State Quantize for Pursuit Approximate Optimal Control using Reinforcement Learning

Huanhuan YU[1,2] and Haohao MAI[1] and Shuling GAO[1] and Xiwen HUANG[1]

and Qiuling YANG[*, 1,3]

[1] Hainan University
[2] 22210812000009@hainanu.edu.cn
[3] qlyang@hainanu.edu.cn

Abstract. In high-speed vehicle motion scenarios, solving optimal control problems faces significant challenges in terms of time and space complexity. Ensuring real-time performance of the controller requires efficient solving algorithms and support from high-performance computing platforms. To reduce the computational cost and approach the performance of optimal control, approximate optimal control has emerged as a feasible solution. In this paper, we propose an approximate optimal vehicle control method that outperforms Model Predictive Control (MPC) in terms of performance. The method combines the pure pursuit algorithm for vehicle path tracking with the Twin Delayed DDPG (TD3) algorithm to generate approximate lookahead distance and velocity control values for the vehicle. Additionally, the vehicle state is quantized and discretized. In our experiments with a vehicle simulator, we compare the MPC control with our proposed method. The results show that while the MPC control remains stable at a vehicle speed of up to 70MPH, our method effectively controls the vehicle even at a speed of 100MPH, with higher control rate and robustness.

Keywords: approximate optimal control, pure pursuit, TD3, quantize

1 Introduction

With the development of science and technology, artificial intelligence technology is more and more widely used in our daily life. The optimal control of vehicles is widely used in the field of autonomous driving and has attracted more and more attention from industry and academia.Ensuring the dynamic stability of the vehicle is a fundamental requirement for achieving optimal control. Dynamic stability, in this context, refers to the ability to avoid lateral drift, tipping, and stopping [1].Pure pursuit control[2] is a widely used front-wheel steering algorithm for path tracking that offers efficient performance at low computational costs and demonstrates good tracking capabilities at low vehicle speeds. However, it operates as a proportional controller and may result in lateral oscillations if the lookahead distance is too short,

while an excessively long distance can lead to inadequate fitting at curved path points. In practical applications, the lookahead distance is often adjusted according to the vehicle speed. Despite its advantages, pure pursuit's control performance becomes significantly limited at relatively low speeds.On the other hand, Model Predictive Control (MPC) exhibits strong robustness and optimal control characteristics that enable effective management of vehicles even in high-speed scenarios[3],[4]. MPC involves predicting future dynamics, solving optimization problems, and controlling the vehicle in a continuous computational loop at each time step. It utilizes a considerably larger model than the original state space model and only obtains the optimal control value for the current step. Consequently,MPC requires high computational hardware performance. Clearly,this approach is not suitable for vehicles equipped with low-performance processors.

For vehicle systems with complex dynamics and nonlinear characteristics, the control performance of traditional controllers is influenced by the accuracy of the modeling. Moreover, in complex and dynamic road environments, vehicle control is challenging to abstract into formulas or simple logic.Reinforcement Learning (RL) learns the mapping from states to actions through interactions with the environment, providing strong nonlinear approximation capabilities and greatly simplifying the design of control systems.Some researchers have used RL for local obstacle avoidance[5], end-to-end autonomous driving[6],and autonomous parking [7].However, RL-based vehicle control often focuses more on longitudinal velocity or lateral control, with less emphasis on the overall optimal control of the vehicle[8]. Additionally, using RL alone for vehicle control often leads to suboptimal performance and usually requires integration with traditional methods for better control outcomes [9].The uncertainty of vehicle control is closely related to sensor noise. Although the sensor data is processed by Kalman filtering algorithm[10],the small amplitude noise may lead to the degradation of RL control performance.

In this paper, we focus on addressing a specific sub-problem in motion planning, which involves controlling the vehicle to closely follow a planned trajectory at an approximately optimal velocity while ensuring dynamic stability. The primary objective is to minimize the time required for the vehicle to reach its destination. In order to develop a controller that is easy to design, computationally efficient, and exhibits high control rates and strong robustness, we propose a RL-based optimal control method called State Quantize for Pursuit Approximate Optimal control (SQ-PAO), which combines the advantages of pure pursuit and RL. The objective of this method is to drive a vehicle along any given path with optimal speed while closely following the path. To facilitate rapid learning, we design a reward function based on prior linear knowledge, which guides the agent's learning process. The input sequence of states for the agent is quantized and discretized, thereby confining the infinite state space within a finite space. By operating within this limited space, the agent can output actions that approximate the behavior for similar states, reducing the behavioral discontinuity caused by minor state variations and improving the model's robustness. Instead of selecting the optimal action for the current state, the agent chooses the action that is most similar to the current state at each time step. Hence, we refer to this approach as approximate optimal control.

2 Problem Statement

The vehicle will drive along the planned path while simultaneously optimizing the velocity and minimizing the tracking error of the path. Pure pursuit controls the steering of the vehicle according to the geometric relationship of the lookahead distance, and the learned Agent determines the lookahead distance and speed.

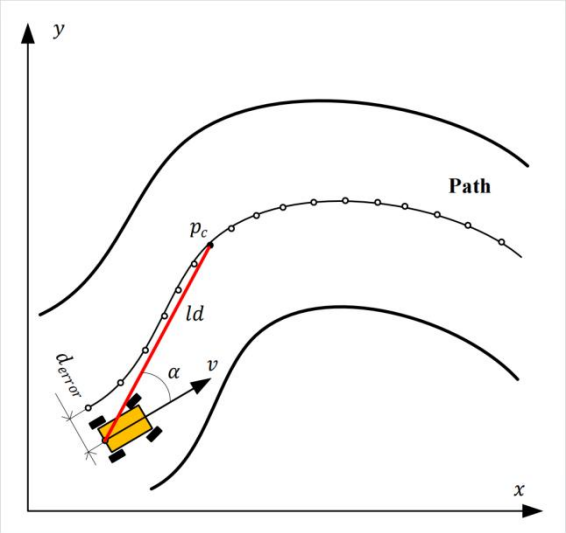


Fig. 1. Driving diagram

In Fig. 1, path consists of a series of waypoints, the waypoints is defined as a sequence of points represented by $P = \{p_1, p_2, p_3, \ldots . p_n\}$. In this context, p_c denotes a specific point in the path P. The variables v represent the current velocity of the vehicle, α represents the angle between the velocity vector and the direction towards the lookahead point. The term d_{error} represents the error or deviation between the vehicle's position and the target waypoint, ld represents the distance between the lookahead point and the center of the rear axle. Pure pursuit uses p_c as a lookahead point to control steering, The Agent's goal is to drive the vehicle at the optimal speed while keeping the path within a certain error range while ensuring stability. At moment k, the wheel angle δ can be expressed as:

$$\delta(k) = arctan(\frac{2Le_y(k)}{ld^2})(1)$$

where L is the distance from the front axle to the vehicle rear axle, and $e_y(k)$ is the lateral error between the lookahead point and the forward direction of the vehicle at time k.

Equation (1) is suitable for fitting the path at low speeds. However, as the speed increases, the corresponding lookahead distance needs to be longer.The growth rate of the denominator ld^2 is much larger than that of the numerator $2Le_y(k)$ .Consequently, the value of $\delta(k)$ becomes smaller.This indicates that the steering ability of the vehicle weakens at higher speeds. As a result, the lateral error $e_y(k)$ of the vehicle increases, and it becomes more prone to deviating from the desired path. To address this issue, we adjusted the order of the denominator to 1.8 in our experiments.

3 Background

3.1 Agent Learning

Fig. 2 illustrates the process of the RL Agent interacting with the environment, with the aim of maximizing the total reward obtained from the environment.At each discrete time step t, given a state $s \in S$, the Agent selects an action $a \in A$ based on its policy $\pi: S \rightarrow A$.The Agent then receives a reward R_{t+1} and transitions to a new state s_{t+1} in the environment. The reward at each time step is calculated as the sum of the future rewards discounted by a factor $\gamma: R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, where γ represents the discount factor.

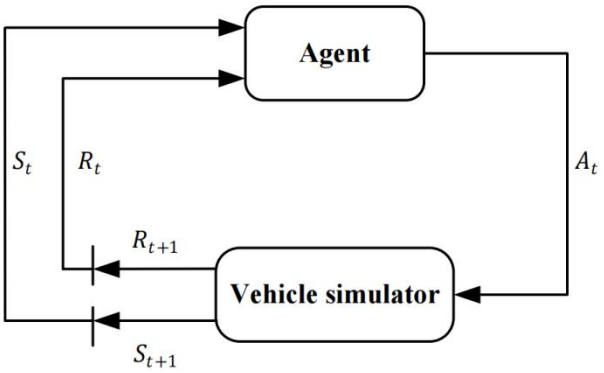


Fig. 2. Agent interacts with the environment

3.2 Twin Delay DDPG

Deep Deterministic Policy Gradients (DDPG,[12]) is an algorithm based on the Actor-Critic framework, which incorporates the deterministic strategy of Deterministic Policy Gradient (DPG, [13]) in the Actor component. In the DDPG algorithm, the Actor is responsible for generating actions directly from the current state using a parametric equation approximation, typically implemented as a neural network. The neural network takes the state as input and produces the corresponding action as output. By learning this mapping from states to actions, the Actor can make

decisions without relying on explicit policy rules or heuristics. On the other hand, the Critic uses the action-state value function Q to evaluate the quality or value of an action in a given state. The Critic takes both the current state and the action as inputs and estimates the expected cumulative rewards associated with taking that action in the given state. By evaluating the Q-value, the Critic provides feedback to the learning process and guides the Actor in selecting actions that are more likely to lead to higher rewards. The goal of the Actor is to maximize the expected sum of current and future discounted rewards, represented as:

$$\pi_{\phi}^{*} = arg \max_{\pi_{\phi}} \sum_{t} E[r(s_t, \pi_{\phi}(s_t)) + \gamma Q_{\theta}(s_{t+1}, \pi_{\phi'}(s_{t+1}))] \quad (2)$$

DDPG can achieve good performance, but in many cases, it is extremely sensitive to the adjustment of hyperparameters and some other parameters. Because the actual Q value Q_{target} may be overestimated like DQN[14], which will lead to the failure of policy learning. TD3[15] solves this problem by learning two Q functions and using the smaller Q value among them to compute the target term in the Bellman error loss function, this approach helps mitigate the overestimation bias and improves the stability of the learning process.

$$Q_{target} = r + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, \pi_{\phi'}(s_{t+1})) \quad (3)$$

3.3 Vector Quantization

For the continuous state space learned by the RL agent, the state S is infinite, which will cause the agent to learn slowly. State sparse discretization can effectively reduce state space features. In two-dimensional continuous space, state discretization can simply be done using one-hot encoding. The high-dimensional state can be sparsely encoded in low-dimension through Tile coding [16],[17]. The limitation of tile encoding is that the generalization ability is limited by the number of layers of tiles. Here, we introduce vector quantization (VQ), a method for discretizing sequences that are related in time and space.

In VQ-VAE [18],[19], VQ technique is utilized to quantize and discretize the latent variables *x* obtained from the encoder. In the quantization process described by (4), a codebook or embedding table is created. This codebook contains a set of embeddings or codewords. The goal is to find the embedding in the codebook that is closest to the input *x* based on the Euclidean distance metric. This closest embedding is then used as a replacement for the original *x*, resulting in the discretization of the latent space. By applying the VQ technique, the continuous-valued latent variables are replaced with discrete embeddings, enabling a more structured and compressed representation of the data.

$$Quantize(x) = e_k \quad where \quad k = arg\,min\left|x - e_j\right| \quad (4)$$

The codebook embeddings are updated to ensure that they are close to the input vectors. In the experiment, the update of the embeddings in the codebook is performed using exponential moving average, as described by (5). This update process aims to gradually adjust the codebook embeddings towards the input vectors, enabling them to better represent the data distribution.

$$N_i^{(t)} = N_i^{(t-1)} * \gamma + n_i^{(t)}(1-\gamma)(5)$$

where $n_i^{(t)}$ is the vector e_i that is closest to the input vector *x* in the codebook, and γ is an update coefficient with a value between 0 and 1.In this experiment, we take $\gamma = 0.99$.

4 Background

This section will introduce the research methodology employed in this paper, which includes vehicle state analysis, the design of the reward fucnction, and the model utilized.

4.1 Vehicle State Space

The state S of the vehicle in this research consists of three main components.

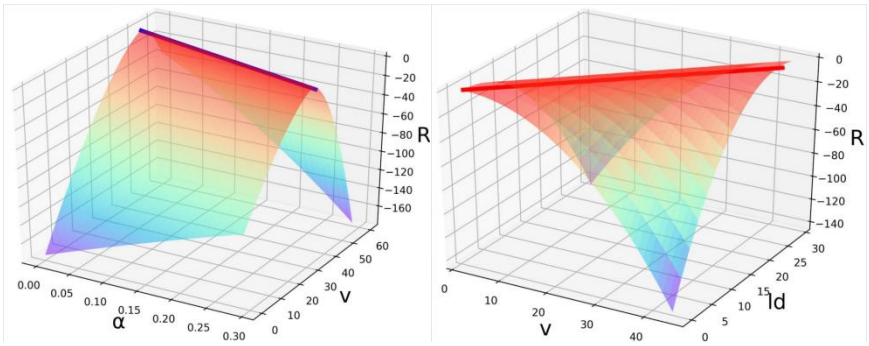
- Angles between waypoints and the vehicle: Denoted as $A = \{\alpha_1, \alpha_2, \alpha_3, \ldots . \alpha_{16}\}$, represents the collection of the angle between the waypoints in front of the vehicle and the vehicle direction, convert the coordinates of the waypoints P to the relative Angle α with $arctan(\frac{y}{x})$.
- Actual turning angle and vehicle speed: Turning angle θ, represents the angle at which the vehicle is currently turning. Vehicle speed v, indicates the current velocity of the vehicle. These parameters provide information about the vehicle's motion and dynamics, which are essential for controlling the vehicle's trajectory.
- Deviation between vehicle driving path and planned path: d_{error}, represents the deviation between the vehicle's actual driving path and the planned path. It quantifies the difference between the vehicle's trajectory and the desired path, indicating how closely the vehicle is following the intended route. This measure of deviation serves as feedback for the control algorithm to make adjustments and minimize the path tracking error.

At each time step t, the three components described above, namely angles between waypoints and the vehicle A, the actual turning angle and vehicle speed $[\theta, v]$, and the deviation d_{error} between the vehicle driving path and planned path, are concatenated into a state vector. This state vector represents the current environmental state of the vehicle and serves as the input to the RL model.

4.2 Prior Reward

In order to speed up the learning process of the Agent, we roughly set the wide range of the lookahead distance and speed according to the prior experience, and design the reward function. The Agent learns in the linear prior area.

In Fig.3(a), $\alpha = \frac{1}{n}\sum_{i=1}^{n} \alpha_i, \alpha_i \in A$ is used to indicate the degree of curvature of the path ahead, and v represents the corresponding empirical speed. The top line represents the linear empirical relationship $v = f_1(\alpha)$ between the average curvature α of the route and the empirical speed v. The function $f_1(\alpha)$ serves as the threshold separating high and low speed penalties, and $R_v = F_1(\alpha, v)$ represents the reward surface associated with high and low speeds.



(a) Velocity reward space (b)ld reward space

Fig. 3 Velocity and ld prior reward diagram

Similarly, in Fig.3(b), the top line represents the linear relationship $ld = f_2(v)$ between the vehicle speed v and the empirical lookahead distance ld. The function $f_2(v)$ acts as the threshold distinguishing between excessively long and excessively short lookahead distances, and $R_{ld} = F_1(v, ld)$ represents the reward surface associated with such lookahead distance deviations.

The reward for the vehicle includes four components: road reward, speed reward, lookahead distance reward, and deviation reward.

- Beyond the road reward: If the vehicle deviates too far from the planned path, it will receive a reward of $R_{over} = -500$ and the episode will end.
- Speed reward: If the absolute difference between the actual speed v_t and the expected speed $f_1(\alpha_1 dt)$ exceeds a threshold Δx, the vehicle will receive a reward based on the function $F_1(\alpha_t, v_t)$. To incentivize higher speeds, a reward $R_{speedup}$ is introduced. This reward is effective at low speeds.
- Lookahead distance reward: If the absolute difference between the actual lookahead distance ld_t and the expected lookahead distance $f_2(v_t)$ exceeds a threshold Δx, the vehicle will receive a reward based on the function

$F_2(v_t, ld_t)$. To encourage longer lookahead distances to adapt to high-speed scenarios, a reward $R_{lengthen}$ for increasing the lookahead distance is introduced. This reward becomes active when the lookahead distance is too short.

- Deviation reward: The reward $R_{deviation}$ is based on the degree of path fitting.

4.3 Structure of Actor-Critic

Fig.4 shows the network structure of Actor and Critic in this paper, the Actor in our approach adopts a simple network structure. We utilize vector quantization to discretize the angle A related to the path in the input model. The discretized representation, denoted as $\widehat{A}$, is then processed by a Temporal Convolutional Network (TCN) [20], followed by a fully connected layer for behavior output. The TCN consists of multiple layers with hidden channels [32, 32, 32, 1] and dilation rates [1, 2, 4, 8] respectively. To address the gradient disappearance issue of the Rectified Linear Unit (ReLU) in the original TCN, we replace it with LeakyReLU activation in each block and center the input of each layer. The output of the last layer of the TCN is concatenated with $[\theta, v, d_{error}]$ along the channel dimension and then fed into a fully connected network for action output. The vehicle steering angle is derived from the lookahead distance of the output behavior using (1). Finally, the computed steering and speed actions are applied to the vehicle controller.

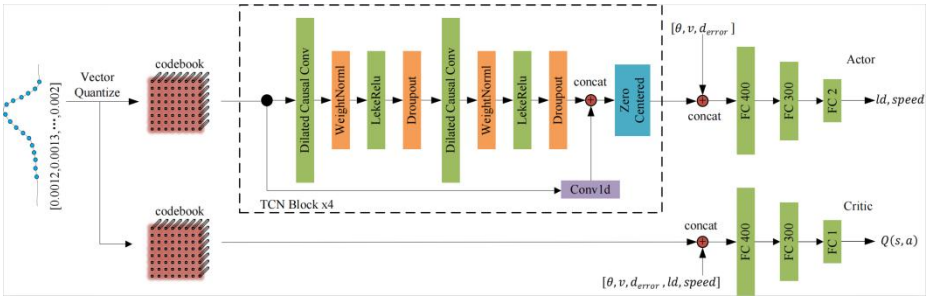


Fig. 4 Actor and Critic network structure

On the contrary, Critic does not require TCN to calculate the Q-value of an state-action pair. Instead, the quantized vector $\widehat{A}$ is concatenated with $[\theta, v, d_{error}, ld, speed]$ and passed through a fully connected network to obtain the approximate state-action value $Q(s, a)$.

During the forward propagation process, the codebook used for vector quantization is updated according to (5). It's important to note that during the backward gradient propagation of the model, the gradients are not propagated to the codebook. This means that the codebook remains unchanged during the training of the Critic and its update is decoupled from the model's parameter updates. The training process of Actor and Critic is shown in Algorithm 1.

Algorithm 1

Algorithm 1 SQ-PAO

Require:

NN $Q_{\theta_1}, Q_{\theta_2}, \pi_\phi$ with θ_1, θ_2, ϕ
target NN $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
codebook q_0, q_1, q_2 for $Q_{\theta_1}, Q_{\theta_2}, \pi_\phi$
target codebook $q_0' \leftarrow q_0, q_1' \leftarrow q_1, q_2' \leftarrow q_2$
replay buffer $\mathcal{B}$ and disable q_0', q_1', q_2' updates

for $epoch = 1$ **to** $MaxEpochs$ **do**
  $s_0, done = env.reset(), false$
  while not done **do**
    Select action $a \sim \pi_\phi(s_0) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$
    $s', r, done = env.step(a)$
    Store transition (s_0, a, r, s') in $\mathcal{B}$
    $s_0 \leftarrow s'$
  end while
  for $i = 1$ **to** t **do**
    ▷Update Critic
    Sample N transitions (s_0, a, r, s') from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
    $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s_0, a))^2$
    if t **mod** d **then**
      ▷Update Actor, target NN and codebook
      $\nabla_\phi J(\phi) = \frac{1}{N} \sum \nabla_\phi Q_{\theta_1}(s_0, \pi_\phi(s_0))$
      $\theta_i' \leftarrow \tau\theta_i' + (1-\tau)\theta_i, \quad \phi' \leftarrow \tau\phi' + (1-\tau)\phi$
      $q_0' \leftarrow q_0, q_1' \leftarrow q_1, q_2' \leftarrow q_2$
    end if
  end for
end for

5 Experiment

The experimental part of this study focuses on the training, testing, robustness, and learning curve analysis of the SQ-PAO Agent.

5.1 Train

The environment used the self-driving car simulator developed by Udacity.The Agent interacted with the environment for 400 rounds, and the maximum interaction step was 8k steps each epoch.

Fig.6 and Fig.7 record the average reward of each step at the end of each round and the Mean Squared Error (MSE) of input and quantization. There is no significant difference in training convergence speed between SQ-PAO Agent and TD3, during the training convergence phase, the average per-step reward obtained by the SQ-PAO Agent is higher than TD3. The MSE between the input and output vectors of the

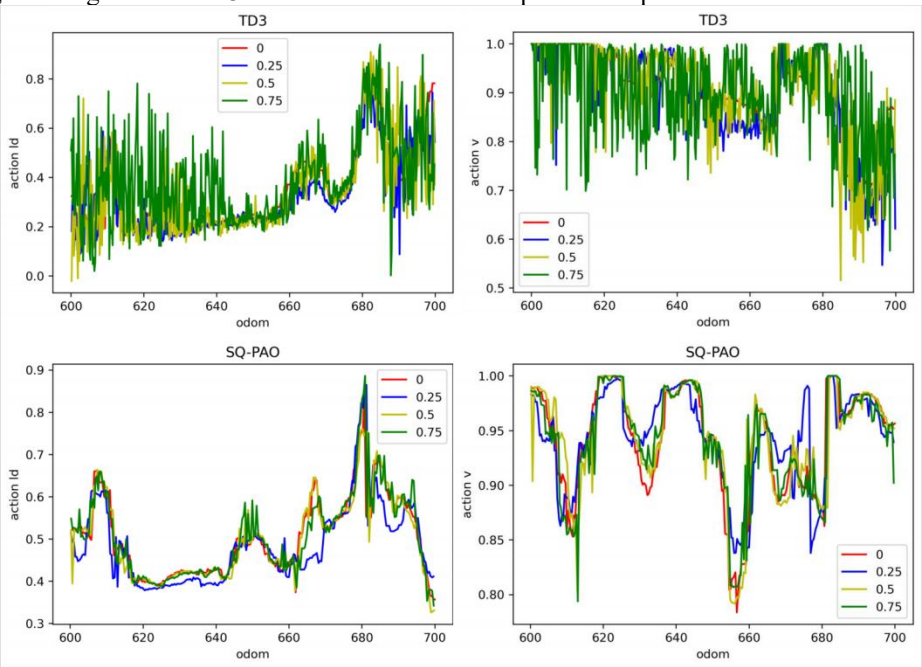


Fig. 5 Robustness comparison, add 25\%, 50\%, 75\% noise respectively, of which the 0 curve is the original curve without noise.

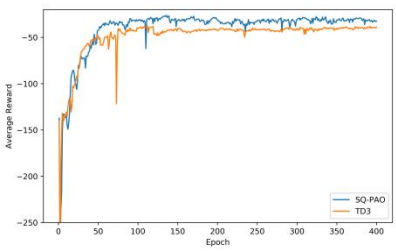


Fig. 6 Training average reward per step

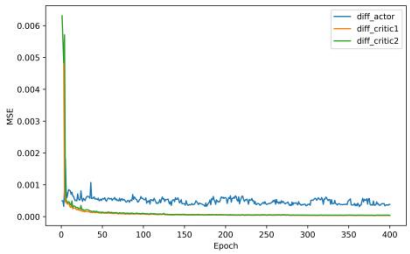


Fig. 7 The vector MSE before and after quantization

codebook starts to converge at epoch 50. The codebook error of the Critic can converge to a range of $[0,1e-4]$, indicating a good approximation of the embeddings. On the other hand, the codebook error of the Actor fluctuates within a small range. This difference in codebook error behavior can be attributed to the

update strategy of the Actor and Critic. The Actor updates the codebook during the interaction with the environment and selection of actions, where it inputs one state and updates the codebook. In contrast, the Critic updates the codebook at the end of each round, and the batch size of the input state is 512. The use of larger batch sizes tends to provide more stable trends compared to smaller batch sizes. Therefore, the Critic exhibits more stability than the Actor when updating the codebook.

Table 1
Action MAE between noise with origin

	nosie	0.25	0.5	0.75
ld	TD3	0.068	0.079	0.083
	SQ-PAO	0.020	0.017	0.017
v	TD3	0.720	0.718	0.696
	SQ-PAO	0.661	0.669	0.666

5.2 Robustness

The addition of Vector Quantization enhances the Agent's robustness to small-amplitude noise in the state sequence set A. To assess its robustness, we introduced random noise to the input state A before the Actor selects an action. Specifically, we applied the following noise perturbation: $\alpha_i = \alpha_i + \eta \cdot rand(-1,1) \cdot \alpha_i$, where $i = 1,2\ldots,16$. This noise perturbation was compared to the performance of the TD3 algorithm. In the noise perturbation equation, α_i represents the individual elements of A, which have been normalized. $rand(-1,1)$ is a random number generated from a uniform distribution ranging from -1 to 1, and η is the noise size coefficient. By applying this random noise to the input state A, we aimed to evaluate the Agent's ability to handle and mitigate the effects of small-amplitude noise in the state sequence.

From Fig.5 and Table 1, we observed interesting results regarding the robustness of the algorithms to noise. When the noise coefficient was set to 0.25, the action output of TD3 exhibited significant and erratic fluctuations, while the action output of SQ-PAO fluctuated around the original action curve. As the noise coefficient increased, the action output of TD3 started to exhibit abrupt jumps at the maximum and minimum output boundaries, while SQ-PAO managed to maintain a smoother trajectory close to the original action curve. These results highlight the advantage of SQ-PAO in handling small-amplitude noise in the state sequence. While TD3's action output was more susceptible to noise and experienced larger deviations, SQ-PAO demonstrated greater resilience and maintained a closer approximation to the original action curve. This robustness can be attributed to the integration of vector quantization, which enables SQ-PAO to effectively quantize and preserve important features of the input state despite the presence of noise.

5.3 Comparison with MPC

In the experiment, SQ-PAO and MPC were autonomously driven for ten rounds, and we recorded the average speed and path fit. The optimization objective of the MPC objective function Γ in the experiment is as follows:

$$\begin{aligned}\min \Gamma = \omega_1 \sum_{k=1}^{N} (|d_{error}|^2 + |epsi_k|^2 + |v_k - v_{ref}|^2) \\ + \omega_2 \sum_{k=1}^{N} (|\delta_k - \delta_{k+1}|^2 + |a_k - a_{k+1}|^2) \\ + \omega_3 \sum_{k=1}^{N} (|\delta_k|^2 + |a_k|^2)\end{aligned}$$

where $epsi$ represents the heading error, which quantifies the angular deviation of the vehicle's heading from the desired direction. v represents the current vehicle speed, while v_{ref} represents the target speed or desired velocity. The term $|v_k - v_{ref}|^2$ in the objective function penalizes deviations between the current speed and the desired speed. Furthermore, the objective function includes terms that penalize the rate of change in the wheel steering angle δ and the vehicle acceleration a The terms $|\delta_k - \delta_{k+1}|^2$ encourage smooth and gradual changes in the steering angle and acceleration. The weights ω_1, ω_2, ω_3 determine the relative importance of each component in the objective function. By adjusting these weights, one can prioritize certain aspects of the control performance over others. The overall goal of minimizing this objective function is to achieve accurate tracking of the desired path, heading, and speed, while also ensuring smooth and stable control actions for steering and acceleration.

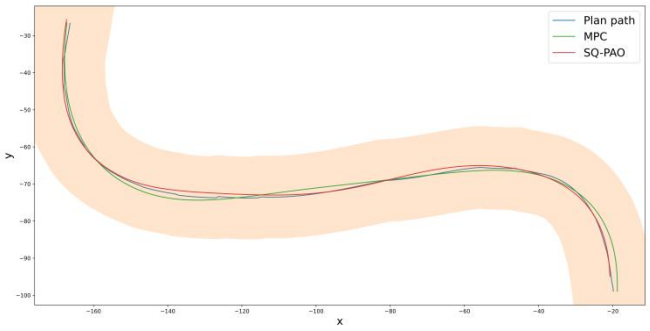


Fig. 8 MPC, SQ-PAO driving path and planning path in curve

According to Fig.8,9,10, it can be observed that SQ-PAO achieves higher speed and smaller path deviation compared to MPC, demonstrating the superior performance of SQ-PAO in terms of speed and path tracking accuracy. SQ-PAO is

capable of driving at higher speeds with smaller path deviation, while MPC performs relatively poorly in comparison. In the Python implementation of the controllers, the control rate of MPC is approximately 30Hz, while SQ-PAO operates at around 250Hz. In the C++ implementation, the control rate of MPC exceeds 1000Hz, while SQ-PAO reaches approximately 1600Hz. This indicates that SQ-PAO can control the vehicle at higher frequencies, further emphasizing its advantages in control performance. In Fig.11, our method effectively controls the vehicle even at a speed of 100MPH.

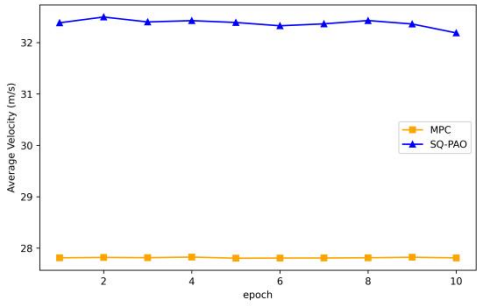


Fig. 9 Average speed

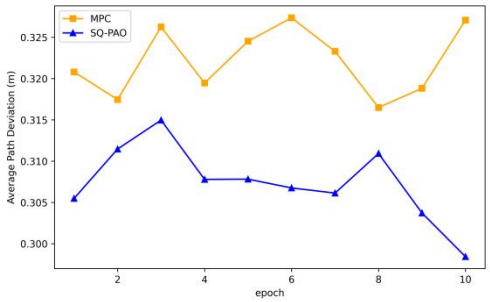


Fig. 10 Mean path deviation

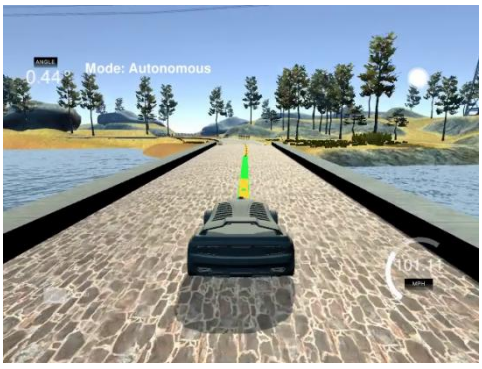


Fig. 11 vehicle moves at high speed in the simulator

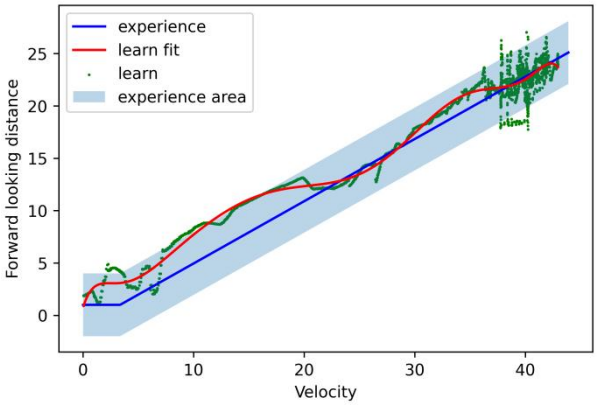


Fig. 12 lookahead distance fitting curve

5.4 Learning Presentation

The Agent learns within a linear prior area, but due to the time it takes for the vehicle to reach the desired speed from the Agent's output action, the output speed of the Agent may be greater or less than the desired speed during this period. This introduces some challenges in analyzing the speed behavior curve of the Agent. On the other hand, the lookahead distance behavior in the Agent's output action is transformed into the wheel angle through geometric transformation, and it is not time-limited. This makes it easier to display the learning fit curve, and the learned curve is expressed as $ld = \Gamma(v)$ for lookahead distance behavior.

In Fig.12 , visualize the Agent learned lookahead distance behavior in the experiment and ploy fit the learning curve of the Agent, the lookahead distance behavior learned by the Agent exhibits a polynomial relationship with the speed behavior. In the experiment, we employ a 9th-order polynomial to fit the Agent's learned lookahead distance behavior. It is worth noting that a significant portion of the Agent's learning interval aligns with our linear empirical interval, indicating a satisfactory alignment between the learned behavior and our established expectations.

6 CONCLUSION

In this paper, we introduce the SQ-PAO method, which combines pure pursuit and reinforcement learning (RL) techniques for the approximate optimal control of vehicles. The Agent learns to output the lookahead distance and speed to effectively control the vehicle.

To expedite the learning process of the Agent, we devise various reward functions based on linear experiences. Additionally, for the collection of angles, denoted as A, we perform Vector Quantization on it. We then utilize TCN to serialize the

quantized vector $\widehat{A}$, which is subsequently fed into a fully connected network for action output. In the experimental section, we conducted a comparative analysis between SQ-PAO and MPC from three perspectives: speed, path deviation, and actual driving route. The results demonstrate that SQ-PAO exhibits superior path fitting ability, achieving a smaller path deviation and enabling the vehicle to drive at higher speeds. Moreover, when implemented using the same programming language, SQ-PAO showcases a higher control rate compared to MPC. Overall, the experimental findings indicate that SQ-PAO outperforms MPC in terms of comprehensive control ability. Furthermore, we introduced different levels of noise to the input and compared the action outputs of TD3 (Twin Delayed Deep Deterministic Policy Gradient) and SQ-PAO. The addition of noise leads to highly unstable action outputs in TD3, resulting in erratic vehicle behavior. Conversely, SQ-PAO maintains relatively stable action outputs, highlighting its superior robustness. Additionally, we visualized the learning curve of the Agent concerning the lookahead distance, revealing that a significant portion of the learning curve aligns with our linear empirical interval.

Lightweight model SQ-PAO method can operate on low-performance processors while delivering superior control performance compared to MPC, along with robustness. However, it is essential to note that the experiments were conducted solely on a simulator and are subject to various physical factors on real roads. Achieving the ideal performance observed in the simulator in real-world scenarios may require further adjustments and considerations.

7 ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under Grant 62362026; in part by the specific research fund of The Innovation Platform for Academicians of Hainan Province under Grant YSPTZX202314.in part by the Scientific Research Project of Hainan Province under Grant Hnky2022-4.

References

1. Jeong hwan Jeon et al., "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," 2013 American Control Conference, Washington, DC, USA, 2013, pp. 188-193.
2. R.C 。 Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST,Tech.Rep., 1992.
3. R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch and M. Diehl, "Towards time-optimal race car driving using nonlinear MPC in real-time," 53rd IEEE Conference on Decision and Control, Los Angeles, CA, USA, 2014, pp. 2505-25101.

4. J. V. Carrau, A. Liniger, X. Zhang and J. Lygeros, "Efficient implementation of Randomized MPC for miniature race cars," 2016 European Control Conference (ECC), Aalborg, Denmark, 2016, pp. 957-962.
5. A. Faust et al., "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning," 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 2018, pp. 5113-5120.
6. U. Patel, N. K. S. Kumar, A. J. Sathyamoorthy and D. Manocha, "DWA-RL: Dynamically Feasible Deep Reinforcement Learning Policy for Robot Navigation among Mobile Obstacles," 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021, pp. 6057-6063.
7. Minghong Hu, Hui Guo, and Xuyuan Ji. 2019. Automatic Driving of End-to-end Convolutional Neural Network Based on MobileNet-V2 Migration Learning. In Proceedings of the 12th International Symposium on Visual Information Communication and Interaction (VINCI '19). Association for Computing Machinery, New York, NY, USA, Article 36, 1–4.
8. Zhang, P.; Xiong, L.; Yu, Z.; Fang, P.; Yan, S.; Yao, J.; Zhou, Y. Reinforcement Learning-Based End-to-End Parking for Automatic Parking System. Sensors 2019, 19, 3996.
9. S. Kuutti, R. Bowden, Y. Jin, P. Barber and S. Fallah, "A Survey of Deep Learning Applications to Autonomous Vehicle Control," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 2, pp. 712-733, Feb. 2021.
10. Lakhani, Ayub I., Myisha A. Chowdhury, Qiugang Lu. 2022. "Stability-preserving automatic tuning of PID control with reinforcement learning" Complex Engineering Systems. 2, no.1: 3. http://dx.doi.org/10.20517/ces.2021.15
11. Kim, Taeklim, and Tae-Hyoung Park. 2020. "Extended Kalman Filter (EKF) Design for Vehicle Position Tracking Using Reliability Function of Radar and Lidar" Sensors 20, no. 15: 4126.
12. Lillicrap, Timothy P. et al."Continuous control with deep reinforcement learning." CoRR abs/1509.02971 (2015): n. pag.
13. D. Silver, G.Lever,N.Heess, T.Degris,D.Wierstra,M. Riedmiller,"Deterministic policy gradient algorithms". In: International conference on machine learning.PMLR,2014,pp387-395.
14. Mnih, Volodymyr et al. "Playing Atari with Deep Reinforcement Learning." ArXiv abs/1312.5602 (2013): n. pag.
15. "Addressing Function Approximation Error in Actor-Critic Methods : Supplementary Material A . Proof of Convergence of Clipped Double Q-Learning." (2018).
16. Alexander A. Sherstov and Peter Stone. 2005. "Function approximation via tile coding: automating parameter choice".In: Proceedings of the 6th international conference on Abstraction, Reformulation and Approximation (SARA'05). Springer-Verlag, Berlin, Heidelberg, 194–205.
17. Sutton, Richard S.. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding." Neural Information Processing Systems (1995).
18. Oord, Aäron van den et al. "Neural Discrete Representation Learning." ArXiv abs/1711.00937 (2017): n. pag.
19. A.Razavi, A.van den Oord, O.Vinyals."Generating diverse high-fidelity images with vq-vae-2" In: Advances in neural information processing systems,2019,pp.14866-14876
20. Lea, Colin S. et al. "Temporal Convolutional Networks: A Unified Approach to Action Segmentation." ECCV Workshops (2016).