Acceleration control strategy for Battery Electric Vehicle based on Deep Reinforcement Learning in V2V driving

Acceleration control strategy for Battery Electric Vehicle based on Deep Reinforcement Learning in V2V driving

Matteo Acquarone
Department of Energetic (DENERG), Politecnico di Torino
Center for Automotive Research and Sustainable mobility (CARS), Politecnico di Torino
Torino, IT
matteo.acquarone@polito.it

Angelo Borneo
Department of Energetic (DENERG), Politecnico di Torino
MCA Engineering s.r.l., Torino
Torino, IT
angelo.borneo@polito.it

Daniela Anna Misul
Department of Energetic (DENERG), Politecnico di Torino
Center for Automotive Research and Sustainable mobility (CARS), Politecnico di Torino
Torino, IT
daniela.misul@polito.it

***Abstract*— The transportation sector is seeing the flourishing of one of the most interesting technologies, autonomous driving (AD). In particular, Cooperative Adaptive Cruise Control (CACC) systems ensure higher levels both of safety and comfort, enhancing at the same time the reduction of energy consumption. In this framework a real-time velocity planner for a Battery Electric Vehicle, based on a Deep Reinforcement Learning algorithm called Deep Deterministic Policy Gradient (DDPG), has been developed, aiming at maximizing energy savings, and improving comfort, thanks to the exchange of information on distance, speed and acceleration through the exploitation of vehicle-to-vehicle technology (V2V). The aforementioned DDPG algorithm relies on a multi-objective reward function that is adaptive to different driving cycles. The simulation results show how the agent can obtain good results on standard cycles, such as WLTP, UDDS and AUDC, and on real-world driving cycles. Moreover, it displays great adaptability to driving cycles different from the training one.**

Keywords*— *Cooperative Adaptive Cruise Control, V2V, Deep Reinforcement Learning, DDPG, Battery Electric Vehicle

I. INTRODUCTION

Advanced driver assistance systems (ADASs) are promising technologies to meet the crucial need of increasing safety, comfort and, potentially, energy savings. Among ADASs technologies [1], cooperative adaptive cruise control (CACC)[2] enables the Ego vehicle to maintain a certain speed and to satisfy other constraints such as the appropriate distance between vehicles. This feature is empowered by communication between the vehicle and the environment (V2X). Among the presented systems, the vehicle-to-vehicle (V2V) interaction considers the Ego vehicle in a car-following scenario, receiving information in terms of speed and acceleration from the vehicle ahead. The benefits that can be achieved by exploiting the aforementioned information for the control strategy of the Ego, are significant for both comfort and energy saving. Although several works are focused on guaranteeing comfort and keeping a safe inter-vehicle distance (IVD) [3], relatively few of them concentrated their attention on the promising energy-saving potential for connected vehicles [4]. There are a variety of algorithms that can be applied to the problem to obtain optimal results with CACC. Firstly, global optimization could be achieved through dynamic programming (DP) [5] which is generally used as a benchmark for energy management strategies, although is not applicable online. To achieve a real-time control strategy, one of the most effective methods is the Equivalent Consumption Minimization Strategy (ECMS)[6]. Moreover, in several works Model Predictive Control (MPC) [7] [8]has been suggested as one of the most promising solutions to CACC problems. In the last years, the innovations made in the field of Reinforcement Learning (RL) have allowed researchers to achieve surprising results in energy management problems [9]. Recently, Deep RL algorithms have been applied to this field to surpass the curse of dimensionality typical of Q-learning and DP due to the discretization of large and continuous state and action spaces [10] [11]. The main goal of this paper is to show the efficacy of the DDPG, a Deep RL algorithm, for optimal acceleration control of an Ego electric vehicle (EV), not only to enhance comfort conditions as in previous work [12], but also to achieve energy savings objective in both standard and real-world driving cycles.

II. VEHICLE MODEL AND DDPG ALGORITHM

A. Vehicle model

The vehicle model, considered for this study, refers to a Battery Electric Vehicle (BEV) [6], whose data are found in the literature [13] and is developed in Python. The vehicle's behaviour is represented through a quasi-static approach. The vehicle is represented through the exploitation of the information regarding instantaneous speed and acceleration to compute the power needed by the battery at each timestep. Higher-order dynamics are neglected. Considering the single-speed transmission of the model, the torque T_{EM}, required at the electric machine to overcome the resistive load of the road and accelerate the vehicle, is obtained as:

$$T_{EM} = \frac{T_{OUT}}{\eta_{transm}^{sign(T_{OUT})}} / \tau_{fin} \quad (1)$$

where τ_{fin} is the final drive ratio of the vehicle, η_{transm} is the transmission efficiency and T_{OUT} is the sum of the inertial torque, road slope and road load torque, which is the overall requested torque at the transmission's outlet:

$$T_{OUT} = m \cdot g \cdot sin\,(\alpha) \cdot r_w + m \cdot a \cdot r_w + T_{RL} \quad (2)$$

$$T_{RL} = (RL_A + RL_B \cdot v + RL_C \cdot v^2) \cdot r_w \quad (3)$$

where RL_A, RL_B, RL_C are the three road load coefficients which model the road resistance; r_w is the wheel radius; α is the road inclination; g is the gravity acceleration; m, v and a are the mass, velocity and acceleration of the vehicle. When the torque requested to the electric machine is known, the battery power request can be computed as the aforementioned torque times the angular velocity of the electric machine. Its power losses and the power required by the auxiliaries are also included. Finally, the following equations have been used to represent the behaviour of the battery. The Rint model allows to calculate the current flow through the battery and the instantaneous change in the State Of Charge ($S\dot{O}C$):

$$I_{batt} = \frac{V_{batt} - \sqrt{V_{batt}^2 - 4R_{batt}P_{batt}}}{2R_{batt}} \quad (4)$$

$$S\dot{O}C = \frac{I_{batt}}{Q_{batt} \cdot \Delta t} \quad (5)$$

Where I_{batt} is the current flowing in the battery, V_{batt} and R_{batt} are its open-circuit voltage and internal resistance, Q_{batt} and Δt are the battery maximum capacity in ampere-seconds and the timestep.

B. DDPG algorithm

RL algorithms train the decision-maker, the agent, in order to obtain an optimal policy in an external environment through a trial and error learning process. At a general time step t, the environment is characterized by an observable state s: the agent takes an action a and receives a reward *r* that tells the agent how good the action taken for that particular state s is. The main objective of RL agents is to maximize the sum of rewards obtained during the driving mission. Since the agent learns directly from the interaction with the environment, as opposed to Supervised Learning, no prior data is required before starting the training process which lasts E_{max} episodes. The latter was set to 2000. The DDPG algorithm [14] used in this work is derived from the simpler Q-learning. The agent tries to estimate the action-value $Q(s, a)$ of the tuples state-action that is defined by the estimate of the discounted sum of rewards, also called discounted return. The discount factor γ is set to 0.99. Some Deep Reinforcement Learning (DRL) algorithms, such as DDPG, estimate the Q-values through deep neural networks (NN), function approximators, solving the problem of discretization of variables that affects tabular Q-learning. In particular, DDPG is an actor-critic algorithm that allows the use of continuous action and state spaces. The agent is made of four different simple feed-forward neural networks: two actors μ and two critics Q characterized by their respective weights θ^{μ} and θ^{Q}. Each net contains a single hidden layer with 64 neurons. The Rectified Linear Unit (ReLU) activation function was used. The output of actor networks is the action given the input state, while the critic nets allow obtaining the estimates of Q values to a particular state and action. It is necessary to use the target actor μ_t and critic Q_t networks in order to achieve stability of Q-values that could otherwise diverge. Their weights θ^{μ_t} and θ^{Q_t}, at first initialized equal to θ^{μ} and θ^{Q} respectively, are updated every episode using soft target update. The complex problem typical of RL algorithms of balancing exploration and exploitation is managed by adding Gaussian noise with mean and standard deviation respectively of 0 and 0.1. Another significant ingredient in the DDPG algorithm is the experience replay memory where the data needed to update neural network parameters during training are stored. In every time step, the replay memory of capacity *N* receives and stores a tuple (s, a, r, s') containing the current state s, the current action a, the presently obtained reward r and the next state s' of the environment. For every training iteration, a batch of n random tuples are sampled from the memory and are used to train the critic and actor networks through the respective loss functions L_c and L_a:

$$y = (r + \gamma Q_t(s', \mu_t(s'|\theta^{\mu_t})|\theta^{Q_t}) \quad (6)$$

$$L_c = \frac{1}{n}\sum_{i=1}^{n}(y - Q(s, a|\theta^{Q}))^2 \quad (7)$$

$$L_a = \frac{1}{n}\sum_{i=1}^{n} Q\big(s, \mu(s)\big) \quad (8)$$

The general framework of the DDPG algorithm is shown in TABLE *I*.

TABLE I. DDPG ALGORITHM FRAMEWORK

Algorithm 1 DDPG Algorithm	
1:	Select the driving cycle of the Lead vehicle
2:	Randomly initialize critic and action network parameters
3:	Initialize experience replay memory
4:	Initialize target networks
5:	**for** episode =1 to Emax do
6:	Receive initial state
7:	**for** t = 1 to time length of the driving cycle T do
8:	Output action from the actor network and add a random noise for action exploration
9:	Execute action *a* and observe reward *r*, new state *s'* from the vehicle model
10:	Store the tuple (*s,a,r,s'*) in the replay memory
11:	Sample a random minibatch of *n* tuples from the replay memory
12:	Update critic networks parameters by minimizing *Lc*
13:	Update actor networks parameters by minimizing *La*
14:	Update the target networks parameters
15:	**end for**
16:	**end for**

III. Control Strategy

The driving scenario of this work comprehends two vehicles travelling in the same direction on a straight road. For simplicity, the driving cycles are considered without slope. The follower vehicle, called Ego, receives information about the other vehicle, called Lead. In this work, we make the strong assumption that the information regarding the Lead are acquired by the Ego instantaneously and without any error since the main focus of the paper is to show the potential of DRL techniques to achieve good results in energy savings and comfort. So, the Lead velocity, acceleration and inter-vehicle distance are passed to the DRL agent together with the information of the Ego. In this work, the authors choose the following state variables observable from the environment: the velocity of the Ego vehicle, the velocity and acceleration of the Lead vehicle and the IVD. This amount of input data allows the agent to have every important information to achieve the desired objectives. The choice of a large number of state variables is possible thanks to the use of deep neural networks which can take several inputs. The main objectives of the RL agent are to obtain maximum energy saving and guarantee comfortable driving conditions for the driver, satisfying pre-imposed inter-vehicle distance limits. The minimum and maximum acceptable IVD are taken from the literature [6] in order to guarantee safety requirements and stable wireless V2V communication. Since the tested driving cycles are mainly urban the maximum IVD is 100m. In this work, due to the high adopted time step of 1 second, the jerk, calculated as the difference between the acceleration at the present time step and acceleration at the previous time step, would be physically far from the derivative of acceleration. For that reason, the root mean square of acceleration is chosen as the comfort variable that the agent tries to minimize. In order to achieve the aforementioned objectives, the following reward function is implemented:

$$r = \frac{w_d r_d + w_{SOC} r_{SOC} + w_{acc} r_{acc}}{w_d + w_{SOC} + w_{acc}} \tag{9}$$

The three terms of the reward are useful to meet different tasks and all have a value between -1 and 1: r_d is necessary to control the inter-vehicle distance, which has to be maintained between the chosen limits; r_{SOC} is useful to reduce the energy consumption of the battery; r_{acc} is introduced to guarantee comfortable driving conditions. Their respective weight coefficients w_d, w_{SOC}, w_{acc} allow choosing which objective is privileged. These weights are initially set to 1, future analysis should consider their tuning in order to obtain a weighted average rather than the arithmetical one. r_d is shaped as a simple linear function giving a lower reward when the distance is closer to distance limits. Moreover, if the distance between the two vehicles goes beyond the chosen limits, the simulation is stopped and the agent receives a negative reward equal to -100. Such a penalty is given in order to make the agent understand to completely avoid those conditions. After such an event a new episode is restarted from the same starting conditions of the chosen driving cycle. Regarding the shape of the three terms of the reward function, the most interesting ones are r_{acc} and r_{SOC}: r_{acc} is shaped as a quadratic function of the acceleration of Ego (acc_{Ego}) with proper weight b and c (in this paper b=1 and c=2) since the objective is the minimization of the rms of acceleration, whereas r_{SOC} has the innovative concept of using the variable z, a function of SOC:

$$z = \frac{SOC_{Ego} - SOC_{Lead}}{SOC_{in} - SOC_{Lead}} \tag{10}$$

$$r_{SOC} = \frac{z}{a} \tag{11}$$

$$r_{acc} = b - c \cdot \frac{acc_{Ego}^2}{\max(acc_{Ego}^2)} \tag{12}$$

where SOC_{Ego} is the state of charge of the Ego vehicle at the current time step, SOC_{in} is the SOC of Ego before starting the driving mission, while SOC_{Lead} is a fictitious variable equal to the SOC that the Ego vehicle would have if it maintained the same velocity as the Lead vehicle for the whole driving cycle. The term z represents the percentage of energy savings achieved; a is a constant representing the percentage of energy savings desired and its value was set to 5%. Further analysis is required in order to understand the influence of this parameter on final results. The reward r_{SOC} is then clipped between -1 and 1 in order to achieve agent stability. This form of reward allows to achieve an adaptive control which autonomously adapts to the driving cycle. If we had used a more intuitive function such as $r_{SOC} = 1 - 2 \cdot \Delta SOC / \Delta SOC_{max}$, function of the change of SOC in a single time interval (ΔSOC) we should have carefully changed ΔSOC_{max} to achieve optimal energy savings in different cycles.

IV. Results

In order to test the adaptive capability of the algorithm, the agent has been trained on different standard driving cycles: Artemis Urban Driving Cycle (AUDC), Urban Dynamometer Driving Schedule (UDDS) and the first 388 seconds of

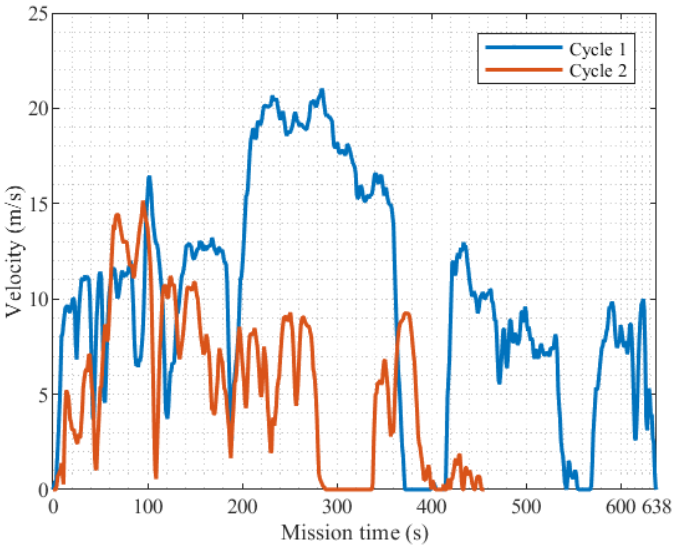

Fig. 1 Real -world driving cycles

Worldwide Harmonized Light Vehicles Test Procedure (WLTP). DDPG is also trained on two different real-world driving cycles as Cycle 1 (RC1) and Cycle 2 (RC2), that Fig.1 shows. The chosen driving cycles are mainly urban because it has been proved in previous works that more significant energy savings can be achieved in urban driving cycles with the hypothesis of neglecting the air drag effect. In this study, different simulations have been carried out: in each simulation, the Lead vehicle follows a chosen driving cycle and the agent learns the optimal acceleration policy of the Ego vehicle through the training process. As previously mentioned, the agent tries and evaluates new actions thanks to the exploration noise added to the action.

For Deep RL algorithms the discounted return and the training loss of the neural networks are two of the most important indicators to analyse how effective the training phase was. Fig.2 clearly shows how the agent correctly learns to achieve higher discounted return as the training progresses for the UDDS cycle. It is interesting to notice that the discounted return at the beginning of the first 200 episodes is very close to -100. This is due to the fact that in this first phase of training the agent is not able to maintain the Ego vehicle between the imposed distance limits and the episode concludes with the penalty reward of -100. Moreover, as shown in the second subplot of Fig.2, the training loss of the critic neural network correctly decreases, demonstrating a progressively more accurate estimate of the Q-values. Therefore, as the training progresses, the agent improves the learned policy. This result is demonstrated also by the energy savings and acceleration rms reduction of the Ego with respect to the Lead, expressed as percentage values, as shown in Fig.3. As the energy savings increases alongside the number of episodes, the reduction of rms of acceleration correctly presents a similar trend. For the sake of clarity, in this representation, the episodes stopped before the end of the cycle are not represented at all. It's interesting to notice that, since the UDDS cycle is characterized by a wider range of velocity and is longer than the other driving cycles, the number of excluded episodes is close to 900 and higher than other simulations. For simpler driving cycles, the distance-related limits are indeed easier to be satisfied and therefore the number of not completed episodes is lower.

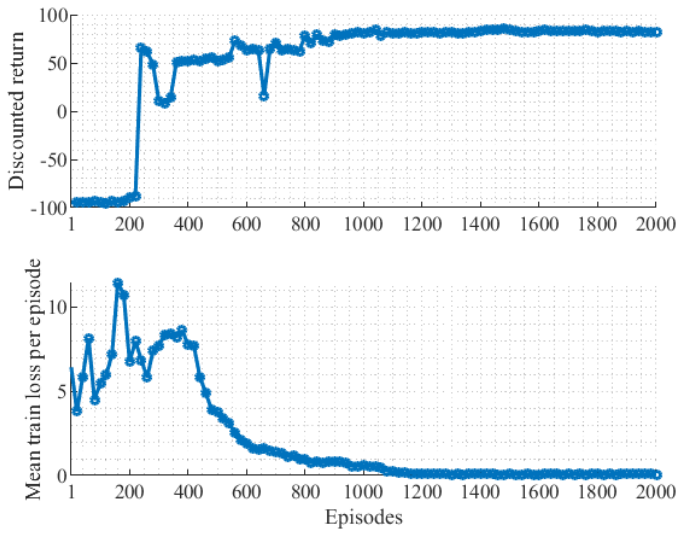

Fig. 2 Discounted return and training loss of training episodes for UDDS

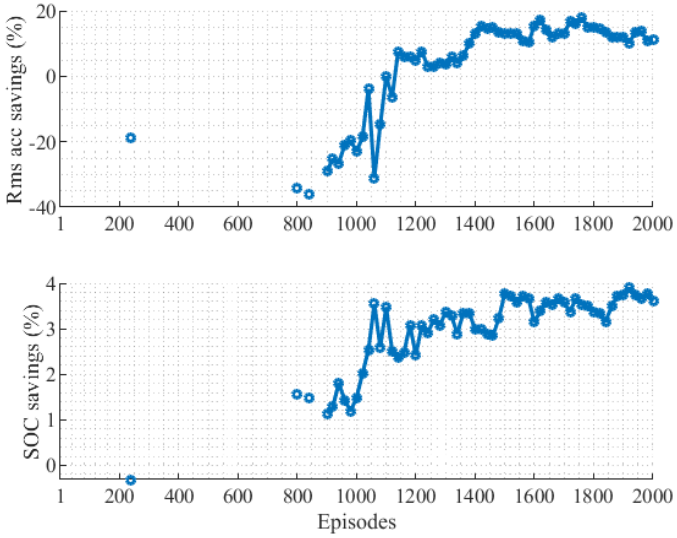

Fig. 3 Rms of acceleration savings and SOC savings of training episodes for UDDS

As a final result, at the end of the training the agent is tested on the same driving cycle with pure exploitation by removing the exploration noise. The agent proves to achieve significant energy savings for different driving missions, enhancing comfort conditions and guaranteeing that the IVD is maintained between the chosen limits. Fig.4 shows the velocity profiles of the Ego and Lead vehicles generated after the training of the agent on UDDS. Since stop-and-go waves deeply affect the fuel economy [15], the agent correctly tries to minimize the number of times the Ego vehicle stops.

Furthermore, in order to demonstrate the adaptability of the algorithm to conditions never seen during training, the agent previously trained on a particular driving cycle is tested on driving cycles different from the one of training. The percentage

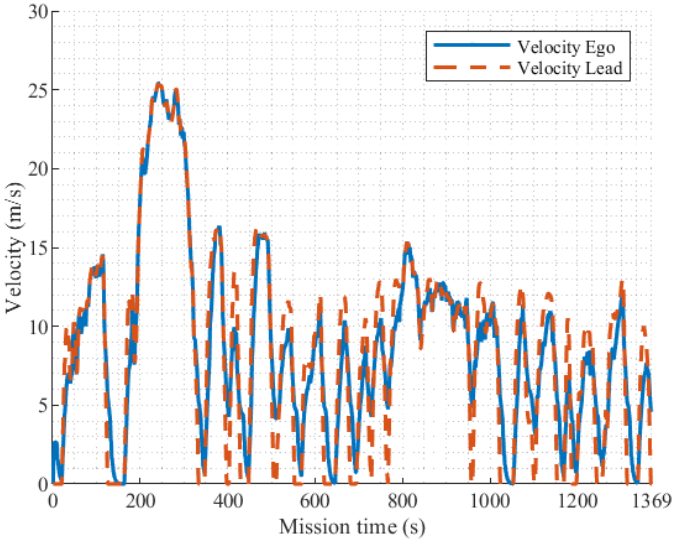

Fig. 4 Velocity of Ego and Lead vehicles for UDDS

SOC savings and acceleration rms reduction of Ego with respect to the Lead for all the simulations are listed in Table I. As shown by the numerical results, the agent is able to generalize the training and achieve significant energy savings also on driving cycles never seen before. As expected, when the agent is trained on a complex and long cycle, it efficiently performs on simpler driving cycles of test. For instance, Fig.5 displays both SOC and comfort advantages for the Ego when the agent is trained on UDDS and tested on the WLTP (388s). Moreover, the agent is surprisingly able to conclude some test driving cycles that are more complex than the training one. For example, the agent trained on the WLTP (388s) cycle can conclude the UDDS which is characterized by a higher maximum velocity and is longer. However, when the agent is trained on WLTP (388s) and tested on the RC1, the Ego can't reach the end of the cycle because of the IVD limits as shown in Fig.6, since the velocity of 20m/s of RC1 is never encountered during WLTP (388s). We can generally conclude that it's better to train the DDPG agent on complex and long cycles in order to let the agent learn how to properly act in front of a wide variety of velocities and accelerations of the Lead vehicle.

TABLE II. RESULTS

Training cycle	Test cycle	SOC savings (%)	Rms acc savings (%)
WLTP (388s)	*WLTP (388s)*	2.3	20
	UDDS	2.3	5
	AUDC	2.8	12
	RC1	-	-
	RC2	1.2	20
UDDS	*WLTP (388s)*	2.5	20
	UDDS	3.5	26
	AUDC	4.1	25
	RC1	1.5	2
	RC2	1.8	26
AUDC	*WLTP (388s)*	2.8	18
	UDDS	3.2	22
	AUDC	4.2	25
	RC1	0.5	5
	RC2	1.5	13
RC1	*WLTP (388s)*	1.0	1
	UDDS	2.7	13
	AUDC	2.6	17
	RC1	1.0	18
	RC2	0.5	17
RC2	*WLTP (388s)*	2.3	8
	UDDS	2.1	3
	AUDC	2.5	8
	RC1	1.1	-5
	RC2	1.8	15

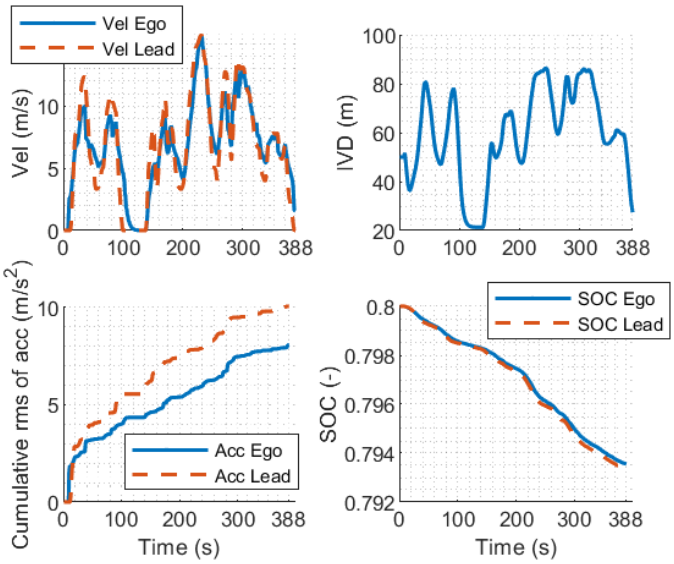

Fig. 5 Results of testing the agent, trained on UDDS, on WLTP (338s)

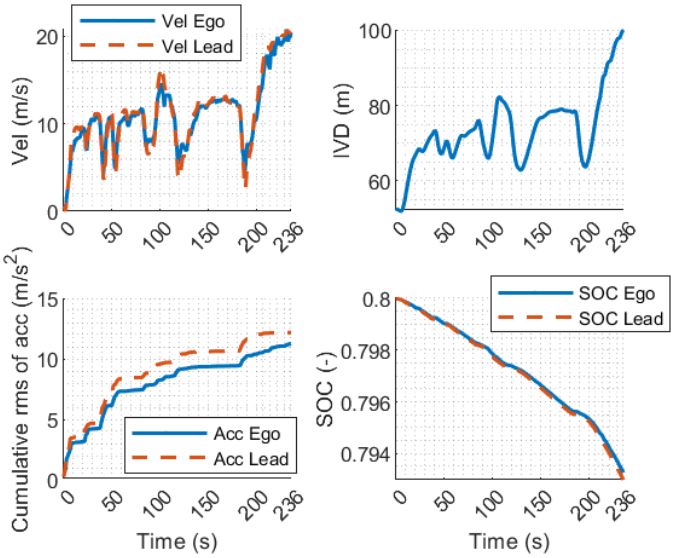

Fig. 6 Results of testing the agent, trained on WLTP (388s), on RC1

V. CONCLUSIONS AND FUTURE WORKS

In this paper, the authors address the problem of acceleration control for an Ego vehicle which is following a Lead vehicle on a straight road. The main objective of this work is to show the potential of DRL algorithms to achieve significant energy savings for the Ego vehicle, enhancing comfort conditions. The first main innovative contribution of this work is the use of a DDPG agent with the main objective of obtaining energy savings, still guaranteeing safe driving conditions. Moreover, the use of an adaptive multi-objective reward function able to achieve good results on different driving cycles without the necessity of tuning parameters is enabled thanks to the creation of a fictitious variable, reproducing the SOC that the Ego vehicle would have if it had the same velocity profile of the

Lead. The agent is trained and tested on different standard driving cycles such as WLTP, UDDS and AUDC and two real-world driving cycles, achieving good results and showing great adaptability also to conditions not seen during training. The main limitations of the work are related to the simple driving scenario and to the idealistic approximations which were made in order to keep the problem simpler. Possible future work might be building a high-fidelity simulation model of the vehicle, allowing also steering actions and including errors or delays in the signals sent in the V2V communication.

ACKNOWLEDGEMENT

This research work was developed in the framework of the activities of the Interdepartmental Center for Automotive Research and Sustainable mobility (CARS) at Politecnico di Torino.

REFERENCES

[1] J. van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," Transportation Research Part C: Emerging Technologies, vol. 89. Elsevier Ltd, pp. 384–406, Apr. 01, 2018. doi: 10.1016/j.trc.2018.02.012.

[2] Z. Wang, G. Wu, and M. J. Barth, A Review on Cooperative Adaptive Cruise Control (CACC) Systems: Architectures, Controls, and Applications; A Review on Cooperative Adaptive Cruise Control (CACC) Systems: Architectures, Controls, and Applications. 2018. doi: 10.0/Linux-x86_64.

[3] A. Alam, A. Gattami, K. H. Johansson, and C. J. Tomlin, "Guaranteeing safety for heavy duty vehicle platooning: Safe set computations and experimental evaluations," Control Engineering Practice, vol. 24, no. 1, pp. 33–41, 2014, doi: 10.1016/j.conengprac.2013.11.003.

[4] A. Vahidi and A. Sciarretta, "Energy saving potentials of connected and automated vehicles," Transportation Research Part C: Emerging Technologies, vol. 95. Elsevier Ltd, pp. 822–843, Oct. 01, 2018. doi: 10.1016/j.trc.2018.09.001.

[5] W. D. Connor, Y. Wang, A. A. Malikopoulos, S. G. Advani, and A. K. Prasad, "Impact of Connectivity on Energy Consumption and Battery Life for Electric Vehicles," IEEE Transactions on Intelligent Vehicles, vol. 6, no. 1, pp. 14–23, Mar. 2021, doi: 10.1109/TIV.2020.3032642.

[6] M. Spano, P. G. Anselma, A. Musa, D. A. Misul, and G. Belingardi, "Optimal real-time velocity planner of a battery electric vehicle in V2V driving," in 2021 IEEE Transportation Electrification Conference and Expo, ITEC 2021, Jun. 2021, pp. 194–199. doi: 10.1109/ITEC51675.2021.9490121.

[7] T. Stanger and L. del Re, "A model predictive Cooperative Adaptive Cruise Control approach," in Proceedings of the American Control Conference, 2013, pp. 1374–1379. doi: 10.1109/acc.2013.6580028.

[8] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of Deep Reinforcement Learning and Model Predictive Control for Adaptive Cruise Control," IEEE Transactions on Intelligent Vehicles, vol. 6, no. 2, pp. 221–231, Jun. 2021, doi: 10.1109/TIV.2020.3012947.

[9] Y. Wu, H. Tan, J. Peng, H. Zhang, and H. He, "Deep reinforcement learning of energy management with continuous control strategy and traffic information for a series-parallel plug-in hybrid electric bus," Applied Energy, vol. 247, pp. 454–466, Aug. 2019, doi: 10.1016/j.apenergy.2019.04.021.

[10] C. Desjardins and B. Chaib-Draa, "Cooperative adaptive cruise control: A reinforcement learning approach," IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 4, pp. 1248–1260, Dec. 2011, doi: 10.1109/TITS.2011.2157145.

[11] M. Li, Z. Cao, and Z. Li, "A Reinforcement Learning-Based Vehicle Platoon Control Strategy for Reducing Energy Consumption in Traffic Oscillations," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 12, pp. 5309–5322, Dec. 2021, doi: 10.1109/TNNLS.2021.3071959.

[12] M. Zhu, Y. Wang, Z. Pu, J. Hu, X. Wang, and R. Ke, "Safe, Efficient, and Comfortable Velocity Control based on Reinforcement Learning for Autonomous Driving," Jan. 2019, doi: 10.1016/j.trc.2020.102662.

[13] United States Environmental Protection Agency, "Data on Cars used for Testing Fuel Economy."

[14] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," Sep. 2015, [Online]. Available: http://arxiv.org/abs/1509.02971

[15] R. E. Stern et al., "Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments," Transportation Research Part C: Emerging Technologies, vol. 89, pp. 205–221, Apr. 2018, doi: 10.1016/j.trc.2018.02.005.