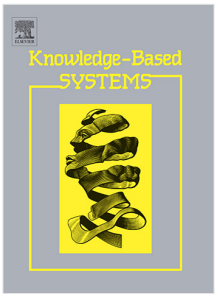
Journal Pre-proof



Formula-E race strategy development using distributed policy gradient reinforcement learning

Xuze Liu, Abbas Fotouhi, Daniel J. Auger

PII: S0950-7051(21)00044-7
DOI: https://doi.org/10.1016/j.knosys.2021.106781
Reference: KNOSYS 106781

To appear in: *Knowledge-Based Systems*

Received date : 21 September 2020
Revised date : 25 November 2020
Accepted date : 14 January 2021

Please cite this article as: X. Liu, A. Fotouhi and D.J. Auger, Formula-E race strategy development using distributed policy gradient reinforcement learning, *Knowledge-Based Systems* (2021), doi: https://doi.org/10.1016/j.knosys.2021.106781.

Knowledge-Based Systems, Volume 216, March 2021, Article number 106781
DOI:10.1016/j.knosys.2021.106781

Formula-E race strategy development using distributed policy gradient reinforcement learning

Xuze Liu*, Abbas Fotouhi, Daniel J. Auger

Advanced Vehicle Engineering Centre, School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, Bedfordshire, MK43 0AL, UK
*Corresponding Author, email: Xuze.Liu @cranfield.ac.uk

Abstract

Energy and thermal management is a crucial element in Formula-E race strategy development. In this study, the race-level strategy development is formulated into a Markov decision process (MDP) problem featuring a hybrid-type action space. Deep Deterministic Policy Gradient (DDPG) reinforcement learning is implemented under distributed architecture Ape-X and integrated with the prioritized experience replay and reward shaping techniques to optimize a hybrid-type set of actions of both continuous and discrete components. Soft boundary violation penalties in reward shaping, significantly improves the performance of DDPG and makes it capable of generating faster race finishing solutions. The new proposed method has shown superior performance in comparison to the Monte Carlo Tree Search (MCTS) with policy gradient reinforcement learning, which solves this problem in a fully discrete action space as presented in the literature. The advantages are faster race finishing time and better handling of ambient temperature rise.

Key words—Energy management; Formula-E race strategy; Deep deterministic policy gradient; Reinforcement leaning

1. Introduction

The popularity of hybrid and electric vehicles has grown rapidly over the recent years. Energy management has always been one of the hottest topics in those advanced electrified vehicles. In top-level motorsport series, the technical regulations introduce stricter restrictions on energy consumption year by year to encourage more high-efficiency powertrain technology development. In the full-electric series Formula-E (FE) championship, energy management has been the most crucial element in race strategy development proved by numbers of winnings and loses witnessed over the past seasons [1].

Published researches addressing energy management problems mainly fall into two categories: (1) real-time controller development, which is the majority [2][3][4][5]; and (2) trip-oriented energy management optimizations [6][7][8][9]. The first category includes various control strategies used to manage the power flow among multiple energy resources in hybrid powertrains, targeting to achieve the maximum overall powertrain efficiency at each timeframe or a certain optimization horizon given a specific power demand from the diver input. In the second category, the target is to minimize the total energy consumption for a pre-

specified route based on the information of the route, vehicle and powertrain and is solved under certain constraints such as minimal finishing time. Energy management problems in motorsport are similar to the second category but with time being the optimization objective and energy treated as an integral constraint to study the maximum achievable performance given an amount of useable energy. Such management problems are usually solved in lap time simulations (LTS) by formulating into an optimal control problem (OCP) [10]. Tremlett et al. [11] studied the optimal tyre usage of Formula-One (F1) cars by including a thermal-dynamic model in the OCP. Limebeer et al. [12] studied the energy management strategy for an F1 hybrid system. Herrmann et al. [13] optimized the electric energy usage for an autonomous race car. And Liu et al. [14] studied the energy management strategy under both energy and battery thermal constraints. Optimal control technique proves to be very reliable for solving such minimal-time management problems for a single lap in motorsport applications.

The energy management problem on a race (multi-lap) level is more difficult than the single lap problem to solve. Although optimal control technique is powerful, it is also a very computational costly method. The computational time could range from minutes to hours depending on the model fidelity inside problem formulation. This leads to hours or even days to solve the multi-lap problem, making it irrational to use optimal control technique for strategic application on a race level. Generally, a good 'race strategic tool' requires fast decision-making capabilities due to the highly dynamic nature of motorsport races.

Race strategy development requires an upper-level view instead of studying the control problem within a single lap. Among the very few publications targeting such analysis, most focus on building race simulation environment discussing how to discretize a race. A popular way is to discretize a race into single laps [15][16] while in some other studies, it is divided into smaller sectors of approximately 150 m length[17] or sub-sectors near the start/finish line to capture potential overtaking opportunities during pitstops [18]. A race can thus be simulated by introducing various influencing factors such as fuel load, tyre aging, car abilities and adding respective time penalties to a baseline sector/lap time. A gap lies in the actual decision making in race strategy development where these studies did not address on. In general, Monte Carlo simulation is one of the most popular methods for strategic planning. It has been applied to the races of either horse racing [19] or motorsport [20]. By introducing probabilities of actions or the environment (i.e. accident probabilities, player abilities), the Monte Carlo methods can generate an approximated solution based on hundreds or thousands of Monte Carlo simulations. However, the quality of such solutions of this exhaustive method is heavily depended on the brute computational force or the accuracy of those probability estimation. To improve the efficiency and reliability of race strategic planning, Liu et al.[19] proposed a technique to solve the race strategy problem by using Monte Carlo Tree Search (MCTS). A Formula-E race is discretized into laps and MCTS is used to shrink the size of the problem by focusing on the relatively more promising actions hence improve the efficiency and quality. The implementation of MCTS proves to be a decent method for planning and improvising given various scenarios during a race.

In Formula E, energy consumption is the major concern. A strategic action in a real-life race is always complex and comprises both continuous and discrete type of actions. The technical regulation of FE racing presented in [22] states that for a complete race, the total amount of energy that can be delivered from battery to the motor is limited to 52 kWh. This amount is

much lower than what is required to be fully aggressive throughout a complete race. Therefore, teams have to decide how much energy to use for each lap (i.e. energy per lap/EPL), which is chosen from a continuous action space (e.g. 1.0kWh-2.2kWh). Another continuous action comes from the thermal management of battery when races are held in hot climates such as those present in Marrakesh and Santiago. Heat is generated both during propulsion (vital for speed) and regeneration (vital for energy efficiency and endurance). Teams have to avoid battery overheating which leads to de-rated power and potentially a Did Not Finish (DNF). As previously pointed out in [14], adopting smooth pedal operation is proved to be an effective technique to manage the temperature rise. The extent to which pedal operation should be smoothed constitutes another continuous action which will be later referred to as Q mode (QM), which is detailed in section 2. Apart from these continuous elements, strategic decisions in FE comprise another important discrete action which is whether to activate Attack Mode or not. In normal race mode settings, the maximum power of motor is limited to 200kW by the regulation. Beyond that, teams are given an option of activating attack mode for a certain amount of time which gives an extra power of 50 kW during this power mode. The number of activations and duration are circuit-dependent and using all the activations is compulsory. Whether to activate the attack mode at a certain stage of a race is not a single independent decision related to that particular lap only. Actually, it also has potential influence on the EPL and QM decisions due to its unique contribution to energy consumption and battery thermal behavior [14]. Therefore, an appropriate strategic tool for FE needs to be capable of generating a hybrid action made of both these continuous and discrete actions.

There are several main weaknesses in the current methods for strategic planning: (1) the use of ordinary Monte Carlo simulations would easily compromise the quality and accuracy of a solution. Among these stochastic simulations, some unlikely moves are included which would rarely happen in a real case. As a result, the solution could potentially be overrated; (2) although according to [19], MCTS takes around 10 seconds to make a decision, it is still relatively slow because of the highly dynamic nature of a motorsport race. Such 10-second duration may compromise the timeliness of the decision made; (3) to implement MCTS, all actions must be discretized. As previously discussed, for the continuous action elements, potential better solutions could be hidden due to the discretization; (4) the online computing methods rely on high computational resource to generate solution with decent accuracy. This bottleneck prevents strategists to investigate a strategy with higher dimensions, which would require significantly more resource otherwise, the solution accuracy and timeliness have to be compromised. To overcome these weaknesses, in this study, reinforcement learning is proposed as a new method for the race strategy problem.

Deep Reinforcement Learning (DRL) is a branch of machine learning algorithms that are designed to optimize actions in an environment in the form of a Markov decision process (MDP) in order to maximize the collected reward. RL-based methods have been proved of great utilities in wide varieties of applications such as robotics [23], games [24], resource management [25], etc. There are several features of interest which makes RL-based approach favorable in motorsport strategic applications: (1) DRL methods are reward based. In motorsport, time as the major concern, can be easily quantified as reward and integrated with other constraint violation penalties; (2) Strategic decision making in a race event conforms to an MDP. A decision is made based on the observation or state of the environment (i.e. the race)

and the objective is to maximize the cumulative reward after a sequence of decisions leading to a terminal state (i.e. the finish of a race). (3) Training a DRL agent may be computational resource consuming but applying the trained outcome can be very simple. In motorsport, apart from the race event, all the rest of time can be used to prepare and train an agent for varieties of scenarios (i.e. offline training before the race). Then the matured agent can be used during the race for decision making at almost no time cost. This nature of DRL suits well the resource allocation and demand for timeliness in motorsport industry. Therefore, DRL has a great potential in motorsport race strategy development.

The very early reinforcement learning focused on simple low-dimensional problems with both discrete states **S** and action spaces **A**. A multi-dimensional table or matrix can thus be created to store the qualities of the state-action pairs **Q(s,a)** which are learned through value iteration algorithm [26]. The policy of this Q-learning method is to simply choose the action with highest Q value in the look-up table. This method is then improved by Deep Q-Learning (DQL) [27] where the Q table is replaced with a deep neural network which broke the bottleneck of high-dimensional and continuous states. DQL has laid the foundation for all the modern value-based deep reinforcement learning algorithms such as Double D-Learning [28] and Dueling Q-learning [29], etc. Instead of value-based methods, another major family of reinforcement learning algorithms are the Policy Gradient (PG) methods. PG methods answer the policy question more directly than the value-based ones which try to choose the best action by approximating the values of state-action pairs. The basic PG method is called REINFORCE [30] whose idea is to train a policy network to make the probabilities distribution more biased to the promising actions. One of its modifications, the Actor-Critic (A2C) algorithm [31] forms another popular family in RL algorithms nowadays. By separating the tasks of value approximation (critic network) and choosing action (actor network), the A2C method overcomes the weakness of noisy gradients and high variance in REINFORCE, hence improves the capability and stability of reinforcement learning technique in wider applications. One of the biggest breakthroughs in recent years is the introduction of Deep Deterministic Policy Gradient (DDPG) [32] which is also an A2C-like algorithm. In DDPG, the actor (policy) network maps the states directly into action values instead of outputting the probability distribution across a discrete action space. This method significantly broadens the application of RL methods in control problems with continuous action spaces like autonomous driving [33].

Traditional single-agent RL algorithms usually suffer from slow training speed when an environment is complex and stochastic in some of its conditions. Formula E races also have such features among which the environmental or ambient temperature has crucial effect on race strategy. Therefore, strategy development in Formula E isn't a single learning task for a single fixed environment. Instead, a range of ambient temperature need to be considered in the training process. This requires a method to scale up the learning process to handle varieties of environmental settings. Distributed stochastic gradient descent has been successfully widely applied in supervised learning [40], while distributing RL tasks only started in the very recent years [41]. The concept of distributed RLs is to deploy multiple workers in separate environments and send either gradients or experiences for a singe learner to update the policy [42][43]. While the early approaches targeted only on tasks featuring only discrete actions. Horgan et al. [44] proposed an architecture called Ape-X which extends prioritized experience replay (PER) [36] to a distributed setting and proves to be a highly-

scalable approach for both discrete and deterministic tasks.

In this study, we investigate a novel approach to solve the race strategy problem of minimizing race finishing time by using distributed DDPG reinforcement learning but with a hybrid action space. With this method, both discrete and deterministic action elements can be optimized simultaneously. This realistic action form guarantees that no potential optimal solution could be hidden due to action space discretization. The proposed framework allows engineers to transfer the strategy optimizations from online to offline. With the significant amount of time between race events, this breaks the online computational resource bottleneck and allow strategists to further investigate higher dimensional and more complex scenarios. This study proposed to the industry with a new approach for decision making not only in race strategies but also potentially in other aspects such as car designs and setups. In this paper, the problem background and previous researches were briefly introduced in this section. Section 2 demonstrates how a race strategy problem is formulated into an MDP environment. In section 3, we modify the DDPG actor network to tackle the hybrid-type decisions in a race environment and the training process will be presented. The result and discussion will be shown in section 4. Finally, the conclusion is presented in section 5

2. Problem formulation

In this study, a race will be discretized in laps to build an MDP environment. An MDP environment comprises four major components, state $\boldsymbol{s}$ of the environment, action $\boldsymbol{a}$ absorbed from agent input, transition model $\boldsymbol{T}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$ which updates the environment state $\boldsymbol{s}$ to a new state $\boldsymbol{s}'$ based on the agent action, and the reward function $\boldsymbol{Q}(\boldsymbol{s})$ which return the reward of a state $\boldsymbol{s}$ to the agent. In this section, states and actions will be firstly clarified. The transition model will be built using neural network prediction models. And the reward function will be formulated.

2.1 States and actions

A state $\boldsymbol{s}$ contains the information accessible during a race including variables that have a significant influence on decision making. Such variables which are contained in a state are stated in table 1.

Table 1 State description

State variable	Description	Initial value	Max value
N_{lap}	Remaining number of laps	34	$N_{lap\_tot} = 34$
E_r	Remaining usable energy	52	E_{max}=52
T_{bat}	Battery temperature	20	$T_{bat\_limit} = 58$
T_{amb}	Ambient temperature	30	-
N_{att}	Available number of Attack Mode(AM) activation	2	$N_{att_{max}} = 2$
PM	Current power mode (1-Race mode;2-Activation mode;3-Attack mode)	1	-
$N_{Rattlap}$	Current remaining number of	0	$N_{lap\_per\_att} = 2$

	AM laps		
t_{race}	Current race time	0	-

In this study, the Marrakesh ePrix track is used. For a 45-minute race, the total number of laps N_{lap_tot} is 34 thus the N_{lap} is 34 at most. The FE technical regulation states that the total amount of usable energy is limited to E_{max}=52 kWh which makes E_r 52kWh at most. The ambient temperature is another important element that affects the cooling of the battery. Overheating the battery (T_{bat} reaching above an upper limit T_{bat_limit}) will lead to a DNF or other forms of unfavorable results; hence it must be avoided. Both E_r and T_{bat} will be accounted in the reward function. N_{att} gives number of times a team can activate attack mode for the remaining laps (N_{lap}) of a race. For Marrakesh ePrix, N_{att} is assigned with initial value of 2, meanwhile each activation lasts 2 laps and $N_{Rattlap}$ gives how many attack mode laps remains during one activation. PM indicates what power mode the car is using. t_{race} denotes the race time starting from the initial state s_0. The third column of table 1 gives an example of initial state values featuring the start of a full race at Marrakesh ePrix. It should be noted that normal race mode (PM=1) and attack mode (PM=3) state that the upper limits for driving power are 200 kW and 250 kW respectively. However, considering that the attack mode is activated at a certain point in the middle of a lap instead of at the start/finish line, and the extra 50 kW of power are only allowed afterwards; a new power mode (PM=2, attack mode activation lap) is introduced between those two modes to describe the performance in the activation lap.

As previously introduced, the hybrid-type action in FE contains two continuous actions namely the EPL and QM. EPL denotes the target energy consumption for the following lap at a given state. Here in this environment, EPL is limited to the range of 1.2-2.2 kWh. This is a rational range on Marrakesh track given by [14]. QM, as stated in section 1, denotes the level of thermal management. Previous research [14] has found that the constraints of ELP and battery temperature have their unique impacts on lap time and thermal management. More specifically, with same amount of available energy, different settings of thermal boundaries may compromise the lap time but offers a more efficient way to manage the temperature rise compared to brutally decreasing the ELP. In this study, QM is defined as a continuous scaler of range 0-3. QM equal to 0 indicates that the battery temperature does not need to be taken care of, while QM>0 means restricting target temperature rise by the magnitude of 0.95^{QM}. For example, if QM=0 has a T_{bat} rise of 4 ℃, then for QM=1, T_{bat} will be expected to rise by 3.8 ℃. The discrete component in an action is relatively simple which delivers the information of the agent's choice on whether to activate attack mode or not. This will be later implemented as a 'one-hot' vector [40]. The complete action space is shown in table 2.

Table 2 Action space

Action type	Continuous		Discrete
Action component	EPL	QM	Activation
Range	1.2-2.2	0-3	[0,1] or [1,0]

2.2 Transition model

As stated earlier, the race is discretized into laps. Therefore, the transition model here should provide information of the effect of an action on the performance of a single lap. While state variables of $N_{lap}, N_{att}, PM, N_{Rattlap}$ can be easily calculated, the lap time, battery temperature and energy consumption during a single lap based on a given action have strong non-linear features which cannot be simply assumed. In a previous research [21], neural networks were proposed to be trained as transition model, which provides decent accuracy. While a commercial simulation software was used in [21] to generate the training data, in this study, the training datasets are collected by formulating each case into an OCP. The reason for doing so is to guarantee the optimality of lap performance of a given input, while commercial simulation software failed to do so with their empirical driver model. An OCP is formulated to minimize a Lagrange cost function of

$$J = \int_{t_0}^{t_f} l(t, x(t), u(t), p)dt \tag{1}$$

which is subject to the constraints of

$$\begin{cases} \frac{dx}{dt} - f\big(t, x(t), u(t)\big) = 0 \\ g\big(t, x(t), u(t)\big) = 0 \\ h(t, x(t), u(t)) \leq 0 \\ g_b(x(t_0), x(t_f), u(t_0), u(t_f)) = 0 \end{cases} \tag{2}$$

In this problem, $x(t) \in R^n$ is the state vector made of vehicle dynamics information and u$(t) \in R^m$ is the control vector of steering and pedals. The system dynamics is described in the vector $f\big(t, x(t), u(t)\big) \in R^n$. Vector $g \in R^{n_g}$ and $g_b \in R^{n_{gb}}$ are the quality constraints and boundary constraints. The inequality constraints are defined in $h \in R^{n_h}$ where the aforementioned EPL and QM are included. This approach has been thoroughly explained in [14] therefore, it is not detailed here. The data collection process through the OCP is shown in figure 1.

The initial usable energy represents the state of charge (SOC) of the battery which influences the thermal dynamics model inside the problem. It should be noted that the energy element appears on both input and output sides as the filled arrow showed in figure 1. The reason for doing so will be explained later. The input range for data collection are defined in table 3 from which the input variables are picked randomly. A total number of 129,000 of datasets are collected to train the neural network transition model. An example of the collected data is shown in figure 2 including the effects of QM and EPL on the lap time. The blocked area in the figure explains why 'energy' has to be on both input and output side. That is because in the situations of high EPL and QM combinations, the requested energy isn't fully consumed due to the strict thermal boundaries. Therefore, there will be a nonlinear mapping between the two sides and the transition model have to be able to capture this feature.

As a result, three individual networks are trained to model the energy consumption, battery temperature rise and lap time separately. This procedure is inherited from [21] thus is not demonstrated here. The structure of the networks and accuracies are shown in appendix A. With these three neural network prediction models, the transition of an environment state can be described in table 4.

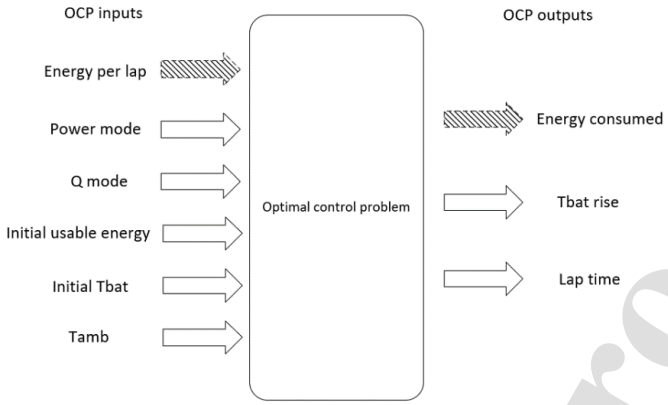


Figure 1 Data collection process through the OCP

Table 3 Input variables' range

Input	**Range**
Energy per lap(kWh)	1.2 - 2.2
Power mode	1,2,3
Q mode	0-3
Initial usable energy(kWh)	0-52
Initial battery temperature(℃)	20-60
Ambient temperature(℃)	15-40

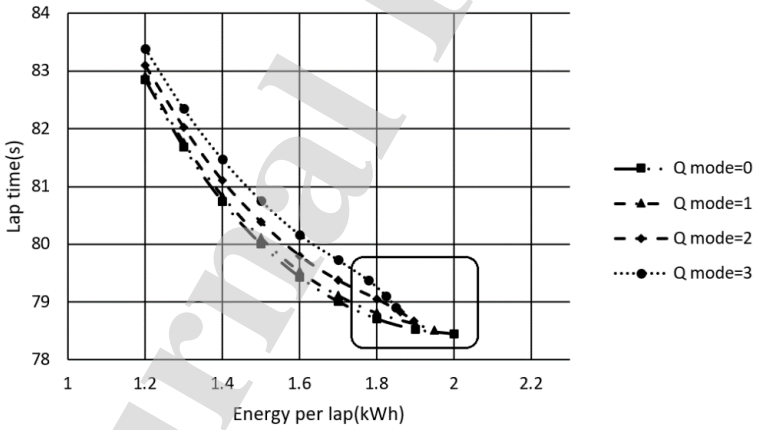


Figure 2 Lap time of different EPL and Q modes (Power mode=1)

Table 4 State description

State *s*	**Transition based on action *a***	**New state *s'***
N_{lap}	-	$N_{lap} - 1$
E_r	Energy consumption network (NN_{Energy})	$E_r - NN_{Energy}(a)$
T_{bat}	Battery temperature network (NN_{Tbat})	$T_{bat} + NN_{Tbat}(a)$
T_{amb}	-	T_{amb}

N_{att}	Action dependent	
PM		
$N_{Rattlap}$		
t_{race}	Lap time network(NN_{tlap})	$t_{race} + NN_{tlap}(a)$

2.3 Reward Function

Generally, RL algorithms are designed to find a policy to maximize the total accumulated reward in an environment [34]. In this study, the aim is to find the policy leading to the fastest possible race finishing time. Therefore, two type of rewards are used in the environment: (1) step reward R_s, which encourages the agent to finish the race, and (2) terminal reward R_T to encourage a faster time. The step reward is returned to the agent after each action is made. It is defined as

$$R_{step} = \begin{cases} 5 & normal\ step\ reward \\ -5 & if\ attack\ mode\ is\ wrongly\ activated \end{cases} \quad (3)$$

This reward means that each time the agent successfully enters the next lap, it will be given a positive value. Meanwhile if attack mode is wrongly activated, a negative penalty will be returned. An activation action is considered as wrong under the circumstances listed in table 5.

Table 5 Activation penalty circumstances

Circumstances	**State condition**
Activate when no available attack modes left	$N_{att} = 0$
Activate when the car is already in attack mode but not on the final lap of attack mode	$PM = 2\ or\ 3$ and $N_{Rattlap} > 0$
Activate in the first two laps of the race (Regulation)	$N_{lap} > 32$
NOT activate when compulsory (Regulation)	$N_{lap} = 2N_{att}$ (Number 2 denotes duration of 2 laps per attack)

In this race environment, a state is considered as terminal when any of these three conditions are met: (1) Battery overheated over the limit: $T_{bat} > T_{bat\_limit}$; (2) Energy is over-consumed: $E_r < 0$; (3) Race finished: $N_{lap} = 0$. If the environment reaches a terminal state, a terminal reward will be added upon the final step reward. The terminal reward is defined as:

$$R_T = R_{Tbat} + R_E + R_{time} \quad (4)$$

Where R_{Tbat} and R_E are the battery temperature and energy penalties respectively and are given by:

$$R_{Tbat} = c \min(0, E_r) \quad (5)$$

$$R_E = c \min(0, T_{bat\_limit} - T_{bat}) \quad (6)$$

c is a large value to magnify the violation of battery temperature limit and energy consumption. The third term in eq.4 is the time reward given by:

$$R_{time} = \max\left(\left(83.5N_{lap\_tot} - \left(t_{race} + 130N_{lap}\right)\right) * 10, exp\left(0.025\left(3010 -\right.\right.\right.$$

$$\left(t_{race} + 130N_{lap}\right)\Big)\Big)\Big) \tag{7}$$

This reward as a function of race time is shown in figure 3.

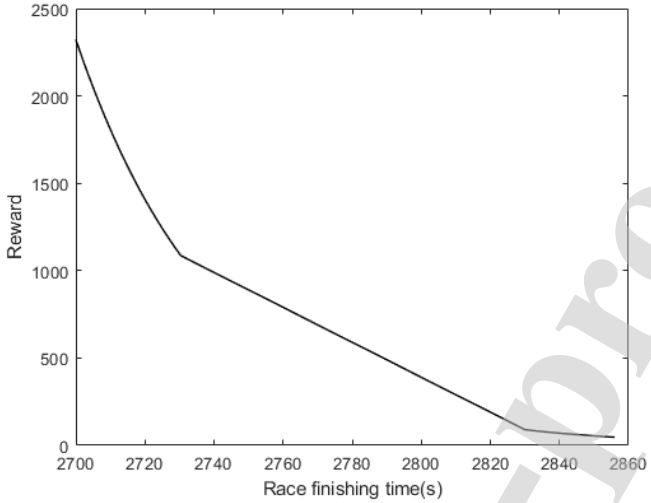


Figure 3 Race time reward

Given by figure 2, a successful lap with extreme energy-saving action is no longer than 83.5 seconds. Hence in equation 7, we give a positive reward for any race finishing performance with lap time quicker than this value. The term $130N_{lap}$ is to add an extra penalty upon t_{race} when a race is terminated unfinished due to flat or overheated battery in which 130 is a much longer time than a normally finished lap. For a 34-lap race, by simple calculation, the average energy consumption is 1.6kWh/lap. According to figure 2, this ends up with a race finishing time around 2733s. The time reward is given a non-linear feature so that when race time is shorter than 2733s, the reward grows exponentially. This high gradient in the reward encourages the agent to ascent faster to a shorter race finishing time. Additionally, this final race time reward has a higher magnitude than step reward. The reason is to guarantee that the final race time has the dominant effect to incent the agent to find faster race finishing solutions. Meanwhile the step reward remains capable of modifying the agent's behavior to correct actions but also not to compromise the final race time.

Other important features in the reward function are the battery temperature and energy penalties. Traditionally, in optimization problems, these violations are usually treated as hard boundaries or with a sharp cut-off in reward calculations. This is not favourable in this study where an A2C-like method is implemented to find the optimal solution, which theoretically lies on the state boundaries (i.e. energy fully used and battery temperature at the limit). In A2C methods, the role of the critic network is to estimate the expected reward which allows the actor to update its policy towards the optimal. Due to the fitting/regression nature of neural networks, it is extremely hard to make the critic network capable of precisely predicting a step feature (shown in figure 4(a)) made by the sharp drop of reward. If hard boundary is used for reward design, the critic is very likely to under-predict or over-predict the boundaries, either of which will not help the actor ascend to the real edge. In contrary, by

using soft boundary penalties, the gradient outside the boundary will favor the critic's performance hence make it easier for the actor finding the optimal policy. The effect of reward design is further discussed in section 4.1.

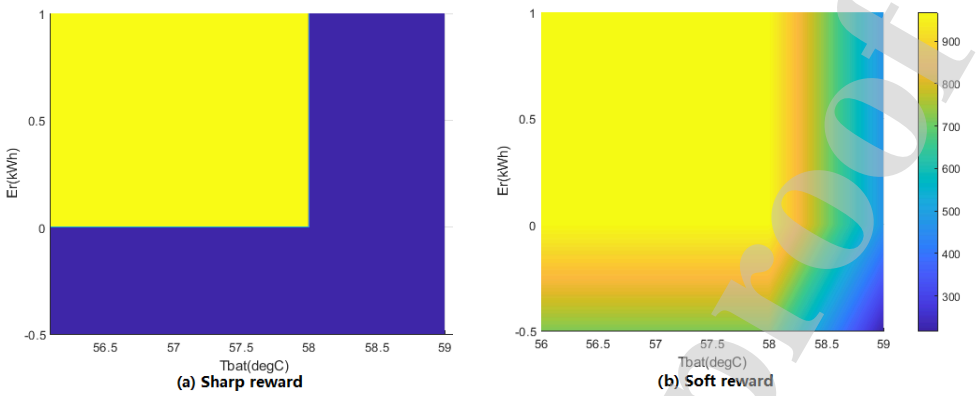


Figure 4 Reward type illustration: (a) Hard boundary cut off (b) Soft boundary penalty The reward is a function of T_{bat}, E_r, t_{race}. This figure only shows the reward on T_{bat} and E_r dimensions assuming t_{race}=2740s

3. FE race strategy development using Ape-X distributed DDPG reinforcement learning

3.1 DDPG algorithm

DDPG is an A2C-like RL algorithm. It has two primary networks, the critic $Q(s,a|\theta^Q)$ with weights θ^Q and the actor $\mu(s|\theta^\mu)$ with weights θ^μ which maps a state s directly to an action μ. Both two primary networks have their subnets called target critic net $Q'(s,a|\theta^{Q'})$ and target actor net $\mu'(s|\theta^{\mu'})$. The critic (i.e. action-value function) describes the expected return after taking an action. In many discrete action RL approaches, it is trained by using recursive relation known as the Bellman equation:

$$Q^\pi(s_t,a_t) = E_{r_t,s_{t+1}\sim\Psi}\left[r(s_t,a_t) + \gamma E_{a_{t+1}\sim\pi}[Q^\pi(s_{t+1},a_{t+1})]\right] \quad (8)$$

where $\gamma \in [0,1]$ is the discount factor in Bellman equation, Ψ is the corresponding expectation distribution for s_{t+1} and r_t in the environment. When the policy distribution π becomes deterministic μ, the bellman equation can be transformed into:

$$Q^\mu(s_t,a_t) = E_{r_t,s_{t+1}\sim\Psi}[r(s_t,a_t) + \gamma Q^\mu(s_{t+1},\mu(s_{t+1}))] \quad (9)$$

This means the expectation only depends on the environment and therefore critic can be trained off-policy from transitions (s_t,a_t,r_t,s_{t+1}) collected from a different policy β. The critic parameter θ^Q therefore can be optimized by minimizing the loss:

$$L(\theta^Q) = E_{s_t\sim\rho^\beta,a_t\sim\beta,r_t\sim\Psi}[(Q(s_t,a_t|\theta^Q) - y_t)^2] \quad (10)$$

Where ρ^β represents the distribution of the state s_t under the current policy β, and y_t is

given by:

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \quad (11)$$

With the aid of critic, the objective of the actor is simply to maximize the expected return:

$$J(\theta^\mu) = E\left[Q(s,a)|_{s=s_t, a=\mu(s_t)}\right] \quad (12)$$

Therefore, the policy updating gradient can be written as:

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{Q_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right] \quad (13)$$

DDPG uses additional two target networks to improve the stability of learning. The target critic Q' and target actor μ' are used to calculate the y_t value in equation (11). This is to avoid critic net being trained and used to calculate target value simultaneously which would easily make the update diverge. The weights of the target networks are slowly updated by tracking the primary critic and actor:

$$\begin{cases} \theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{cases} \quad (14)$$

where $\tau \ll 1$ is a small number. In this way, y_t is constrained to change slowly thus learning stability is improved.

3.2 Prioritized experience replay

DDPG is an off-policy method which requires a replay buffer to store the transitions collected from old policies and to sample a mini-batch of transitions for a network updating step. Since the actor updates its policy solely dependent on critic, the accuracy of the critic value approximation is crucial for DDPG performance. Based on eq.11 the criterion for critic performance in DDPG is usually the temporal-difference (TD) error given by:

$$\delta_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_t)) - Q(s_t, a_t) \quad (15)$$

Conventional off-policy methods use uniform random sampling to select which transition experience to replay. Hou et al. [35] pointed out that such uniform sampling without considering TD-error would very likely to select large amount of experiences of low errors. Training on such batches may fall into a local optimal solution. It is natural that experiences with higher TD-errors should be given higher priorities. However, greedily selecting experiences with largest TD-errors suffers from shortcomings such as time-costly sweeps of entire replay buffer, lack of experience diversity and over-fittings. Therefore, in this study, a stochastic prioritization technique called prioritized experience replay (PER) is used to make a trade-off between uniform random sampling and greedy TD-error prioritization.

According to [36], the probability of transition i in the replay buffer being selected, is calculated by:

$$P_i = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (16)$$

where p_i is the priority of transition i. Exponent α describes how much prioritization to be used. A case of $\alpha = 0$ means uniform random sampling. In this study, we use the direct, proportional prioritization where $p_i = |\delta_i| + \epsilon$, with ϵ being a small positive constant to avoid the probability of a transition becoming zero and thus, it will not be revisited once its TD-error becomes zero after network updates.

Prioritized replay inevitably introduces a bias which changes the distribution the critic expected to converge to. To unbias such an effect, importance-sampling (IS) weights [37] are added:

$$\omega_i = (\frac{1}{N} \cdot \frac{1}{P_i})^{\beta} \quad (17)$$

Where N is the mini-batch size. β is a correction exponent; with $\beta = 1$ is to aggressively correct the probability P_i. In typical RL cases, the unbiased feature of updates is crucial when the RL training process is approaching to the end. Practically in this paper, $\beta = 0.5$ is assigned in the early stages of the training and then will be increased linearly to $\beta = 1$ when approaching the end. The updates of the critic will be based on $\omega_i \delta_i$ instead of δ_i with ω_i being normalized by $\frac{1}{max_i \omega_i}$ for stability reasons [36].

3.3 Network layout and exploration

As previously discussed, the actions in FE comprises both discrete and deterministic components. This is very different from conventional RL problems where the actions are either fully discrete or deterministic. Moreover, it needs to be accounted that the discrete actions have their effect on the deterministic ones. Instead of conventional networks, which takes state input in the first layer and output actions in the final layer, we use a different layout to accommodate this hybrid-type actions and their mutual effect. The actor network layout, which is used in this study, is shown in figure 5.

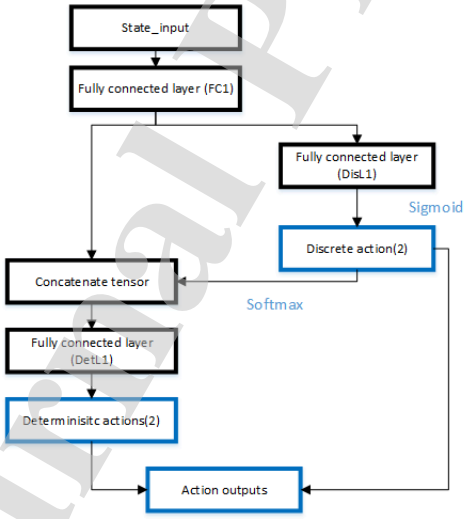


Figure 5 Actor network layout

The actor network takes in the state information and directly pass through the first fully-connected layer *FC1*. The output of *FC1* is shared by two tensor paths. On the right-hand side, *FC1* is passed through a second fully connected layer and activated using Sigmoid function. This tensor (size of 2) is used as the discrete action output of actor and passed to the environment to generate a one-hot vector as a signal for attack mode activation. Meanwhile,

this tensor is converted into a probability distribution tensor with sum of one using Softmax function and concatenated with the *FC1* output on the left-hand side path. Then through another fully-connected layer, this tensor will be used as the deterministic action output.

The critic network is relatively simple thus is not demonstrated here. Details of the networks are shown in appendix B.

An RL agent learns its policy based on exploring the action space. Each different RL algorithm has its suitable exploration technique. For example, epsilon-greedy [26] algorithm is widely used in discrete action space applications. The agent does random selections occasionally with probability ϵ and follows its own policy for most of the time with probability $(1-\epsilon)$. The Upper Confidence Bounds (UCB) algorithm [38] is usually used in tree searches such as Monte Carlo Tree Search (MCTS) to balance the visits of nodes based on the visit history and collection of rewards. To encourage exploration in continuous action space, the actor deterministic outputs are usually added with noises such as Ornstein-Uhlenbeck (OU) process [39]. The FE strategy development problem has several unique features which make direct implementation of any of these classical methods inappropriate. The FE environment is usually a short-episode environment. In this study, a full episode has 34 steps at most. Finding the optimal policy is a delicate job due to the energy and temperature constraints. During the ending phase of RL, big noise effect would easily violate the constraints, which prevent the agent from reaching the optimum. Epsilon-greedy method cannot be applied to continuous actions and adding noises to discrete actions is not effective enough due to its magnitude. To overcome this, we use a hybrid-type exploration method by integrating epsilon-greedy method with continuous noise under a recursive decay policy.

For most of the time with probability of $(1-\epsilon_{EO})$, an episode will be performed as an exploration episode; while occasionally an episode is performed by directly following the actor policy with probability of ϵ_{EO}. Inside an exploration episode, there are also occasional non-exploration action steps determined by probability of ϵ_{EI}. Therefore, the probability of an exploration action being taken is $(1-\epsilon_{EO})(1-\epsilon_{EI})$. If an action is to explore, the discrete actions and deterministic actions are treated differently. We use η to denote the level of randomness for exploring. For deterministic actions, we add basic zero-mean Gaussian noise to the actor output:

$$\eta\mu'_{Det} = \mu\left(s_t\middle|\theta_t^{\mu}\right) + \mathbb{N}_{Gau}(\eta) \tag{18}$$

Where η demotes the standard deviation of the Gaussian distribution. For discrete actions, we still use greedy method. With probability of (1- η), a random action will be picked from the action space which in this case is either [0,1] or [1,0]. By this way, we guarantee enough exploration needed to learn the policy meanwhile, we ensure that large noise does not compromise the convergence near the end. We recursively decay the η through episodes:

$$\eta_i = \frac{\eta_0}{1+\varepsilon\cdot(i \bmod N_{decay})} \tag{19}$$

where i is the number of current episode, η_0 is the initial randomness level, ε is the decay factor, and N_{decay} is the recursive decay period.

3.4 Ape-X distributed reinforcement learning

An Ape-X architecture comprises three major components, namely a learner, an experience

replay memory and numbers of workers. This is shown in figure 6.

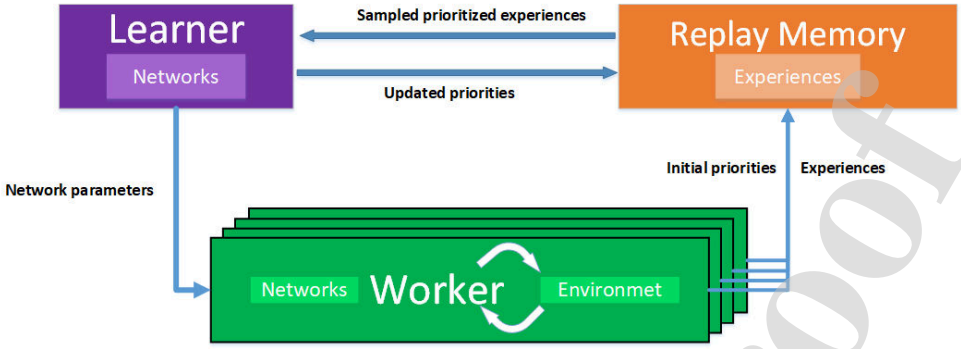


Figure 6 Ape-X architecture

In Ape-X, a centralized replay memory is used to store the transition experiences for the learner. These experiences were periodically received from the workers along with the initial priorities calculated also by the respective workers. The workers periodically obtain the latest network parameters from the learner and use those to interact with their own environments. The job of learning is solely completed by the learner who updates the networks parameters based on experiences from the replay memory. After each update, the new calculated priorities are sent back to the replay memory to update the priorities of the sampled transitions. In this study, each worker interacts with an environment what is initialized with random ambient temperature ranging from 20℃ to 35℃ in order to guarantees that the RL agent robustly learns from varieties of ambient temperatures.

3.5 Implementation

The combined implementation of Ape-X integrated with DDPG, PER and exploration decay is described in a pseudocode as follows. See appendix for additional details.

Algorithm 1 Ape-X Worker

procedure WORKER(Learner, PER)

Initialize worker network parameters $Q_w(s,a|\theta^{Q_w})$, $Q'_w(s,a|\theta^{Q'_w})$, $\mu_w(s|\theta^{\mu_w})$, $\mu'_w(s|\theta^{\mu'_w})$ from Learner $Q_L(s,a|\theta^{Q_L})$, $Q'_L\left(s,a\middle|\theta^{Q'_L}\right)$, $\mu_L(s|\theta^{\mu_L})$, $\mu'_L\left(s\middle|\theta^{\mu'_L}\right)$

while run **do**

Initialize environment with random ambient temperature $s_0 \leftarrow$ Environment. Initialize()

Decide if explore based on probability $(1-\epsilon_{EO})$

Compute decay randomness level η if explore, else $\eta = 0$

for t=1 **to** T **do**

Select action $a_t = \mu_w(s|\theta^{\mu_w})$ according to current policy and add random move $\mathbb{N}(\eta)$ if explore

Obtain reward r_t and new state s_{t+1}

Locally store transition, LocalBuffer. add(s_t, a_t, r_t, s_{t+1})

end for

Periodically **do**

Sample transitions batch (s_B, a_B, r_B, s_{B+1}) from local buffer

Compute batch TD-error $\delta_B = r_B + \gamma Q_w^{'}(s_{B+1}, \mu_w^{'}(s_B)) - Q_w(s_B, a_B)$

Compute batch priorities $p_B = |\delta_B| + \epsilon$

Send batch transitions and priorities to PER replay memory

Obtain latest network parameters from learner and update worker networks

Algorithm 2 Ape-X Learner

Procedure LEARNER(PER)

Initialize critic network $Q_L(s, a|\theta^{Q_L})$ and actor network $\mu_L(s|\theta^{\mu_L})$ with weights parameter θ^{Q_L}, θ^{μ_L}

Set target network $Q_L'\left(s, a\middle|\theta^{Q_L'}\right) = Q_L(s, a|\theta^{Q_L}), \mu_L'\left(s\middle|\theta^{\mu_L'}\right) = \mu_L(s|\theta^{\mu_L})$

Set exponent $a = 0.5, \beta = 0.5,\ \eta_0 = 0.4$, minibatch size K

for t=1, T **do**

Sample a prioritized transition batch K from PER replay memory with probability $P_j = \frac{p_j^\alpha}{\sum_i p_i^\alpha}$

Compute importance-sampling weight $\omega_j = (\frac{1}{N} \cdot \frac{1}{P_i})^{\beta} \cdot \frac{1}{max_i \omega_i}$

Compute TD-error $\delta_j = r_j + \gamma Q_L^{'}(s_{j+1}, \mu_L^{'}(s_j)) - Q_L(s_j, a_j)$

Update the critic network by minimizing the loss: $L_K = \frac{1}{K}\sum_i \omega_i \delta_i^2$

Update the actor network using policy gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{K}\sum_i \nabla_a Q_L(s_t, a_t|\theta^{Q_L})|_{s=s_i, a=\mu_L(s_i)} \nabla_{Q_{\mu_L}} \mu_L(s_t|\theta^{\mu_L})|_{s_i}$

Update the target networks:

$$\theta^{Q_L'} = \tau\theta^{Q_L} + (1-\tau)\theta^{Q_L'}$$
$$\theta^{\mu_L'} = \tau\theta^{\mu_L} + (1-\tau)\theta^{\mu_L'}$$

Update the priorities of transitions in PER replay memory according to $\left|\delta_j\right|$

end for

To demonstrate the advantage of continuous action space in FE strategy development, this method will be implemented and later compared against the Monte Carlo Tree Search with policy gradient reinforcement learning method, which was proved to be reliable and stable for strategy development in the literature [21]. The pseudocode of MCTS approach is detailed in appendix C.

4 Results and discussion

In this section, we first show the effect of reward shaping mentioned in section 2.3. Then the examples of race strategy solutions will be demonstrated and compared against MCTS

approach.

4.1 Reward shaping effect and performance

In section 2.3, we proposed two ways of reward formulation. Figure 7 shows the convergence performance of the two methods. In order to avoid poor readability caused by looking into a wide range of ambient temperatures studied using Ape-X, we create an additional background thread to constantly perform noise-free episodes using the latest network parameters of the learner. For demonstrating the reward shaping effect, the background performer uses temperature of 30°C, which is a common ambient temperature when a race is held in Merrakesh (i.e. the track used in this study).

As shown in Figure 7, In the first 18k episodes, the performances are similar for the two methods. During this stage, the agents were unable to finish a race due to either energy or thermal violations. Then, gradually the agents became capable of conservatively finishing with a slow race time. Between 20k and 60k episodes, two methods performed differently. While the performance of sharp-reward method (referred to as SHARP) stopped further improving and maintaining a conservative finishing level, the smooth-shaping method (referred to as SMOOTH) began to find faster and faster race time solutions. Later in the ending phase between 60k and 80k episodes, SHARP still failed to improve and also started showing some instability. In contrary, SMOOTH started to stabilize at a high-solution-quality level.

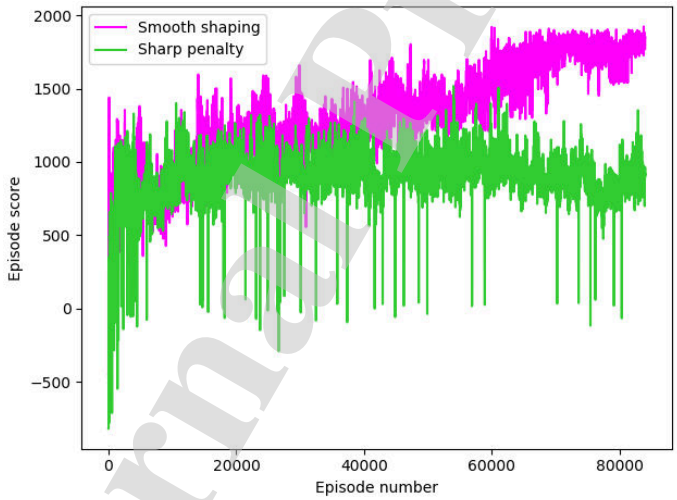


Figure 7 Convergence performance of different reward formulations

As mentioned previously, the performance is largely dependent on the critic network predictions. Figure 8 below shows the prediction by critics from both methods. Because the final lap/step is where constraint violations mostly occur and cause change in reward magnitude, and in order to avoid the effect of discount factor in Bellman equation on early laps/steps, the critics are tested based on an identical second-to-last lap state.

$$\left[N_{lap}, E_r, T_{bat}, T_{amb}, N_{att}, PM, N_{Rattlap}, t_{race}\right] = [56.6, 1.26, 1, 0, 0, 30, 2635.96]$$

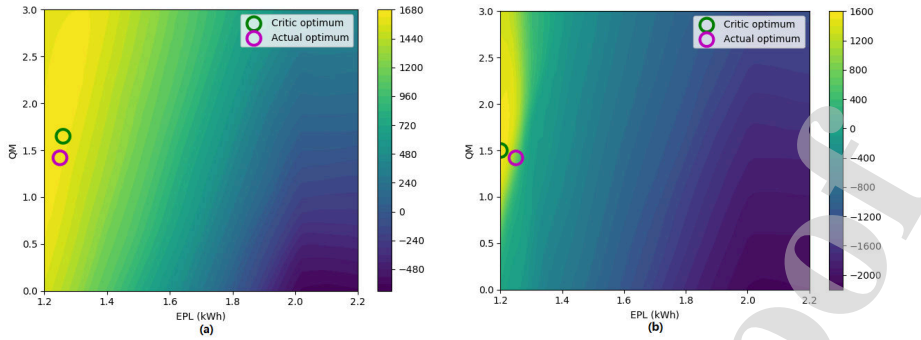


Figure 8 Comparison of critic network predictions: (a) Smooth-reward critic; (b) Sharp-reward critic

We ran exhaustive searches to find what are the optimal values suggested by the critics and the actual optimal action based on the transition models. It was observed that from both methods, the critic optimal points are all very close to the actual optimum, even SHARP is closer. However, the contour in figure 8 shows that SHARP critic has much lower prediction values than the SMOOTH critic. Because this is the final step, this behavior can be further detailed by comparing the critic values and the actual rewards calculated by respective reward functions. The relative error is shown in figure 9.

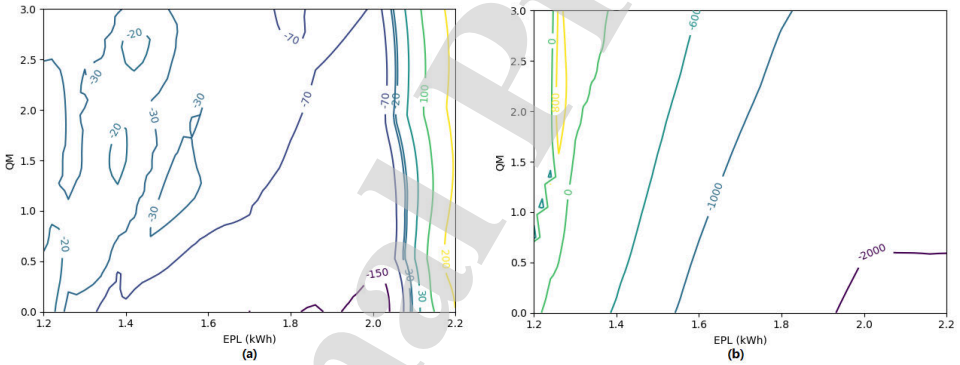


Figure 9 Relative error between critic value and actual reward: (a) Smooth-reward critic; (b) Sharp-reward critic

It can be clearly observed that SMOOTH has much more accurate estimations of the actual rewards than SHARP. Moreover, huge gradient can be seen in the SHARP error. Note that despite SHARP has direct cut off reward for constraint violations, the time reward term in equation (7) remains identical for both methods. By joining figure 8 and figure 9, it can be seen that SHARP's overall estimations based on the given state is much lower than that of SMOOTH and the high values locate on the high QM and low EPL actions which are very conservative actions. This is a reflection of SHARP's poor evaluation on the tested state that SHARP evaluates it as a very bad state with large chance of constraint violation thus should be avoided earlier in an episode by taking more conservative actions. This is a major cause of

SHARP failing to find faster race solutions as shown in figure 7. This can also be proved by the actual complete episode solution SHARP generated, which will be discussed in the next section.

4.2 Race strategy solutions

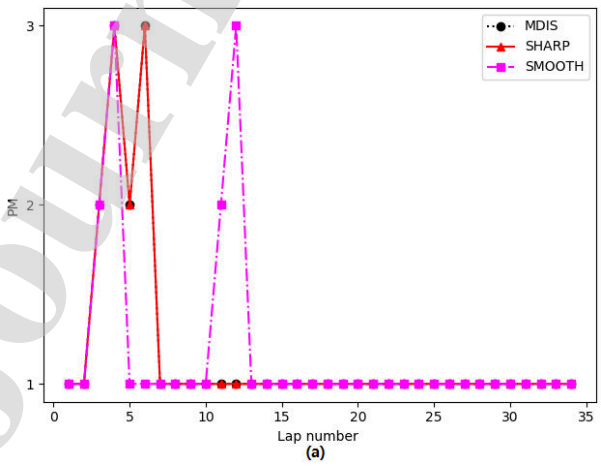
In this section, we compare the solutions generated from three methods, the SHARP and SMOOTH which are DDPG based with different reward shaping, and MCTS based reinforcement learning with completely discretized action space (referred to as MDIS). Two different cases are presented. The first case is given relatively higher ambient temperatures which make thermal management a major concern while the second case, in contrary, has a lower ambient temperature.

4.2.1 Case 1: ambient temperature of 30℃ and 32℃

Table 6 reveals the terminal state information from the strategies generated by the three methods. It can be seen that SMOOTH outperformed the other two methods in terms of both boundary management (i.e. smaller margin to the boundaries means better use of the resource) and the final race time. The solution from continuous-action-based SMOOTH finished a race 3 seconds faster than the discrete-action-based MDIS and nearly 6 seconds faster than SHARP. This 0.1 % advantage might seems small in magnitude but 3 seconds of time in motorsport races is a significant improvement and would make huge difference in a race. Figure 10 show the strategic actions generated by these methods.

Table 6 Terminal states using different methods with T_{amb}=30℃

Method	Remaining energy (kWh)	Battery temperature (℃)	Race finishing time (s)
SMOOTH	0.025	57.944	2714.28
SHARP	0.176	57.578	2720.21
MDIS	0.295	57.886	2717.26



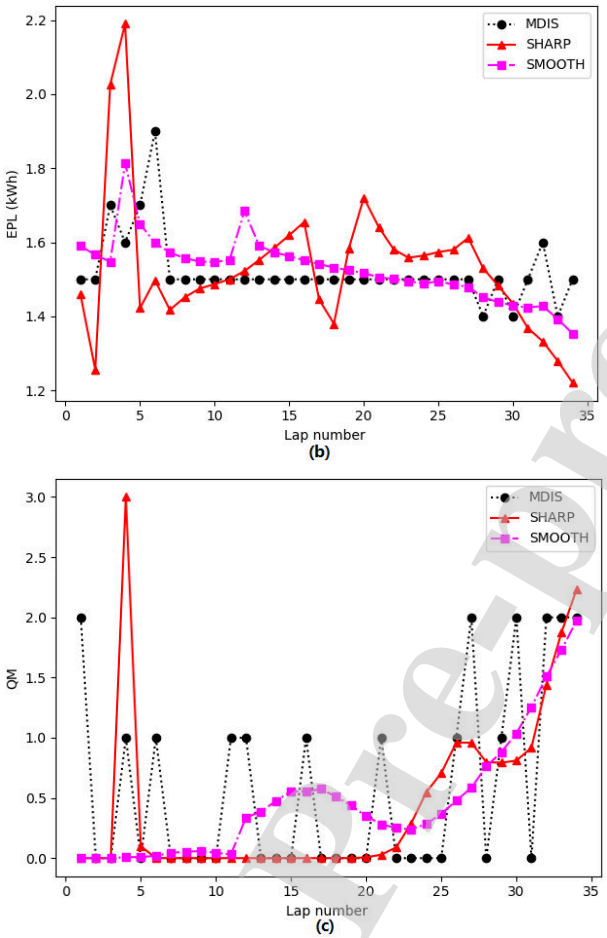


Figure 10 Race strategic solutions for T_{amb}=30°C: (a) Power mode (PM); (b) Energy per lap (EPL); (c) Q mode (QM)

The first obvious difference is the use of attack mode. All three methods suggest that the first attack mode should be activated in the third lap, which is the earliest time the regulation allows. Then SHARP and MDIS activate the second attack mode at lap number 5 immediately after the first one finishes while SMOOTH waits until the eleventh lap to activate the second attack. Features can be observed in EPL actions, which are directly related to the attack mode. For MDIS and SMOOTH, it can be seen that when attack mode is used (lap 3-7 for MDIS; lap 3,4,11,12 for SMOOTH), the EPL is significantly higher than the other regular laps. While SHARP actions is more chaotic, it is still clear that SHARP makes a much higher EPL action during its attack mode laps (i.e. lap 3,4). It should also be noted that MDIS prefers to have a relative constant EPL action while the fastest solution from SMOOTH suggests a general decrease trend as race progresses. Another feature can be found, shared by all three methods, that the EPL values all start to drop towards the end of the race (i.e. after lap 27). The cause of these early uses of attack mode and EPL drop in the end can be explained referring to the battery thermal dynamics model (appendix D) used in the OCP. According to the battery model, the heat generation is proportional to the magnitude of current, which will increase in

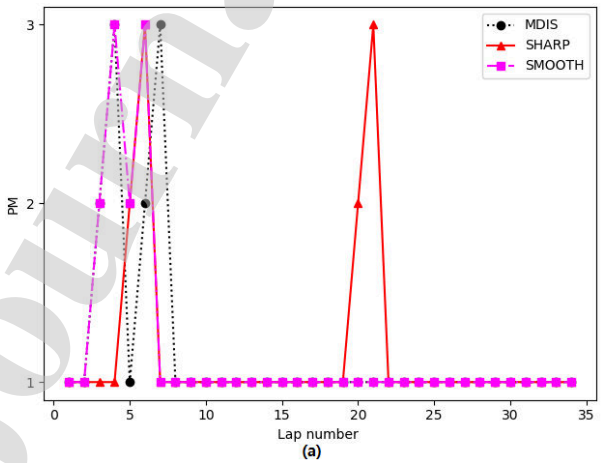
the later stages of a race due to lower open circuit voltage (OCV) caused by SOC drop. The heat generation rate will be higher during the ending phase of a race thus need to be properly managed. Additionally, the 50kW of higher power of attack mode would more likely lead to thermal crisis if activated in the later stages. Therefore, attack modes are suggested to be activated in the earlier laps of a race. This temperature managing phenomenon can be jointly proved by the QM actions. Higher QM actions can be observed in the later stage of the race suggesting stricter temperature management measures need to be taken. Another feature of interest is that instead of monotonically increasing the QM towards the end, both continuous-action-based methods suggest a hump before the final increase. The fastest solution from SMOOTH suggests a hump in the middle of a race. The reason is assumed to be related to the battery entropy coefficient change (appendix D) in the middle range of SOC, which would cause additional heat generation.

Then, we have further increased the ambient temperature to 32℃, which would be more challenging for temperature management. The terminal information corresponding to this case is shown in table 7.

Table 7 Terminal states using different methods with T_{amb}=32℃

Method	Remaining energy (kWh)	Battery temperature (℃)	Race finishing time (s)
SMOOTH	0.092	57.927	2717.67
SHARP	0.302	57.481	2728.53
MDIS	0.016	57.936	2722.57

With higher ambient temperature, which requires more thermal management, it is natural that the race times are slower than those in case one. Among the three methods, SHARP remains the slowest one. It can also be observed that the advantage of SMOOTH becomes larger over the other two with nearly 5 seconds faster than MDIS and over 10 seconds faster than SHARP. In the action solutions shown in figure 11, some features can be observed different from the 30℃ case.



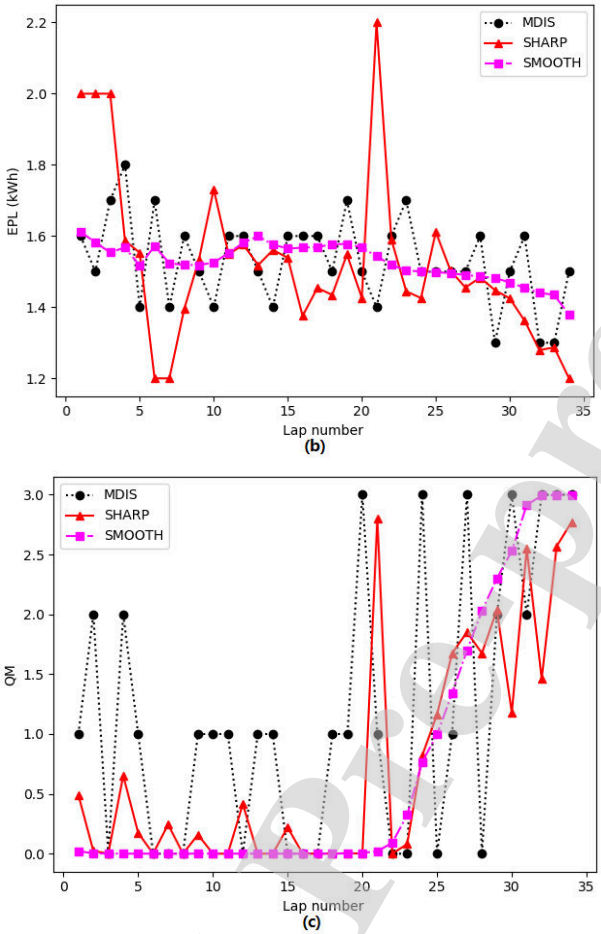


Figure 11 Race strategic solutions for T_{amb}=32℃: (a) Power mode(PM); (b) Energy per lap (EPL); (c) Q mode (QM)

On attack mode usage, both SMOOTH and MDIS activate it in the early laps while SHARP gives a big delay for the second activation. The general dropping trend of EPLs remains similar as case 1. However, high EPL for attack mode laps cannot be clearly seen in case 2. Only small increases can be observed in lap 4 and 6 where SMOOTH is on the attack mode laps. The biggest difference between the two cases lies in the QM actions that the hump in case 1 does not happen in case 2. While MDIS's and SHARP's QM actions have oscillated in the early and middle race stages before rising in the end, SMOOTH suggests no measurements for thermal management during these stages, and it generates higher QM actions only after lap 21 with magnitude monotonically increasing towards the end.

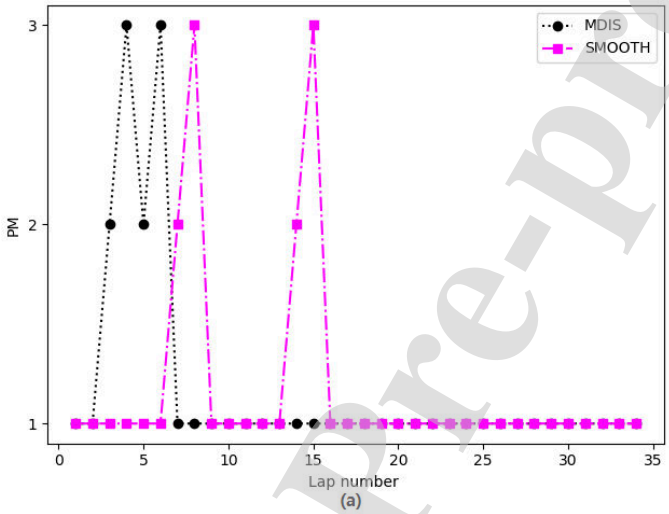
4.2.2 Case 2: ambient temperature of 25°C

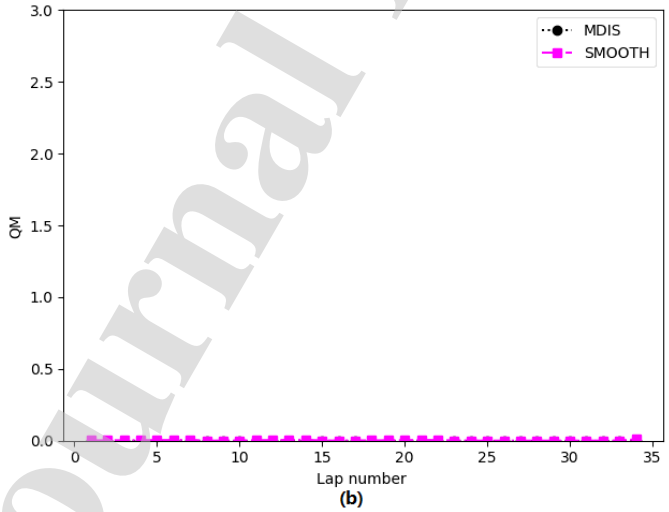
In this case, we lower the ambient temperature to 25°C, which would make the T_{bat} constraint of 58°C a less concern in the decision making process. Only SMOOTH and MDIS are compared in this case since SHARP failed to deliver strong performances in the previous

section. The terminal information corresponding to this case is shown in table 8.
It can be seen that in this case, two methods performs quite similar. SMOOTH is faster than MDIS but with gap of only 0.27 seconds. Figure 12 shows the detailed actions solutions.

Table 8 Terminal states using different methods with T_{amb}=25°C

Method	Remaining energy (kWh)	Battery temperature (°C)	Race finishing time (s)
SMOOTH	0.0022	57.995	2711.34
MDIS	0.003	57.202	2711.61





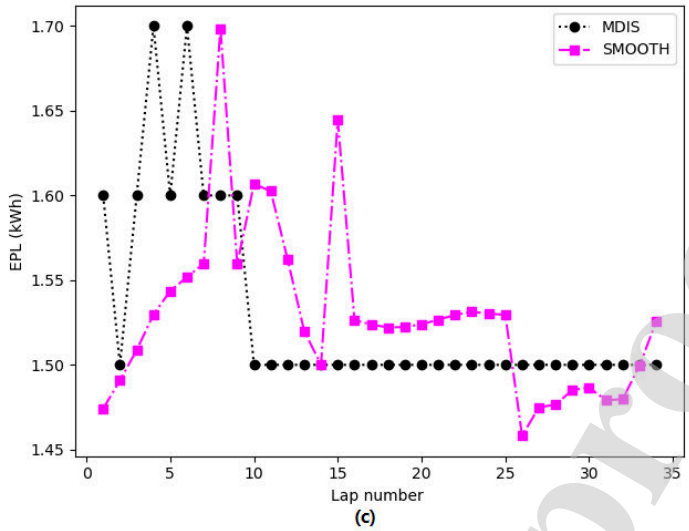


Figure 12 Race strategic solutions for T_{amb}=25℃: (a) Power mode(PM); (b) Q mode (QM); (c) Energy per lap (EPL)

The effect of lower ambient temperature can be clearly observed in QM action. Higher QM actions are suggested in case 1 to manage the battery temperature. While given T_{amb}=25℃, both SMOOTH and MDIS suggest that no QM actions need to be taken. This also explains why two methods have similar solutions. Regarding the fact that no QM action is needed, the total action space is reduced to an extent, which makes the problem simpler to the algorithms. Although two methods generate similar level of race times, the two strategies are clearly different. MDIS activates the attack modes at the very beginning (figure 12a), while SMOOTH activates separately and later than MDIS. Both high EPLs are observed when using attack modes (i.e. lap 3-7 for MDIS, and lap 7,8,15 for SMOOTH). Following the attack laps, MDIS suggests a constant EPL of 1.5kWh afterwards while SMOOTH shows a different pattern.
Overall, SMOOTH performs best and has clear advantage over MDIS when thermal management is a major concern. When thermal risk lowers, the continuous method and discrete method have similar level of performance. The summary of performance of SMOOTH and MDIS is shown in table 9.

Table 9 Comparison of performance in continuous and discrete action space

T_{amb}	MDIS	SMOOTH	Difference(compared to MDIS	
T_{amb}=25	2711.61(s)	2711.34(s)	-0.27(s)	-0.01%
T_{amb}=30	2717.26(s)	2714.18(s)	-3.08(s)	-0.113%
T_{amb}=32	2722.57(s)	2717.67(s)	-4.9(s)	-0.180%

More validation tests on different ambient temperatures are presented in appendix E.

5 Conclusion

In this study, the Formula-E race strategy was formulated into an MDP decision making

problem. A new layout of actor model was used to accommodate both discrete and continuous action types in the strategic decisions. Deep deterministic policy gradient method was implemented under Ape-X distributed architecture and integrated with prioritized experience replay. The reward was shaped in favor of the reinforcement learning algorithm. It was shown that reward shaping with soft penalties of constraint violations clearly benefits the performance of RL algorithm. The smooth-shaped reward fundamentally makes it easier for the critic to estimate the reward thus helps actor better to find optimal solutions in this study. Continuous-action-based method successfully found superior strategies of 3-4 seconds faster than discrete-action-based method. This advantage is more significant when overheating risk is severe in higher ambient temperature environment. Overall, a race can be finished clearly faster when the strategy is developed in continuous action space.

Declaration of Conflict of Interest:

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Reference

[1] "FIA Formula E." Energy Management 101: The Importance of Energy in Formula E | FIA Formula E, 18 Sept. 2020, www.fiaformulae.com/en/news/2020/march/formula-e-energy-management.

[2] Wieczorek, Maciej, and Mirosław Lewandowski. "A mathematical representation of an energy management strategy for hybrid energy storage system in electric vehicle and real time optimization using a genetic algorithm." Applied energy 192 (2017): 222-233.

[3] Montazeri, Morteza, Abbas Fotouhi, and Akbar Naderpour. "Driving segment simulation for determination of the most effective driving features for HEV intelligent control." Vehicle system dynamics 50.2 (2012): 229-246.

[4] Chen, Zheng, et al. "Energy management of a power-split plug-in hybrid electric vehicle based on genetic algorithm and quadratic programming." Journal of Power Sources 248 (2014): 416-426.

[5] Song, Chuanxue, et al. "Energy management of parallel-connected cells in electric vehicles based on fuzzy logic control." Energies 10.3 (2017): 404.

[6] Gong, Qiuming, Yaoyu Li, and Zhong-Ren Peng. "Trip-based optimal power management of plug-in hybrid electric vehicles." IEEE Transactions on vehicular technology 57.6 (2008): 3393-3401.

[7] Du, Yongchang, et al. "Trip-oriented stochastic optimal energy management strategy for plug-in hybrid electric bus." Energy 115 (2016): 1259-1271.

[8] Zhang, Chen, and Ardalan Vahidi. "Route preview in energy management of plug-in hybrid vehicles." IEEE Transactions on Control Systems Technology 20.2 (2011): 546-553.

[9] Martinez, Clara Marina, et al. "Energy management in plug-in hybrid electric vehicles: Recent progress and a connected vehicles perspective." IEEE Transactions on Vehicular Technology 66.6 (2016): 4534-4549.

[10] Perantoni, Giacomo, and David JN Limebeer. "Optimal control for a formula one car with

variable parameters." Vehicle System Dynamics 52.5 (2014): 653-678.
[11] Tremlett, A. J., and D. J. N. Limebeer. "Optimal tyre usage for a formula one car." Vehicle System Dynamics 54.10 (2016): 1448-1473.
[12] Limebeer, David JN, Giacomo Perantoni, and Anil V. Rao. "Optimal control of formula one car energy recovery systems." International Journal of Control 87.10 (2014): 2065-2080.
[13] Herrmann, Thomas, et al. "Energy management strategy for an autonomous electric racecar using optimal control." 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, 2019.
[14] Liu, Xuze, Abbas Fotouhi, and Daniel J. Auger. "Optimal energy management for formula-E cars with regulatory limits and thermal constraints." Applied Energy 279 (2020): 115805.
[15] Sulsters, Claudia, and R. Bekker. "Simulating Formula One Race Strategies." (2018).
[16] Choo, Christopher Ledesma Weisen. Real-time decision making in motorsports: analytics for improving professional car race strategy. Diss. Massachusetts Institute of Technology, 2015.
[17] Bekker, James, and W. Lotz. "Planning Formula One race strategies using discrete-event simulation." Journal of the Operational Research Society 60.7 (2009): 952-961.
[18] Heilmeier, Alexander, Michael Graf, and Markus Lienkamp. "A Race Simulation for Strategy Decisions in Circuit Motorsports." 2018 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2018.
[19] Conboninvestment. “Using Monte Carlo Simulations To Make Horse Racing Selections.” White Knight Racing, 20 Apr. 2014, whiteknightracing.wordpress.com/2014/04/22/using-monte-carlo-simulations-to-make-horse-racing-selections/.
[20] Heilmeier, Alexander, et al. "Application of Monte Carlo Methods to Consider Probabilistic Effects in a Race Simulation for Circuit Motorsport." Applied Sciences 10.12 (2020): 4229.
[21] Liu, Xuze, and Abbas Fotouhi. "Formula-E race strategy development using artificial neural networks and Monte Carlo tree search." Neural Computing and Applications (2020): 1-17.
[22] de l’Automobile, F´ed´eration Internationale. 2020 FIA Formula E Championship Technical Regulations and Sporting Regulations. Paris: F´ed´eration Internationale de l’Automobile, 2019
[23] Kober, Jens, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." The International Journal of Robotics Research 32.11 (2013): 1238-1274.
[24] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
[25] Mao, Hongzi, et al. "Resource management with deep reinforcement learning." Proceedings of the 15th ACM Workshop on Hot Topics in Networks. 2016.
[26] Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge (1998)
[27] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
[28] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." arXiv preprint arXiv:1509.06461 (2015).
[29] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." International conference on machine learning. 2016.
[30] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3-4 (1992): 229-256.

[31] Konda, Vijay R., and John N. Tsitsiklis. "Actor-critic algorithms." Advances in neural information processing systems. 2000.
[32] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
[33] Xiong, Xi, et al. "Combining deep reinforcement learning and safety based control for autonomous driving." arXiv preprint arXiv:1612.00147 (2016).
[34] Grondman, Ivo, et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42.6 (2012): 1291-1307.
[35] Hou, Yuenan, and Yi Zhang. "Improving DDPG via Prioritized Experience Replay." no. May (2019).
[36] Schaul, Tom, et al. "Prioritized experience replay." arXiv preprint arXiv:1511.05952 (2015).
[37] Mahmood, A. Rupam, Hado P. van Hasselt, and Richard S. Sutton. "Weighted importance sampling for off-policy learning with linear function approximation." Advances in Neural Information Processing Systems. 2014.
[38] Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem." Machine learning 47.2-3 (2002): 235-256.
[39] Uhlenbeck, George E., and Leonard S. Ornstein. "On the theory of the Brownian motion." Physical review 36.5 (1930): 823.
[40] Harris, Sarah, and David Harris. Digital design and computer architecture: arm edition. Morgan Kaufmann, 2015.
[41] Nair, Arun, et al. "Massively parallel methods for deep reinforcement learning." arXiv preprint arXiv:1507.04296 (2015).
[42] Clemente, Alfredo V., Humberto N. Castejón, and Arjun Chandra. "Efficient parallel methods for deep reinforcement learning." arXiv preprint arXiv:1705.04862 (2017).
[43] Babaeizadeh, Mohammad, et al. "Reinforcement learning through asynchronous advantage actor-critic on a gpu." arXiv preprint arXiv:1611.06256 (2016).
[44] Horgan, Dan, et al. "Distributed prioritized experience replay." arXiv preprint arXiv:1803.00933 (2018).

Appendix

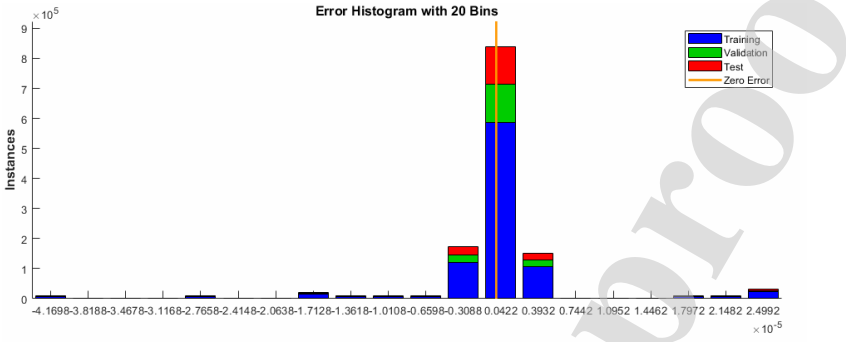
Appendix A Neural network transition models

The MCTS uses three neural networks as its transition models to separately predict the energy consumption, battery temperature rise and lap time.
In this study, fully connected shallow networks are used with adequate of prediction accuracy. The number of neurons and accuracy are shown in table A1 and figure A1.

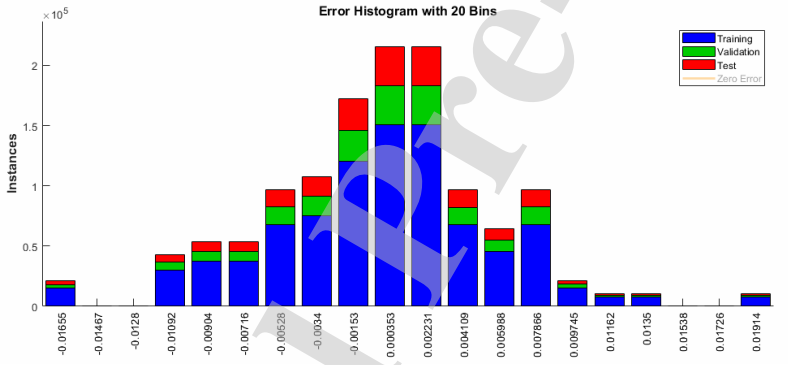
Table A1 Neural network structure

Prediction	Inputs	Hidden neurons	Output neurons	Activation function

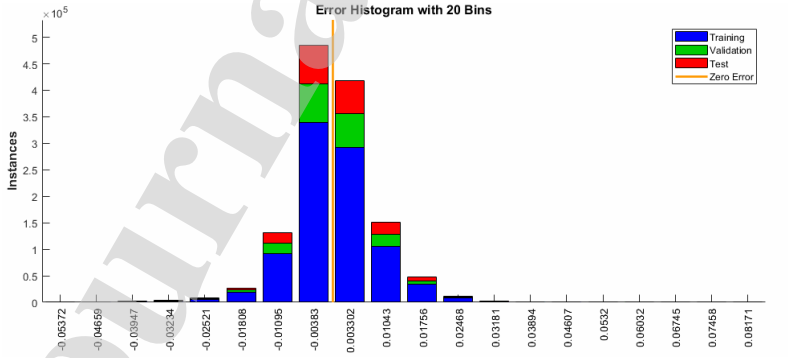
Energy consumption	EPL, PM, QM	6	1	Sigmoid
Lap time				
Battery temperature rise	EPL,PM,QM,SOC,Tbat,Tamb	10		



a) Energy consumption error



b) Lap time error



c) Battery temperature error

Figure A1 Prediction accuracy

Appendix B Actor and critic network parameters

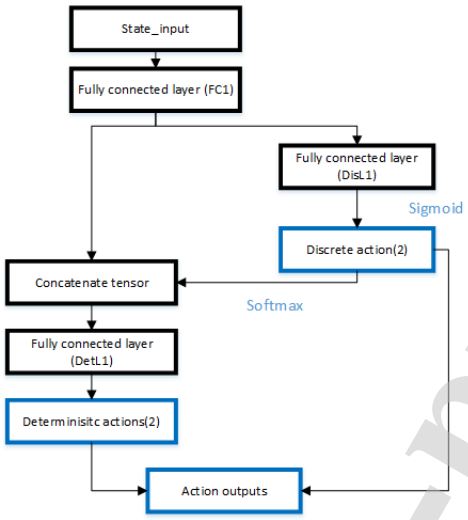


Figure B Actor network layout

The FC1 layer has 400 unites, followed by a Rectifier activation. The DisL1 has 300 unites, also with Rectifier activation and then is mapped into a size-2 discrete action activated using Sigmoid function. The discrete action is further activated using Softmax function and concatenated with the output of FC1. The tensor is then passed through a 300-unite layer followed by Rectifier and then though a 200-unite layer with sigmoid function becomes the size-2 deterministic actions.

The critic network has a simply fully-connected layout with size of 7(state size)-500-400-1 with Rectifier activation for all layers.

Appendix C Pseudocode for MCTS with policy gradient reinforcement learning

Algorithm 2 On-Policy reinforcement learning

Initialize policy network parameter θ

For epoch=1, M **do**

Initialize replay buffer R

Repeat episode

Initialize $s_0 = [N_{lap}, E_r, T_{bat}, T_{amb}, N_{att}, N_{Rattlap}]$

Store result sequence $Q(A,S), A, S \leftarrow MCTS(s_0, \theta)$ in R

Until episode number reached

for t = 1, T **do**

Sample a random minibatch of N transitions (S, A, $Q(A,S)$) from R

Update the policy using the sampled policy gradient

$$\theta = \theta + \alpha\nabla_\theta\big(Q(A,S) - B_Q\big)log\pi_\theta(A|S) + c\|\theta\|^2$$

end for

end for

Algorithm 3 Monte Carlo Tree Search

Function MCTS(s_0 , θ)

Create root node v_0 with state s_0

while within computational budget do

$v_l \leftarrow$ TreePolicy (v_0 , θ)

$\Delta \leftarrow$ Simulation($s(v_l)$, θ)

Backup (v_l, Δ)

return highest-reward sequence $Q(A,S), A, S$

Function TreePolicy(v, θ)

while v is not terminal **do**

if v is not expanded **then**

$v \leftarrow$ Expansion(v, θ)

else

$v \leftarrow$ Bestchild(v) based on $UCT\_MaxR(v)$

return v

Function Expansion(v, θ)

For i=1,Max Expansion Number **do**

choose a∈untried actions from A($s(v)$) based on policy $\pi_\theta(a|s)$

add a new child v' to v

with s(v')=Transition(s(v),a)

end for

$v \leftarrow$ first child $v'(1)$

return v

Function Simulation (s, θ)

while s is not terminal **do**

choose a∈A($s(v)$) based on policy $\pi_\theta(a|s)$

$s \leftarrow$Transition(s,a)

return reward for terminal state s

Appendix D Battery thermal dynamics model used in the OCP

$$m_b C_b \frac{d}{dt} T_b(t) = I^2 R(SOC) - IT_b\left(\frac{dU_{oc_b}}{dT_b}\right)(SOC) - h_{comb}(V)\, A_b(T_b - T_{amb}) \qquad \text{(D1)}$$

Where which m_b is the battery total cell mass, C_b is the specific heat capacity of the cell, T_b is the battery temperature, I is the battery current, $R(SOC)$ is the sum of polarization and

ohmic resistance of the battery as a function of SOC and $(\frac{dU_{oc_b}}{dT_b})(SOC)$ denotes the entropy coefficient which is also a function of SOC.

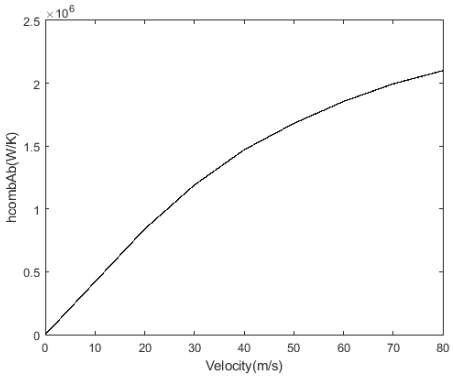


Figure D1

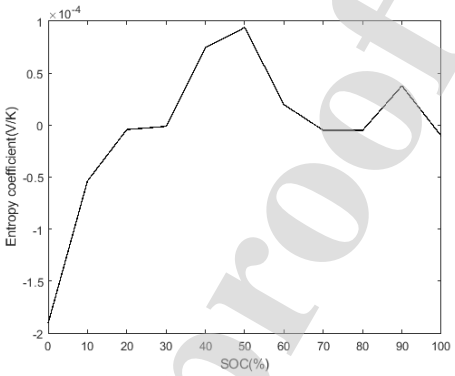


Figure D2

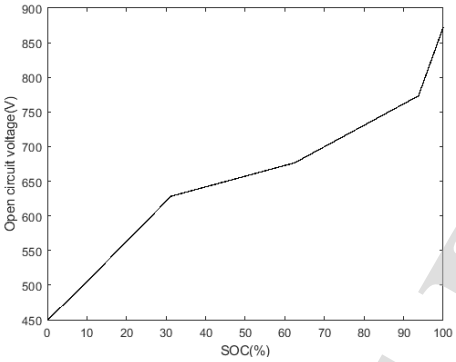


Figure D3

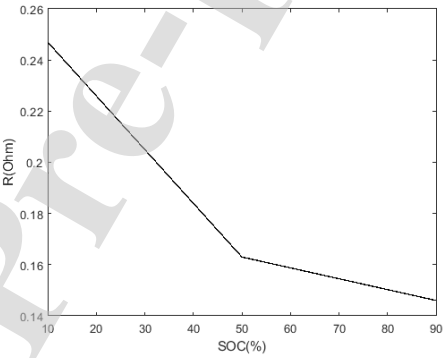


Figure D4

Figure D1 is the 'Combined heat transfer coefficient $h_{comb}(V)\ A_b$' appeared in equation D1

Figure D2 is the 'Entropy coefficient $(\frac{dU_{oc_b}}{dT_b})(SOC)$

Figure D3 is the 'Open circuit voltage U_{oc_b}' profile of the battery package in equation D1

Figure D4 is the 'Combined resistance $R(SOC)$' appeared

The table below shows other information in the model for result reproduction

Table D1

Parameters	Value
Battery cell number	209
Battery mass	317 kg
Battery specific heat capacity	1015 J/(kg· K)
Coolant mass	20 kg
Coolant specific heat capacity	1800 J/(kg· K)

Appendix E: Solutions on wider range of ambient temperatures

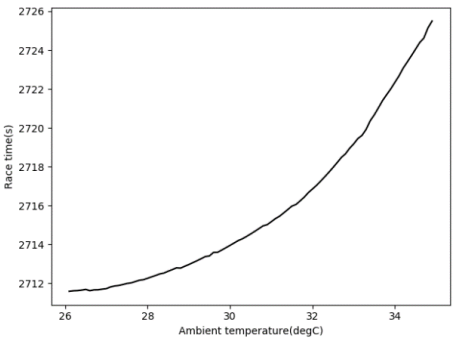


Figure E1

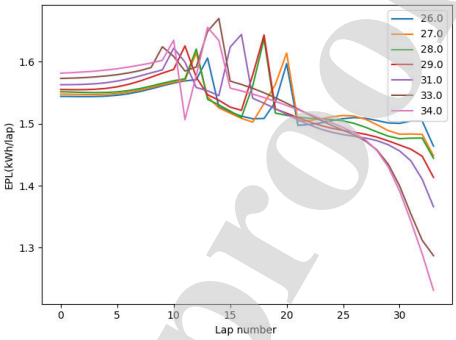


Figure E2

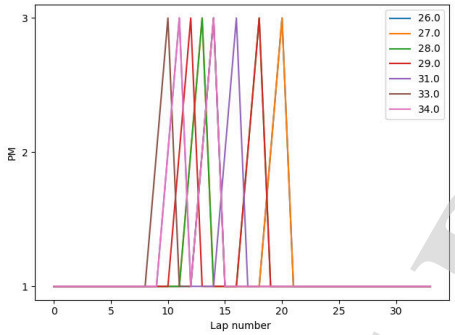


Figure E3

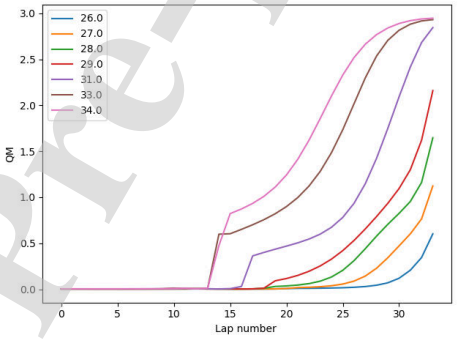


Figure E4

Figure E1 Race time at different ambient temperatures

Figure E2 Energy per lap solutions at different ambient temperatures

Figure E3 Power mode solutions at different ambient temperatures

Figure E4 QM actions at different ambient temperatures

As shown in figure E1, the race pace is significantly slower at higher ambient temperatures. A race can take more than 10 seconds longer at 34°C than at 26°C. To manage the battery temperature, as suggested by figure E3 and E4, much radical QM actions are taken in the final stage of a race. Meanwhile, with higher ambient temperatures, high EPL actions are biased to the early stages and EPL is reduced in the later stages in order to co-act with high QM actions to control the battery temperature within the regulatory limit.

Appendix F: Ape-X configuration and computing resource

This study is completed on a workstation with two Intel E5-2620 and a Tesla-K80 GPU. We tested different number of workers, batch size and replay memory size. The learning progress are shown in the following figures.

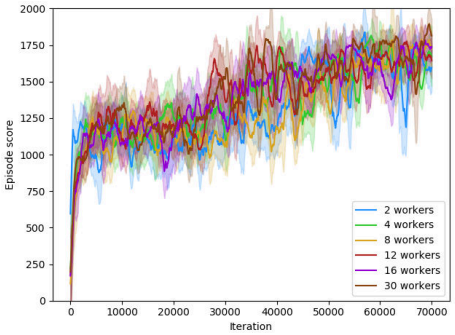


Figure F1

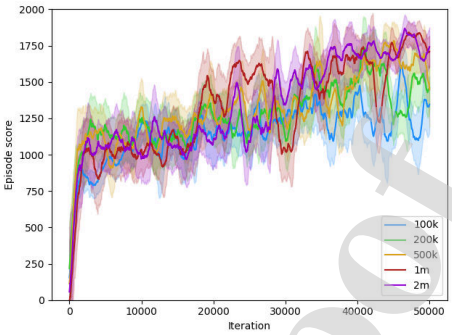


Figure F2

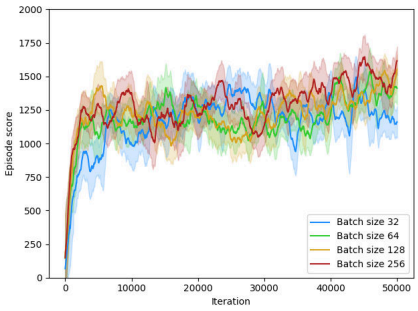


Figure F3

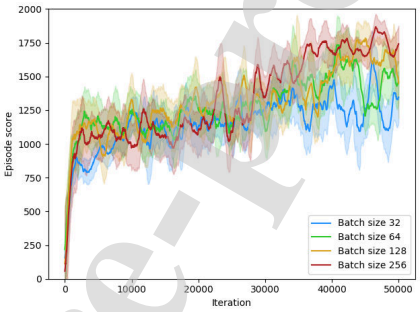


Figure F4

Figure F1: Performance with different number of workers
Figure F2: Performance with different replay memory size
Figure F3: Performance with different batch size (memory size 100k)
Figure F4: Performance with different batch size (memory size 2000k)

The cases showed in this section are trained for approximately 12 hours. As can be observed from figure F1, higher worker numbers have better performance. This advantage is believed to be more significant when larger number of workers is applied (e.g. 128 workers) as demonstrated in [44]. However, due to the limit of the workstation, the maximum worker number tested was 32. For Ape-X, as an off-policy approach, the performance relies on the diversity of collected experience, we further investigated the effect of size of replay memory. It can be cleared observed that with larger memory size containing more diverse data, the performance is much better. On the contrary, with a very small memory size (e.g. 100k, 200k), the performance improves very slowly, As can be observed from figure F3 and F4, bigger batch size can also improve the performance. By comparing figure F3 and F4, a combination of large memory size and batch size proves to be a stronger configuration in this study.

- Targeting the most popular race strategy problem in Formula-E races
- Propose a novel race strategy approach using distributed reinforcement learning architecture
- Reward shaping improves the reinforcement learning performance
- Modification of actor neural network layout for hybrid-type action generation

Xuze Liu: Conceptualization, Methodology, Software, Formal analysis, Writing - Original Draft, Visualization
Dr Abbas Fotouhi: Writing - Review & Editing, Supervision
Dr Daniel Auger: Supervision

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Formula-E race strategy development using distributed policy gradient reinforcement learning

Liu, Xuze

2021-01-20

Liu X, Fotouhi A, Auger DJ. (2021) Formula-E race strategy development using distributed policy gradient reinforcement learning. Knowledge-Based Systems, Volume 216, March 2021, Article number 106781

https://doi.org/10.1016/j.knosys.2021.106781