electronics

MDPI

Article

Autonomous Vehicle Fuel Economy Optimization with Deep Reinforcement Learning

Hyunkun Kim [1], Hyeongoo Pyeon [1], Jong Sool Park [2], Jin Young Hwang [2] and Sejoon Lim [3,*]

[1] Graduate School of Automotive Engineering, Kookmin University, Seoul 02707, Korea; newhyunkun@kookmin.ac.kr (H.K.); 20153439@kookmin.ac.kr (H.P.)
[2] Transmission Research Lab, Hyundai Motor Group, Hwaseong, Gyeonggi-do 18280, Korea; rrchart@hyundai.com (J.S.P.); jiny33@hyundai.com (J.Y.H.)
[3] Department of Automobile and IT Convergence, Kookmin University, Seoul 02707, Korea
* Correspondence: lim@kookmin.ac.kr; Tel.: +82-02-910-5469

Received: 15 October 2020; Accepted: 11 November 2020; Published: 13 November 2020

Abstract: The ever-increasing number of vehicles on the road puts pressure on car manufacturers to make their car fuel-efficient. With autonomous vehicles, we can find new strategies to optimize fuels. We propose a reinforcement learning algorithm that trains deep neural networks to generate a fuel-efficient velocity profile for autonomous vehicles given road altitude information for the planned trip. Using a highly accurate industry-accepted fuel economy simulation program, we train our deep neural network model. We developed a technique for adapting the heterogeneous simulation program on top of an open-source deep learning framework, and reduced dimension of the problem output with suitable parameterization to train the neural network much faster. The learned model combined with reinforcement learning-based strategy generation effectively generated the velocity profile so that autonomous vehicles can follow to control itself in a fuel efficient way. We evaluate our algorithm's performance using the fuel economy simulation program for various altitude profiles. We also demonstrate that our method can teach neural networks to generate useful strategies to increase fuel economy even on unseen roads. Our method improved fuel economy by 8% compared to a simple grid search approach.

Keywords: fuel economy optimization; reinforcement learning; autonomous vehicle; motion planning; velocity profile; numerical differentiation

1. Introduction

An increase in the fuel economy of a vehicle can reduce environmental impact and maintain cost. From 1990 to 2017, the average kilometers per liter (km/L) for all light-duty vehicles in the US was increased by 18% [1]. Nevertheless, in the same period, greenhouse gas emission from transportation increased by 71%, making it the second-fastest-growing emission source overall [2]. These statistics show that an increase in the number of new cars with better fuel economy adds more to the total emission than older cars' retirement subtracts. One central area we can improve in this situation is how we drive the car along a given trajectory. Designing velocity profiles pertinent to given road grade information can reduce fuel consumption considerably. Since many vehicles introduce the autonomous driving mode, the autonomous driving system in fully controls the vehicle from path planning to selecting what speed it will take for a given trip. Accordingly, it is possible to make the vehicle more fuel efficient in the autonomous vehicle mode [3–6].

Studies to optimize energy efficiency on electric vehicles (EV) are being carried on to reduce the impact on the environment. A precise prediction and planning of energy consumption on EV is critical because of limited battery capacity and long recharging time. The effect of the reference

cornering response on energy saving as both control allocation is studied in [7], and reference understeer characteristics can cause wheel torque distribution and generate yaw moments, impacting energy consumption. Their experiment on a four-wheel-drive EV showed that more energy could be saved through the appropriate sharing of the reference cornering response. A more comprehensive overview of EVs' existing torque distribution strategies with independently actuated drivetrains can be found in Reference [8]. In addition, Reference [9] derived a solution for energy-optimal routing by using the A* search framework. Their solution-based on A* search is faster than the generic framework with the Dijkstra or Pallottino strategy.

Compared to human driving, using a cruise control system can increase fuel economy as it holds the vehicle speed steady. Advanced cruise control using model predictive control (MPC) can further optimize fuel economy. In Reference [10], the authors proposed a system to improve fuel economy on congested roads where traffic conditions change frequently and have multiple intersections. This system captures the relevant information of the traffic status and road grades and predicts the future moving of the preceding vehicles. Then, it computes the optimal vehicle control input using MPC. They showed that MPC can increase 15% fuel economy compared to the Gipps model [10,11].

The MPC methods can optimize fuel economy for the finite time intervals with numeric methods and works well with linear systems. However, MPC is not suitable for long-term plans, and internal combustion engines (ICE), which are the most common in automobiles. The ICE models are highly nonlinear, making it hard to apply MPC in a straight forward manner.

A numerical method in ordinary differential equations and differential-algebraic equations for nonlinear MPC (NMPC) of heavy-duty trucks was introduced in Reference [12]. This method gives a chance for multiple predictive gear choices. However, the combination of nonlinear dynamics, constraints, and objectives with the gear choice's hybrid nature results in a challenging combinatorial prediction problem.

Both linear and nonlinear MPC-based models may lead to unreliable results in the real world driving conditions if their parameters are not adjusted for varying conditions such as vehicle mass, weather conditions, and fuel type. In Reference [13], the authors validated the concept of adaptive nonlinear model predictive controller (ANLMPC) by implementing their algorithm in a vehicle equipped with a standard production powertrain control module. In their ANLMPC approach, the vehicle and fuel model parameters were adjusted automatically. During a real-world test on a sport utility vehicle (SUV), ANLMPC improved fuel economy up to 2.4% on average compared to a production cruise controller with the same time of arrival. The works of References [14,15] incorporated quadratic programming to handle nonlinearity in improving fuel economy while cruising. Since these nonlinear MPC (NMPC) is difficult to design and can take a lot of CPU power, an approach to train a deep neural network (DNN) controller architecture on data from well-designed NMPC was proposed [16]. The DNN controller removes the process of solving complex optimization problems by stating estimation problems in real-time. Additionally, DNN allowed fast computation using GPU. They showed that the MPC could be successfully replaced with the trained DNN controller.

Another approach to optimize fuel in ICE is using dynamic programming (DP) to search velocity profile on the whole trip. In Reference [17], their method generated the optimal velocity profile in the distance domain using DP algorithm. They used cloud servers generated route once the driver tells the destination. Traffic information is also used to solve the fuel optimization problem using the DP algorithm. They improved 5–15% in fuel economy without considerable time loss. Reference [18] also studied to minimize trip time and fuel consumption using DP algorithm for a heavy diesel truck. They reduced 3.5% of fuel consumption without a time increment on a 120 km route. However, DP algorithm faces a crucial setback when the vehicle strays from its calculated optimal velocity profile. The DP algorithm has to calculate the whole velocity profile from the current position until it reaches a destination, which the vehicle may not follow again.

Deep reinforcement learning can overcome the above-mentioned issues, the high complexity, and other environmental factors. A famous example of a deep learning application is the Deep Mind's

AlphaGo [19]. Deep Mind's AlphaGo was expected to take decades for artificial intelligence to beat humans because of its vast search space and complexity. However, AlphaGo showed us that, with deep reinforcement learning, it won 4–1 against Lee Se-dol, which is one of the best Go players in the world. AlphaGo showed that reinforcement learning could overcome high complexity and respond appropriately to changing situations. In Reference [20], the authors proposed a deep reinforcement learning method that controls the vehicle's velocity to optimize traveling time without losing its dynamic stability. In Reference [21], deep reinforcement learning is used to control the electric motor's power output, optimizing the hybrid electric vehicle's fuel economy.

As previously mentioned, researchers tried to optimize fuel consumption of vehicles equipped with ICE. However, they confronted some difficulties like high nonlinearity with a computational load. Thus, we propose a deep reinforcement learning-based algorithm that trains autonomous vehicles to optimize fuel while following a given trajectory. The learning algorithm is expected to generate the optimal velocity profile after the training phase is done despite of high nonlinearity and to calculate in short time with the help of high-performing GPU. This optimum speed profile based on the given trajectory's altitude profile will maximize the effect of improving fuel economy for autonomous vehicles or vehicles equipped with some partial autonomous driving functionalities, such as smart cruise control system. Hyundai Motor's fuel economy simulation program (FESP) was used for the evaluation of the learned velocity profile in the process of training and testing, which calculates the accurate fuel consumption for the given trip. Fuel economy simulation, connected to the learning algorithm and treated as a black box is used as a proxy for measuring the goodness of the learning algorithm. We developed a novel method of using black box numerical differentiation and backpropagation over our DNN to train our learning algorithm. Furthermore, we developed a deep reinforcement learning algorithm that considers both trip road information and vehicle hysteresis status and generates an optimal strategy by effectively reducing the dimension of velocity profiles with a suitable parameterization for fast convergence. Our evaluation demonstrates that our method can teach a deep neural network to optimize fuel even on unseen roads.

The proposed deep reinforcement learning algorithm takes grade information of road ahead of a given journey and time budget (TB) as input while making an optimal velocity profile in the time domain. More specifically, the grade information is given as sequences. Our model generates velocity blocks of T second-interval for an output, and the velocity profile is made by concatenating them. Our model is trained on a realistic fuel consumption simulation program for ICE, treated as a black box for the purpose of calculating the gradient of reward (representing the goodness of our DNN) in terms of the network weights. The contributions of this paper are as follows.

- We developed a deep reinforcement learning based algorithm suitable for a fuel economy optimization problem.
- We developed a method for training deep neural network weights by combining backpropagation and numeric differentiation when a heterogeneous program (written in Fortran in our case) intervenes in the process of training the network.
- We represented the velocity profile in the time domain rather than in the spatial domain so that we can utilize the vehicles' time-varying states and control parameters. We further utilized parameterized representation for the vehicle speed control strategies rather than directly generating the velocity profile for faster convergence to the optimal solution.

2. Problem Formulation

For an autonomous vehicle or a vehicle equipped with smart cruise control to travel a predetermined distance within a predetermined time, there are numerous speed profiles.

In this paper, we aim to find a speed profile that satisfies the specified time condition and maximizes fuel economy while the autonomous vehicle moves from A to B. A velocity profile is a velocity plan over a trip, often expressed as a sequence of distance or hourly velocity divided into sufficiently satisfactory levels.

A trip involves starting a vehicle at a specified velocity through a given road until it reaches its end-point. Our proposed method generates a velocity profile V_p, while a vector composed of velocities v_t sampled evenly with sampling interval *Ts*, as output. Given the road's grade information and time budget as input, V_p should lead to minimum fuel consumption for the autonomous vehicle. The road grade information is given by an array of distance (m) and grade (%) pairs. We call this array, $Grd = [(d_1, g_1), (d_2, g_2), \ldots, (d_N, g_N)]$, where distances d_i is sampled with Ds making $d_i = (i-1)\cdot$Ds. We list the variable names and their meanings in Table 1.

Table 1. Terminologies and definition used in the fuel optimization problem.

Expression	Definition	Unit
d_k	Distance from the trip starting-point	meter
g_k	Grade at d_k	slope 100 (%)
p_t	Position at time t	meter
v_t	The velocity of the vehicle at time t	kilometer per hour (kph)
a_t	Acceleration of vehicle at time t	m/s^2
Grd	$[(d_1, g_1), (d_2, g_2), \ldots, (d_N, g_N)]$	-
$Grd(p_s)$	Grd for the road ahead where distance is measured from the position p_s	-
$\mathbf{V_p}(0, t)$	$[v_0, v_{0.1}, \ldots, v_t]$	-

When the velocity profile is generated for a given trip, it is passed to a vehicle dynamics simulator to calculate the fuel consumption and fuel economy FE. In this study, we used the fuel economy simulation developed by the Hyundai motor company in the calculation of the fuel consumption and fuel economy. Figure 1 shows the whole process from producing a velocity profile to calculating the fuel economy.

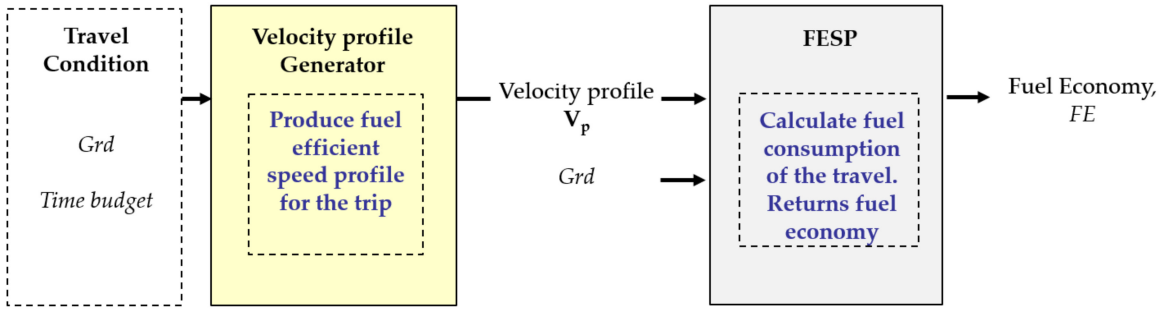


Figure 1. Input/output diagram of the proposed method. The velocity profile generator takes road grade information and time budget as input and returns the velocity profile as output. The FESP calculates fuel economy at the end of the trip. Our goal is to generate velocity profiles that maximize FE.

2.1. Fuel Economy Simulation Program

The Transmission Research Lab of the Hyundai Motor company developed the FESP for this research, shown in Figure 2. This program calculates fuel economy with vehicle data such as Engine BSFC (brake specific fuel consumption), transmission efficiency, vehicle road load, road grade, and control characteristics, according to driving conditions.

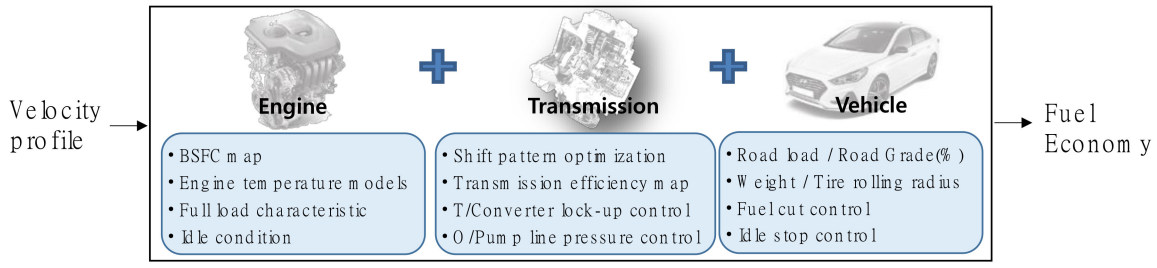


Figure 2. Fuel economy simulation program developed by the Hyundai motor company.

The calculation time of this program is 55 ms in the FTP-75 mode. As shown in Figure 3, this program's fuel consumption rate is almost equivalent in most of the situations to that of a commercial program.

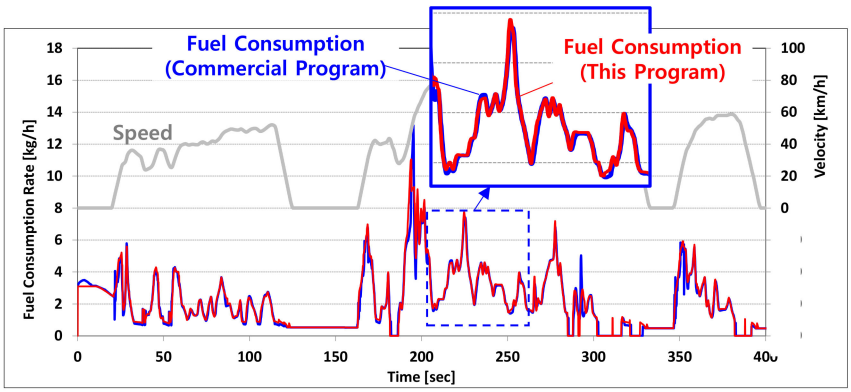


Figure 3. Comparison of the fuel consumption rate calculated by our program and another commercial program.

In addition, the shift pattern is optimized by FESP. That is, a dynamic shift pattern is used to enable a smooth shift step strategy, according to real-time optimal point driving and a long-term driving strategy. As a result, the number of shifts is reduced, and busy shifts are prevented, improving fuel economy and drivability.

We treat this simulation program as a black box not to use backpropagation through fuel consumption directly. This choice makes further research free from a particular choice fuel consumption model.

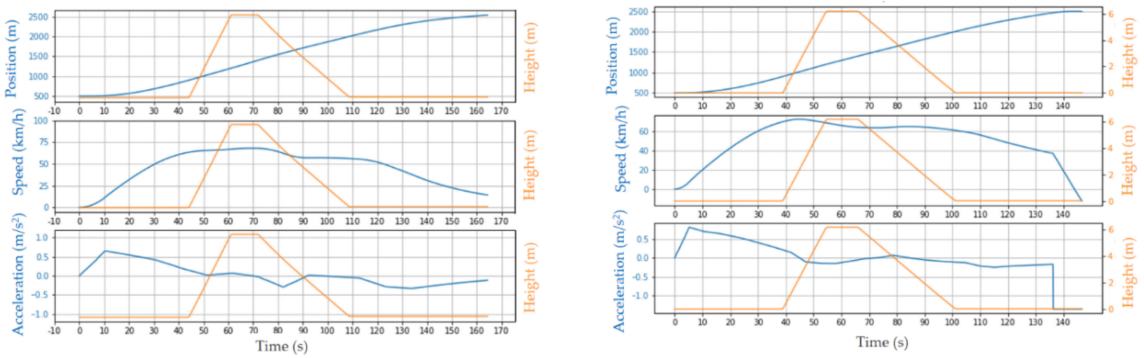
2.2. Problem Constraints

The vehicle starts at rest, fixing the first element in V_p to be 0. A simple rule-based program such as one shown in Figure 4 can be used to ensure the vehicle stops strictly at the end-point. However, it is allowed to finish the trip with speed since our work is more focused on optimizing fuel economy. Therefore, stopping the vehicle can be handled separately for this task.

We limit the maximum speed to 150 km per hour (km/h) and the minimum speed to 0 km/h. i.e.,

$$0 \leq v_t \leq 150 \tag{1}$$

To ensure that the diverse V_p is explored in finding a fuel-efficient strategy, we gave a sufficiently long *TB* and 100 s per 1-km distance. For example, a trip of 2 km length should take no more than 200 s to finish. It is okay to generate V_p that travels 2 km in less than 200 s, but it should not take any longer than 200 s in this case.



(**a**) (**b**)

Figure 4. (**a**) The velocity at the end-point is not constrained. (**b**) The velocity is strictly 0 at the end-point.

3. Deep Reinforcement Learning Approach

3.1. Reinforcement Learning

Reinforcement learning is a framework that is used to describe agents in a given environment by maximizing its cumulative reward. The state, S_t, denote all the information needed to characterize the environment at time t. The agent will choose an action A_t according to the state S_t. This action will result in a new state $S_{t+\Delta T}$, where ΔT is a fixed time length that one action takes. Determining which action to take is governed by the policy π. The policy, $\pi(A_t, S_t)$ denotes the probability that the agent will take action A_t at a given state S_t. Whenever the agent visits a state S_t, the agent is given a reward r_t. The episode is the sequence of state, action, and reward sets from the start to the end state. The cumulative reward R is the sum of all r_t in the episode.

$$R = \Sigma_t r_t \tag{2}$$

In our study, we want a DNN, which we call the velocity block generator, to generate a speed profile for a trip. The current state, S_t, contains a vehicle position, *Grd* for the road ahead, velocity profile up until current time $V_p(t)$, and the remaining time allowed for the trip $TB - t$. The velocity block generator determines the policy in a deterministic way, i.e., $\pi(A_t, S_t)$ is 1 for the output action and 0 for other actions. The reward is given only at the terminal state, making R straight forward *FE* for the trip time is no more than *TB*. In case it takes more time to finish the trip, we gave a small penalty by how much distance was left to the destination.

$$\text{penalty} = \text{distance left (km)} \tag{3}$$

$$R = FE - \text{penalty} \tag{4}$$

The terminologies of reinforcement learning in our problem are organized in Table 2.

Table 2. Reinforcement learning terminologies and explanations.

Expression	Definition	Relation to This Problem
S_t	State at time t	Road grades, velocity profile, time left
A_t	Action taken at S_t	neural network output,
r_t	Reward for visiting S_t	0 or *FE*—penalty
R	Cumulative reward	*FE*—penalty

In this reinforcement learning framework, our velocity block generator takes S_t and returns A_t that generate acceleration for time t and $t + \Delta T$, generating velocities for this period. These velocities, called velocity block $\mathbf{V_p}(t, t + \Delta T)$, make up the whole $\mathbf{V_p}$ once the terminal state is reached. In Figure 5,

the velocity profile generator calls the velocity block generator multiple times to generate $\mathbf{V_p}$, where the velocity block generator is our DNN model. Figure 6 shows this process in more detail.

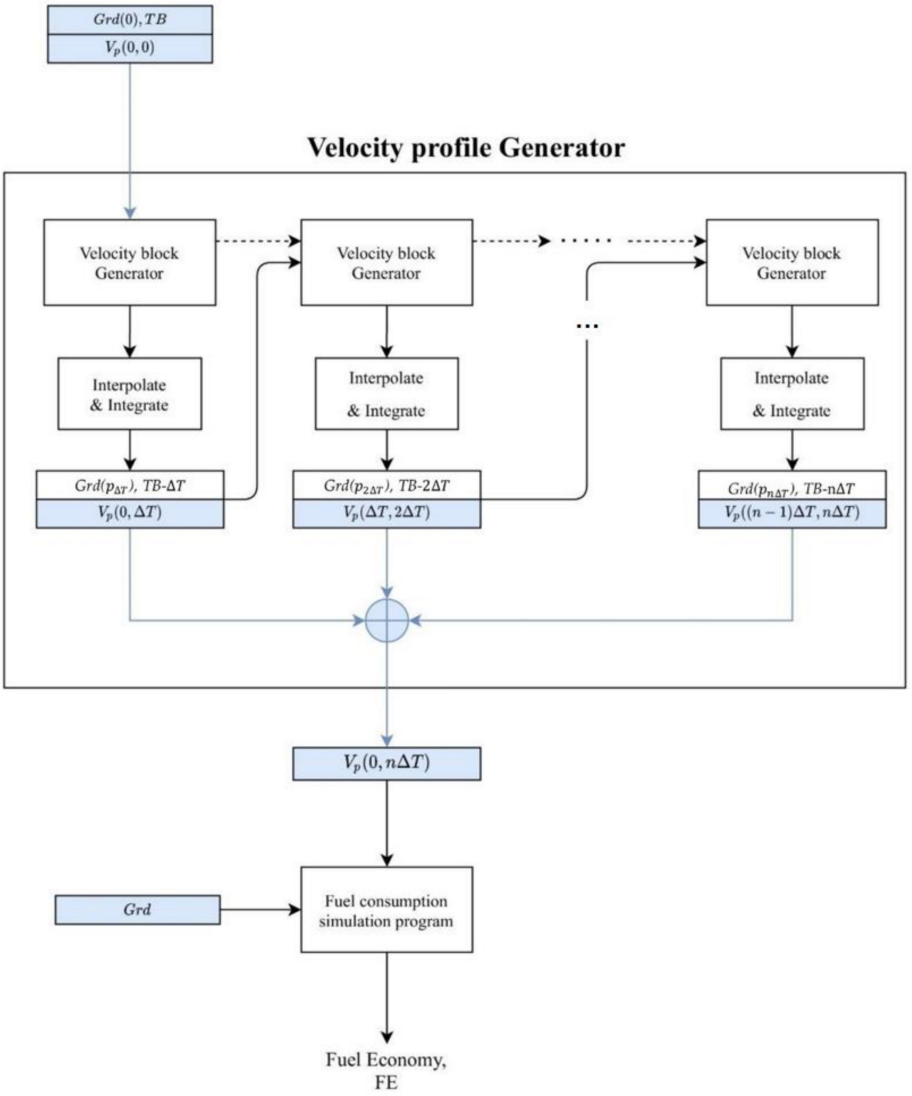


Figure 5. The velocity profile generator calls the velocity block generator multiple times until the trip is finished. The velocity block generator is made with DNN, and the dotted arrows represent hidden states of LSTM layers inside this neural network. It takes the state as input and outputs M values. These values are interpolated to be accelerations, which is integrated to be a velocity block.

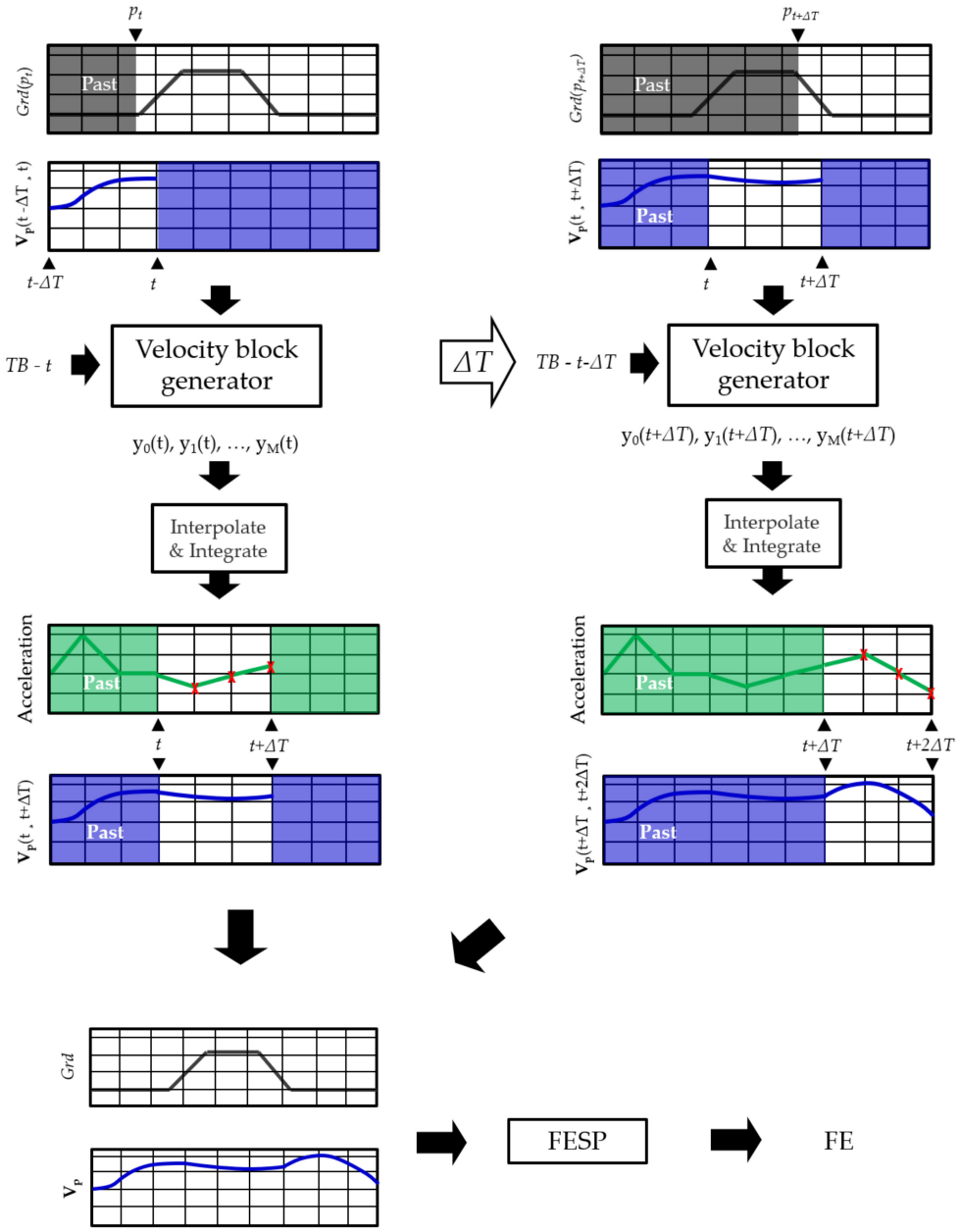


Figure 6. The DNN, which is the velocity block generator, takes the state by direct inputs and by its hidden state in long short-term memory (LSTM) layers. Its output will generate accelerations and velocities, $\mathbf{V_p}(t, t + \Delta T)$. These velocity blocks make the whole $\mathbf{V_p}$ for the trip. The fuel consumption simulation program takes this $\mathbf{V_p}$ along with *Grd* to calculate the fuel economy.

3.2. Deep Neural Network Architecture

The DNN model used, shown in Figure 7, is structured appropriately to handle all the data that is about to be given. These input data includes the present condition in this drive, past velocity profiles, and grade information for the road ahead. Only *Grd* of the road ahead and $\mathbf{V_p}(t - \Delta T, t)$ are passed through long short-term memory (LSTM) layers to reduce dimensionality. The rest of the past velocity profiles are given indirectly by the previous state of the LSTM layers. The LSTM layers that process *Grd* and $\mathbf{V_p}$ are called the Road model and the Car model, respectively. Through these models, high-level features of the road and the vehicle state are made. These feature vectors are then concatenated with conditions such as how much time is left on the clock. They are all processed together through fully

connected layers, known as dense layers, to return $\mathbf{Y_t}$. The action $\mathbf{Y}_t$ generates the velocity profile block $\mathbf{V_p}(t, t + \Delta T)$. We have freedom in representing this period with a finite number of parameters. However, there are a few things to consider when choosing the right parameterization method.

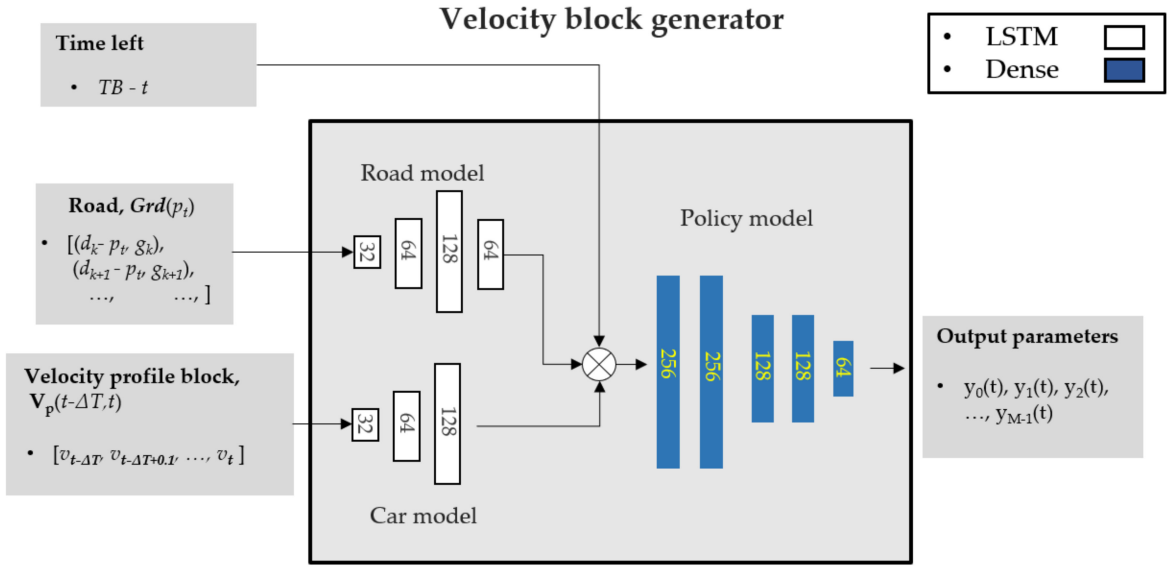


Figure 7. Deep neural network (DNN) model structure. Sequential data is passed through LSTM layers by reducing its dimensionality. LSTM layers are suitable for processing sequential data and extracting meaningful features. These feature tensors are concatenated with non-sequential inputs and processed all together through the dense layers to return the output tensors.

First, even random outputs should be interpreted as a plausible vehicle movement. The DNN, in its first stage of the training, will make random outputs. If this is not a plausible movement, this episode is terminated, and DNN will not learn anything about fuel-saving strategies. Second, the cumulative reward R and its gradient should be as smooth as possible for DNN outputs. Otherwise, the numeric differentiation of R is not suitable to train DNN. Third, the number of parameters should be as small as possible to represent diverse vehicle movement to shorten the time it needs to calculate R's numerical differentiation for the outputs. Figure 8 is shown to clearify these three points.

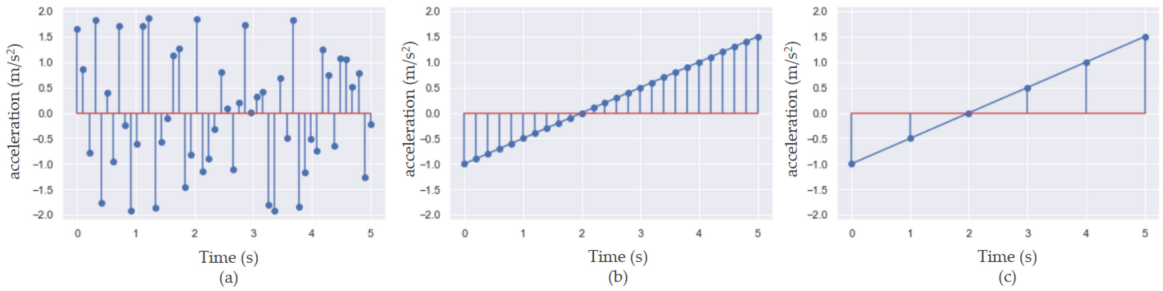


Figure 8. (**a**) Not a plausible vehicle acceleration and result in an error during the learning process. (**b**) A plausible vehicle acceleration. However, 26 numbers are used to represent this simple line. (**c**) This represents the same vehicle acceleration. It uses only six numbers and likely a better parameterization of this acceleration profile.

We scheme to make acceleration values at a few fixed time points, fill in the intermediate values with a straight line, and then integrate it to make $\mathbf{V_p}(t, t + \Delta T)$. Concretely, the DNN output $\mathbf{Y_t} = [y_0, y_1,$

$y_2, \ldots, y_{M-1}]$, the number y_0 determines the acceleration magnitude at $t + \Delta T$, and the other numbers, $y_1, y_2, \ldots, y_{M-1}$ determine how the transition from a_t to $a_{t+\Delta T}$ takes place.

$$-2 \le y_0 \le 2, -0.5 < y_1, y_2, \ldots, y_{M-1} < 0.5 \quad (5)$$

The 1/M, 2/M, ... , (M − 1)/M of the change in acceleration happens at the $(1 + y_1)/M$, $(2 + y_2)/M$, ... , $(M - 1 + y_{M-1})/M$ of time difference. These M points, including the end that divides time and the acceleration, are called key points. All the intermediate acceleration values between key points are interpolated linearly as shown in Figure 9.

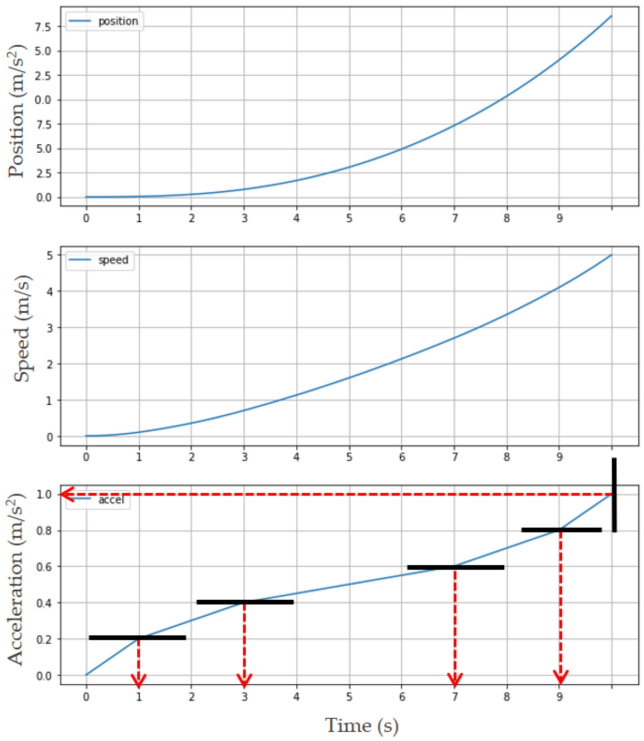


Figure 9. Parameterization example of $\Delta T = 10$ and $M = 5$. In this case, the value of y_0 is 1.0 m/s^2, and the values of y_1, y_2 are −0.49 m/s^2, and y_3, y_4 are 0.49 m/s^2. With these values, we determined the key points, shown as red arrows in the diagram.

3.3. Numerical Differentiation

To update θ, the DNN's learnable parameters, we use the gradient ascent algorithm directly on R with learning rate λ.

$$\theta^{new} = \theta^{old} + \lambda \cdot \partial R(\theta^{old})/\partial\theta \quad (6)$$

In this case, R represents as a function of θ as it determines the action, which determines R.

We have to differentiate the FESP program to calculate the last term in (6), the gradient of R for θ. We cannot use automatic differentiation in this step since FESP is a black box program. Instead, we use numerical differentiation.

The derivative of a real function f at x is defined as a limit below.

$$f'(x) = \lim_{h \to 0} [f(x + h) - f(x)]/h \quad (7)$$

The term in the limit, called Newton's difference quotient, approximate $f\,'$ (x) with small nonzero number h. The error, for this approximation, is of order O(h). Similarly, we use a symmetric difference quotient to approximate real derivatives.

$$f\,'(\mathrm{x}) \approx [f\,(\mathrm{x}+\mathrm{h}) - f\,(\mathrm{x}-\mathrm{h})]/2\,\mathrm{h} \tag{8}$$

In this study, approximation error is of order $O(\mathrm{h}^2)$ since the first-order errors cancel each other. Thus, we will use Equation (8) to calculate R's gradient. However, another level of indirection is used to calculate this gradient. This is because the deep learning network has millions of parameters.

We will use the chain rule to divide the term $\partial \mathrm{R}/\partial\theta$ into a multiplication of two gradients.

$$\partial \mathrm{R}/\partial\theta = \partial \mathrm{R}/\partial \mathbf{Y} \cdot \partial \mathbf{Y}/\partial\theta \tag{9}$$

Here, Y is the tensor of all the outputs $\mathrm{Y_t}$. The gradient $\partial \mathrm{R}/\partial \mathrm{Y}$ is relatively easy to apply Equation (8), as all the outputs of DNN on a trip are of the order of hundreds. Furthermore, Y can be differentiated automatically using standard deep learning libraries such as TensorFlow and PyTorch.

Algorithms 1–3 are pseudo codes that show how numeric differentiation is implemented in the training process. The Velocity_Block_Generator (VBG) algorithm takes a velocity block of a previous timestep, time left, and let the DNN generate $\mathrm{V_p}$ for this trip.

Algorithm 1. Generate_Velocity_Block

Inputs: Grade profile ahead *G*
Velocity profile for the previous block **Vb**
Allowed time left *ATL*
Outputs: Action for this velocity block $\mathbf{Y_t}$
Let Road_model be the LSTM layers, which take grade *G* data
Let Car_model be the LSTM layers, which take velocity profile block **Vb**
Let Policy_model be the LSTM layers, which take Road_model's output, Car_model's output and *TL*
1: road_vector = Road_model(*G*)
2: car_vector = Car_model(**Vb**)
3: $\mathbf{Y_t}$ = Policy_Model(road_vector, car_vector, *ATL*)
4: **RETURN** $\mathbf{Y_t}$

Algorithm 2. Generate_Velocity_Profile

Inputs: Grade profile for the entire trip *Grd*
Time budget *TB*
Outputs: Actions for the entire trip **Y**
1: initialize time *t* as 0, velocity block index *n* as 0
2: $\mathbf{Y_0}$ = Generate_Velocity_Block(*Grd(0)*, **Vp**(0, *0*), *TB)*
3: **Vp**(0, ΔT) = Interpolate_and_Integrate($\mathbf{Y_0}$)
4: **Y** = **Y** concat $\mathbf{Y_0}$,t = t + ΔT, n = n + 1
5: **WHILE** t >= *TB* OR Position_At(t) >= total_distance **DO**
6: $\mathbf{Y_t}$ = Generate_Velocity_Block(*Grd(*Position_At(*t*)), **Vp**(*t* − ΔT, *t*), *TB-t*)
7: **Vp**(t, *t* + ΔT) = Interpolate_and_Integrate($\mathbf{Y_t}$)
8: **Y** = **Y** concat $\mathbf{Y_t}$, *t* = *t* + ΔT, *n* = *n* + *1*
9: **RETURN Y**

Algorithm 3. Calculate_Reward

Inputs: All the actions of trip $\boldsymbol{Y}$
Grade profile *Grd*
Time budget *TB*
Outputs: Reward for the trip
Let Fuel Economy Simulation Program bee FESP
1: $\mathbf{Vp}(0, n\Delta T)$ = Interpolate_and_Integrate($\mathbf{Y}$)
2: fe= FESP($\mathbf{Vp}(0, n\Delta T)$, *Grd*)
3: **IF** travel_time >= *TB* **DO**
4: penalty = distance_left
5: Reward = fe—penalty
6: **RETURN** Reward

The Backpropagete algorithm, Algorithm 4, updates DNN's learnable parameters for a given amount of times using Algorithms 1–3 and Equations (6), (8), and (9).

Algorithm 4. Bakcpropagate

Inputs: All the actions of trip: $\mathbf{Y}$
Learnable weights of DNN: θ
Grade profile *Grd*
Time budget *TB*
Outputs: Updated DNN weights $\theta^{\mathbf{new}}$
Let number of ΔT blocks be n
1: **FOR** $\mathbf{Y}_t$ in $[\mathbf{Y}_0, \mathbf{Y}_{\Delta T}, \mathbf{Y}_{2\Delta T}, \ldots, \mathbf{Y}_{n\Delta T}]$ **DO**
2: **FOR** y_k in $[y_0, y_1, \ldots, y_{M-1}]$ **DO**
3: $\mathbf{Y^+}$ = $\mathbf{Y}$ except y_k is increased by h
4: $\mathbf{Y^-}$ = $\mathbf{Y}$ except y_k is decreased by h
5: Reward^+ = Calculate_Reward($\mathbf{Y^+}$, *Grd, TB*)
6: Reward^- = Calculate_Reward($\mathbf{Y^-}$, *Grd, TB*
7: $\partial R/\partial \mathbf{Y}$ [*n*, k] = [Reward^+—Reward^-]/2 h (refer to **Equation** (8))
8: $\partial \mathbf{Y}/\partial \theta$ = **autograd**($\mathbf{Y}$, θ)
9: $\theta^{\mathbf{new}} = \theta + \lambda \cdot \partial R/\partial \mathbf{Y} \cdot \partial \mathbf{Y}/\partial \theta$ (refer to **Equations** (6) and (9))
8: **RETURN** $\theta^{\mathbf{new}}$

Algorithm 5. Policy_Search

Inputs: Number of iterations to train *train_iters*
Let θ be all the weights of the Velocity Profile Generator DNN Model
1: **FOR** iteration **IN** [1, 2, . . . , *train_iters*] **DO**
2: $\mathbf{Y}$ = Generate_Velocity_Profile(*Grd, TB*)
3: $\theta^{\mathbf{new}}$ = Backpropagate($\mathbf{Y}$, θ, *Grd, TB*)
4: Apply $\theta^{\mathbf{new}}$ to the Velocity profile Generator

4. Learning Algorithm Evaluation

4.1. Time Interval Experiment

The time interval, ΔT, has a profound impact on the training step, and the trained results with itself. With a bigger ΔT, less action is needed to finish the trip, resulting in faster training iterations. However, a small number of actions in a drive reduces the velocity block generator's controllability of the vehicle. It might settle in sub-optimal strategies that are less reflexive to the environment. Thus, we can balance training time and training quality by adjusting ΔT. We have experimented with ΔT = 5, 10, and 20 s on an easy 2 km long road with one hill and compared their results. To get reliable metrics

for training results, we repeated numerous training for all the three ΔT. This repetition is necessary since the initial parameters of velocity block generator significantly impact the training outcomes.

We repeat 50 policy search rounds for each ΔT, where a policy search round, as shown in Algorithm 5, is defined as a single velocity profile generation trial with random initialization of a velocity block generator composed of playing and learning from 20 episodes. We repeat 50 policy search rounds for a fixed ΔT. Comparing each policy search round will clarify the impact of ΔT in the time and quality of the training. All three results will also confirm our method's ability to train the velocity block generator on a black box fuel consumption program on ICE. The time sampling Ts and distance sampling Ds is set to 0.1 (s) and 5 m, respectively.

4.2. Generalization Experiment

The advantage of the deep learning method is that it can be generalized to unseen roads. We have fixed ΔT to 5 s and made ten 5 km roads. The velocity block generator had to learn from nine of these roads (road no. 1~9) and was tested on the unseen one road (road no. 10).

In this case, during a policy search round, the velocity block generator make trips in all ten roads, including the test road, to see how it does during the training. The velocity block generator's learnable parameters get updated from all nine trips at once. After 20 updates, by evaluating how it did on the test road for 20 iterations, we will see if it had be generalized. The time sampling Ts and distance sampling Ds is set to 0.1 s and 5 m, respectively.

4.3. Evaluation Metrics

We measured Average FE, Top 5 FE, Best FE, and average search time for the 50 policy search rounds. For each policy search, the highest FE value is selected over 20 episodes. Then, on average, the top 5 average, and maximum is calculated over 50 policy search rounds, respectively. The FE increase is the increase rate of best FE compared to FE made from a constant speed cruise control velocity profile. The constant velocity cruise control velocity profile is made by accelerating with α (m/s^2) until the top speed ν km/h is reached. Then the velocity is kept constant until the trip ends. The values of α and ν are determined by a simple grid search, which is explained in Appendix A. If the FE of the last five trips on a policy search round is higher than the FE of the first five trips on average, we mark this policy search round as a success. The success rate is the percentage of successful searches in 50 policy search rounds. This rate will show us how stable the training is or is not. We list these metrics in Table 3 to clarify their definitions.

Table 3. Policy search round evaluation metrics.

Metrics	Definition
Average FE	Average FE of all policy search rounds
Top 5 FE	Top 5 FE of all policy search rounds
Best FE	Best FE of all policy search rounds
Success rate (%)	Rate of policy search rounds where the last five episodes had higher FE than the first five episodes
Avg search time	Average time to finish a policy search round
FE increase (%)	Best FE's increase rate over a simple constant velocity cruising

4.4. Implementation Environment

The training algorithm codes were mainly written with python version 3.6 and TensorFlow version 2.3 on Ubuntu 18.04 OS. The FESP, which calculates an episode's fuel economy, was written in Fortran programming language. This is accessible to the main python program by using NumPy's F2py module, which enables Fortran subroutines to be executed in python programs. For the hardware, the Titan RTX graphics card and intel i9-9980XE CPU with 128 GB ram were utilized. Although the Titan RTX graphics card was used in the training, TensorFlow's GPU usage was limited to 4 GB.

5. Training Results

5.1. Time Interval Experimental Results

The training results with varying ΔT had a very high variance across the set's 50 policy search rounds. Nevertheless, in Figure 10, averaging the fuel economy of the trip in the set shows a tendency to rise as the train progresses. The overall performance of the set is listed in Table 4, also the best case and detailed analysis of the best case is shown in Figures 11 and 12 respectively. Our result shows that the training was stable during training with a success rate of 100% for all ΔT. The set with ΔT = 5 s was better on average, but the very best FE was found in the set with ΔT = 20 *s*. At the beginning of training, larger ΔT seems to be better on average, but this is reversed at the end stage of training, where smaller ΔT seems to do better on average. The FE increase was at least 5.1% when compared to a constant speed cruise control. The average rep times show an exponential increase as ΔT gets smaller.

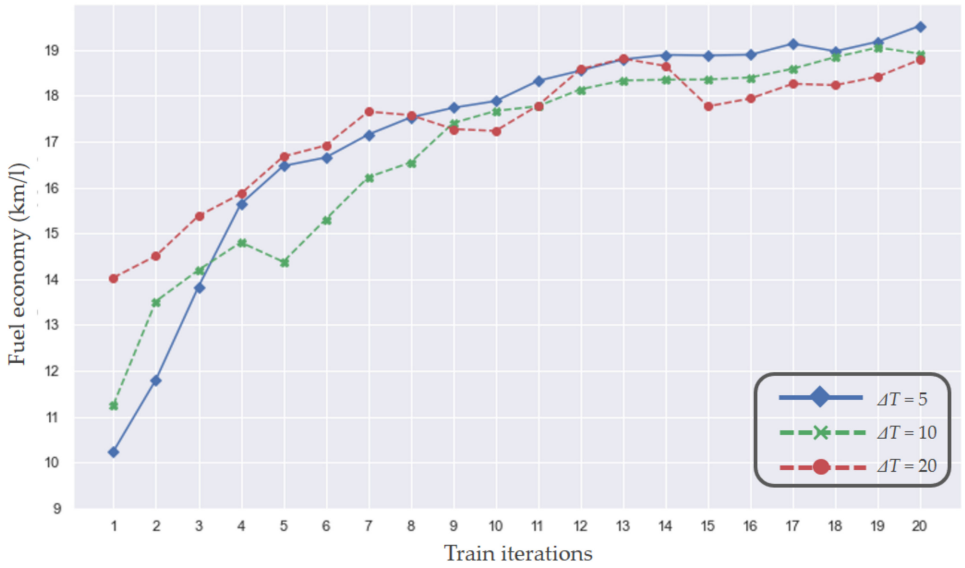


Figure 10. Fuel economy changes during different training iterations. The points in the graph are the average of 50 different policy search rounds in the set.

Table 4. Policy search round evaluation.

Test Set	Success Rate (%)	Average FE (km/L)	Top 5 FE (km/L)	Best FE (km/L)	FE Increase (%)	Avg Search Time (s)
ΔT = 5 s	100	20.34	20.91	21.06	5.1	332
ΔT = 10 s	100	19.93	20.89	21.00	4.8	186
ΔT = 20 s	100	19.87	21.02	21.64	8.0	108

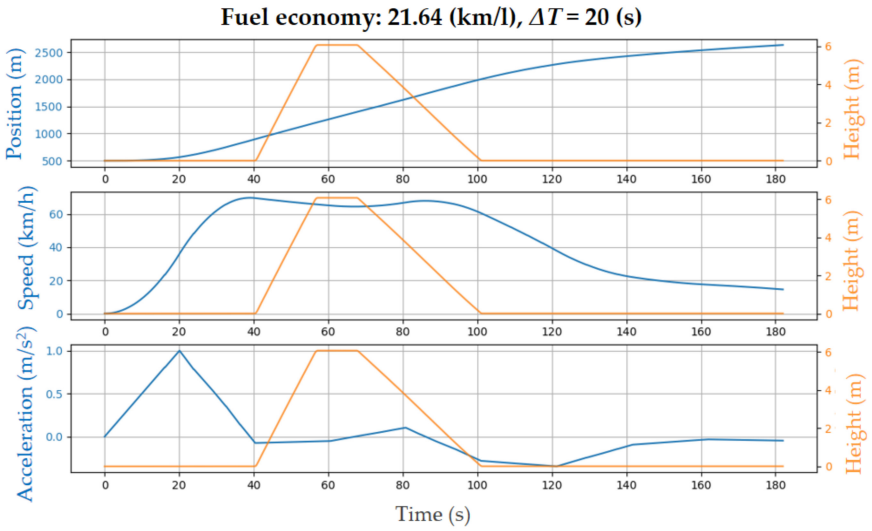


Figure 11. The best trip in the three sets. The velocity generator found the strategy to accelerate up to 70 km/h until the uphill starts and slowly decelerates except for slight acceleration that is given during the downhill.

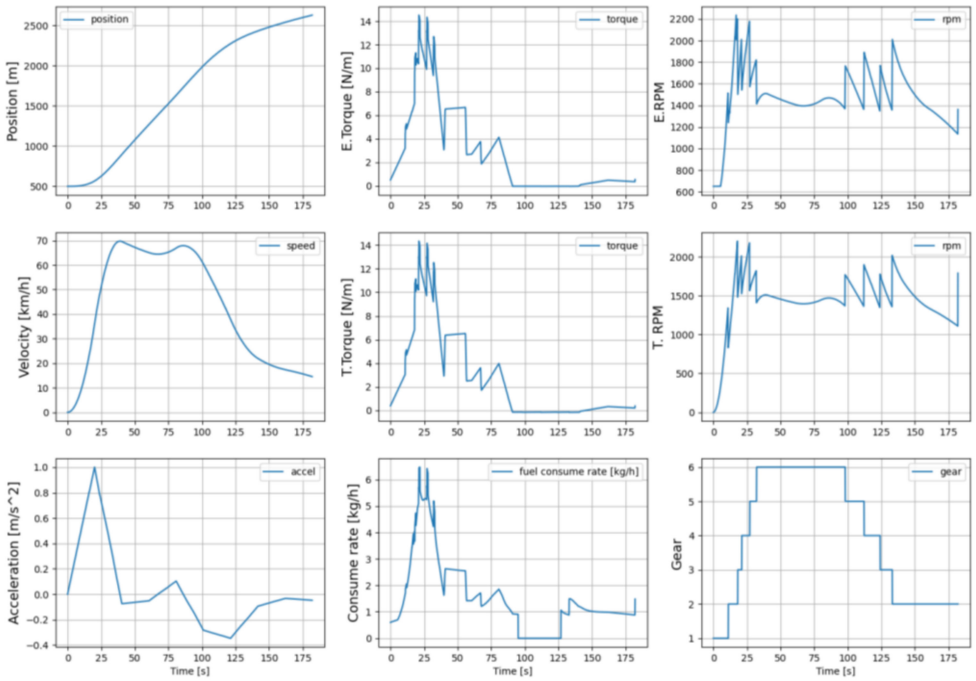


Figure 12. Detailed analysis on the best trip in the three sets.

5.2. Generalization Experiment Results

The DNN learned to optimize FE in all 10 roads. The $\mathbf{V_p}$ generated in these roads are shown in Figure 13. It seems that it has learned to keep the speed to accelerate to 70 km/L until it reaches uphill or downhill and decelerate or accelerate accordingly. The FE values resulting from constant velocity cruising and that of our velocity profile generator are compared in Table 5.

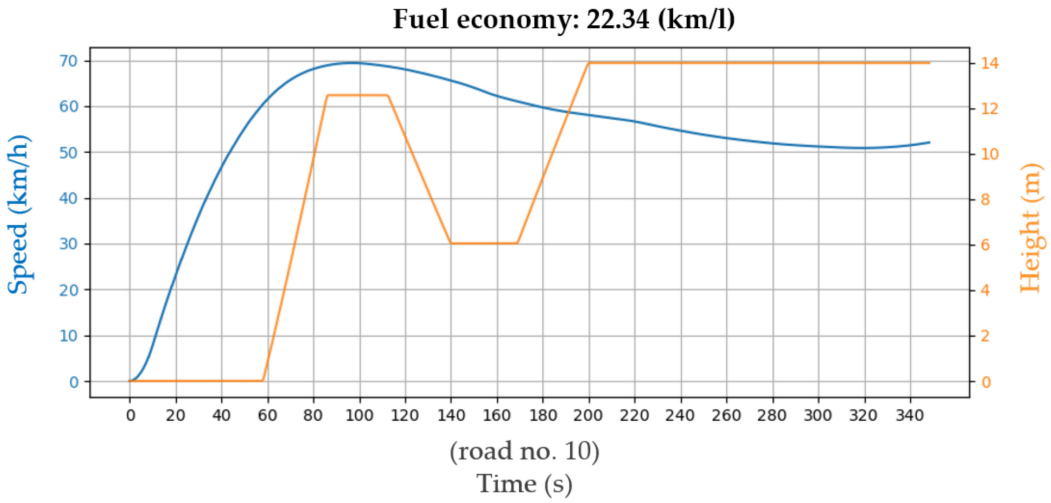


Figure 13. Generalization results in road no. 10. The trip made at the final iteration of total 20 iterations of training. Roads no 1–9 were used in the training whereas road no. 10 was not. All the trips including road no. 10 seems to have a fuel saving velocity profile for the road. The results for roads no 1–9 is shown in Appendix B.

Table 5. Comparison between constant velocity cruise and our algorithm.

Road	FE of Constant Velocity Cruise (km/L)	FE of Velocity Profile Generator (km/L)	FE Increase (%)
No. 1	25.01	26.24	4.9
No. 2	22.30	22.77	2.1
No. 3	24.99	25.60	2.4
No. 4	22.82	23.38	2.5
No. 5	24.41	24.76	1.4
No. 6	23.14	23.16	0.1
No. 7	24.22	25.72	6.2
No. 8	21.65	21.76	0.5
No. 9	24.16	24.33	0.7
No. 10	22.32	22.34	0.1

We also verified DNN's generalization through another experiment. The DNN uncovered some strategies for the training roads (road no. 1~9) and made a speed profile for the unseen road (road no. 10) with reasonable FE. Unfortunately, we encountered some difficulties during the training loops. DNN sometimes falls into the local maximum FE strategy.

6. Conclusions and Future Work

We proposed a novel method for improving fuel economy given road grade information for the trip and the allowed time budget based on the deep reinforcement learning. Our algorithm was trained using various examples of road grades and time budgets, and the output velocity profile was evaluated using Hyundai Motor Group's Fuel Economy Simulation Program. We evaluated our algorithm and demonstrated that our algorithm increased, at most, in 8% fuel economy than normal constant velocity cruising. We also learned that, contrary to our prediction, smaller ΔT is not always better. As mentioned in the result section, Figure 10 shows that larger ΔT seems to be better in the earlier stage of training. This seems to be because larger ΔT inherently generates smoother velocity profiles.

As shown in Table 4, for ΔT = 10 s, it takes about 186 s to generate a velocity profile and train the velocity profile generator for 20 times. Thus, only 9.3 s is needed to generate and learn on average, and it takes only about 3 s to plan for the whole trip of 2 km and only 0.2 s for planning for the next

10 s. This is fast enough to adjust for unexpected events during the trip, such as a change in route during travel. Thus, our algorithm is fast enough so it can be integrated with in-vehicle systems.

Our future work includes strictly imposing the time budget and end-point velocity and coping with dynamic environments, such as the vehicles ahead and varying traffic conditions.

Author Contributions: Conceptualization, S.L. and J.S.P. Methodology, S.L. and H.K. Software, H.K., H.P., J.S.P., and J.Y.H. Validation, S.L. and J.S.P. Formal analysis, H.K. and J.S.P. Investigation, H.K., S.L., J.S.P., and J.Y.H. Data curation, H.K., H.P., and J.Y.H. Writing—original draft preparation, H.K. and J.Y.H. Writing—review and editing, S.L. Visualization, H.K. and H.P. Supervision, S.L. Project administration, S.L. and J.S.P. Funding acquisition, S.L. and J.S.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Hyundai Motor Group and also supported by the Technology Innovation Program (No. 10083646) and the Competency Development Program for industry specialists (No. N0002428) funded by the Ministry of Trade, Industry, and Energy (Korea).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The constant velocity cruising is used as a comparison to our method. This constant velocity cruise is determined by two parameters α and ν. They are accelerated from the start (m/s^2) and cruising velocity (km/h). The value α is chosen from [0.1, 0.2, ... , 0.9, 1.0] and ν is chosen from [50, 55, 60, ... , 90, 95]. By checking all the combinations of α and ν, the one with the best FE is compared with our method.

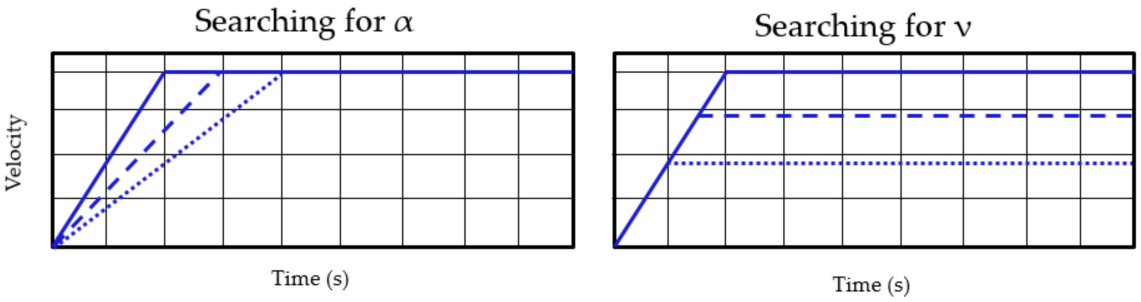


Figure A1. Grid search is done by searching every combination of α and ν. Then, we chose the combination with the best FE. This is then compared with FE made from our method.

Appendix B

Here, we present the generalization result for the other nine roads.

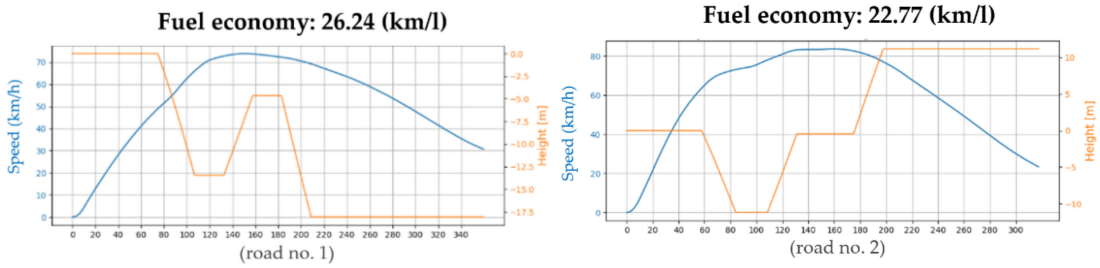


Figure A2. *Cont.*

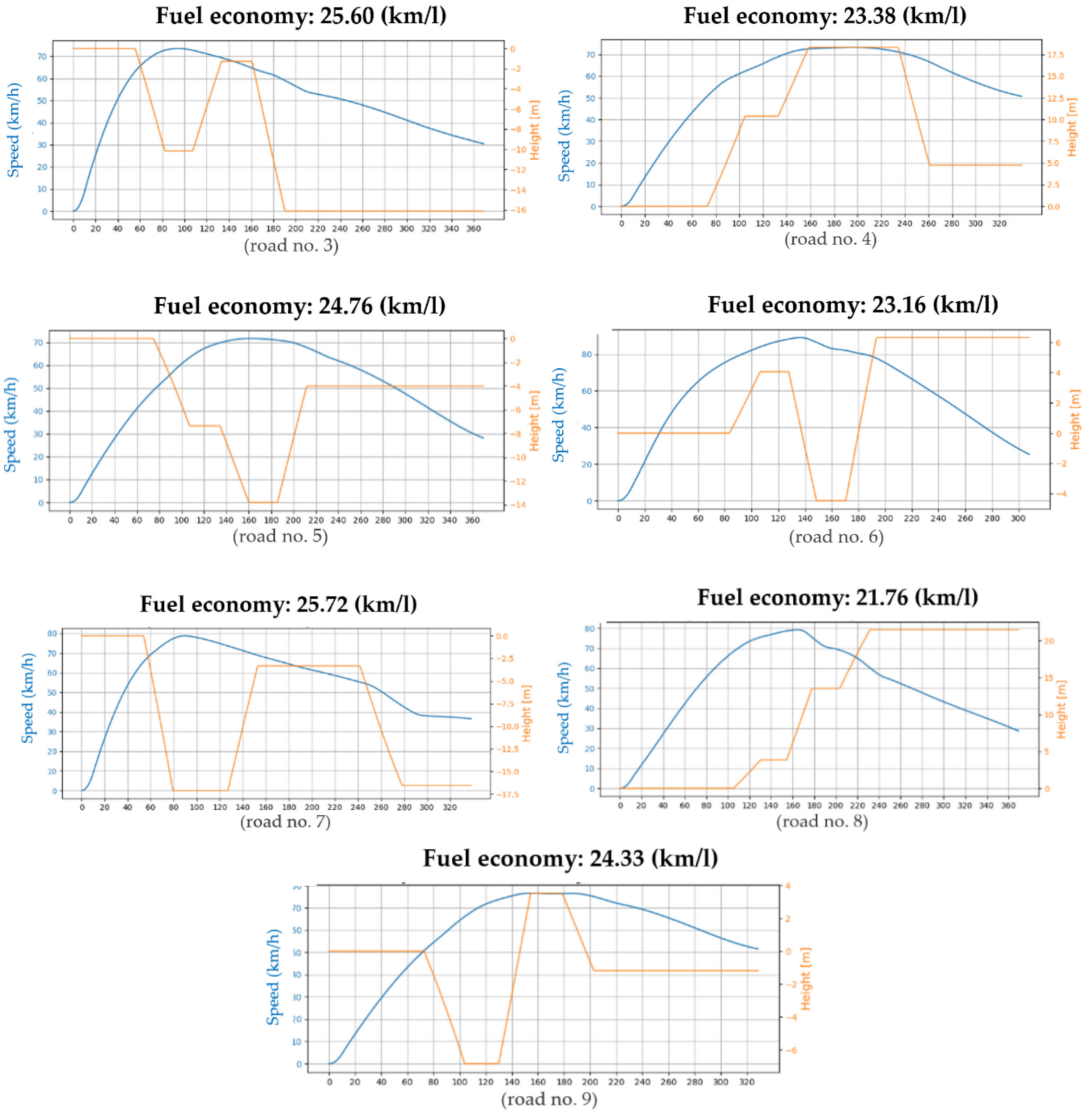


Figure A2. The final result for the nine roads, road no. 1–9, which are used in the training process.

References

1. Average Fuel Efficienct of US Light Duty Vehicles. Available online: https://www.bts.gov/content/average-fuel-efficiency-us-light-duty-vehicles (accessed on 24 September 2020).
2. 4 Charts Explain Greenhouse Gas Emissions by Countries and Sectors. Available online: https://www.wri.org/blog/2020/02/greenhouse-gas-emissions-by-country-sector (accessed on 24 September 2020).
3. Guanetti, J.; Kim, Y.; Borrelli, F. Control of connected and automated vehicles: State of the art and future challenges. *Annu. Rev. Control.* **2018**, *45*, 18–40. [CrossRef]
4. Doan, V.-D.; Fujimoto, H.; Koseki, T.; Yasuda, T.; Kishi, H.; Fujita, T. Simultaneous Optimization of Speed Profile and Allocation of Wireless Power Transfer System for Autonomous Driving Electric Vehicles. *IEEJ J. Ind. Appl.* **2018**, *7*, 189–201. [CrossRef]
5. Bae, S.; Kim, Y.; Guanetti, J.; Borrelli, F.; Moura, S. Design and implementation of ecological adaptive cruise control for autonomous driving with communication to traffic lights. In Proceedings of the 2019 American Control Conference (ACC), Philadelphia, PA, USA, 10–12 July 2019; pp. 4628–4634.
6. Prakash, N.; Kim, Y.; Stefanopoulou, A.G. A Comparative Study of Different Objectives Functions for the Minimal Fuel Drive Cycle Optimization in Autonomous Vehicles. *J. Dyn. Syst. Meas. Control* **2019**, *141*, 071011. [CrossRef]

7. De Filippis, G.; Lenzo, B.; Sorniotti, A.; Sannen, K.; De Smet, J.; Gruber, P. On the Energy Efficiency of Electric Vehicles with Multiple Motors. In Proceedings of the 2016 IEEE Vehicle Power and Propulsion Conference (VPPC), Hangzhou, China, 17–20 October 2016; pp. 1–6.
8. Sforza, A.; Lenzo, B.; Timpone, F. A state-of-the-art review on torque distribution strategies aimed at enhancing energy efficiency for fully electric vehicles with independently actuated drivetrains. *Int. J. Mech. Control* **2019**, *20*, 3–15.
9. Martin, S.; Martin, L.; Andreas, A.; Julian, H. Efficient energy-optimal routing for electric vehicles. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, CA, USA, 7–11 August 2011; pp. 1402–1407.
10. Kamal, A.S.; Mukai, M.; Murata, J.; Kawabe, T. Model Predictive Control of Vehicles on Urban Roads for Improved Fuel Economy. *IEEE Trans. Control. Syst. Technol.* **2012**, *21*, 831–841. [CrossRef]
11. Gipps, P.G. A behavioral car-following model for computer simulation. *Transp. Res. B-Meth.* **1981**, *15*, 105–111. [CrossRef]
12. Kirches, C.; Bock, H.G.; Schlöder, J.P.; Sager, S. Mixed-integer NMPC for predictive cruise control of heavy-duty trucks. In Proceedings of the European Control Conference, Zürich, Switzerland, 17–19 July 2013.
13. Santin, O.; Pekar, J.; Beran, J.; D'Amato, A.; Ozatay, E.; Michelini, J.; Szwabowski, S.; Filev, D. Cruise Controller with Fuel Optimization Based on Adaptive Nonlinear Predictive Control. *SAE Int. J. Passeng. Cars Electron. Electr. Syst.* **2016**, *9*, 262–274. [CrossRef]
14. Santin, O.; Beran, J.; Pekar, J.; Michelini, J.; Jing, J.; Szwabowski, S.; Filev, D. Adaptive Nonlinear Model Predictive Cruise Controller: Trailer Tow Use Case. *SAE Tech. Paper Ser.* **2017**, *1*, 2017-01-0090.
15. Santin, O.; Beran, J.; Mikulas, O.; Pekař, J.; Michelini, J.; Szwabowski, S.; Mohan, S.; Filev, D.; Jing, J.; Özgüner, Ü. On the Robustness of Adaptive Nonlinear Model Predictive Cruise Control. *SAE Tech. Paper Ser.* **2018**, 2018–01–1360.
16. Kumar, S.S.P.; Tulsyan, A.; Gopaluni, R.B.; Loewen, P.D. A Deep Learning Architecture for Predictive Control. *IFAC-PapersOnLine* **2018**, *51*, 512–517. [CrossRef]
17. Ozatay, E.; Onori, S.; Wollaeger, J.; Ozguner, U.; Rizzoni, G.; Filev, D.P.; Michelini, J.; Di Cairano, S. Cloud-Based Velocity Profile Optimization for Everyday Driving: A Dynamic-Programming-Based Solution. *IEEE Trans. Intell. Transp. Syst.* **2014**, *15*, 2491–2505. [CrossRef]
18. Hellström, E.; Ivarsson, M.; Åslund, J.; Nielsen, L.R. Look-ahead control for heavy trucks to minimize trip time and fuel consumption. *Control. Eng. Pr.* **2009**, *17*, 245–254. [CrossRef]
19. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Driessche, G.V.D.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nat. Cell Biol.* **2016**, *529*, 484–489. [CrossRef] [PubMed]
20. Hartmann, G.; Shiller, Z.; Azaria, A. Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge. In Proceedings of the 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR, USA, 4–6 November 2019; pp. 186–193.
21. Liessner, R.; Schroer, C.; Dietermann, A.; Bäker, B. Deep Reinforcement Learning for Advanced Energy Management of Hybrid Electric Vehicles. In Proceedings of the 10th International Conference on Agents and Artificial Intelligence, Funchal, Portugal, 16–18 January 2018.