Safe-ParkDRL for School Zones

Andy Seoho Yun
University of Washington
andyseo@cs.washington.edu

Xinlei Liu
University of Washington
xllegion@cs.washington.edu

Chanseok Oh
University of Washington
chanoh5@cs.washington.edu

Abstract

Human-to-car interaction for autonomous parking is a research area that has received significantly less attention than car-to-car interactions. However, preventing harm to people is equally, if not more, important than avoiding collisions with other vehicles or structures. To address this issue, we introduce our Soft Actor-Critic (SAC) model, which is a modification of Stable Baselines 3. Our model detects both human-to-car and car-to-car interactions and safely parks the car in the Highway.env simulation. Through this work, we aim to contribute to safe autonomous parking tasks.

1. Introduction

Autonomous driving has experienced great improvement in various areas of the field. However, there are diverse constraints that often discourage people from applying such models. As a result, many companies like Tesla and Mercedes offer autonomous parking as it is subject to fewer constraints. Unfortunately, the data and simulations used for training these models are often challenging to access or confidential. Therefore, it is important to establish a general and universally accepted approach to evaluate the performance of such models. A possible option is open-source simulations but the primary objective of these simulations is often to train car-to-car interactions and ensure the model can park safely without causing damage to any neighboring cars. It is equally, if not more, essential to address human-to-car interactions as well. The agents especially need to understand the dynamics of children's movement and act accordingly. To solve the issues described above, we introduce, the "Parking in a School Zone" problem as a holistic challenge. Here, we added more Human-to-Car features to a well-known open-source simulation, "Highway.env" and trained a Soft-Actor-Critic(SAC) model that captures both human-to-car and car-to-car interactions. To focus on our objective by minimizing the variables to handle, we narrowed down our problem to autonomous parking. In other words, we did not consider the friction of the road, parking lot temperature, the characteristics(brand) of the cars, etc. All of these factors can be incorporated into our model for future research. One major challenge is that there is no theoretical proof that guarantees SAC is the best model for our problem. Although SAC is known to be sample-efficient, stable, and used heavily in practice, it does not guarantee optimal performance mathematically. The other major challenge is that the simulation models the dynamics of the children's movement as a function of a few variables which may not accurately represent actual behaviors.

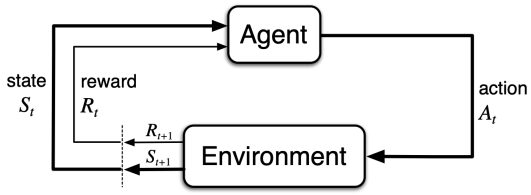


Figure 1. General RL process from Sutton et al.

2. Related work

2.1. preliminaries

As described in Figure 1, Reinforcement Learning (RL), a machine learning framework of Markov Decision Process (MDP), trains an agent to perform actions, denoted as A, in a state, denoted as S, that rewards the agent. Then, a ground truth transition function, denoted as T, leads the action to the next state. However, in a highly complicated environment like autonomous parking, obtaining such a transition function is nearly impossible. Moreover, classical methods like value and policy iterations are not feasible due to continuous state and action spaces.

Fortunately, this is not the end of the story. Deep Reinforcement Learning (Deep RL) utilizes deep neural networks to approximate the system and obtain meaningful parameters of environment and policy, leading to significant advancement in handling complex states and actions. There are two main different methodologies in Deep RL: Deep Q-Learning and Policy gradient. Deep Q-learning [7] extends the principles of (traditional) Q-learning and its

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

Figure 2. Algorithm of SAC from Harrnoja et al.

convergence property and applies them to continuous state spaces. Essentially, instead of Q-tables, the neural networks approximate the Q-values for diverse actions, A_i. There are many descendants of this approach like DPG (Deep Deterministic Policy Gradient) [11] and DDPG (Deep Deterministic Policy Gradients) [5]. The other approach is Policy Gradient [12] which approximates the likelihood ratio gradient and uses bootstrapping to estimate the value function for policy updates. The descendants of this family include actor-critic [6] and PPO (Proximal Policy Optimization) [10].

2.2. Soft Actor-Critic(SAC)

To address these drawbacks, Haarnoja et al. introduced the Soft Actor-Critic (SAC), an off-policy, maximum entropy, actor-critic algorithm derived from DDPG. This algorithm effectively resolves both the instability of the DQN-based models and the sample inefficiency of the policy gradient-based models.

Many of these models from both methodologies described above possess two major challenges. Firstly, DQN-based models, which have a deterministic policy, often yield highly unstable results when the environment gets overly complicated. Secondly, policy gradient-based models are on-policy, meaning they require new samples to be collected for each gradient step. On top of everything, both families of models are extremely sensitive to hyperparameters so a lot of resources are required to get the right set of hyperparameters for a specific task. To mitigate all of these drawbacks, Haarnoja et al. proposed Soft Actor-Critic(SAC), a DDPG-descended off-policy maximum entropy actor-critic algorithm, solves both challenges [2].

To do so, SAC has a slightly distinct objective function. Unlike classical reinforcement learning, the objective of SAC is to get the maximum policy that satisfies

$$\sum_t \mathbb{E}_{(s_t,a_t)\sim\rho}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{1}$$

where α is the temperature parameter and $\mathcal{H}(\pi) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log(\pi(a|s))]$(we intentionally excluded the discount factor γ to make the equation simple as done in Haarnoja's [3]). By including an entropy term, the maximized policy is able to adapt stochasticity and explore more effectively, since a uniform distribution yields the highest entropy.

The benefit of SAC is that it can handle both discrete and continuous settings. Since autonomous driving has both continuous state and action spaces, we use the continuous version and approximate the Q-function and the policy.

To simplify the process, we consider the soft Q function that depends on θ(in practice, often two soft Q functions are implemented to prevent overfitting) and the tractable policy π that depends on ϕ. Here, each soft Q-function is modeled using a neural network that processes both states and actions from a trajectory.The policy is also modeled as a Gaussian distribution. The mean and covariance of this dissolution are determined by inputting only states into a different neural network. Then, the mean and covariance are often transformed using hyperbolic tangent to get meaningful intervals [3].

Mathematically, the Soft Q-function parameters can be trained to minimize the soft Bellman residual:

$$\mathbb{E}_{(s_t,a_t)}[\frac{1}{2}(Q_\theta((s_t, a_t) - (r(s_t, a_t) + \gamma\mathbb{E}_{s_t}[V_{\bar{\theta}}(S_{t+1})])^2] \tag{2}$$

where $\bar{\theta}$ is the moving average of the soft Q weights.

The policy parameters can be learned by directly minimizing the expected KL-divergence, however since the derivative of KL-divergence is difficult to deal with, we use the surrogate of it instead.

$$\mathbb{E}_{s_t}[\mathbb{E}_{a_t\sim\pi}[\alpha\log(\pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t)] \tag{3}$$

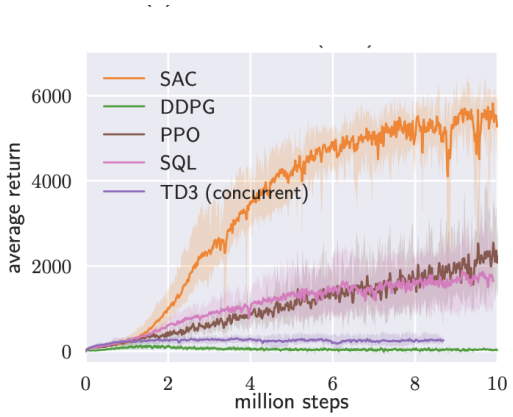
Also, Haarnoja used the empirical way to optimize the temperature parameter, α [3].

$$\mathbb{E}_{a_t\sim\pi_t^*}[-\alpha_t\log(\pi_t^*(a_t|s_t;\alpha_t) - \alpha_t\bar{\mathcal{H}}] \tag{4}$$

Comparing these elements, the complete Soft Actor-Critic (SAC) algorithm is detailed in Figure 2. Empirically, as shown in Figure 3, SAC (represented by the orange line) demonstrates the fastest increase and convergence, along with a superior average return in complex simulations such as Humanoid (rllab). This result is supported by the SAC algorithm, which is more sample-efficient, stable, and adaptable in complex environments.

2.3. Autonomous parking

There have been numerous approaches to solve autonomous parking. Among many approaches, the one developed by Özeloğlu et al [13] stands out. Their approach uses two neural networks: one network that determines a



(f) Humanoid (rllab)

Figure 3. Performance of SAC from Moreira et al.

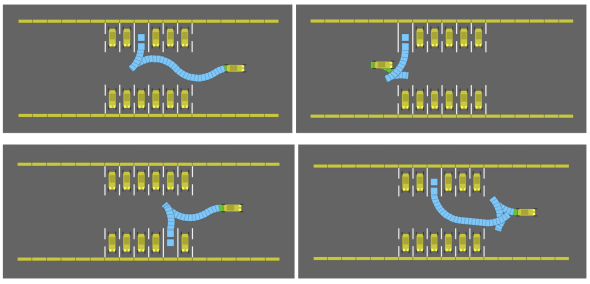


Figure 4. Performance of SAC from Haarnoja et al.

parking slot and another that moves the vehicle. However, as the models struggle to adapt the dynamic nature of object detection and environment stochasticity, there may be scenarios where the two networks do not align well. Simply, whenever the vehicle moves, the vision model needs to readjust accordingly. On the other hand, our approach (with a vision architecture) can leverage the relationship between the critic and the actor to learn more effectively, even in real-world scenarios.

2.4. Simulation: Highway.env

This is one of the most common simulations for autonomous parking and driving tasks due to its optimized features [4]. Previously, Moreira et al. implemented the SAC model, along with DDPG and TD3, in highway.env [8]. Similar to their approach, the agent manages the relative positions of objects around it and acts based on the policy, π in our simulation. It is important to note, as highlighted in our introduction and illustrated in Figure 4, that their research is concerned with only car-to-car interactions and kinematics involved in vehicle controls. In contrast, our study encompasses both car-to-car and human-to-car interactions and therefore presents more realistic scenarios by adding more dynamic objects (which are further described in the Methods section).

2.5. Stable-Baseline3

Stable Baseline-3 is an extension of Stable Baselines and is highly compatible with Highway.env [9]. Also, it has been optimized for the SAC algorithm for both continuous state and action spaces. Specifically, the SAC version that is implemented here uses a double Q-learning trick which was derived from Haaronja et al [3] and TD3 [1]. More importantly, this architecture corresponds well with most current Gymnasium simulations, including Highway.env.

3. Methods

We built upon highway.env's existing parking.env implementation for a parking scenario(parking-v0). The original implementation includes only pre-existing parked vehicles and the agent vehicle controlled by the model. For our primary objective, we defined two new types of RoadObjects called "Child" and "Adult". It is important to note that some of the following parameters are subject to tuning and will soon be available in our GitHub repository.

3.1. Child Object

These are implemented as a collidable and solid RoadObject with a length of 1 meter and a width 1 meter. A "Child" object has a constant speed of 3 m/s and a randomly initialized heading, indicating the direction the child is going. For every time step of the simulation, we update the position of the "Child" objects according to their heading and speed to simulate the dynamics of a running child in a school zone.

3.2. Adult Object

These are implemented as a collidable and solid RoadObject with length 1.5 meters and width 1.5 meters. An"Adult" object has a speed of 1 m/s and a randomly initialized heading, indicating the direction the adult is going. For the lifetime of the simulation, an "Adult" object will not change in heading and speed for simplicity. We make this assumption that a walking adult is more predictable than a running child in school zones. For every time step of the simulation, we update the position of the "Adult" objects according to their heading and speed.

3.3. Model Vehicles

The model-controlled vehicle has a length of 5 meters, a width of 2 meters, and a maximum speed of 40 m/s. It has the following:

State Space: this consists of the location of the desired parking spot, and all Child, Adult, and Vehicle objects in

the simulation. The sizes, positions, headings, and speeds of all objects are provided to the model. Note that this is different from real-life scenarios, where views of objects' movements may be obstructed and not all information will be available.

Action Space: the controlled vehicle has two parameters that the model can change: steer, which affects the heading, and acceleration. Therefore for our controlled vehicle, we have a continuous action space with two ranges: a steering range between -45 and 45 degrees, and an acceleration range between -5 to 5 m/s^2

3.4. Reward Function

The reward is calculated using a p-norm (default p = 0.5). We calculate our reward using the achieved goals and desired goals. The achieved goals will be a vector $(x_c, y_c, vx_c, vy_c, dx_c, dy_c)$ which represents the position of the controlled vehicle in x and y, velocity of the controlled vehicle in x and y, and the direction of the controlled vehicles' heading in x and y. The desired goals will be a similar vector representing the target position, desired velocity when parking, and the desired direction of the controlled vehicles' heading. Let the list of achieved goals be g_a, the list of desired goals be g_d, the list of weights be w. Then the reward r is calculated as

$$r = -\sqrt{|g_a - g_d| \cdot w} \tag{5}$$

We also penalize collisions of cars with humans and cars. When the controlled vehicle hits a "Child", "Adult" or parked vehicle object, we add a penalty term p_c:

$$r = -\sqrt{|g_a - g_d| \cdot w} + \mathbb{1}\{\text{Collision}\}p_c \tag{6}$$

Since we do not want the car to stay in the original position, we add a penalty p_s when the car's velocity is below a certain threshold.

$$r = -\sqrt{|g_a - g_d| \cdot w} + \mathbb{1}\{\text{Collision}\}p_c + \mathbb{1}\{\text{Stationary}\}p_s \tag{7}$$

The reward function is subject to hyperparameter tuning. Currently, the following values are used:

$$w = (1, 0.5, 0, 0, 0.0001, 0.0001)$$

$$p_c = -10$$

$$p_s = -0.5$$

Essentially, the vehicle's position is most important, not the velocity of the vehicle when parking.

4. Experiments

The goal of our experiment is to achieve significant performance in safely parking our car in more challenging environments. To achieve this goal, we broke our experiments down into three steps. First, we trained a Stable Baselines 3 model on an environment without obstacles. Then, using the same hyperparameters, we trained a baseline model in an environment with cars. Lastly, we trained a model on an environment with both cars and the "Adult" and "Child" objects. We then applied both the baseline model trained from above and training from scratch.

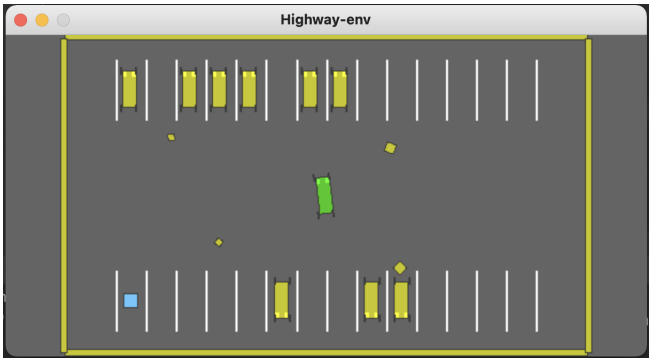


Figure 5. One possible starting state for the simulation. The green rectangle represents the controlled vehicle, big yellow rectangles represent the other parked vehicles, small yellow squares represent Child objects, medium yellow squares represent Adult objects, and the blue square represents the desired parking spot.

Figure 6 displays the results of these experiments. The rewards have been increasing with the number of time steps, indicating that both model with only parked cars and the model with both parked cars and human objects have learned not to crash into people. However, the model with parked cars and the model with both cars and human objects did not exhibit higher success rates. We suspect multiple factors contribute to this phenomenon.

The first is the bottleneck of computational resources. Most Reinforcement Learning models require millions of time steps, but we capped our time steps at 100,000 due to limitations of computational resources. Another limitation is the extent of hyperparameter tuning. Owing to computational resource constraints, we performed manual tuning of the reward function and learning rates, instead of using an extensive grid search. With more computational resources, we anticipate that experimenting with more hyperparameters would enhance our model's success rate.

5. Discussion

For future experiments, one improvement we plan to make to the simulation environment is to upgrade the dynamics of objects and the environment. Here, the children and adult objects move at a constant speed; we could add sudden stops and change the velocity depending on the time and position of the objects. Additionally, we did not consider factors like the friction of the road, parking lot temperature, and the characteristics (brand) of the cars. Adding

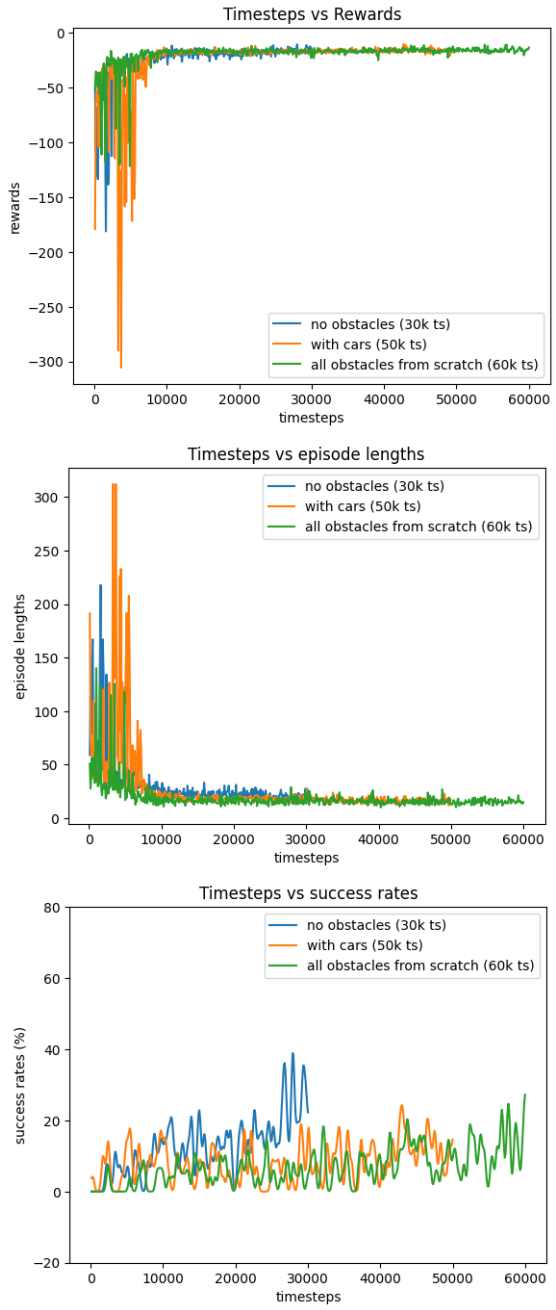


Figure 6. Performance of model with various environments

these new features can create more realistic settings.

The second modification is to add more human and car objects into the environment during training. During some training episodes, we noticed that the environment had significant open areas between the controlled vehicle and the desired parking spot, which made the model's learning less efficient since the agent would not crash into human objects and cars when acting as if there were no human objects or cars.

Another potential improvement is modifying our reward function. Currently, when we set the heading difference term in the reward function, the model learns to go directly to the parking spot, then fails to maneuver and park the car in the correct direction to crashes. This is because the current reward function only values forward parking. In reality, one might want to park backward as well. We could change the function to allow both parking straight and backing into the parking spot. We also should differentiate the collision penalty for crashing into human objects and crashing into cars.

6. Acknowledgements

We would like to thank Aaron Walsman for offering both academic and emotional support. His invaluable feedback helped us overcome the tough learning curve of deep RL. We are also deeply appreciative of the guidance and resources provided by Professor Ali Farhadi and Aditya Kusupati for this experiment.

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. 3

[2] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. 2

[3] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019. 2, 3

[4] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018. 3

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. 2

[6] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. 2

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. 1

[8] Dinis Moreira. Deep reinforcement learning for automated parking. 3

[9] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. 3

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. 2

[11] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. 2

[12] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, May 1992. 53k Accesses, 3680 Citations, 42 Altmetric. 2

[13] Alican Özeloğlu, İsmihan Gül Gürbüz, and İsmail San. Deep reinforcement learning-based autonomous parking design with neural network compute accelerators. *Concurrency and Computation: Practice and Experience*, 34(9):e6670, 2022. 2