Final Project Report

MPC-SAC: A Hybrid Autonomous Parking Motion Planner

Zhao Yimin

Matric Number: A0285282X

ME5418 Machine Learning in Robotics

National University of Singapore

November 22, 2024

Contents

1 Introduction

As autonomous driving advances, self-parking has become a key function in intelligent driving systems. Efficient parking requires vehicles not only to accurately identify parking spaces but also to execute a safe and smooth parking path. This demands a high level of motion control capability, enabling vehicles to navigate complex environments like narrow spaces and obstacles.

Traditional motion control methods, often based on predefined path planning and rule-based control, is more reliable than the learnable planners. However, it is computational intensive and struggling with the uncertainties present in real-world environments. On the contrary, learning-based methods are suitable for complex dynamic scenarios and only require forward propagation for inference, with relatively small computational requirement. However, due to their lack of interpretability and high dependence on data and extensive training, the output actions have a certain of uncertainty.

In that case, hybrid planning methods is becoming increasingly popular in the industrial and researching community. In 2023, Predictive Driver Model (PDM) proposed by Dauner et al. [3] had already justified the feasibility of this approach. In terms of the principle of PDM, the planning trajectory of the first half of the duration is computed by traditional high-performance method Intelligent Driver Model (IDM), and the second half is predicted by a MLP network. This planner achieved rank 1 in the 2023 nuPlan challenge. Inspired by this, we combine advantages of conventional planners and learnable planners, and design a novel motion planner MPC-SAC, which is reliable, stable, and adaptable to the environment.

2 Literature Review

2.1 Model Predictive Control

Model Predictive Control (MPC) is an advanced control strategy that utilizes a dynamic model of the system to predict future behavior and optimize control actions accordingly. By solving a finite horizon optimization problem at each control step, MPC can handle multivariable systems with constraints on inputs and outputs, making it particularly suitable for complex and constrained systems [2]. Its ability to anticipate future events and incorporate feedback makes MPC a powerful tool in various applications, including process control, automotive systems, and robotics [9].

2.2 Deep Q Learning

Deep Q-Learning has played an important role in advancing reinforcement learning by enabling agents to learn optimal policies in high-dimensional state spaces, which is crucial for applications like motion control in autonomous driving. Deep Q-Learning approximates the optimal action-value function using deep neural networks. The key update rule in Deep Q-Learning is expressed as [8]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{1}$$

where s_t is the state at time t, a_t is the action taken, r_t is the reward received, α is the learning rate, and γ is the discount factor. In the autonomous driving, Deep Q Learning can be employed to learn optimal motion control policies by mapping LiDAR and camera sensor inputs, or map data after Simultaneous Localization And Mapping (SLAM). The trained model can be used to control actions like steering and acceleration. This approach allows the vehicle to make decisions that adapt to dynamic environments, enhancing its ability to navigate complex scenarios involving obstacles and narrow spaces [8].

However, Deep Q-Learning's limitation lies in its design for discrete action spaces, making it less suitable for tasks that require continuous control actions, such as precise steering or acceleration in autonomous vehicles. To overcome this, algorithms like **Deep Deterministic Policy Gradient (DDPG)**, **Twin Delayed Deep Deterministic Policy Gradient (TD3)**, and **Soft Actor-Critic (SAC)** have been developed. The interrelation between these algorithms lies in their shared foundation of deep reinforcement learning and their goal of enabling continuous control. DDPG extends Deep Q-Learning to continuous action spaces using deterministic policies. TD3 builds upon DDPG by introducing methods to reduce overestimation and improve learning stability. SAC, while also operating in continuous spaces, differs by using stochastic policies and entropy maximization to enhance exploration. All three algorithms address the limitations of Deep Q-Learning in continuous domains and contribute to the advancement of motion control in autonomous driving by enabling vehicles to learn and adapt to complex, real-world environments.

Deep Deterministic Policy Gradient

DDPG builds upon Deep Q-Learning by combining it with deterministic policy gradients, allowing for continuous action selection [7]. It employs an actor-critic architecture where the actor network outputs continuous actions, and the critic network evaluates them. This approach effectively extends Deep Q-Learning's value function approximation to continuous domains, enabling the learning of precise control policies necessary for smooth vehicle operation.

Twin Delayed Deep Deterministic Policy Gradient

TD3 further refines DDPG by addressing issues like overestimation bias and function approximation errors that can occur in actor-critic methods [4]. It introduces techniques such as clipped double Q-learning, where two critic networks are used, and the minimum of their estimates is taken to reduce bias. Delayed policy updates and target policy smoothing are also implemented to enhance learning stability. TD3's enhancements make it more robust for continuous control tasks, improving upon DDPG's performance in complex environments.

Soft Actor Critic

SAC takes a different approach by incorporating entropy maximization into the reinforcement learning objective, aiming to improve both the exploration and robustness of the learned policies [5]. By maximizing the expected reward while also maximizing the entropy of the policy, SAC encourages more stochastic and exploratory behavior. This is particularly advantageous in autonomous driving scenarios where the environment is dynamic and uncertain, requiring the agent to adapt to unforeseen changes.

3 Methodology

3.1 Fixing Spinning Ego Agent

In previous work, the model did not perform well. It kept spinning in all kinds of parking scenarios. Therefore, we changed the RL algorithm from our own designed DDPG to the Soft Actor Critic (SAC) algorithm in the stable-baseline3 library [10], in order to determine whether there are any problems with the environment, reward, and training process we designed.

Environmental Fine-tuning

Other static vehicles are removed and random goal position is changed to fixed goal for making simulation environment simple to learn. For vehicle model, in order to make the ego agent motion control more flexible, the max steering limit and max steering change per step are both appropriately increased.

Reach Goal Detection

In previous work, this detection was based on reward computation, which means that when the reward exceeds a certain threshold, it will be deemed as successfully completing the automatic parking task, and the environment will be reset. The used reward function was mentioned in proposal and can be written as [6]:

$$R(s,a) = -\|s - s_g\|_{W,p}^{p} - b \cdot \text{collision} \tag{2}$$

where $\|s - s_g\|_{W,p}$ is the weighted p-norm, encouraging the vehicle to move closer to the target spot and align correctly with the desired heading without collision. However, using reward thresholds to determine success heavily relies on the design of the reward function. If the reward function is not accurately or comprehensively designed, it may lead to misjudgment of success. In addition, due to the complexity of the reward design, how to set the threshold is also a problem. In that case, we changed criteria to more direct and simple ones.

$$\|\mathbf{p}_c - \mathbf{p}_g\|_2 < d_{th} \tag{3}$$

$$\mathbf{h}_c \cdot \mathbf{h}_g > \sigma_{th} \tag{4}$$

, where $\mathbf{p}_c$ and $\mathbf{h}_c$ are current position and heading, and $\mathbf{p}_c$ and $\mathbf{h}_c$ are goal position and heading. In addition, d_{th} is position distance threshold and σ_{th} is heading similarity threshold. Practically, we set threshold variances $d_{th} = 15$ and $\sigma_{th} = 0.90$. For goal heading, we set $\mathbf{h}_g = \pi/2$ because all parking lots are vertical in the simulation environment. For goal position, it is initiated in a random parking lot.

3.2 Algorithms Deployment

In new environment, SAC performs well which means problems in simulation environment are solved. In that case, we first **randomly reset goal** for every episode and **randomly generates static obstacle cars** in spaces that are not goal with 0.1 rate. Secondly, we deployed, train, and simulate self-designed DDPG, TD3, MPC-SAC algorithms. We abandon

DDPG-Net described in proposal because even DDPG itself can not converge in this task, and developed the advanced version of DDPG – TD3. Furthermore, we design MPC-SAC planner to combine the advantages of MPC and SAC. All experiments are implemented on NVIDIA RTX 3080 GPU.

Algorithm 1 DDPG Network Architecture

```
 1: Initialize actor $\mu(s|\theta^{\mu})$ and critic $Q(s,a|\theta^{Q})$ network with random weights
 2: Initialize target networks $\mu'$ and $Q'$ with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$
 3: Initialize replay buffer R
 4: for each episode do
 5:     Initialize exploration noise N
 6:     Set initial state $s_1$
 7:     for t = 1, T do
 8:         Process state: if dict, extract 'achieved_goal'
 9:         $x \leftarrow \text{ReLU}(\text{Linear}_{256}(s))$          ▷ Actor: First hidden layer
10:         $x \leftarrow \text{ReLU}(\text{Linear}_{256}(x))$          ▷ Second hidden layer
11:         $a \leftarrow \tanh(\text{Linear}(x)) \times \text{max\_action}$
12:         Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
13:         Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
14:         Sample random minibatch of N transitions from R
15:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}))$
16:         Update critic by minimizing loss: $L = \frac{1}{N}\sum(y_i - Q(s_i, a_i))^2$
17:         Update actor using policy gradient
18:         Update target networks:
19:         $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
20:         $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
21:     end for
22: end for
```

Soft Actor Critic

By implementing SAC from Stable-Baseline, the training pipeline is constructed easily. In terms of key update rule, the equation is presented as [5]:

$$J(\pi) = \sum_{t} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha\mathcal{H}\left(\pi(\cdot \mid s_t)\right)\right] \tag{5}$$

Practically, the SAC model is configured to use the MultiInputPolicy for the environment, with HER Replay Buffer. The replay buffer size is set to 1,000,000, with a learning rate of 0.001, a discount factor of 0.95, a batch size of 2048, and a soft update coefficient of 0.05. The policy network architecture consists of 3 layers, each with 1024 neurons. Learning begins after 10,000 steps, and the training step is set to 300,000.

Deep Deterministic Policy Gradient

For DDPG, the key update rule equation is presented as [7]:

$$\nabla_{\theta^{\mu}} J \approx \mathbb{E}_{s_t}\left[\nabla_a Q(s_t, a|\theta^{Q})\big|_{a=\mu(s_t)} \nabla_{\theta^{\mu}}\mu(s_t|\theta^{\mu})\right] \tag{6}$$

This update is implemented in training iteration, which is written as Algorithm 1. Practically, the HER replay buffer size is set to 1,000,000, with an actor learning rate of 1.0e-4, a critic learning rate of 1.0e-3, a discount factor of 0.99, a batch size of 256, and a soft update coefficient of 0.005. The policy network architecture consists of 2 layers, each with 256 neurons. The training step is set to 1.0e6.

Twin Delayed Deep Deterministic Policy Gradient

For TD3, the key update rule equation is presented as [4]:

$$y = r_t + \gamma \min_{i=1,2} Q_{\theta^{Q_i}} \left(s_{t+1}, \mu_{\theta^{\mu}}(s_{t+1}) + \epsilon\right) \tag{7}$$

This update is implemented in training iteration, which is written as Algorithm 2. Practically, hyper-parameters are same to those of the DDPG.

Algorithm 2 TD3 Network Architecture

```
1:  Initialize actor network $\pi(s|\theta^{\pi})$ and critic networks $Q_1(s,a|\theta^{Q_1}), Q_2(s,a|\theta^{Q_2})$
2:  Initialize target networks $\theta^{\pi'} \leftarrow \theta^{\pi}$, $\theta^{Q_1'} \leftarrow \theta^{Q_1}$, $\theta^{Q_2'} \leftarrow \theta^{Q_2}$
3:  Initialize replay buffer R
4:  **for** each episode **do**
5:      Initialize exploration noise N
6:      Set initial state $s_1$
7:      **for** t = 1, T **do**
8:          Process state: if dict, extract 'achieved_goal'
9:          $x \leftarrow \mathrm{ReLU}(\mathrm{Linear}_{256}(s))$                ▷ Actor: First hidden layer
10:         $x \leftarrow \mathrm{ReLU}(\mathrm{Linear}_{256}(x))$               ▷ Actor: Second hidden layer
11:         $a \leftarrow \mathrm{max\_action} \times \tanh(\mathrm{Linear}(x))$
12:         Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
13:         Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
14:         Sample random minibatch of N transitions from R
15:         // Compute target actions with noise and clipping
16:         $\tilde{a} \leftarrow \pi'(s_{t+1}) + \mathrm{clip}(\epsilon, -c, c)$, $\epsilon \sim \mathcal{N}(0, \sigma)$
17:         // Compute target Q-value as minimum of both critics
18:         $y \leftarrow r + \gamma \min_{i=1,2} Q_i'(s_{t+1}, \tilde{a})$
19:         // Update critics
20:         Update $Q_1, Q_2$ by minimizing $(y - Q_1(s,a))^2$ and $(y - Q_2(s,a))^2$
21:         **if** t mod policy_delay = 0 **then**                    ▷ Delayed policy updates
22:             Update actor by maximizing $Q_1(s, \pi(s))$
23:             Update target networks:
24:             $\theta^{\pi'} \leftarrow \tau\theta^{\pi} + (1-\tau)\theta^{\pi'}$
25:             $\theta^{Q_1'} \leftarrow \tau\theta^{Q_1} + (1-\tau)\theta^{Q_1'}$
26:             $\theta^{Q_2'} \leftarrow \tau\theta^{Q_2} + (1-\tau)\theta^{Q_2'}$
27:         **end if**
28:     **end for**
29: **end for**
```

MPC-SAC

For Model Predictive Control, the key is optimizing the following objective function [2]:

$$\min_{u(k)} \sum_{i=1}^{N_p} \|y(k+i) - r(k+i)\|_Q^2 + \sum_{i=1}^{N_c} \|u(k+i-1)\|_R^2 \tag{8}$$

This MPC Algorithm 3 is integrated into environment to control ego vehicle every 5 steps. For remaining steps, SAC is used to control the ego vehicle.

Algorithm 3 MPC-SAC Architecture

```
 1: Initialize MPC controller with horizon $H$, timestep $\Delta t$
 2: Initialize SAC controller
 3: $step \leftarrow 0$                                   ▷ Track current simulation step
 4: while simulation is running do
 5:     if $step \mod 5 = 0$ then                    ▷ Every 5th step, use MPC for control
 6:         $control \leftarrow$ MPC-Control(current_state, previous_action)
 7:     else                                               ▷ Use SAC for other steps
 8:         $control \leftarrow$ SAC-Control(current_state)
 9:     end if
10:     Apply $control$ to the ego vehicle
11:     $step \leftarrow step + 1$
12: end while
13: function MPC-Control(state, action)
14:     $u_0 \leftarrow$ repeat(action, $H$)          ▷ Initial guess for MPC optimisation
15:     optimal_u $\leftarrow \min_u$ ObjectiveFunction($u$, state) return first action pair from optimal_u
16: end function
17: function SAC-Control(state)
18:     return SAC policy decision based on current state
19: end function
```

3.3 Training Agents Modularization

As we design two algorithms, a base training agent is created to contain implementation of **Hindsight Experience Replay (HER) replay buffer**, exploration **noise** and other abstract methods. HER is a technique that enhances the learning efficiency of reinforcement learning agents in environments with sparse rewards by allowing them to learn from unsuccessful attempts as if they were successful [1]. For the DDPG agent, it adds an Ornstein Uhlenbeck Action Noise (OU noise). While TD3 adds a Gaussian noise. Combining with training script, algorithm is able to be **switched** to train by changing parameters behind python bash running commands.

4 Evaluation

4.1 Experiments Details

In Agent Training Report we provided a TensorBoard interface, but it lacks some important monitoring indicators such as success rate, average step length and average reward. Meanwhile, we remove meaningless distance to goal, episode length, episode reward, and average Q value. We finally complete experiments on the 4 algorithms introduced above. The final monitoring panel results are presented by Fig. 1, Fig. 2, and Fig. 3.

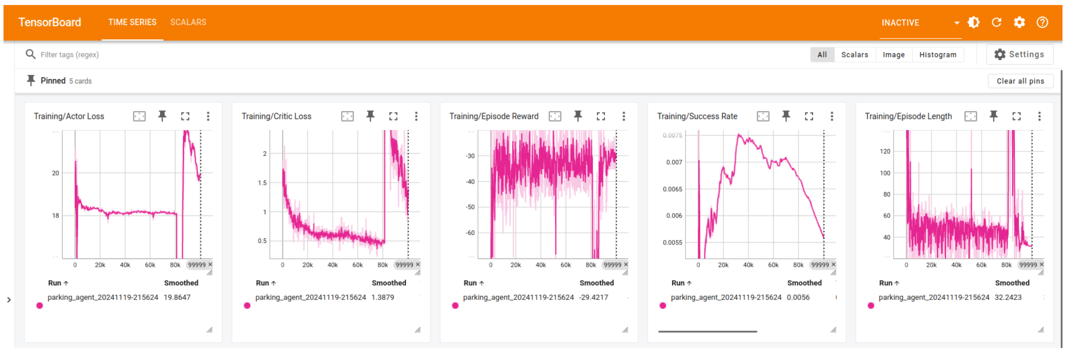


Figure 1: DDPG training indicators are monitored by TensorBoard.

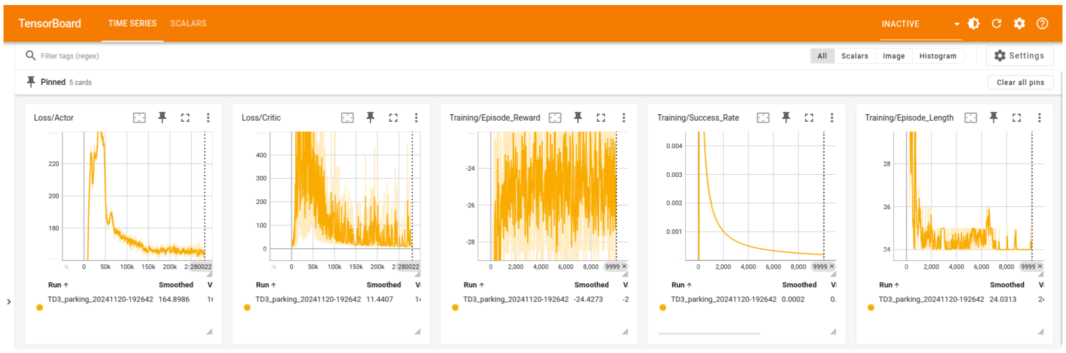


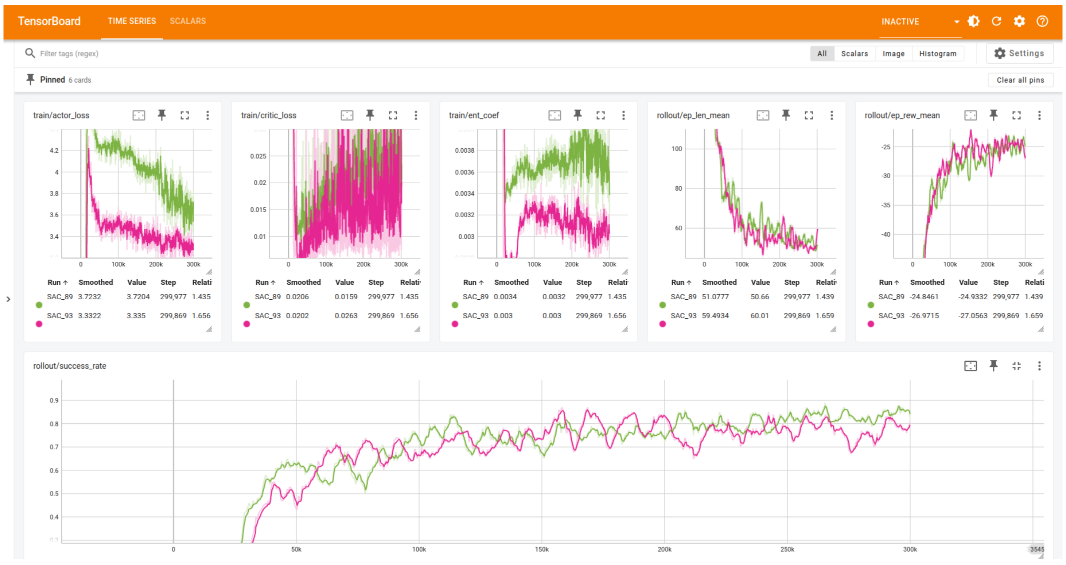
Figure 2: TD3 training indicators are monitored by TensorBoard.

By referring the metrics in these figures, we illustrate the Tab. 1 to compare and evaluate all methods implemented. All data recorded is smoothed average value. As Tab. 1 indicated, DDPG and TD3 are failed to converge in this task and obtain 0 success rate. However, SAC and MPC-SAC converge and our novel MPC-SAC motion planner performs better in success rate. Although loss metrics of SAC are lower, those of MPC-SAC decrease more dramatically.

Furthermore, Fig. 3 shows that the success rate fluctuation of MPC-SAC is smaller than that of SAC. Additionally, MPC-SAC success rate still appears to be on an upward trend, which means that as the number of episodes further increases, the performance gap between MPC-SAC and SAC will also further widen.

Planner	Actor Loss ↓	Critic Loss ↓	Episode Length ↓	Episode Reward ↑	Success Rate ↑
DDPG	diverge	diverge	diverge	diverge	0.00
TD3	diverge	diverge	diverge	diverge	0.00
MPC-SAC (Ours)	3.7232	0.0206	**50.66**	**-24.85**	**0.8414**
SAC Baseline	**3.3322**	**0.0202**	60.01	-27.06	0.7951

Table 1: Training metrics comparison among DDPG, TD3, SAC, MPC-SAC algorithms



train/actor_loss

	Run ↑	Smoothed	Value	Step	Relati
🟢	SAC_89	3.7232	3.7204	299,977	1.435
●	SAC_93	3.3322	3.335	299,869	1.656

train/critic_loss

	Run ↑	Smoothed	Value	Step	Relati
🟢	SAC_89	0.0206	0.0159	299,977	1.435
●	SAC_93	0.0202	0.0263	299,869	1.656

train/ent_coef

	Run ↑	Smoothed	Value	Step	Relati
🟢	SAC_89	0.0034	0.0032	299,977	1.435
●	SAC_93	0.003	0.003	299,869	1.656

rollout/ep_len_mean

	Run ↑	Smoothed	Value	Step	Relati
🟢	SAC_89	51.0777	50.66	299,977	1.439
●	SAC_93	59.4934	60.01	299,869	1.659

rollout/ep_rew_mean

	Run ↑	Smoothed	Value	Step	Relati
🟢	SAC_89	-24.8461	-24.9332	299,977	1.439
●	SAC_93	-26.9715	-27.0563	299,869	1.659

rollout/success_rate

Figure 3: Comparison of training indicators between SAC (pink) and MPC-SAC (green).

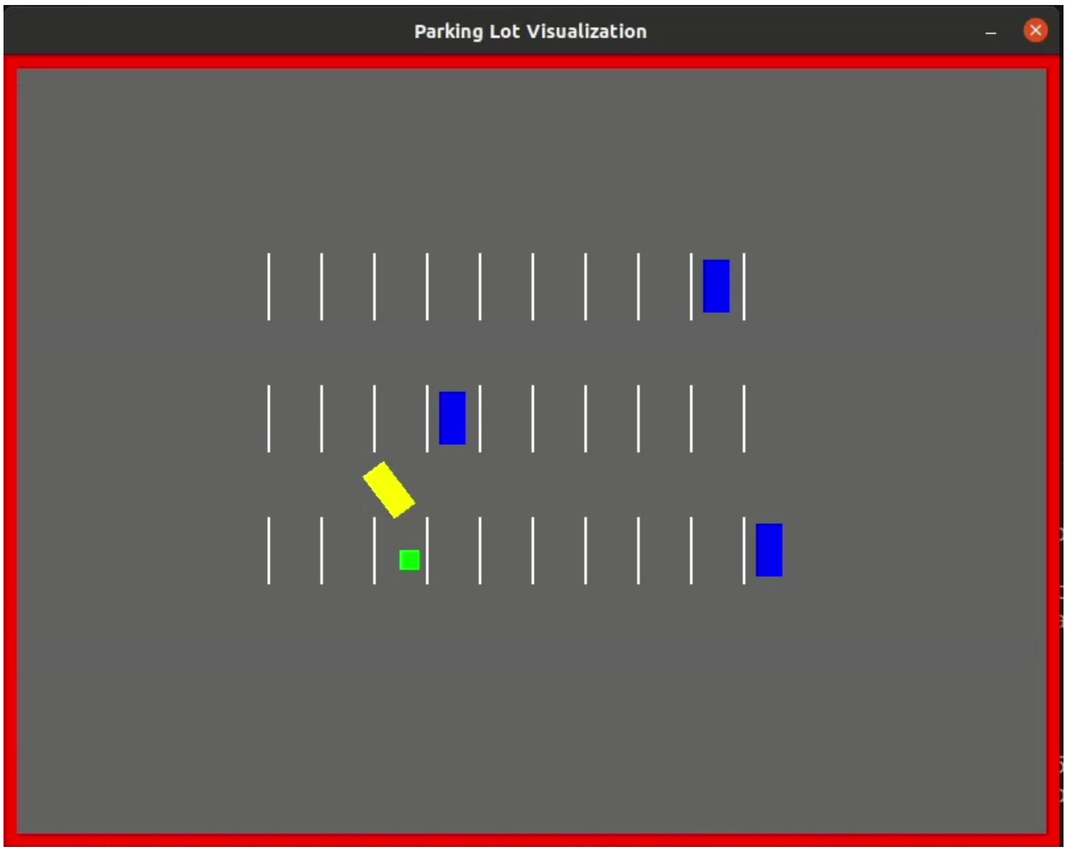


Figure 4: Simulation: Autonomous vehicle is driving forward the goal position

In the end, we also find the speed of MPC-SAC is slower than that of SAC baseline. The addition of MPC requires more computation resource. However, due to the fact that MPC is only used every 5 steps, the speed of MPC-SAC is much faster than using MPC directly. This new method cleverly achieves a **balance** between the planner's **speed** and **accuracy**.

5 Simulations

By loading model and rendering the simulation environment, inference performance are illustrated intuitively. The video is included in the submission fold, and Fig. 4 depicts the scene of yellow autonomous driving vehicle heading towards the green goal position.

6 Conclusion

In conclusion, this work proposes a novel motion planner for autonomous parking called MPC-SAC. By using SAC as the main method and MPC as the auxiliary correction, the performance of MPC-SAC is significantly higher than those of the purely learning-based methods in the autonomous parking task. Specifically, while DDPG and TD3 algorithms are so time-consuming and finally failed to converge, both stable-baseline SAC and our MPC-SAC achieved convergence. Notably, the MPC-SAC planner achieved a higher success rate of 84.14% compared to the SAC baseline's 79.51%, along with **shorter episode lengths** and **improved stability**. The success rate of MPC-SAC also exhibited less fluctuation during training, indicating **enhanced reliability**. This hybrid approach not only improves performance but also contributes to bridging the gap between traditional control methods and modern reinforcement learning algorithms. The code of our project is open-source on GitHub: `https://github.com/ztony0712/5418_autopark_env`.

7 Future Work Discussion

First enhancement could be deployed on environment. Friction might be used and four-wheeled car model would be more realistic than bicycle model. Dynamic obstacles and varying parking lot environment could improve robustness and adaptability of motion planner. Secondly, sensor input or map prior knowledge should be added into observation for making ego agent understand the positions of obstacles. At last, exploring implementations of more kinds of reward functions and learning-based methods is helpful to further improve motion planner.

References

[1] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., Zaremba, W.: Hindsight experience replay. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)

[2] Camacho, E.F., Bordons, C.: Model Predictive Control. Advanced Textbooks in Control and Signal Processing, Springer, London (2007). https://doi.org/10.1007/978-0-85729-398-5

[3] Dauner, D., Hallgarten, M., Geiger, A., Chitta, K.: Parting with misconceptions about learning-based vehicle motion planning. In: Proceedings of The 7th Conference on Robot Learning. pp. 1268–1281. PMLR (Dec 2023)

[4] Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods (Oct 2018). https://doi.org/10.48550/arXiv.1802.09477

[5] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (Aug 2018). https://doi.org/10.48550/arXiv.1801.01290

[6] Leurent, E.: An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env` (2018)

[7] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (Jul 2019). https://doi.org/10.48550/arXiv.1509.02971

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (Feb 2015). https://doi.org/10.1038/nature14236

[9] Qin, S.J., Badgwell, T.A.: A survey of industrial model predictive control technology. Control Engineering Practice **11**(7), 733–764 (Jul 2003). https://doi.org/10.1016/S0967-0661(02)00186-7

[10] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021), `http://jmlr.org/papers/v22/20-1364.html`