Optimising PID Control with Residual Policy Reinforcement Learning

Andrew Hynes[1], Elena Sapozhnikova[2], Ivana Dusparic[1]

[1] Trinity College Dublin, Ireland {hynesa1,duspari}@tcd.ie
[2] ZF Friedrichshafen, Graf-von-Soden-Platz 1, 88046 Friedrichshafen, Germany
elena.sapozhnikova@zf.com

Abstract. Suspension control systems in cars are a vital component in modern vehicles tasked with enhancing ride comfort for passengers. However, these systems rely on system controllers to dictate the damping rate. Commonly PID controllers are used for this purpose, but these controllers have a number of drawbacks such as their linearity and inability to adapt. On the other hand, many of these weaknesses are the strengths of Reinforcement Learning (RL) techniques. In this research, a novel approach is taken to combine both PID controllers and RL through a technique known as Residual Policy Reinforcement Learning (RPRL). This approach is evaluated on a quarter car suspension system model in a variety of common scenarios and is shown to improve the suspension control of a car, enhance the performance of a PID-controller, and adapt to environmental changes. However, it is also shown that the algorithm's performance is highly reliant on the performance of the base policy, i.e., initially selected PID parameters.

Keywords: PID controller · RL · System Control · Suspension System Control

1 Introduction

Suspension control systems are an integral part of modern vehicles, with ride comfort and passenger satisfaction being directly linked to the performance of these systems. One popular suspension system used is the semi-active suspension system, which relies on a system controller to monitor and govern the damping rate of the damper. Whilst there is a wide range of controllers available for this purpose, one of the most commonly used controllers is the Proportional Integral Derivative (PID) Controller. Even though this controller has many benefits, including its simplicity and cost, it also has many drawbacks. Some of these include the linear nature of the controller, which makes it unsuitable for use in complex nonlinear systems, and the sensitivity of the controller to its parameters. These parameters are environment specific and need to be tediously tuned to maximise the controllers performance. Additionally, they can suffer dramatic drops in performances due to changes in environmental conditions such as temperature change.

In contrast, many of the difficulties associated with PID Controllers can be considered the strengths of reinforcement learning (RL) techniques, as they are self training and generally show high levels of adaptability. Additionally, when using Deep RL it is possible to overcome complex nonlinear systems. As a result, they are able to overcome a lot of the issues related to PID controllers.

However, RL itself also has its drawbacks with difficulties associated with divergence and stability being common in Deep RL algorithms. [1] showed that this can be overcome, at least in some cases, by combining RL with a strong base policy through Residual Policy Reinforcement Learning (RPRL). In this case the base policy refers to the traditional control technique used alongside the Reinforcement Learning algorithm. Therefore, in order to create a system that benefits from both the strengths of the PID controller and RL techniques our research combines both methods together using RPRL.

This is, to the best of our knowledge, the first time PID control and RL are combined through RPRL. Therefore, the main contributions of this paper are:

- Combining PID control and RL through the use of Residual Policy Reinforcement Learning. In particular, we enriching PID control with the Deep Deterministic Policy Gradient (DDPG) variant of RL.
- Evaluating RPRL on a simulation of quarter-car suspension system.

The rest of the paper is structured as follows. Section 2 provides an introduction to the two main components of this approach: PID control and RL. Section 3 outlines related work and alternative approaches to overcome some of these issues. Section 4 illustrates how suspension control was modelled as an RL problem, whilst Section 5 describes the evaluation method and presents the results. Finally, Section 6 concludes the paper.

2 Background

2.1 PID Controller

Proportional Integral Derivative (PID) Controllers are a form of closed loop controller which aim to reduce the error between the target and current output values of a system by altering a variable known as the manipulated variable. In order to do this, the PID controller relies on 3 components; the proportional component, the integral component and the derivative component. The proportional component receives the error, i.e. the difference between the current and target output value, and performs the operation shown in equation 1.

$$Proportional\,Output = K_p e(t) \tag{1}$$

where:
K_p = the proportional gain
$e(t)$ = the error between the actual process output and target process output

The Integral component also receives the error, however, it applies the operation shown in equation 2.

$$Integral\,Output = K_i \int_0^t e(\tau)d\tau \tag{2}$$

where:
K_i = the integral gain
$e(\tau)$ = error

Similarly the derivative component outputs the derivative of the error times the Derivative gain (equation 3).

$$Derivative\,Output = K_d \frac{de(t)}{dt} \tag{3}$$

where:
K_d = the derivative gain
$e(t)$ = the error
The final output is the sum of these three components.

The fact that the final output is the sum of relatively straightforward mathematical operations is one of the reasons the PID controller is so popular. However, this also contributes to some of its downfalls. Specifically, the performance of the PID controller is heavily dependent on its hyperparameters, also known as gains. These are often incredibly hard to tune, requiring complex tuning methods such as the Ziegler-Nichols methods [2]. In addition, once tuned, gains are often highly environment-specific so any changes to the environmental characteristics can result in massive performance drops. The controllers are also highly linear, making them unsuitable for complex nonlinear systems.

2.2 Reinforcement Learning

RL represents a family of algorithms in which an agent interacts and learns from an environment directly, by taking actions in a given state and receiving a reward outlining how good or bad that action was.

By combining RL with Deep Learning it is possible to create highly complex models capable of controlling nonlinear systems [3]. In addition, due to the models ability to self-learn, it is also possible to create dynamic agents, capable of adapting to changes in the environment.

However, as Deep RL relies on Deep Learning, it is also susceptible to issues which face Deep Learning, namely, the instability and divergence issues. One cause of these divergence issues is the Deadly Triad which was studied extensively in [4]. To overcome these issues, more advanced RL algorithms, such as the Deep Deterministic Policy Gradient (DDPG) [5], have been proposed.

DDPG is an actor-critic RL approach which is essentially an extension of the Deep Q-network algorithm proposed by Mnih et al [3]. It makes use of a critic to approximate the Q-value of actions in given states and an actor network to make

optimum actions. However, it has been found in previous work [6] that the DDPG algorithm alone can be unstable and diverge when applied to suspension control. Therefore, this paper proposes the use of RPRL to overcome these problems. RPRL is an algorithm which combines RL with traditional control techniques by combining both approaches through summation, as can be seen in equation 4.

$$\pi_\theta(s) = \pi(s) + f_\theta(s) \tag{4}$$

where:
π_θ(s) = final policy
$\pi(s)$ = base policy (Traditional Technique)
f_θ = residual policy (Reinforcement Learning Technique)

Using this equation, it can be seen that the gradient of the policy, with respect to the parameters θ, depends on the residual policy as opposed to the base policy. Therefore, it is possible to use policy gradient methods alongside residual policy RL.

This approach has been shown to improve the performance of the DDPG algorithm when used with a reactive hook base policy. It was also shown in [7] to improve the performance of a P controller.

3 Related Work

This work builds on a magnitude of other research which attempted to tackle some of the problems associated with a PID control.

[8] proposed an Adaptive PID controller (APID) which makes use of the gradient descent algorithm to update the parameters on the fly. This online tuning method allowed the PID controller to avoid pre-tuning, whilst helping it adjust to environmental changes. Similar online tuning methods were proposed in [9] and [10].

Other approaches have involved tuning the PID controller using an RL agent. [11] used an incremental Q-Learning strategy to tune the PID controller in an online fashion. This algorithm dynamically grows a Q-value table in both the action space and state space direction through discretization techniques, in order to create a discrete yet accurate tuning model. Other approaches involved using the Continuous Action RL Automata (CARLA) algorithm [12] and a Radial Base Function (RBF) Actor Critic network [13] to tune a PID controller. However, neither method for online tuning, nor those that combine PID with RL, eliminate issues relating to the linearity of the PID controller.

Finally, approaches have also been made to completely replace PID controllers with RL. [14] successfully applied the DDPG algorithm for an intelligent control strategy for transient response of a variable geometry turbocharger system. Similarly, [15] and [16] also proposed RL models to replace PID controllers. Whilst these models proved to be capable of outperforming and replacing PID

controllers in their given task, they do not directly address the divergence and stability issues associated with Deep RL techniques.

As such, there is a need for an algorithm which can overcome the weaknesses of both PID control and RL techniques, and achieve superior performance.

4 Modelling Suspension Control as a RL Problem

As suspension control systems are closed loop feedback systems it is possible to formulate them as an RL problem. In this case, a quarter-car suspension control simulation was provided by ZF Friedrichshafen AG in the form of an OpenAI GYM environment. This allowed the suspension system behaviour to be simulated, whilst also ensuring that the RL agent and the suspension system could have the same interaction as seen in an RL problem. In other words, it allows for the agent to view the current state of the suspension systems, take an action and receive a reward. The DDPG algorithm (as shown in Algorithm 1) was used, and the state space, action space and reward are outlined below.

4.1 State Space

The state space captures the system's wheel velocity and position. The wheel velocity and position change 1ms ahead of the chassis velocity and position, and therefore using this as the state space gave the agent a chance to act and protect the chassis from impact.

4.2 Action Space

The action space was the suspension system's damping rate. This damping rate described the stiffness of the damper and was in the range [100, 5000], with 100 being a very loose damper, and 5000 being a very stiff damper. The agent was only allowed take an action every 35ms due to real world physical constraints which would prevent the agent from taking an action more frequently. This prevents the agent from reacting immediately to road disturbances that occur during the 35ms window, however, it better reflects the real world constraints.

4.3 Reward Function

The reward received by the agent reflects how successful the agent is at reducing chassis acceleration and as such is expressed as:

$$Reward\,Function = -((clip((abs(a_{chassis}) - 0.315m/s^2), 0, 1) \times 100)) \quad (5)$$

where;
$a_{chassis}$ = chassis acceleration

This reward function was chosen following extensive testing which showed its improved performance over alternatives, including reward functions which

gave an immediate punishment of -100 when accelerations exceeded $0.315m/s^2$ as opposed to the gradual build up in punishment seen in the chosen reward function. In addition, it is based on the ISO 2631 Riding Comforts Standards which highlights that the vehicle ride becomes uncomfortable following accelerations in excess of $0.315m/s^2$. Therefore, the agent receives maximum reward for accelerations below this threshold.

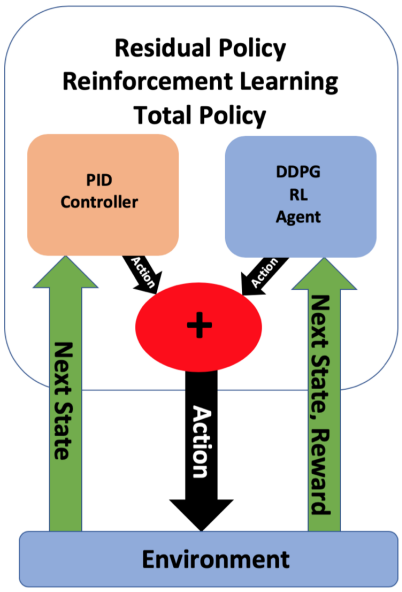

Fig. 1: Residual Policy Reinforcement Learning Agent's Interaction with Environment

4.4 Terminal Goal

The end of the simulation occurs when a terminal goal is reached. In this case the terminal goal occurs when chassis acceleration falls to below $0.2m/s^2$ for 200 consecutive timesteps. This is done to determine when the chassis has settled down following the road disturbance. When this is achieved the simulation ends.

5 Experimental Evaluation and Analysis

5.1 Implementation

The RPRL algorithm was created using a PID controller as the base policy and a DDPG RL agent as the residual policy. The PID controller was pre-tuned using

an iterative method which tested a series of parametric values for each gain in the range [-1000, 1000], with iteration size 0.1. Using this method the optimum parameters were found as follows: *Proportional* = 0, *Integral* = 0, *Derivative* = 0. This means the PID controller outputs a constant target value of 0. Whilst this may seem peculiar it is believed to be caused by the simulation, rather than the tuning method, which requires the PID controller to select a target acceleration rather than a damping rate.

The RL agent was implemented using the DDPG algorithm. The actor's architecture consisted of a single neuron input layer, a 300 neuron hidden layer and a single neuron output layer. All layers used the ReLU activation function, except for the final layer which used a TanH activation function. This allowed for the actor to have a zero centred action space, which could then be multiplied by 5000 to give the RL agent an action space of [-5000, 5000]. Note: This action space is different to the environment action space, as the actor's output is combined with the PID controllers, i.e., it does not represent the action of the RPRL agent alone.

Algorithm 1 DDPG Algorithm Pseudocode [5]

Randomly initialise critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu$ with weights θ^Q and $\theta^m u$.
Initialise target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialise replay buffer R
for episode = 1, M **do**
  Initialise a random process N for action exploration
  Receive initial observation state s_1
  for t=1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}$
    Update critic by minimising the loss: L = $\frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s_i \tag{6}$$

    Update the target networks"

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{7}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{8}$$

  end for
end for

The critic was composed of two input layers made of a single neuron. The first input layer took in the state and connected to hidden layer 1, composed of 400 neurons. Hidden layer 1 then connected to hidden layer 2 which was composed of 300 neurons. The second input layer, which received the actor's action, also connected directly to hidden layer 2. Hidden layer 2 then connected to the single neuron output layer. Again, all layers used ReLU activation functions except for the output layer which used a linear activation.

5.2 Experiments

Experimental scenarios were selected with the following evaluation objectives:

1. To evaluate whether the RPRL algorithm is capable of enhancing a PID-controller's performance in terms of accepted metrics.
2. To evaluate whether the RPRL algorithm has the ability to adapt and re-optimise the PID-controller's performance following changes to the environment.
3. To evaluate whether the RPRL algorithm has the ability to optimise the performance of untuned PID-controllers, reducing the need for tedious tuning methods.

The performance is analysed using the following metrics: Maximum Chassis Acceleration, Maximum Chassis Velocity, Maximum Chassis Movement, Reward Function Score and ISO 2631 Riding Comfort Standard.

Evaluation Scenarios The following three scenarios were evaluated, corresponding to the three evaluation objectives specified above:

1. **Performance Versus a Standard PID** - The algorithm was tested against a PID controller using the same parameters as seen above on 6 different simulations to compare overall performance. This performance was analysed based on the metrics above. Whilst 6 simulations were used (2.5cm sine wave disturbance, 5cm sine wave disturbance, 2.5cm pothole disturbance, 5cm pothole disturbance, 2.5cm hybrid disturbance, 5cm hybrid disturbance), only the results for the 2.5cm hybrid road disturbance[3] will be presented in this paper in the interest of space.
2. **Mass Change** - The algorithm was also tested on its ability to deal with environmental change, which was simulated by altering the mass of the chassis. This mass change was from 650kg to 600kg which is similar to unloading a vehicle. This was tested on the 2.5cm hybrid road disturbance.
3. **Untuned PID** - Finally the algorithm was also tested on its ability to optimise the performance of an untuned PID-controller. In order to do this the PID gains were randomly selected. Again the model was tested only on a 2.5cm hybrid road disturbance.

[3] Hybrid refers to a road disturbance which is a combination of a sine wave and a unit step function.

Training For each simulation the RPRL agent was trained for 30,000 episodes. During training Early Stopping and Performance Threshold were used to improve performance. Early stopping was used to end training when convergence had been reached. This was useful as during training it was found that the RPRL agent could fall out of convergence and diverge to a bad policy. Performance Threshold was used to ensure a minimum was reached. In this case the PID performance level was chosen, and only models performing better than this minimum performance level were saved. In addition the performance threshold (expressed the reward obtained) was updated when a model outperformed, so that only models performing better than this new performance threshold were saved. This meant that only the best performing models would be saved and would further protect against divergence during training.

5.3 Results and Analysis

Performance Versus a Standard PID The results comparing RPRL and traditional PID controller are shown in Tables 1 and 2

Table 1 Evaluation: 2.5cm hybrid disturbance results

Metric	PID-controller	RPRL
Maximum Chassis Acceleration (m/s^2)	0.52	0.49±0.005
Maximum Chassis Velocity (m/s)	0.01	0.0097±0.0001
Maximum Chassis Movement (m)	0.0014	0.0013
Reward	-335	-280 ±15

Table 2 2.5cm hybrid disturbance results as per ISO 2631 ride comfort standard **Note:** the numbers represent the number of milliseconds spent in each comfort level

Comfort Level	PID-controller	RPRL
Comfortable	796	798
Little Uncomfortable	38	42
Fairly Uncomfortable	6	0
Uncomfortable	0	0
Very Uncomfortable	0	0
Extremely Uncomfortable	0	0

We observe that the RPRL agent outperformed the traditional PID controller on all metrics. Table 1 shows that RPRL achieves a better reward score, whilst also experiencing less maximum chassis acceleration, velocity and movement, all indicators of improved chassis comfort. Whilst these improvements may seem

small, they are significant enough to maintain improved comfort levels when compared using the ISO 2631 standard. This is also seen in Table 2, where it is shown that the RPRL agent obtains increased comfort when compared using the ISO 2631 Ride Comfort Standard. Similar results were obtained when tested on the 5cm hybrid, 2.5cm and 5cm sine wave and 2.5cm and 5cm pothole road disturbances, which, in the interest of space, are not shown here.

Mass Change The algorithm was also tested on its ability to deal with environmental change. In this case the model was trained for 30,000 episodes using a vehicle mass of 650kg. Following 30,000 episodes the mass was changed to 600kg and the model was tasked with adjusting to this change and re-optimising. The results can be seen in Figure 2.

Fig. 2: RPRL Agent Responding to Mass Change

The RPRL shows the ability to re-adjust and re-optimise the performance of the suspension control system. This can be seen at the 30,000 episode mark where the mass change occurs. The purple line drops substantially below that of the PID controller, however, it quickly adjusts and achieves performance superior to that of the PID controller. Whilst the oscillation around the PID controller's performance line is not ideal, it is not problematic due to the use of the performance threshold and early stopping techniques described above. As such, this simulation shows the RPRL algorithm's ability to adapt to change.

Untuned PID The final test involved investigating the RPRL ability to optimise a poorly tuned PID controller. In order to do this, a poorly tuned PID controller was used as the base policy. The PID parameters used were selected at random as follows: *Proportional* = 12, *Integral* = -5, *Derivative* = 7.

As can be seen in Figure 3, the RPRL Policy was unable to optimise the poorly tuned PID controller. In fact, the subsequent performance proved to be worse than the poorly tuned PID controller itself. It is thought that this

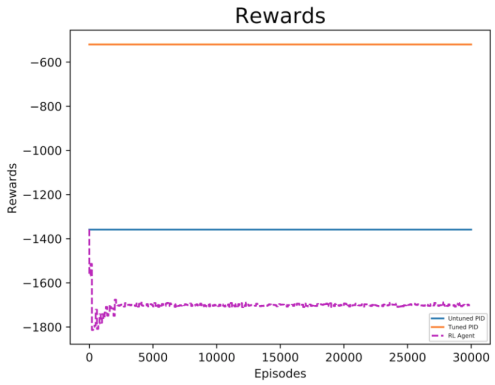

Fig. 3: RPRL Agent with Poorly Tuned PID controller base policy

is because the poorly tuned PID controller offers very little guidance on good actions or even guidance on places to explore for good actions, which was stated in the original paper [1] to be one of the benefits of algorithm. As such, the resultant performance is disappointing.

6 Conclusion

The goal of this work was to use Residual Policy Reinforcement Learning to combine PID control and Reinforcement Learning in a complimentary fashion, so as to maximise the advantages of each system. Whilst it was successfully shown that this system is capable of optimising and enhancing the performance of a PID controller and improving the adaptability of a PID controller, it was also shown that it was incapable of compensating for poorly tuned PID controllers and therefore is highly reliant on the base policy used.

These findings are not only important for future work involving Reinforcement Learning and damper control, but for all work involving system control. At present PID control is one of the most popular system control methods and as such the findings here can be applied to many other use cases.

Future work should focus on applying this technique to more complex simulations, including, where possible, simulations that make use of real world data. In this case a simplified simulation was used and as such the results should be verified on more realistic models.

7 Acknowledgements

This publication is supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number 16/SP/3804

References

1. Silver, Tom, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. "Residual Policy Learning." arXiv (2018): arXiv-1812
2. John G Ziegler, Nathaniel B Nichols, et al. Optimum settings for automatic controllers. Trans. ASME, 64(11), 1942.
3. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518 (7540):529–533, 2015. Published as a conference paper at ICLR 2016.
4. van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N. and Modayil, J., 2018. Deep Reinforcement Learning and the Deadly Triad.
5. Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." arXiv (2015): arXiv-1509.
6. Andrew Hynes. The applications of reinforcement learning techniques to car control. Technical Report, ZF and Trinity College Dublin. June 2019.
7. Schoettler, Gerrit, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. "Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards." arXiv (2019): arXiv-1906.
8. Laiq Khan, Shahid Qamar, and Umair Khan. Adaptive PID control scheme for full car suspension control. Journal of the Chinese Institute of Engineers, 39(2):169–185, 2016.
9. Hamed Khodadadi and Hamid Ghadiri. Self-tuning PID controller design using fuzzy logic for half car active suspension system. International Journal of Dynamics and Control, 6(1):224–232, 2018.
10. Rodrigo Hernández-Alvarado, Luis Govinda García-Valdovinos, Tomás Salgado-Jiménez, Alfonso Gómez-Espinosa, and Fernando Fonseca-Navarro. Neural network-based self-tuning pid control for underwater vehicles. Sensors, 16(9):1429, 2016.
11. Ignacio Carlucho, Mariano De Paula, Sebastian A Villar, and Gerardo G Acosta. Incremental q-learning strategy for adaptive PID control of mobile robots. Expert Systems with Applications, 80:183–199, 2017.
12. Mark N Howell and Matt C Best. On-line PID tuning for engine idle-speed control using continuous action reinforcement learning automata. Control Engineering Practice, 8(2): 147–154, 2000.
13. Xue-Song Wang, Yu-Hu Cheng, and Sun Wei. A proposal of adaptive PID controller based on reinforcement learning. Journal of China University of Mining and Technology, 17(1):40–44, 2007.
14. Bo Hu, Jie Yang, Jiaxi Li, Shuang Li, and Haitao Bai. Intelligent control strategy for transient response of a variable geometry turbocharger system based on deep reinforcement learning. Processes, 7(9):601, 2019.
15. Weinan Deng, Hao Li, and YuanQiao Wen. Data-driven unmanned surface vessel path following control method based on reinforcement learning. In 2019 Chinese Control And Decision Conference (CCDC), pages 3035–3040. IEEE, 2019.
16. Le Jiang, Yafei Wang, Lin Wang, and Jingkai Wu. Path tracking control based on deep reinforcement learning in autonomous driving. In 2019 3rd Conference on Vehicle Control and Intelligence (CVCI), pages 1–6. IEEE, 2019.