Automated Parking with Proximal Policy Optimization

1st Lijiao Wang
Khoury College of Computer Sciences
Northeastern University
Boston, US
wang.liji@northeastern.edu

1st Ryan M. Dunn
Khoury College of Computer Sciences
Northeastern University
Boston, US
dunn.rya@northeastern.edu

1st Meet M. Katrodiya
Khoury College of Computer Sciences
Northeastern University
Boston, US
katrodiya.m@northeastern.edu

***Abstract*—Full self driving systems are not complete without automatic parking capabilities, but parking poses unique challenges requiring flexibility and precision. This study utilized proximal policy optimization to train a vehicle agent to park in a simulated parking environment while avoiding occupied spaces. The observation and action spaces in the simulated environment were both continuous. After training with relatively little computational power the agent was able to park at a rate of 92% with a stochastic policy and 95% with a deterministic policy.**

***Index Terms*—Self-driving, Proximal Policy Optimization, Neural Network, Gradient Descent.**

I. Introduction

A. Problem Statement

Parking poses unique challenges to drivers, requiring precise maneuvering in limited space and creativity navigating through spaces without strict road guidelines. Automated parking is mandatory to achieve full end-to-end self driving (FSD) but is often viewed as a convenience due to decreased danger level relative to road navigation. The demand for improved automatic parking systems will increase alongside FSD adoption rates. Automated parking systems also open the door for collaborative parking agents, in contrast to the traditionally competitive nature of parking.

This project seeks to develop a self-driving parking algorithm in a simulated environment that is extensible to multi-agent collaboration.

B. Related Works

Simulating traffic scenarios in parking lots presents unique challenges due to the complex, low-speed interactions and the presence of vulnerable road users such as pedestrians. Several recent studies have attempted to address this issue using reinforcement learning (RL)-based approaches.

A closely related open-source project is described in [1], which proposes a distributed algorithm for resolving cooperative multi-vehicle conflicts in highly constrained parking lot spaces. The authors formulated the problem as a multi-agent reinforcement learning task and simulated cooperative parking behavior for up to four autonomous vehicles on a single parking lane. While this work makes a significant contribution, it is limited to scenarios where all agents are actively searching for parking spaces. More realistic and diverse situations—such as vehicles exiting parking spots, bypassing others, or navigating among parked vehicles—are not considered. The parking component of `highway-env` has also been utilized to evaluate the effectiveness of Deep Deterministic Policy Gradient (DDPG) and the Soft Actor Critic (SAC) algorithm for training parking agents [2].

In our project, we focus on training a **single-agent** parking policy using the HighwayEnv library [3], specifically its `parking-v0` environment [4]. This environment provides a simplified yet flexible framework for autonomous parking tasks, supporting continuous control of steering and acceleration and easy customization of parking layouts. Based on this foundation, we implemented a training framework using Proximal Policy Optimization (PPO) to simulate and optimize the parking behavior of an individual autonomous vehicle. Extending this to a multi-agent setting remains an important direction for future work.

II. Methods

A. Environment

The simulated environment must meet the following requirements: kinematic simulation of a controlled vehicle, random generation of obstacle vehicles, and collision detection. Furthermore, compatibility with Gymnasium [5] is desirable to facilitate a machine learning interface and promote extensibility.

After initially developing a custom environment framework, an existing environment was selected that met the environment requirements as-provided. The `highway-env` environment [3] includes multiple simplified driving simulation including a parking simulation and was built around Gymnasium compatibility. The environment uses pixels as a primary distance unit, and seconds as the time unit. Kinematic time steps (Δt) are derived from the adjustable frequency of the simulation.

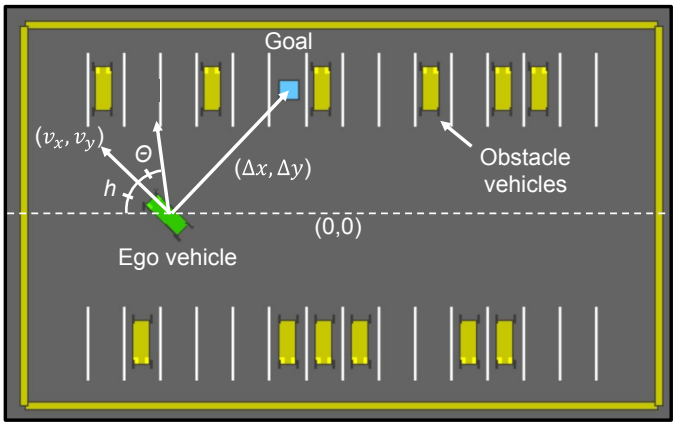

Fig. 1. Labeled visualization of the simulated parking environment. θ is the steering action value, presented by the angular offset between the vehicle's heading and the tire direction.

B. Action Space

The ego-vehicles action space consists of a continuous acceleration state state and a continuous steering state. The range of these actions spaces are [-1, 1] and are mapped to acceleration and steering ranges during kinematics simulation. The selected range is [-5, 5] (units/s^2) and the selected steering range is $[-\pi/4, \pi/4]$ radians (a 90 degree cone.) The heading (h) of the vehicle is updated based on velocity and steering action according to the formula:

$$h' = h + v\sin(\beta)/N \quad \beta = \arctan(\tan(s)/2) \quad N = L/\Delta t$$

where s, L, and v are the scaled steering value, length, and velocity of the vehicle respectively. Because θ is the angular offset between the vehicle's heading and the "tire" heading, velocity is updated according to the formula $v' = v + \Delta t * a$.

C. Observation and Reward

While fully realized SD systems rely on sensor data such as LiDAR, our simulated environment is fully observable. The agent has access to it's position, velocity, and heading, along with the position of the goal. All non-goal parking spaces were treated as occupied in the environment which simplifies the observation space but has drawbacks discussed in Section V-B. The original observation space of the ParkingEnv class included in the `highway-env` package did not have obstacle vehicles in the observation so this optional functionality was added via our CustomParkingEnv subclass of ParkingEnv.

The reward function $R(obs)$ has three weighted components: A collision penalty c, a success reward s, and a heuristic r representing goal proximity in terms of position, velocity, and heading. Episodes are terminated when c is applied, and terminated as a success when s is applied. r is applied at every timestep and is always negative. Accumulation of negative rewards encourage the agent to complete the episode in fewer timesteps.

Two methods for calculating r were used. The first method is simply the weighted sum of euclidean velocity, euclidean distance to goal, and $|\cos(\theta)|$. $\theta = 0$ represents a heading perpendicular to the parking spaces meaning that either vertical vehicle orientation maximizes reward.

$$R_1 = w_v\sqrt{v_x^2 + v_y^2} + w_p\sqrt{\Delta x^2 + \Delta y^2} + |\cos(\theta)| + s + c$$

The second and more effective reward function R utilized the weighted p-norm between the vehicle's state and the goal state:

$$R = ||w(obs - goal)||_{0.5} + c + s$$

A weighted p-norm with $p < 1$ is used to create a reward "spike" near the success state. The elements of obs and $goal$ are $[x, y, v_x, v_y, cos(\theta), \sin(\theta)]$. The ultimate weight selection is discussed in Section III and Section V.

D. Model Selection

In this project, We used Proximal Policy Optimization (PPO) [6] to train an autonomous parking agent. Although our course introduced Deep Q-Networks (DQN), DQN is limited to discrete action spaces, which makes it unsuitable for tasks like vehicle control that involve continuous actions, such as steering angles and acceleration. In contrast, PPO supports both discrete and continuous action spaces, making it a better fit for autonomous driving applications.

PPO is a type of policy gradient algorithm that improves the policy directly using gradient ascent. It introduces a clipped surrogate objective function that ensures training stability by preventing overly large policy updates. This helps balance learning performance and stability, which is especially important in real-world-inspired tasks.

Additionally, PPO is sample-efficient: it reuses collected trajectories across multiple epochs using mini-batch stochastic gradient descent. This makes it ideal for complex control tasks such as parking, where sample efficiency can significantly reduce training time.

Given its compatibility with continuous control, training stability, and wide adoption in autonomous driving scenarios, PPO was a natural choice for building a robust and realistic autonomous parking model.

III. Training

We trained a single-agent automatic parking policy using Proximal Policy Optimization (PPO) from Stable Baselines3 library [7]. For better performance, the training process was parallelized across separate environment instances with SubprocVecEnv which creates a separate process for each environment.

The training process required tuning a number of hyperparameters related to both the environment and the model.

A. Environment Parameters

Here, we discuss some environment parameters that were tuned:

- **collision_reward:** Negative reward when an agent crashes into other parked vehicles or the boundary walls. This encourages safe driving.
- **reward_weights:** A list of 6 weights denoting the contribution of different components in the total reward $(\Delta x, \Delta y, v_x, v_y, \sin\Theta, \cos\Theta)$. A lower weight was used for the velocity to encourage the agent to move.
- **simulation_frequency:** Frequency at which the simulation physics and rendering are updated. Higher values yield smoother simulations.
- **action_frequency:** Number of decisions the agent makes in a second. The ratio of simulation frequency to action frequency represents how many timesteps of the simulation pass before the actor makes a new decision. Action frequency cannot exceed simulation frequency.
- **static_vehicles:** Number of parked vehicles in the environment. Used to create static obstacles and simulate realistic parking scenarios.

The initial value for collision reward was -5, and 0 and -10 were also tested. Setting the collision reward to 0 was meant to encourage the agent to explore without fear of penalty but that issue was better solved by increasing the entropy value, forcing exploration directly. Ultimately the collision penalty of -10 was used to prevent the agent from quickly crashing to mitigate accumulation of negative reward at each timestep.

Several combinations of simulation and action frequency were tested to understand the tradeoff between model performance and training efficiency. Allowing the agent to make a decision at every timestep increased the computational cost of the algorithm without significant improvement in agent effectiveness. The ratio of 3:1 timesteps per action was found to be a suitable tradeoff. The simulation frequency primarily affected the relationship between the action space ranges and the kinematic simulation. Any value below 1 Hz made precise control impossible, and any value below 10 Hz produced choppy visualizations. Ultimately a 15 Hz simulation frequency with a 5 Hz action frequency produced suitable and stable results. With more computation power there should be no drawback to using much higher frequencies.

Reward weight and static vehicle parameter selections are both discussed in Section V-A and II-C.

B. PPO Hyperparameters

The following PPO hyperparameters were considered during training iterations and evaluated against agent success rate.

- **n_envs:** Number of parallel environments used during training. The environment run in parallel to increase efficiency.
- **batch_size:** Total number of transitions (state-action pairs) collected before each update to the policy. Larger batches lead to stable gradient estimates.
- **n_steps:** Number of steps each environment runs before synchronization. The effective batch size becomes n_envs × n_steps.
- **learning_rate:** Controls the steps size at which model paramters are updated. Lower value prevents overshooting.
- **timesteps:** Total number of steps an agent takes in the environment during training. A large value helps in better generalization.
- **gamma:** The discount factor for future rewards. A lower value helps in short episodes to encourage the agent to maximize immediate rewards.
- **epochs:** Number of training iterations for each mini batch. A lower value (5 or 10) makes training faster and suffices for this task.
- **ent_coef:** Entropy coefficient to use during loss computation. A higher value promotes exploration and helps in escaping local optima.
- **policy_kwargs:** The architecture and activation function used in policy neural network. A compact architecture is sufficient for this task.

TABLE I
HYPERPARAMETER CONFIGURATIONS

Hyperparameter	Value
collision_reward	-10
reward_weights	[1, 0.3, 0.00, 0.00, 0.02, 0.02]
action_frequency	5
simulation_frequency	15
n_envs	8
batch_size	64
n_steps	batch_size * 30
learning_rate	0.001
timesteps	6,000,000
gamma	0.95
ent_coef	0.005
epochs	5
policy_kwargs	{net_arch: [64, 64], activation_fn: tanh}

PPO utilizes several additional hyperparameters that were kept at the default values outlined in [7]. Gamma was reduced from a default value of 0.99 to 0.95 based on episode length observations during initial training. Successful episodes using a 15 Hz simulation frequency take less than 20 timesteps meaning rewards would only ever be discounted to 80% at most over the entire episode. A gamma of 0.95 allows that value to reach 35%. If simulation frequency was raised, gamma must also be raised appropriately.

One often important hyperparameter is the clipping factor which limits how much a model parameter can change at each update. The default clipping factor of 0.2 was deemed effective by monitoring the clipping rate during training. If the clipping rate ever remained at 0 or 1 during training it may have been necessary to raise or lower the clipping parameter to allow only the potentially unstable changes to be clipped.

Table I shows the values of hyperparamters were used in the most successful training run discussed in Section IV.

The training was performed completely on the CPU. Models were saved at the end of each training for evaluation or continuing training. Logging was performed using TensorBoard to monitor the learning curve for success rate. Furthermore, the Monitor wrapper [7] was used during training to gather episodes length and rewards in all parallel environments.

To measure the progress of training, we chose the success rate as our primary metric. Training was considered complete when the success rate qualitatively leveled off. Due to the similarity between our training and evaluation environment, training success always closely matched evaluate success.

IV. Results

A. Training

Plots for reward over time and episode length over time were used to monitor the agent's behavior during training. The figures in this section were produced using data from successful training, the logs of which are available in the repository in Section VII

Reward Over Time — This plot shows the rolling average of episode rewards throughout the training process. Each point represents the average over the window of previous 50 episodes. This helps to smooth out the curve.

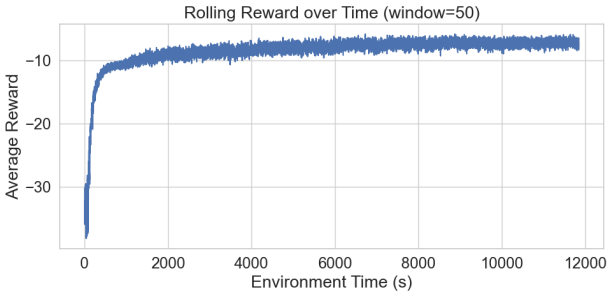

Fig. 2. Mean episode reward throughout the training. The rewards increase fast initially and then slows down.

Figure 2 shows that the agent receives high negative rewards initially as it explores the environment without any optimal policy. These extreme negative values are caused by the accumulation of negative reward while the vehicle remains mostly stationary due to the acceleration range centering at zero. Over time, the reward increases and stabilizes as the agent learns from the experiences. Occasional dips in the rewards may suggest difficult scenarios or random exploration, but the overall upward trend continues suggesting improved performance.

Episode Length Over Time — Figure 3 depicts the length (number of steps) of each episode throughout the training. As with reward, the values are rolling averages to observe change in how long it takes the agent to finish an episode.

Early in the training, the episode lengths are very high as the vehicle is not moving fast and episode is truncated due to time limit. An early dip occurs as the agent begins exploring without an optimal policy, resulting in short episodes due to crashes. Finally, as the agent learns optimal policy to park successfully, the length reaches plateaus. The episodes towards the end are longer than crashes suggesting quick parking and purposeful behavior.

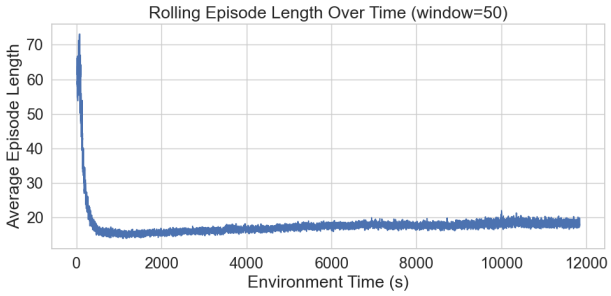

Fig. 3. Mean episode length throughout the training. The episode length is higher initially, then drops to minimum, and finally reaches plateaus a bit higher than minimum

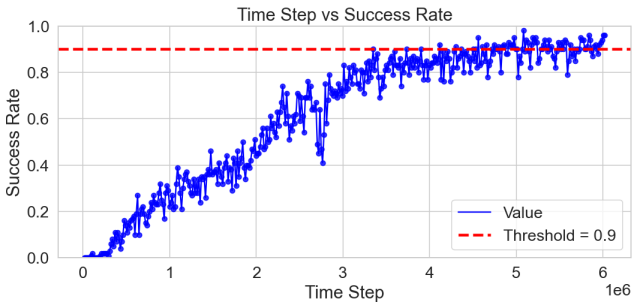

Fig. 4. Mean agent success rate of each update batch for 6 million timesteps of training. The success rate gradually increased before beginning to plateau around 92%.

Figure 4 illustrates the agent's success rate over the course of 6 million timesteps. Initially, the success rate is close to zero, as the agent explores the environment with little understanding of effective strategies. As training progresses, particularly after the first 1.5 million steps, the success rate begins to rise significantly, indicating that the agent is learning to complete the parking task successfully. Between 1.5 and 3 million timesteps, the agent experiences a phase of rapid improvement. After approximately 3 million steps, the success rate starts to plateau around 92%, suggesting that the agent has developed a reasonably stable and effective policy. Minor fluctuations in the curve are due to continued exploration and variability in the environment, but the overall trend confirms steady learning and convergence toward optimal behavior.

B. Evaluation

We evaluated the agent trained with PPO over 1000 test episodes using a custom evaluation script. The purpose was to measure how generalized the learned policies are.

The policies can be used in two ways: Deterministic and Stochastic. Deterministic policies uses the most probable action given a state. This provides more stable but less exploratory results. Stochastic policies, on the other hand, samples the next action from the probability distribution learned during training. This provides more robust results.

A suitable metric to measure the evaluation performance is the proportion of episodes where the agent parked successfully. With models like PPO that generate stochastic policies, using deterministic policies leads to better performance. We observed the same during our model evaluation. The success rate was 95% using deterministic policies and 92% using stochastic policies.

In our evaluation script, we also provide a way to record the test episodes in GIF format. If RECORD is set to True, the GIFs for test episodes are stored in res directory. Some sample GIFs from our test run can be found on our GitHub repo linked in section VII.

V. Conclusions

A. Challenges

A number of challenges were encountered while investigating the model's sensitivity to environmental parameters and PPO hyperparameters. Often times the challenge was addressed by returning the parameter to a default value or mitigating the effect of the hyperparameter by either disabling a feature of the model or "zeroing out" the parameter.

Penalizing velocity — when the reward weight for velocity (w_v) was set to any significant value (e.g. $w_v = w_{\Delta p}$) episodes would mostly truncate at the timestep limit and training progress would not be made. It is likely that the policy was adverse to moving at all because of the velocity penalty. Training progress was made when w_v was set so low that it was functionally zero. As a result w_v was set to 0 for the remainder of training. There are other possibilities to address this obstacle, including restructuring the reward function so that the velocity penalty only has significant impact when the ego vehicle is close to the destination. Feeding human generated reference training data could also help overcome the policy's initial aversion to moving.

Wobbling steering — during evaluation we observed that even a relatively well trained agent would often rapidly alternate steering directions leading to behavior that appears erratic. While the behavior is inconsistent with human driving, in the simulation it simply acts as a speed retardant by causing the vehicle to oscillate around a straight path. A more realistic kinematic model would make this behavior harder by preventing the wheel position from instantaneously changing. Alternatively, the behavior could be penalized directly by applying a negative reward for large changes in steering value.

Stable training with poor results — Early iterations of training would converge on a stable policy that attempted to park but never fully succeeded. We observed that the vehicle would stop parallel to the goal but fail to adjust its lateral position consistent with the motion required for parallel parking. This issue was exacerbated when running the policy in a deterministic mode. In order to "parallel park" the vehicle must first pull away from the destination before approaching from a more favorable angle but the penalty for pulling away from the destination was too large. Increasing the success reward and adjusting the $p-$value of the reward function helped but ultimately increasing the entropy co-efficient from $1e{-}4$ to $1e{-}3$ eliminated this issue.

B. Future Work

Several iterative improvements are proposed to extend the model described in this study. The environment extensions and training classes were developed with an object oriented approach to promote re-usability.

Observation extensions — As mentioned in Section II, the observation space was simplified by treating all non-goal spaces as occupied for purposes of collision and omitting the position of the other vehicles from the observation. This simplification dramatically reduced computation time by eliminating the need to run kinematic simulations on dozens of vehicles that should not move. However, omitting obstacles from the direct observation means that the agent will be trained to avoid parking spaces but cannot learn to avoid obstacle vehicles that are moving or outside of a space. With additional computation power or time, the obstacle vehicles could be added back to the observation.

Generalized layouts — The training in this study was performed on a single parking lot layout consisting of two lanes of parking spaces. Because the observations are relative, the trained agent should not be sensitive to random starting locations but would be sensitive to different parking lot layouts. Adding obstacles to the observation space and training the agent on randomized *layouts* should produce a generalized agent capable of navigating arbitrary layouts.

Multi-Agent Parking — We have developed a prototype model for multi-agent parking, where several controllable vehicles attempt to park simultaneously. This setup introduces additional complexity into the training environment, as agents must coordinate while avoiding collisions and competing for limited space. Our initial experiments did not yield satisfactory results — training was unstable and policies failed to converge to effective behaviors. As future work, we intend to explore better hyperparameter configurations and redesign the model architecture to enhance multi-agent coordination and learning.

VI. Contributions

Each team member actively contributed to all parts of the project. We worked together closely on planning, coding, testing, and writing, ensuring that every stage benefited from our shared input and discussion.

VII. Source code

The source code along with detailed logs and evaluation are available in our GitHub repository: https://github.com/MKatrodiya/Automatic-Parking

Acknowledgment

This project was advised by Mr. Justin Feldman and Dr. Rajagopal Venkatesaramani.

References

[1] Shen Xu et al. Conflict resolution in cooperative multi-agent parking scenarios. https://github.com/XuShenLZ/conflict_rez, 2023. GitHub repository. Accessed: 2025-06-25.

[2] Moreira Dinis. *Deep Reinforcement Learning for Automated Parking*. PhD thesis, 2021.

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

[4] Farama Foundation. Highwayenv documentation - parking environment, 2023. Accessed: 2025-06-25.

[5] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[7] 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.