

An Online Evolving Framework for Advancing Reinforcement-Learning based Automated Vehicle Control^{*}

Teawon Han^{*} Subramanya Nageshroa^{**} Dimitar P. Filev^{***}
Ümit Özgüner^{*}

^{*} *Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210, USA (e-mail: han.394@osu.edu, ozguner.1@osu.edu)*

^{**} *Ford Greenfield Labs, Palo Alto, CA 94304, USA (e-mail: snageshr@ford.com)*

^{***} *Ford Motor Company, Dearborn, MI 48121, USA (e-mail: dfilev@ford.com)*

Abstract: In this paper, an online evolving framework is proposed to detect and revise a controller's imperfect decision-making in advance. The framework consists of three modules: the evolving Finite State Machine (e-FSM), action-reviser, and controller modules. The e-FSM module evolves a stochastic model (e.g., Discrete-Time Markov Chain) from scratch by determining new states and identifying transition probabilities repeatedly. With the latest stochastic model and given criteria, the action-reviser module checks validity of the controller's chosen action by predicting future states. Then, if the chosen action is not appropriate, another action is inspected and selected. In order to show the advantage of the proposed framework, the Deep Deterministic Policy Gradient (DDPG) w/ and w/o the online evolving framework are applied to control an ego-vehicle in the car-following scenario where control criteria are set by speed and safety. Experimental results show that inappropriate actions chosen by the DDPG controller are detected and revised appropriately through our proposed framework, resulting in no control failures after a few iterations.

Keywords: evolving controller, reinforcement learning, automated vehicle, machine learning.

1. INTRODUCTION

For many decades, various methodologies have been proposed for safe or efficient automated vehicle (AV) control under different situations. Rule-based, supervised-learning, and reinforcement-learning are widely studied and applied to control AVs.

As a rule-based approach, the Finite State Machines (FSMs) which consist of the finite number of predefined states and transition-conditions has been implemented for decision-making under known situations. In the FSMs, either actions or control-strategies are described in each state, and the transitions occur when pre-determined conditions are met. So, the FSMs based controller returns the appropriate action based on the system's conditions. Kurt and Özgüner (2008) and Redmill et al. (2008) implement the Hierarchical FSMs as a high-level decision-maker within the Hybrid State System (HSS) to control automated vehicles (AVs) under the given several scenarios in the 2007 DARPA Urban Challenge. Although the AV (called OSU-ACT) is controlled via appropriate decision-making under complex situations, the capability of the rule-based controller is fully subordinate to the initially defined rules and conditions. For example, if unexpected situations are encountered, the controller can neither rec-

ognize the situations nor make the right decision due to no existing proper rules.

The Supervised Learning (SL) is focused on seeking the best coefficients or weights of the predefined control models by using a given cost-function and ground truth data. The data consists of pairs of inputs (situations or conditions) and outputs (optimal actions), and the optimal coefficients or weights are obtained by exploring the configuration which minimizes differences between the ground truth and the model's output. In Gadepally et al. (2013); Kuge et al. (2000), the Hidden Markov Model (HMM) is implemented for obtaining a general driver model that enables AV controllers to predict other vehicles' behaviors, which is helpful for safer driving control. The HMM-based driver model is designed by the fixed number of hidden states and observations, and the transition and emission probabilities are identified via the Viterbi algorithm. In addition, various types of Deep Neural Network (DNN) are implemented to create AV control or driver model for predicting future traffic situations or choosing optimal actions respectively. Han et al. (2019b) proposes to train DNN by using raw observations and estimated general driving characteristics to predict other vehicle's lane-change behavior on the highway. Also, the end-to-end learning-based driving controller, which maps raw camera-images with optimal controls via the Convolutional Neural Network

^{*} This work was supported by the Ford Motor Company

(CNN), is derived by Bojarski et al. (2016). Al-Qizwini et al. (2017) implements the GoogLeNet to obtain accurate affordance parameters which are essential to take the optimal driving actions. Even though the SL approach has been widely used to get optimal models, a large amount of labeled data and off-line training procedure are required.

The Reinforcement Learning (RL) approach has been suggested to get an optimal controller which can improve its performance by updating its policy iteratively based on experiences. Unlike the SL, the ground truth data and pre-training process are not necessary, but a reward-function needs to be pre-defined appropriately. While the RL based controller explores or exploits actions at different states, its policy is updated repeatedly in a way to get the highest value (sum of future discounted rewards) at every state. The policy is represented by matrices or DNN. Since the DNN can well approximate value-function, various types of multilayer perceptron (MLP) have been proposed to represent the RL's policy. Mnih et al. (2015) proposes Deep Q Network (DQN), and Van Hasselt et al. (2016) suggests Double Deep Q-Network (DDQN) which can prevent the over-optimistic value estimation problem observed in the DQN. Since DQN and DDQN can only handle the discrete type of actions, Deep Deterministic Policy Gradient (DDPG) is proposed by Lillicrap et al. (2015) for the continuous type of actions. Nagesh Rao et al. (2019b) proposed the DRL agent control architecture, which implements DDQN for safe and adaptive AV control.

Due to the model-free attribution, the RL with MLP controller can find the optimal actions under unexpected situations, unlike the rule-based and SL approaches. However, a well-designed reward-function is still required. In this paper, instead of paying efforts for designing a perfect model or reward-function by hand, we propose an evolving methodology (called evolving Finite State Machine), which can derive a stochastic model from scratch by observing variations of system-conditions. Also, the derived stochastic model is used within the online evolving framework to check and revise the action which is chosen by the controller.

This paper is organized as follows: Section 2 describes the details of three modules within an on-line evolving framework. Section 3 introduces the DDPG AV controller and the car-following scenario settings. Section 4 explains the on-line evolving framework settings for the DDPG controller. Section 5 analyzes car-following performances of the DDPG controller w/ and w/o the suggested framework. Lastly, our contributions and further discussion are summarized in section 6.

2. AN ON-LINE EVOLVING FRAMEWORK

An on-line evolving framework is proposed to inspect or revise an action selected by a controller. In order to inspect the action, it needs to predict future states depending on the action. In the framework shown in Fig. 1, there are a controller, evolving Finite State Machine (e-FSM), and action-reviser modules for choosing an optimal action, creating a stochastic model, and inspecting & revising the controller's actions respectively.

A set of observations z_t at time-step t is fed into the controller and e-FSM modules. The control module makes

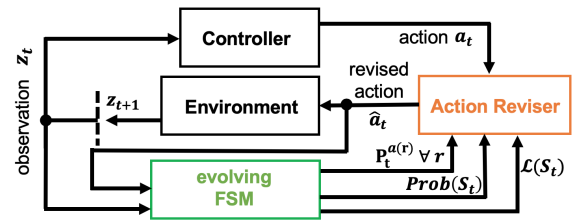


Fig. 1. An on-line evolving framework for controllers

a decision to choose the best action; the e-FSM module derives a stochastic model which consists of states and state-transition dynamics based on the applied actions; the action-reviser module checks validity of the chosen action a_t and revises the action if necessary. Note that a_t , $Prob(S_t)$, $P_t^{a(r)}$, and $L(S_t)$ refer to an action chosen by the controller at time t , probability distributions over states (e.g., driving situations) which are determined by time t , identified state-transition probabilities with respect to actions, and a set of state attributions. In the following sub-sections, details of the e-FSM and the action-reviser modules are described.

2.1 evolving Finite State Machine(e-FSM)

The e-FSM, an on-line evolving method designed for supporting optimal decision-making, is a hybrid Markov model with states representing specific situations. The states are identified as clusters in the state space, and state dynamics is determined through a set of transition probability matrices associated with the inputs (actions). The framework of e-FSM is shown in Fig. 2, and some of the key features (online state-determination, state flagging and recognition, and online transition-identification) are described in this paper; see Han et al. (2019a) for e-FSM's further specific properties and descriptions.

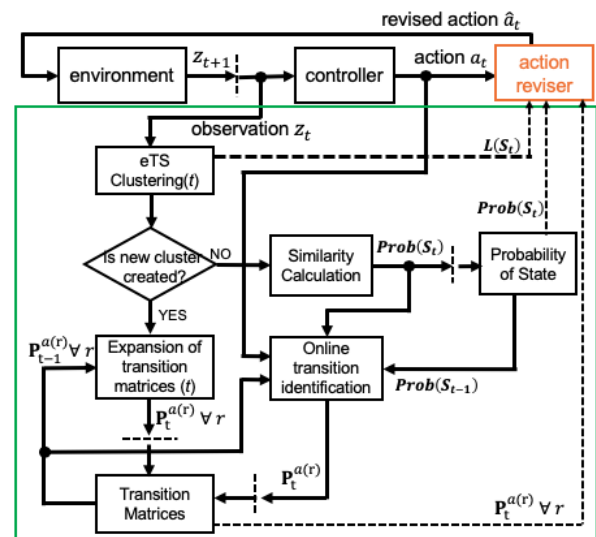


Fig. 2. The framework of an evolving Finite State Machine

- *Online State Determination* In the e-FSM, states represent situations that a controller could be encountered. Due to the difficulty of pre-determining all possible situations, the e-FSM aims to determine a state whenever a new situation is observed over time. This feature enables the controller to have opportunities to find the best action by recognizing initially unexpected situations.

To realize the online state determination, a set of observations $z_t = [o_t(1), \dots, o_t(k)]^T$ at every time t is clustered, and each cluster-center is considered as a state, representing individual situation. The evolving Takagi-Sugeno (eTS) online clustering method proposed by Filev and Kolmanovsky (2010) is implemented so that infinite number of unknown states can be determined; the eTS can have infinite number of clusters. A state-set can be written by $\mathcal{S}_{t \geq 1} = \{s_t(1), \dots, s_t(n_t)\}$ where the states $s_t(i), 1 \leq i \leq n_t$, are cluster-centers, and n_t is the total number of the states determined by time t . This feature enables the structure of the stochastic model to evolve.

- *Flagging and Recognition of State* The e-FSM determines states, in which each represents a distinguished situation. Instead of labeling all states, the states are flagged by either unfavorable states or not, where the unfavorable states are defined based on the given criteria. For example, if a given criterion of AV control is safety, the recognized state at the moment of collision is flagged as the unfavorable state. A set of the state attributions at time t , $\mathcal{L}(S_t)$, is transferred to the action-reviser as shown in Fig. 1, where $\mathcal{L}(s_t(i)) \in \{none, unfavorable_j\}, 1 \leq i \leq n_t, 1 \leq j \leq c_n; c_n$ is the total number of given criteria.

Given z_t , the current situation is represented by probability distributions over the states determined by time t , denoted as $Prob(S_t) = [Prob(s_t(1)), \dots, Prob(s_t(n_t))]^T$, where each probability of state $Prob(s_t(\cdot))$ is obtained by calculating a similarity, $\lambda_t^i(z_t)$, between z_t and $s_t(i), 1 \leq i \leq n_t$ as defined in Eq.(1).

$$\lambda_t^i(z_t) = \frac{\eta_t^i(z_t)}{\sum_j \eta_t^j(z_t)} \quad (1)$$

$$\text{where } \eta_t^i(z_t) = \exp\left(-\frac{(z_t - s_t^i)^T(z_t - s_t^i)}{\text{var}(s_t^i)}\right)$$

- *Online State-Transitions Identification* The identification of state-transitions w.r.t. actions is important to predict future states precisely. For example, if an AV controller knows in advance what the future state would be based on the chosen action, it can get more chances to make a better decision. In the e-FSM, the state-transitions are represented by transition probability matrices (TPMs), of which each matrix is correlated to individual action in the action-set.

The action-set written by $\mathcal{A}_d = \{a(1), \dots, a(q)\}$ is initially defined to have the fixed q number of elements unlike the state-set \mathcal{S}_t which is a variant set. The e-FSM uses discrete action-set but is compatible to controllers that have continuous action-set (\mathcal{A}_c) by encoding to discrete action-set (\mathcal{A}_d) with an arbitrary interval δ . For example, the continuous action-set $\mathcal{A}_c = [-2.0 \ 2.0]$ can be encoded with the interval $\delta = 1.0$ to the discrete action-set such as $\mathcal{A}_d = \{a(1), a(2), a(3), a(4)\}$ where $a(1) = [-2.0 \ -1.0), a(2) = [-1.0 \ 0.0), a(3) = [0.0 \ 1.0), a(4) = [1.0 \ 2.0]$.

The state-transitions w.r.t. actions are represented by the q number of TPMs. And, each TPM $\mathbf{P}_t^{a(r)} \forall r \in R^{n_t \times n_t}$ is correlated to each discrete action $a(r) \in \mathcal{A}_d$, where $1 \leq r \leq q$ and q is the total number of discrete actions in \mathcal{A}_d . The TPMs are defined by:

$$\mathbf{P}_t^{a(r)} = \{P_t^{a(r)}(i, j)\}, 1 \leq i, j \leq n_t$$

$$\text{where } P_t^{a(r)}(i, j) = Prob(s_{t+1} = s(j) | s_t = s(i), a_t = a(r)) \quad (2)$$

When a new state is determined, the dimension of all TPMs is expanded by adding a column and a row, following the proposed method; see detail steps and an example given by Han et al. (2019a). Otherwise one of TPMs which is correlated to a chosen action is identified via Eq.(3) and (4) in a recursive way, designed by Filev and Kolmanovsky (2010). Note $\tau(t)$ and $\gamma(t)$ are probability distributions over states at time $t-1$ and t , $Prob(S_{t-1})$ and $Prob(S_t)$; φ and 1_{n_t} are a learning rate and the n_t -dimensional vector of ones respectively. In the beginning, $F^{a(r)}(0)$ and $F_o^{a(r)}(0)$ are initialized by $\bar{\epsilon}$ which is a small non-negative constant.

$$\mathbf{P}_t^{a(r)} = \text{diag}(F_o^{a(r)}(t))^{-1} F^{a(r)}(t) \quad (3)$$

$$F^{a(r)}(t) = F^{a(r)}(t-1) + \varphi(\tau(t)\gamma(t)^T - F^{a(r)}(t-1))$$

$$F_o^{a(r)}(t) = F_o^{a(r)}(t-1) + \varphi(\tau(t)\gamma(t)^T 1_{n_t} - F_o^{a(r)}(t-1)) \quad (4)$$

By using identified TPMs, the future states can be predictable by calculating probability distributions over states at time $t+1$. Given probability distributions over states at t and an action selected by the controller at t , probability distributions over state at $t+1$ or $t+k$ can be obtained by Eq.(5) and (6). As shown in Eq.(7), \mathbf{P}_t^* is a marginal TPM, where $Prob(a_t = a(r))$ is set by uniform distribution such as $1/q \forall r$.

$$Prob_{pred}(S_{t+1}) = \mathbf{P}_t^{a(r)} \cdot Prob(S_t) \quad (5)$$

$$Prob_{pred}(S_{t+k}) = \mathbf{P}_t^* \cdot Prob(S_{t+1}) \quad (6)$$

$$\mathbf{P}_t^* = \{P_t^*(i, j)\}, 1 \leq i, j \leq n_t, \text{ where}$$

$$P_t^*(i, j) = \sum_{r=1}^q Prob(s_{t+1} = s(j) | s_t = s(i), a_t = a(r)) \cdot Prob(a_t = a(r)) \quad (7)$$

2.2 Action-Reviser Module

This module inspects and revises an action a_t chosen by a controller if required. The revision of chosen action is decided based on predicted probabilities of unfavorable states. Given $\mathcal{L}(S_t)$ and a_t , if predicted probabilities of the unfavorable states are higher than a threshold ϱ_t , a_t will be revised. Otherwise, a_t is applied.

The predicted probability distributions of states at $t+1$, $Prob_{pred}(S_{t+1})$, can be obtained by using Eq. (5), where $\mathcal{L}(S_t)$, $Prob(S_t)$ and $\mathbf{P}_t^{a(\cdot)}$ are provided from the e-FSM module, and the action $a(r)$ is chosen by a controller. The variant threshold ϱ_t is set such that if a variance of $Prob_{pred}(S_{t+1})$ is larger, then ϱ_t is smaller and vice versa as shown in Alg. 1.

Algorithm 1 *getVariantThreshold* ($Prob_{pred}(S_{t+1})$)

- 1: **procedure** $\varrho_t = f_{threshold}(Prob_{pred}(S_{t+1}))$
 - 2: $X \leftarrow Prob_{pred}(S_{t+1})$
 - 3: $X \leftarrow \text{Sort } X \text{ by descending order}$
 - 4: $E[X] = \sum_{j=1}^{n_t} j \cdot X(j)$
 - 5: $\varrho_t \leftarrow X(\text{floor}(E[X]))$
-

For revising the chosen action, the action-reviser inspects other possible actions. For example, assuming

as the unfavorable states, which represent either unsafe (collision) or slow-speed driving situation (*headway_{dist}* is longer than 200[m]). At every time-step, $s_t(i)$, where $\{s_t(i) \in S_t | i = \text{argmax}_i(\text{Prob}(s_t(i)))\}$, is flagged by:

$$\mathcal{L}(s_t(i)) = \begin{cases} l_n(\text{none}) : \text{neither crash nor large-distance} \\ l_s(\text{unfavorable}_{\text{safety}}) : \text{crash} \\ l_{ld}(\text{unfavorable}_{\text{speed}}) : \text{large-distance} \end{cases}$$

As shown in Eq.(13), three observations are selected to represent states in the e-FSM. The coefficients of eTS within the e-FSM module, ρ and ϵ , are set by 0.7 and 0.3 respectively. Since the DDPG controller uses the continuous action-space, the continuous action-set is encoded to the discrete action-set by using an interval $\delta = 0.2[m/s^2]$. The continuous action set $A_c = [-2.0 \ 2.0][m/s^2]$ is given for the both vehicles in the scenario, and encoded to the $q = 20$ number of discrete actions.

$$z_t = \{v_t(\text{veh}_{ego}), \text{headway}_{\text{dist}}, v_t(\text{veh}_{lead})\} \quad (13)$$

4.2 Action-Reviser Configurations

If it is predicted that the selected action a_t causes a transition to one of the unfavorable states, the action-reviser explores and selects another action as discussed in section 2.2.

The DDPG controller chooses an action $a_t \in A_c$ at every time-step, and a_t is encoded to a discrete action $a(r) \in A_d$. Then, future probability distributions over states are calculated via Eq.(5) to inspect $a(r)$. In the predicted probability distributions, if any of the unfavorable states (l_s or l_{ld}) have a higher probability than the dynamic threshold ρ_t , the action-reviser revises $a(r)$. Otherwise, a_t is returned and applied. The action-reviser continues exploring a discrete action space (A_d) until an appropriate action is found. Specific steps are shown in Alg. 2.

After action-revising process, $a(r)$ is decoded to a continuous action \hat{a}_t by calculating a mean of corresponded range since each discrete action represents an individual

Algorithm 2 *actionsReviser*

```

1: procedure  $\hat{a}_t = f_{AR}(\text{Prob}(S_t), a_t, A_d, \mathbf{P}_t^{a(\cdot)}, \mathcal{L}(S_t))$ 
2:    $r \leftarrow \text{encode}(a_t, A_d)$ 
3:    $\text{Prob}_{\text{pred}}(S_{t+1}) \leftarrow \mathbf{P}_t^{a(r)} \cdot \text{Prob}(S_t)$ 
4:    $I \leftarrow f_{IA}(\text{Prob}_{\text{pred}}(S_{t+1}), \mathcal{L}(S_t))$  ▷ Alg. 3
5:   if  $I = 0$  then ▷  $l_s$  is expected
6:     while  $I = 0$  &  $r > 0$  do
7:        $r \leftarrow r - 1$ 
8:        $\text{Prob}_{\text{pred}}(S_{t+1}) \leftarrow \mathbf{P}_t^{a(r)} \cdot \text{Prob}(S_t)$ 
9:        $I \leftarrow f_{IA}(\text{Prob}_{\text{pred}}(S_{t+1}), \mathcal{L}(S_t))$ 
10:     $\bar{a}_t \leftarrow \text{decode}(a(r), A_d) + \hat{\mathcal{N}}(0, V)$ 
11:    return  $\hat{a}_t \leftarrow \min(\max(\bar{a}_t, a_{\min}), a_{\max})$ 
12:   else if  $I = 1$  then ▷  $l_{ld}$  is expected
13:     while  $I = 1$  &  $r < q$  do
14:        $r \leftarrow r + 1$ 
15:        $\text{Prob}_{\text{pred}}(S_{t+1}) \leftarrow \mathbf{P}_t^{a(r)} \cdot \text{Prob}(S_t)$ 
16:        $I \leftarrow f_{IA}(\text{Prob}_{\text{pred}}(S_{t+1}), \mathcal{L}(S_t))$ 
17:     $\bar{a}_t \leftarrow \text{decode}(a(r), A_d) + \hat{\mathcal{N}}(0, V)$ 
18:    return  $\hat{a}_t \leftarrow \min(\max(\bar{a}_t, a_{\min}), a_{\max})$ 
19:   else ▷  $l_n$  is expected
20:     return  $\hat{a}_t \leftarrow a_t$ 

```

Algorithm 3 *inspectAction*

```

1: procedure  $I = f_{IA}(\text{Prob}_{\text{pred}}(S_{t+1}), \mathcal{L}(S_t))$ 
2:    $\rho_t \leftarrow f_{\text{threshold}}(\text{Prob}_{\text{pred}}(S_{t+1}))$  ▷ Alg. 1
3:   for  $i = 1, 2, \dots, n_t$  do
4:     if  $\text{Prob}_{\text{pred}}(s_{t+1}(i)) \geq \rho$  then
5:       if  $\mathcal{L}(s_{t+1}(i)) = l_s$  then
6:         return  $I \leftarrow 0$ 
7:       else if  $\mathcal{L}(s_{t+1}(i)) = l_{ld}$  then
8:         return  $I \leftarrow 1$ 
9:       else
10:        return  $I \leftarrow 2$ 

```

partial range of the continuous action-set. For example, if the selected discrete action is $a(r) = [0.2 \ 0.4]$, 0.3 is returned as a revised action \hat{a}_t . In addition, the Gaussian noise with a decreasing variance $\hat{\mathcal{N}}(0, V)$ is defined as shown in Eq.(14) and implemented to explore the action space for identifying the e-FSM's TPMs.

$$\text{where } V = \frac{a_{\max}}{\max(1, (K * \text{Iter}_{\text{episode}}))} \quad (14)$$

Initially, V is set by the maximum acceleration of the vehicle, but it is decreased over the number of the episode iterations. Note K, a_{\max}, a_{\min} , and $\text{Iter}_{\text{episode}}$ are an invariant constant, the maximum/minimum accelerations of the vehicle, and the number of episode iterations. Since the e-FSM does not have any states in the beginning, appropriate inspection and revision of the chosen action are not possible. Thus, the action-reviser module is activated after the ξ number of episode iterations; K and ξ are set by 0.001 and 50 in this study.

5. EXPERIMENTAL RESULTS

The car-following control performances of the DDPG w/ and w/o the on-line evolving framework are compared in terms of *safety* and *speed* criteria. In the beginning of every first episode, the DDPG and e-FSM modules are initialized such that weights of critic and actor networks in the DDPG are randomly assigned, the state-set of e-FSM is set by empty-set $S_0 = \{\}$, and $F_o^{a(r)}(0)$ and $F^{a(r)}(0)$ are set by $\bar{\epsilon}$ for all r . While running the scenario repeatedly, the critic and actor networks are updated, and the e-FSM derives a stochastic model by determining states and identifying state-transitions over time.

Since every episode is terminated when the controller fails to avoid collisions or maintain *headway_{dist}* less than 200[m], the controller's performance can be analyzed by comparing the total number of simulated steps in each episode, where the maximum step is 800. The Fig. 5 plots the evolution of the car-following performances for DDPG w/ and w/o the on-line evolving framework averaged over 10 different runs; 1500 episodes are iterated in each run. The different number of succeeded and failed episodes (in total 15000 episodes) are observed depending on the implementation of the proposed framework as below table.

Type	Success	Large-distance	Collision
DDPG	12249 (81.6%)	1251	1500
Our Approach	14719 (98.1%)	76	205

As shown in Fig. 5, the control performances of the both approaches are enhanced over iterations. But, the DDPG

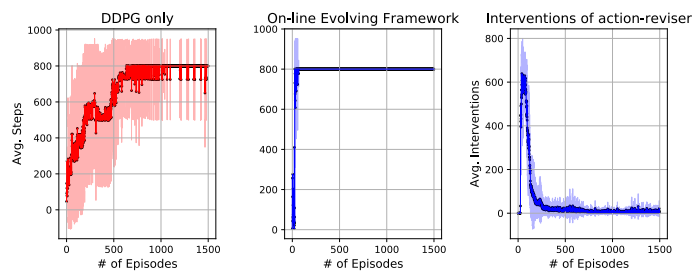


Fig. 5. The control performance evolution in car-following by DDPG w/o (left) and w/ on-line evolving framework (center). The averaged number of interventions for revising actions chosen by DDPG (right).

w/ the on-line evolving framework (proposed approach) achieves the given control-goal (driving safe and fast) with fewer iterations. Also, our approach does not fail after 68 iterations, but the DDPG w/o the framework does. It is observed that the number of interventions for revising actions is decreasing as DDPG policy is more optimized; there are no interventions during the first 50 episodes because the action-reviser is inactivated. These results imply that the DDPG controller's wrong decision-making is detected and revised appropriately within our proposed framework. Besides, it is noticed that a stochastic model is properly created by the e-FSM module, enabling precise prediction of future states based on the choice of actions.

Since the available maximum speed of ego vehicle depends on the lead vehicle's speed in the car-following, the best way to satisfy the speed criterion is driving as fast as the lead vehicle. Thus, the speed difference between ego and lead vehicles among succeeded episodes is quantified. The overall mean and variance of the velocity difference are 2.0460 and 1.7800 for the DDPG with the on-line evolving framework, while 1.9497 and 1.8057 for the DDPG only.

Although the DDPG w/ the on-line evolving framework controls the ego vehicle slightly slower than the DDPG only, it achieves both criteria, safety and speed, more successfully by trading off priority of the two criteria.

6. DISCUSSION AND CONCLUSION

In this work, the on-line evolving framework is proposed to advance the decision-making capability of controllers. Since the e-FSM module can create a stochastic model by evolving the model's structure and identifying state-transition dynamics through experiences, it becomes possible to predict future states based on the choice of actions. The action-reviser module checks if an action chosen by the controller causes unfavorable situations in the future. And, if one of the unfavorable situations is highly expected, it explores the given action-space to choose another action.

In the given car-following scenario, DDPG w/ and w/o the on-line evolving framework are implemented to control the ego vehicle as *fast* and *safe* as possible. As shown in the experimental results, the crashes and large-distances does not occur with our proposed framework after few episode iterations whereas the control failures are continuously occurred w/o the framework. It shows that the e-FSM generates a stochastic model precisely which enables to detect incorrect decisions effectively and revise them appropriately within the on-line evolving framework.

REFERENCES

- Al-Qizwini, M., Barjasteh, I., Al-Qassab, H., and Radha, H. (2017). Deep learning algorithm for autonomous driving using googlenet. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 89–96. IEEE.
- Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.
- Filev, D.P. and Kolmanovsky, I. (2010). Markov chain modeling approaches for on board applications. In *Proceedings of the 2010 American Control Conference*, 4139–4145. IEEE.
- Gadepally, V., Krishnamurthy, A., and Ozguner, U. (2013). A framework for estimating driver decisions near intersections. *IEEE Transactions on Intelligent Transportation Systems*, 15(2), 637–646.
- Han, T., Filev, D., and Ozguner, U. (2019a). An online evolving framework for modeling the safe autonomous vehicle control system via online recognition of latent risks. *arXiv preprint arXiv:1908.10823*.
- Han, T., Jing, J., and Özgüner, Ü. (2019b). Driving intention recognition and lane change prediction on the highway. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, 957–962. IEEE.
- Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kuge, N., Yamamura, T., Shimoyama, O., and Liu, A. (2000). A driver behavior recognition method based on a driver model framework. Technical report, SAE Technical Paper.
- Kurt, A. and Özgüner, Ü. (2008). Hybrid state system development for autonomous vehicle control in urban scenarios. *IFAC Proceedings Volumes*, 41(2), 9540–9545.
- Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.
- Nagesh Rao, S., Costa, B., and Filev, D. (2019a). Interpretable approximation of a deep reinforcement learning agent as a set of if-then rules. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 216–221. IEEE.
- Nagesh Rao, S., Tseng, H.E., and Filev, D. (2019b). Autonomous highway driving using deep reinforcement learning. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2326–2331. IEEE.
- Redmill, K.A., Ozguner, U., Biddlestone, S., Hsieh, A., and Martin, J. (2008). Ohio state university experiences at the darpa challenges. *SAE International Journal of Commercial Vehicles*, 1(2008-01-2718), 527–533.
- Uhlenbeck, G.E. and Ornstein, L.S. (1930). On the theory of the brownian motion. *Physical review*, 36(5), 823.
- Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.