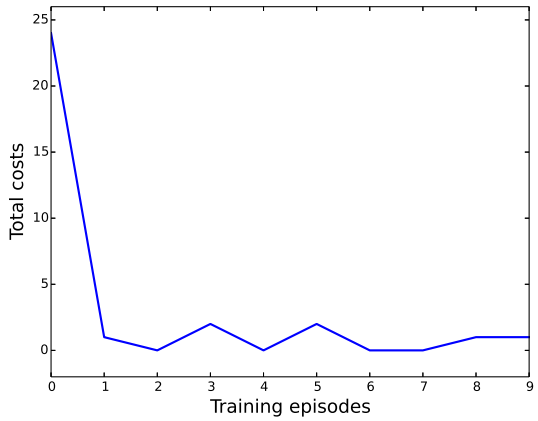


Fig. 7. Total cost vs. training episode during the training procedure of IAQL.

such as making the MDP non-stationary. This idea can be used in many control applications with any learning models (especially useful for data-consuming models such as deep neural networks).

B. Benchmark Comparison: Optimal Velocity Model

We use a widely used car-following model, the optimal velocity model (OVM) [26]–[29], as a benchmark to compare with our IAQL control. Specifically, the driver is modeled as:

$$\begin{aligned} \dot{d} &= v_l - v_f, \\ \dot{v}_f &= \bar{\alpha}(V(d(t-\tau)) - v_f(t-\tau)) \\ &\quad + \beta(v_l(t-\tau) - v_f(t-\tau)), \end{aligned} \tag{38}$$

where $\bar{\alpha}$ represents the driver gain to match the actual velocity to a distance-dependent reference velocity, while β is the driver gain to match the velocity to that of the lead vehicle. The variable τ represents human reaction time, ranging from 0.4 to 1 sec [26]. The function $V(d)$ is a velocity policy mapping from distance headway to reference velocity. In particular, the following monotonically increasing function is adopted:

$$V(d) = \begin{cases} 0 & \text{if } d \leq d_{st} \\ \frac{1}{2} v_{\max}\left[1 - \cos(\pi \cdot \frac{d - d_{st}}{d_{go} - d_{st}})\right] & \text{if } d_{st} < d \leq d_{go} \\ v_{\max} & \text{if } d \geq d_{go} \end{cases} \tag{39}$$

The function $V(d)$ describes the desired speed as a function the distance headway. Specifically, when the distance headway is smaller than d_{st}, the vehicle is expected to stop; while for large distance headway $d > d_{go}$, it is expected to travel with the maximum speed; the desired speed monotonically increases with the headway when between d_{st} and d_{go}.

In this paper, we use $\tau = 1$ and the discretized model with $\Delta = 1$ becomes

$$\begin{aligned} d^+ &= d + (v_l - v_f), \\ v_f^+ &= v_f + \alpha(V(d(t-1)) - v_f(t-1)) \\ &\quad + \beta(v_l(t-1) - v_f(t-1)), \end{aligned} \tag{40}$$

In the simulation, we use the optimized parameters reported in [27], $d_{st} = 10\ m$, $d_{go} = 40\ m$, $v_{\max} = 30\ m/s$, $\bar{\alpha} = 1$, and $\beta = 1.05$.

TABLE II
PERFORMANCE STATISTICS OF DIFFERENT CONTROL METHODS

Control method	Aggressive	Moderate	Conservative	Total
OVM	101	74	44	219
Adaptive OVM	31	19	20	70
IAQL	5	6	3	14

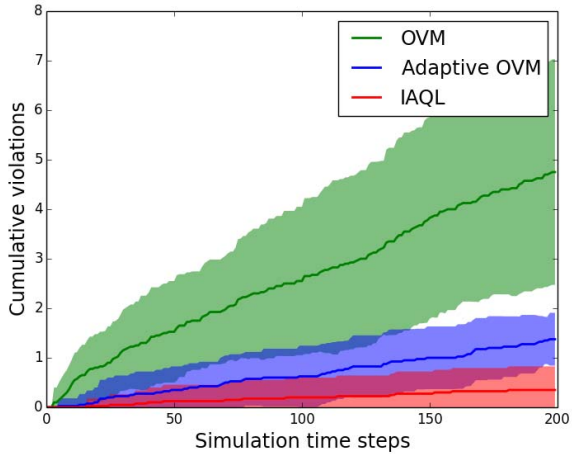


Fig. 8. Performance evaluation of different control policies. The curve shows the mean value and the shade shows the standard deviation of the cumulative cost.

Since the constraints are more towards time headway, to improve the OVM performance, we make d_{st} and d_{go} adaptive as:

$$d_{st}(t) = T_{d,min} v_f(t), \quad d_{go}(t) = T_{d,max} v_f(t). \tag{41}$$

To make the comparison, we use our trained IAQL controller and run the 200-second episode 40 times, together with the OVM and the adaptive OVM. Table II shows the number of constraint violations classified by the lead vehicle driving mode. Each cell records the number of constraint violations. Note the total step number is 8000. It can be seen that our IAQL algorithm clearly outperforms the other two.

The mean and standard deviation of cumulative number of violations over the episodes for the three policies are illustrated in Figure 8. The IAQL algorithm clearly performs better than the heuristic control strategy, as well as its adaptive version. This demonstrates the capability of the proposed IAQL approach when dealing with a fairly complex lead vehicle dynamics (hybrid Markov chain). Although real-world training related safety issues need to be carefully addressed, the developed IAQL system is very applicable to the ACC system.

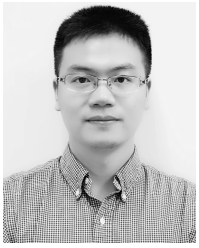
VI. CONCLUSIONS

In this paper, we showed that the drift counteraction optimal control (DCOC) problem can be tackled using reinforcement learning (RL) algorithms, removing requirements of disturbance measurement and knowledge of the disturbance transition. Specifically, we handle the problem of adaptive cruise control with a DCOC formulation. An informed approximate Q-learning algorithm (IAQL) algorithm is developed and the control performance is compared with the optimal vehicle model (OVM) as a benchmark. We show that the IAQL clearly outperforms the benchmarks.

REFERENCES

[1] I. V. Kolmanovsky, L. Lezhnev, and T. L. Maizenberg, "Discrete-time drift counteraction stochastic optimal control: Theory and application-motivated examples," *Automatica*, vol. 44, no. 1, pp. 177–184, Jan. 2008.
[2] R. A. E. Zidek and I. V. Kolmanovsky, "Deterministic drift counteraction optimal control and its application to satellite life extension," in *Proc. 54th IEEE Conf. Decision Control (CDC)*, Dec. 2015, pp. 3397–3402.
[3] I. V. Kolmanovsky and A. A. Menezes, "A stochastic drift counteraction optimal control approach to glider flight management," in *Proc. Amer. Control Conf.*, Jun. 2011, pp. 1009–1014.
[4] I. V. Kolmanovsky and D. P. Filev, "Stochastic optimal control of systems with soft constraints and opportunities for automotive applications," in *Proc. IEEE Control Appl. (CCA), Intell. Control (ISIC)*, Jul. 2009, pp. 1265–1270.
[5] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Univ. Cambridge, Cambridge, U.K., 1989.
[6] A. R. Rosyadi, T. A. B. Wirayuda, and S. Al-Faraby, "Intelligent traffic light control using collaborative Q-Learning algorithms," in *Proc. 4th Int. Conf. Inf. Commun. Technol. (ICoICT)*, May 2016, pp. 1–6.
[7] B. Luo, D. Liu, T. Huang, and D. Wang, "Model-free optimal tracking control via critic-only Q-learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 10, pp. 2134–2144, Oct. 2016.
[8] S. Kosunalp, "A new energy prediction algorithm for energy-harvesting wireless sensor networks with Q-learning," *IEEE Access*, vol. 4, pp. 5755–5763, 2016.
[9] G. Arslan and S. Yüksel, "Decentralized Q-learning for stochastic teams and games," *IEEE Trans. Autom. Control*, vol. 62, no. 4, pp. 1545–1558, Apr. 2016.
[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[11] F. S. Melo and M. I. Ribeiro, *Q-Learning With Linear Function Approximation*. Berlin, Berlin, Germany: Springer, 2007, pp. 308–322.
[12] C. M. C. O. Valle, R. Tanscheit, and L. A. F. Mendoza, "Computed-torque control of a simulated bipedal robot with locomotion by reinforcement learning," in *Proc. IEEE Latin Amer. Conf. Comput. Intell. (LA-CCI)*, Nov. 2016, pp. 1–6.
[13] T. Chu, J. Wang, and J. Cao, "Kernel-based reinforcement learning for traffic signal control with adaptive feature selection," in *Proc. 53rd IEEE Conf. Decision Control*, Dec. 2014, pp. 1277–1282.
[14] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 664–671.
[15] J. Lygeros, D. N. Godbole, and S. Sastry, "Verified hybrid controllers for automated vehicles," *IEEE Trans. Autom. Control*, vol. 43, no. 4, pp. 522–539, Apr. 1998.
[16] R. Muller and G. Nocker, "Intelligent cruise control with fuzzy logic," in *Proc. Intell. Vehicles Symp.*, Jun. 1992, pp. 173–178.
[17] D. Moser, R. Schmied, H. Waschl, and L. del Re, "Flexible spacing adaptive cruise control using stochastic model predictive control," *IEEE Trans. Control Syst. Technol.*, to be published, doi: 10.1109/TCST.2017.2658193.
[18] M. Wang, M. Treiber, W. Daamen, S. P. Hoogendoorn, and B. van Arem, "Modelling supported driving as an optimal control cycle: Framework and model characteristics," *Proc.-Social Behavioral Sci.*, vol. 80, pp. 491–511, Jun. 2013.
[19] I. V. Kolmanovsky, L. Lezhnev, and T. L. Maizenberg, "Discrete-time drift counteraction stochastic optimal control: Theory and application-motivated examples," *Automatica*, vol. 44, no. 1, pp. 177–184, 2008.
[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning—An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[21] F. S. Melo. *Convergence of Q-Learning: A Simple Proof*. Accessed: Nov. 3, 2016. [Online]. Available: http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf
[22] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, vol. 12. Cambridge, MA, USA: MIT Press, 2000, pp. 1057–1063.
[23] D. P. Filev and I. Kolmanovsky, *Markov Chain Modeling and On-Board Identification for Automotive Vehicles*. London, U.K.: Springer, 2012, pp. 111–128.

[24] J. Odentrantz, "Markov chains: Gibbs fields, Monte Carlo simulation, and queues," *Technometrics*, vol. 42, no. 4, pp. 438–439, 2000.
[25] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1054–1062.
[26] G. Orosz, "Connected cruise control: Modelling, delay effects, and nonlinear behaviour," *Vehicle Syst. Dyn.*, vol. 54, no. 8, pp. 1147–1176, 2016.
[27] N. I. Li, C. R. He, and G. Orosz, "Sequential parametric optimization for connected cruise control with application to fuel economy optimization," in *Proc. 55th IEEE Conf. Decision Control*, Dec. 2016, pp. 227–232.
[28] N. I. Li and G. Orosz, "Dynamics of heterogeneous connected vehicle systems," *IFAC-PapersOnLine*, vol. 49, no. 10, pp. 171–176, 2016.
[29] G. Orosz, R. E. Wilson, and G. Stépán, "Traffic jams: Dynamics and control," *Philos. Trans. Roy. Soc. London A, Math. Phys. Sci.*, vol. 368, no. 1928, pp. 4455–4479, 2010.



Zhaojian Li (S'15–M'16) received the B.S. degree in civil aviation from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2010, and the M.S. and Ph.D. degrees from the Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA, in 2014 and 2016, respectively. From 2010 to 2012, he was an Air Traffic Controller with the Shanghai Area Control Center, Shanghai, China. From 2014 and 2015, he was an Intern with Ford Motor Company, Dearborn, MI, USA. Since 2017, he has been an Assistant Professor with the Department of Mechanical Engineering, Michigan State University. His current research interests include optimal control, system modeling, estimation, and intelligent transportation systems. He was a recipient of the National Scholarship from China.

Tianshu Chu received the B.S. degree in physics from Weseda University, Tokyo, Japan, in 2010. He is currently pursuing the Ph.D. degree with the Department of Civil and Environmental Engineering, Stanford University, Stanford, CA, USA.

He is currently a member of the Stanford Center for Sustainable Development and Global Competitiveness. His research interests include reinforcement learning, deep learning, multi-agent learning, and large-scale intelligent traffic signal control.

Ilya V. Kolmanovsky (F'08) received the M.S. and Ph.D. degrees in aerospace engineering and the M.A. degree in mathematics from the University of Michigan, Ann Arbor, MI, USA, in 1993, 1995, and 1995, respectively.

He is currently a Full Professor with the Department of Aerospace Engineering, University of Michigan. His current research interests include control theory for systems with state and control constraints, and control applications to aerospace and automotive systems.

He was with Ford Research and Advanced Engineering, Dearborn, MI, USA, for close to 15 years. He was a recipient of the Donald P. Eckman Award of American Automatic Control Council. He is named as an Inventor on 92 U.S. patents.

Xiang Yin was born in Anhui, China, in 1991. He received the B.Eng. degree from Zhejiang University in 2012, the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 2013 and 2017, respectively, all in electrical engineering. He is currently with the Department of Automation, Shanghai Jiao Tong University. His research interests include supervisory control of discrete-event systems, model-based fault diagnosis, formal methods, and security and their applications to cyber and cyber-physical systems.

Dr. Yin received the Outstanding Reviewer Award from AUTOMATICA in 2016, the IEEE Conference on Decision and Control Best Student Paper Award Finalist in 2016, and the Outstanding Reviewer Award from the IEEE *Transactions on Automatic Control* in 2017. He is the Co-Chair of the IEEE CSS Technical Committee on Discrete Event Systems.