International Journal of Intelligent Communication and Computer Science
Vol. 1, No. 2, 2023, Pages 1-18.

Autonomous parking in the perpendicular parking lot, using the Deep reinforcement algorithm: Proximal Policy Optimization with Generative Adversarial Imitation Learning and Behavior Cloning

Iryna Tyshchenko [1] and Mohammed Nazeh Alimam[1]

Research Paper

University of Europe for Applied Sciences, Berlin, GERMANY
Email: tyschenk20@gmail.com
Received: 30 Sep 2023, Revised: 15 Oct 2023, Accepted: 29 Oct 2023

Abstract:

This research paper focuses on the implementation of an autonomous parking system in Unity's simulated environment. The primary objective is to enable safe parking in designated slots without collisions. The implementation of this simulated model in Unity is discussed, highlighting the key classes and components responsible for agent behavior, car spawning, and control. The study leverages Generative Adversarial Imitation Learning (GAIL) to refine the agent's performance by incorporating expert demonstrations. Metrics such as Stimulative Reward, Episode Length, and GAIL loss are employed to evaluate the system's performance during training. Additionally, grid-search optimization is utilized to fine-tune the Proximal Policy Optimization (PPO) algorithm's hyperparameters. This research contributes to the development of efficient autonomous parking systems and offers insights into PPO hyperparameter optimization using grid search. The simulation framework supports the evaluation of autonomous parking performance under diverse scenarios, promoting a better understanding of the system's capabilities and limitations. By comparing the residual energy between normal nodes and energy harvested nodes, the effectiveness of the solar energy harvesting approach can be assessed.

Keywords: autonomous parking, Unity simulation, reinforcement learning, extrinsic rewards, behavior-driven approaches, Generative Adversarial Imitation Learning, Proximal Policy Optimization, grid-search optimization

1. Introduction

Parking in urban areas is a challenging process due to high vehicle density, incurring energy costs, contributing to the environmental footprint, and hampering productivity. Governments implement policies to discourage vehicle ownership and promote public transportation. Finding suitable parking spaces involves considering factors like proximity and layout. Drivers must navigate safely and choose from various parking slot orientations. The increase in vehicle numbers has elevated the risk of accidents, resulting in substantial losses and damages. To address human errors and improve safety, autonomous driving technology has emerged as a solution. Autonomous driving involves the use of computer systems to control different aspects of a vehicle without human intervention. An essential component of autonomous driving is automated parking, which enables a vehicle to navigate within a parking lot without the need for human control. This functionality relies on two interconnected systems: the smart parking lot and the smart car. The smart parking lot system plays a crucial role in guiding vehicles to their designated parking spots and optimizing the parking journey for drivers. Meanwhile, the smart car utilizes parking position data to detect obstacles and navigate the parking lot while avoiding barriers and obstructions. Both systems heavily rely on intelligent

learning systems, such as artificial intelligence and machine learning, to perform their tasks efficiently and make informed decisions.

2. Literature Survey

Autonomous driving technology has made significant advancements in recent years, aiming to enhance road safety, reduce traffic congestion, and improve the overall driving experience. This study builds upon several influential papers that have contributed to the field. Notably, in [1], an actor-critic (A3C-PPO) system for autonomous parking is introduced. Manual adjustments of hyperparameters, such as tuning the mini-batch size and fine-tuning the Generalized Advantage Estimate (GAE) factor, were performed to optimize the agent's final rewards. The study by [2] provides an effective method for autonomous rear parking (ARP) using reinforcement learning, path planning, and path following. This approach demonstrates effectiveness in reducing path following errors, holding promise for real-world applications across different industries. In [3], the authors present an approach that incorporates an imaginative model for predicting outcomes before parking. They employ an enhanced rapid-exploring random tree (RRT) for planning efficient trajectories from a given starting point to a parking location. The study results indicate that, compared to traditional RRT, the proposed approach performs better and provides superior results. The paper [4] focuses on the non-linearity of vehicle dynamics limitations and introduces the Deep Proximal Policy Optimization with Imitation Learning (DPPO-IL) approach, employing the Proximal Policy Optimization algorithm. This framework enhances parking spot exploration, path planning, and path tracking while utilizing intrinsic reward signals. It combines imitation learning with deep reinforcement learning for improved performance. In [5], a new approach is introduced that utilizes reinforcement learning for perpendicular parking. This enables the vehicle to learn optimal steering angles for various parking scenarios, achieve human-like intelligent parking, and address challenges such as path tracking errors and network convergence. The project [6] focuses on creating an autonomous car parking environment for multi-agent reinforcement learning. The author applies both Q-Learning and IPPO, examining competitive and collaborative behaviors among agents. PPO and grid-search parameter tuning are also applied, with the results showing a high success rate of at least 99.5% for parking with up to 7 agents.

A. Objective

The goal of this research in autonomous parking systems is to develop technology that enables automobiles to park themselves without human assistance. This will be achieved by proposing a combination of the Deep Reinforcement Learning algorithm and Imitation Learning. Additionally, the study aims to optimize the algorithm's hyperparameters using the grid-search technique. Ultimately, the objective is to revolutionize the parking experience, making it safer, more efficient, and more convenient for all.

B. Scenario

The study concentrates on a simulated vertical parking environment with two obstacles near the parking spot. The coordinate system of the parking environment is established with the center of the parking space as the origin, as illustrated in Figure 1.

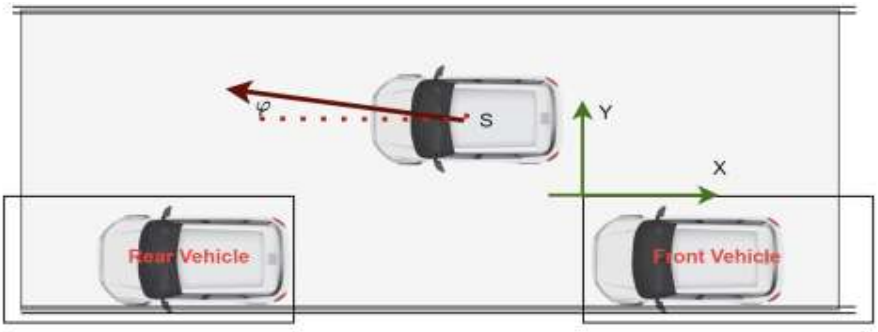

Figure 1: Scenario for autonomous vehicle parking [3]

3. Theoretical Background

A. Reinforcement learning overview

Artificial intelligence (AI) plays a vital role in various aspects of our lives and society, being integrated into electronic devices for automation and serving as a fundamental component of utility software. Among the machine learning techniques, reinforcement learning has gained significant popularity. It involves learning through trial and error, accumulating experiences, and adjusting the model's behavior based on rewards received for each experience. Rewards indicate the effectiveness of chosen actions in the current environment. Due to intrinsic control faults, traditional approaches based on path planning and path tracking may produce suboptimal parking orientations. There are methods that use "human-like" parking and reinforcement learning to address this problem. This end-to-end approach avoids errors brought on by path tracking by directly mapping the environment to actions. Additionally, it enables continuous learning and experience accumulation from multiple parking attempts, allowing the system to improve over time [7].

B. Main algorithm of the reinforcement learning

Reinforcement learning (RL) algorithms operate based on a reward system, where agents are rewarded for correct actions and penalized for incorrect ones. Unlike supervised learning, RL doesn't rely on labeled data, and it differs from unsupervised learning, which is focused on pattern finding. In RL, agents interact with an environment, taking actions to transition between states, as shown in Figure 2. Feedback or rewards may be delayed, and the current action influences future data. The agent's state guides its decision-making, and its history comprises observations, actions, and rewards. In fully observable environments, the agent's state aligns with the environment's state, modeled using a Markov Decision Process (MDP). In partially observable environments, a Partially Observable Markov Decision Process (POMDP) is used, constructing the state from history and assumptions about the environment's state [8]. An agent in reinforcement learning is made up of a model, a value function, and a policy. The agent's behavior is determined by the policy, which maps actions from states. Based on expected future rewards, the value function assesses the quality of states or actions. The model represents the agent's understanding of the environment and is used for simulating potential scenarios. Together, these components enable the agent to learn and make better decisions [2, 9].

Figure 2: Reinforcement Learning cycle [10]

C. Reinforcement learning environment

Observation Space

The reinforcement learning environment's observation space contains observable parameters. The heading angle (θe) and the position errors (Xe and Ye) of the vehicle's ego pose in relation to the target pose make up the observation space in this situation. Additionally, the readings from the LiDAR sensor are observed to determine if the vehicle is parked correctly or not.

State Space The vector between the autonomous car and its environment, in this case the parking lot, is represented by the state space. The state space can be defined as a combination of the vehicle's position

errors (*Xe* and *Ye*) with respect to the target position, the distance error (*d*) between the vehicle and the target spot, and the orientation (*O*) of the vehicle relative to the target pose [11] [12].

$$S = \begin{pmatrix} X_e \\ Y_e \\ \theta_e \\ d \\ O \end{pmatrix} \tag{1}$$

Action Space The collection of activities that the autonomous car is capable of performing in the parking lot environment is represented by the action space. The action signal specifically consists of distinct steering angles with step intervals of 15 degrees and a range of ±45 [1].

D. Deep reinforcement learning

In real-world applications, learning in complex environments with large state and action spaces can be challenging due to memory and computational limitations. Discretization of these spaces can be a potential solution, but it may lead to either limited performance with large discretization steps or a large state-action space with impractical sampling requirements with small discretization steps. To address this issue, function approximation techniques are actively researched in the field of reinforcement learning. Function approximation allows for generalization across states and actions, enabling the storage and retrieval of estimates using approximator functions. Neural networks, particularly deep neural networks, have also been widely used for function approximation, giving rise to the paradigm of deep reinforcement learning (DRL) [6] [12].

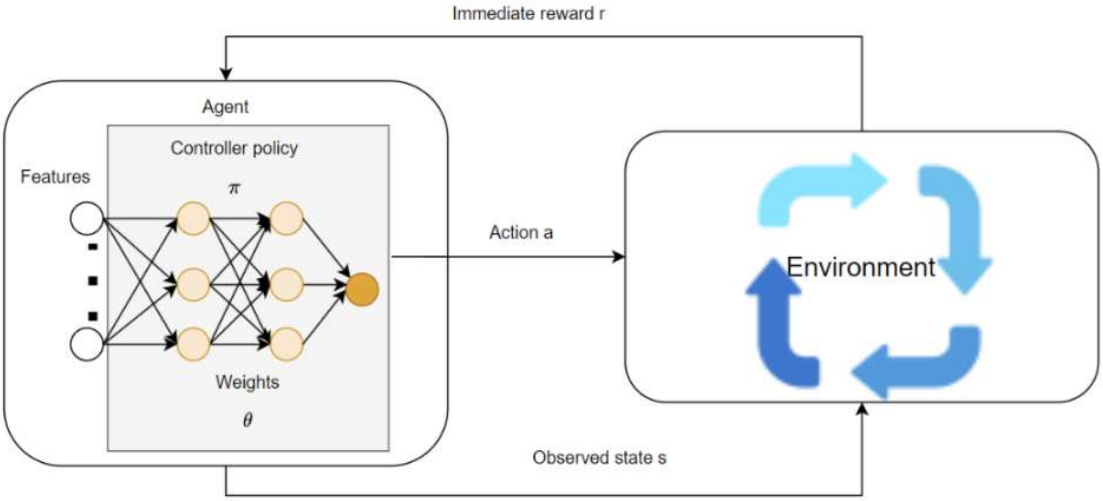

Figure 3: Deep reinforcement learning process [12]

E. Variations of deep reinforcement learning

Deep reinforcement learning is a popular approach for training self-driving vehicles without relying on manually labeled data. Two main categories of reinforcement learning methods are used for controlling autonomous vehicles: value based, policy-based, and actor-critic methods [12]. Value-based methods, such as deep Q-learning (DQN), aim to estimate the value of each possible action in a given state. These methods use a deep neural network to approximate the Q-value function, which represents the expected cumulative reward for taking a particular action in a specific state. The agent then selects actions based on the highest estimated Q values. Value-based methods are effective for problems with discrete action spaces, such as choosing

between predefined maneuvers like turning left, turning right, or going straight. However, they may struggle with problems that involve continuous action spaces, such as controlling the precise steering angle or the vehicle's velocity [13]. By computing the gradient of the predicted reward with respect to the policy's parameters, policy-based approaches directly optimize the policy. These techniques can handle continuous action spaces with high dimensions, but they may suffer from excessive variance in policy gradient estimates, which can lead to unstable training. However, improvements in algorithms and methodologies, such as variance reduction techniques, have helped to address these difficulties and increased the dependability of policy-based solutions. Policy-based methods directly optimize the policy itself by calculating the gradient of the expected reward with respect to the policy's parameters. These methods can handle high dimensional continuous action spaces but may suffer from high variance in policy gradient estimation, which can make training unstable. However, advancements in algorithms and techniques, such as variance reduction methods, have helped mitigate these issues and make policy-based methods more reliable [13]. Actor-critic methods, on the other hand, combine elements of both value-based and policy-based approaches. In actorcritic methods, there are two components: the actor and the critic. The actor is responsible for selecting actions based on the current policy, while the critic evaluates the chosen actions by estimating their expected rewards. By leveraging the critic's evaluations, the actor can update its policy to improve decision-making. This combination of value estimation and policy optimization allows actor-critic methods to handle both discrete and continuous action spaces. Popular actor-critic algorithms include Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO), and Actor-Critic with Experience Replay (ACER).
In this project, considering the limitations of value-based methods for continuous action spaces, the idea is to apply an actor-critic method PPO. This approach combines the benefits of both value-based and policy-based methods, leading to more stable training and improved control of autonomous vehicle parking.

F. Markov process

Reinforcement learning is a decision-making framework where an agent interacts with an environment. This process can be described using the Markov decision process (MDP) notation, which consists of a set of states (S), a set of actions (A), state transition probabilities (P), and a reward function (R). The set of states (S) represents the possible configurations or situations in which the agent can find itself. The set of actions (A) represents the available choices for the agent in a given state, determining how it interacts with the environment. The state transition probabilities (P) define the likelihood of transitioning from one state to another based on the current state and action. The reward function (R) provides a scalar feedback signal to the agent, indicating the desirability or quality of its actions. In reinforcement learning, episodes are simulated for a bounded number of time steps (τ). Each episode starts with the agent in an initial state (s_0) and ends at the last time step ($t = \tau - 1$). The dynamics of an MDP specify that the agent, at each time step, is in a state (s), takes an action (a), transitions to a new state (s') with a certain probability (P), and receives a reward (r) from the environment. The agent's goal is to learn a policy that maximizes the cumulative reward over time by exploring the environment, making decisions, and receiving feedback [6]

G. Imitation Learning

Imitation learning (IL) is a method where a policy is learned directly from expert demonstrations, without accessing the expert policy itself. The agent is provided with a dataset (D) containing expert trajectories, obtained from an expert policy interacting with the Markov decision process (MDP). The objective is to learn the best policy (π_{IL}) using this dataset without requiring access to the expert policy (π_{exp}).
In IL, there are two well-known algorithms: Behavior cloning and Generative Adversarial Imitation Learning (GAIL). Behavior cloning minimizes the loss function between the agent and the expert policy by using negative log likelihood, without collecting samples from the environment. On the other hand, GAIL minimizes the loss function using cross-entropy loss and collects samples from the environment to improve its policy [10].

H. Imitation Learning process

Imitation learning serves as an alternative to reinforcement learning, particularly when the aim is to make the AI behave more like a human rather than pursuing machine-like perfection. The framework involves a Teacher agent performing the task and a Student agent imitating the Teacher. The learning process starts with the Teacher playing the task for a specific duration, varying based on the task's difficulty. The Teacher

can be a neural network, a human, or a deterministic algorithm, with human Teachers often yielding the best results. Meanwhile, the Student agent observes the actions of the Teacher and attempts to imitate its behavior. Imitation learning allows the Student agent to learn by observing and mimicking the Teacher's actions, enabling the acquisition of similar behaviors and decision-making processes. This approach enables the development of AI systems that exhibit human-like characteristics and behaviors, making them suitable for applications where human-like performance is desirable [9].

I. Behaviour Cloning

Behaviour Cloning (BC) is a form of imitation learning that treats the problem as a supervised learning task. It does not require any interaction with the Markov decision process (MDP). In BC, each expert state-action pair is treated as a training example for a regression problem, where the state is the input and the action is the output. The policy is learned by minimizing the supervised training loss, which aims to make the agent's predicted actions match the expert actions.

$$\min_{\pi} \; J_{BC}(\pi) = -\frac{1}{N}\sum_{k=1}^{N} \log \pi\left(a_k \left| s_k\right.\right) \tag{2}$$

where π represents the policy that is being optimized, $J_{BC}(\pi)$ is the objective function, N represents the number of data points in the dataset, s_k represents the state of the environment at the $k-th$ data point, a_k represents the action taken by the expert at the $k-th$ data point and $\log \pi\left(a_k \left| s_k\right.\right)$ is the logarithm of the probability of taking action a_k in-state s_k according to the policy π.

J. Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) consists of two networks: the generator and the discriminator D_{φ}. The generator produces a distribution of state-action pairs, while the discriminator, a secondary neural network, generates a distribution from the expert demonstration samples p. GAIL aims to acquire near-optimal behaviours directly from expert demonstrations and self-exploration, without requiring the design of task-specific reward functions. By encouraging the generator to confuse the discriminator, GAIL enables the agent to mimic a policy that exhibits human-like behaviour by performing similar states and actions as demonstrated by the expert. GAIL optimizes the min-max cross-entropy objective, as presented in Equation below.

$$\min_{\pi} \max_{\varphi} V(\theta,\varphi) \triangleq \mathrm{E}_{\pi\theta}\left[\log\left(1-D_{\varphi}(s,a)\right)\right] + \mathrm{E}_{\pi e}\left[\log D_{\varphi}(s,a)\right] \tag{3}$$

There is a generative model G and a discriminative classifier D competing against each other. The objective of D is to differentiate between trajectories generated by G and those generated by the expert E. The success of G is determined by its ability to generate trajectories that are indistinguishable from the expert's trajectories. To achieve this, the formula that should be maximized by D is given by:

$$\mathrm{E}_{\pi}\left[\log D_{\varphi}(s,a)\right] + \mathrm{E}_{\pi_E}\left[\log\left(1-D_{\varphi}(s,a)\right)\right] \tag{4}$$

E_{π} in this formula takes a trajectory D composed of state s and its corresponding action a as input. The $\log D(s,a)$ function returns a continuous value ranging from 0 to 1. A higher value indicates that the input trajectory resembles an expert's trajectory. By maximizing this value, D can provide a reward signal to train G to generate trajectories similar to those of the expert. This gradient update allows G to adjust its parameters θ in the direction that increases the likelihood of generating trajectories that are perceived as expert-like by D [12].

K. Proximal Policy Optimization

PPO is a widely used actorcritic algorithm in the field of deep reinforcement learning. It has gained recognition for its simplicity of implementation and competitive performance compared to other state-of-theart approaches. PPO has become a popular choice due to its effectiveness and ease of use. Unlike supervised learning, where defining a cost function and applying gradient descent often leads to good results,

reinforcement learning is more complex and requires careful tuning. RL algorithms can be challenging to debug and often require significant effort to achieve satisfactory performance. PPO addresses these challenges by offering a balanced approach. It aims to strike a balance between implementation simplicity, sample efficiency, and ease of tuning. By computing updates at each step that minimize the cost function while ensuring limited deviation from the previous policy, PPO efficiently explores the policy space and finds optimal policies. PPO has been successfully applied in various domains, including automated parking, where it demonstrated robust performance against noise and emphasized smooth trajectory planning. While some limitations may exist in specific experimental setups, PPO has proven to be a reliable and effective algorithm in many applications. PPO falls under the category of policy-gradient methods, specifically actor-critic methods. In these methods, the policy is parameterized by weights θ and optimized using stochastic gradient ascent on a loss function L (θ). Additionally, the agent's value function, which approximates the Q-values in Q-Learning, is simultaneously approximated. The actor approximates the policy, determining which action to take, while the critic approximates the value function, providing feedback on the quality of the chosen action and guiding adjustments. The PPO algorithm updates the weights θ based on the loss function using a learning rate hyperparameter α and a random batch of trajectories from an experience buffer. This process shown in Figure 7 is repeated for a specified number of epochs, with hyperparameters such as buffer size, batch size, and network architectures (layers and hidden units) influencing the training dynamics. PPO also introduces a minimum size for trajectories to be added to the buffer, known as the time horizon [13].

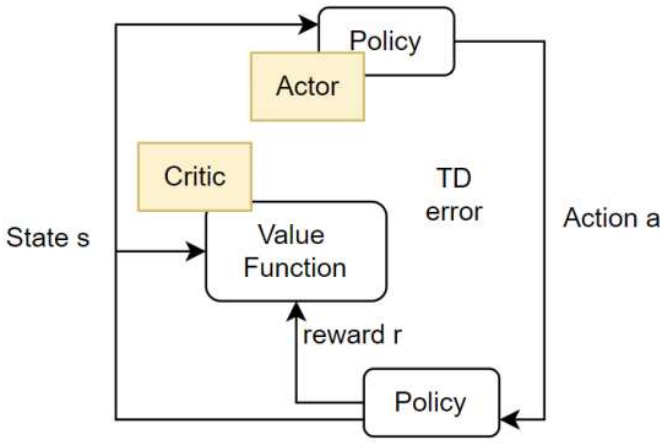

Figure 4: The actor-critic architecture [14]

4. Methodology

In this study, the experiments were conducted using reinforcement learning techniques, specifically focusing on the Unity game engine and the ML-Agents reinforcement learning platform. The chosen algorithm for reinforcement learning was PPO. PPO was selected as the reinforcement learning algorithm due to its advantages as a policy-based approach compared to value-based algorithms like Q-Learning. Given the wide range of possible actions for the learning agent, using a value-based algorithm would require significant memory resources. The agent's input consists off field-of-view images captured from both the front and rear in-vehicle cameras. The network model utilized in the study is illustrated in Figure 5, incorporating three convolutional layers. The performance evaluation of the learning process is based on measures such as cumulative reward and entropy [15].

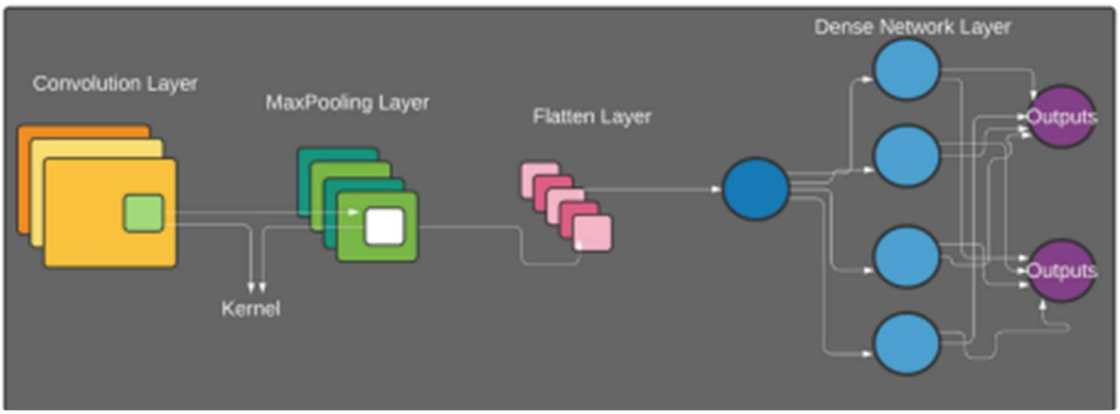

Figure 5: Deep Neural Network Model [16]

A. Reinforcement Learning

This project involves an agent, represented by a car, operating in an environment with specific actions and receiving rewards based on its actions as shown in Figure 6. The agent's objective is to find a parking spot. It follows a reinforcement learning cycle, where it takes actions, receives rewards or punishments, and transitions to the next state. The agent learns over time which actions lead to positive rewards and avoids actions that result in punishments. The cycle continues until the agent either reaches its goal (a parking spot) or reaches a predetermined step limit. Positive rewards are given when the agent successfully reaches the parking spot, while negative rewards are assigned for collisions with other parked cars.

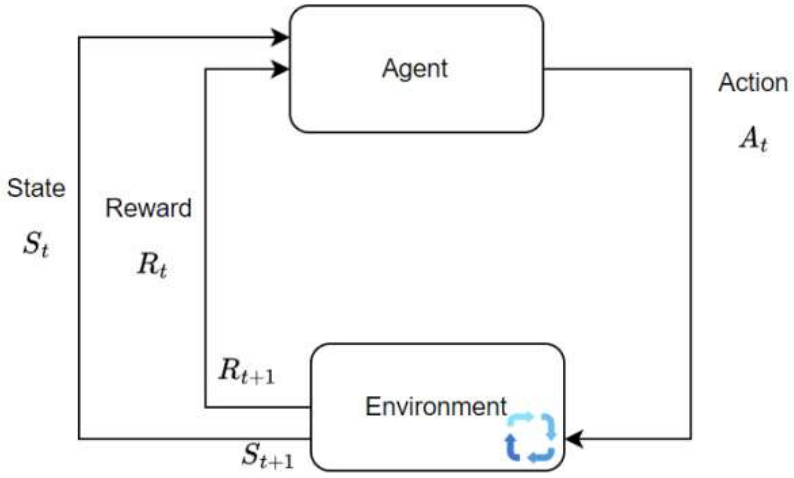

Figure 6: Agent-Environment Interaction cycle [15]

B. GAN Policy

The architecture of the GAN policy is shown in Figure 7. The simplified process includes the following steps:

1. Train the discriminator D using expert and generated trajectories.
2. Update the policy G with a gradient.

Expert trajectories can be either generated from a human expert, an algorithm or a policy that has already mastered the target task [16].

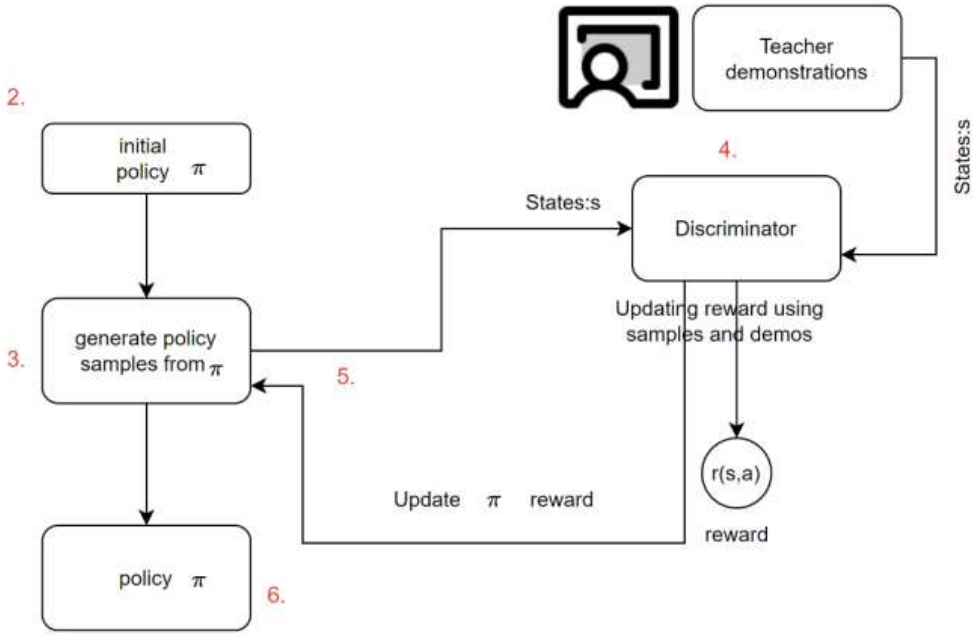

Figure 7: GAN policy architecture [16]

Proximal policy optimization architecture

In Figure 8, the actor represents the policy network responsible for determining the policy function $\pi(a/s)$, while the critic evaluates the chosen policy by estimating the state-value function Q (s,a). Using information from the energy market, the PPO algorithm develops the theta and varpimodel parameters for the actor and critic networks. The current policy π is used to sample finite-length trajectories D from the training data during each training iteration (k), as opposed to investigating the full episode. Based on the existing policy for each trajectory, the simulation generates bidding decisions (a) for the energy and frequency regulatory markets. As a result, utilizing the reward function, each time-step (t) experiences a profit or loss $\{r(t)\}$ depending on the state $\{s(t)\}$. The value function calculates the trajectory's expected discounted reward.

C. Unity

The study's learning environment was created using the Unity game engine, in particular the Unity Machine Learning Agents Toolkit (ML-Agents). Unity is known for creating interactive 2D and 3D games. The reinforcement learning (RL) community favours ML-Agents because of its visualizations and support for concurrent agent training. ML-Agents provide tools for real-time 2D/3D Markov Decision Process (MDP) simulations with varying agent numbers. It implements RL algorithms (e.g., single/multi-agent PPO) and integrates with Tensor Board for visual analysis. Its Python Low-Level API enables researchers to interface with Unity using custom RL algorithms [9, 13].

D. Visualization

Tensor Board is the main tool used for data collection in order to monitor crucial metrics like average rewards and policy loss. Additional metrics are also written to files using the built-in CSV module.

E. Metrics

The selected metrics for evaluating performance are:

Total Reward: this metric measure the cumulative rewards collected by the agent throughout the training and testing phases. An increase in the total reward indicates an overall improvement in the agent's policy.

Success Rate: the success rate metric calculates the average number of times the agent successfully completes the task, which in this case involves parking the vehicle correctly in the designated lane without collisions. A higher success rate indicates better performance.

Estimating the distance of driving length: Estimating the distance of driving length is depicted in Figure 9 is another important metric to consider. This metric provides information about the total distance covered by the agent during the training and testing phases. It can be useful for evaluating the efficiency and effectiveness of the agent's navigation and decision-making capabilities [16].

In addition to the aforementioned metrics, monitoring the vehicle velocity is indeed an important factor. By tracking the velocity of the ego vehicle, we can assess its speed and ensure that it remains within the desired range of behaviour. Deviations from the expected velocity range can indicate issues such as overly cautious or aggressive driving behaviour, which may require adjustments to the agent's policy or parameters. Monitoring and analysing the vehicle velocity can help optimize the agent's performance and ensure safe and efficient driving.

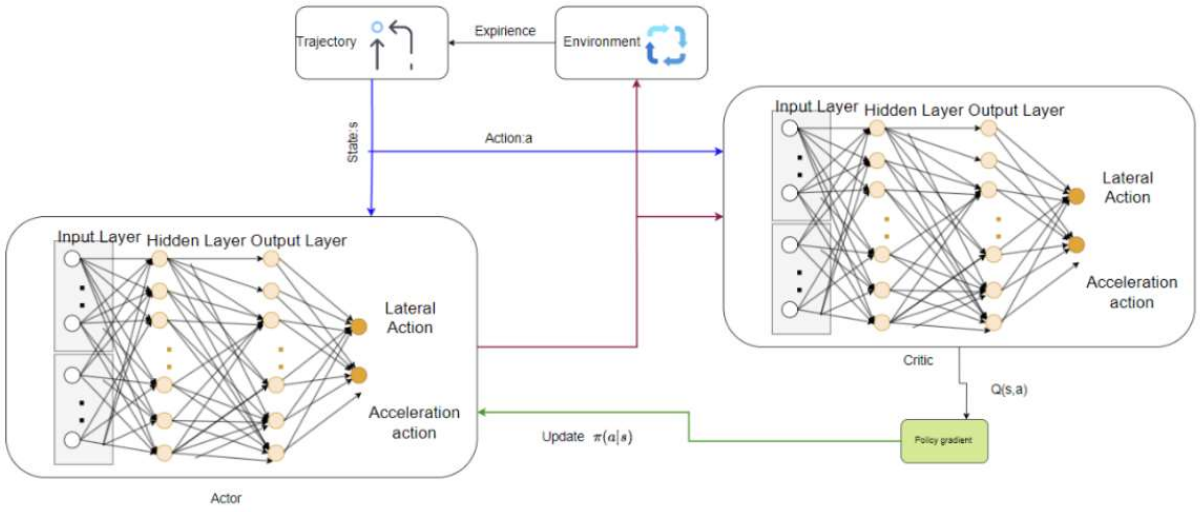

Figure 8: The proximal policy optimization algorithm's high-level diagram [17]

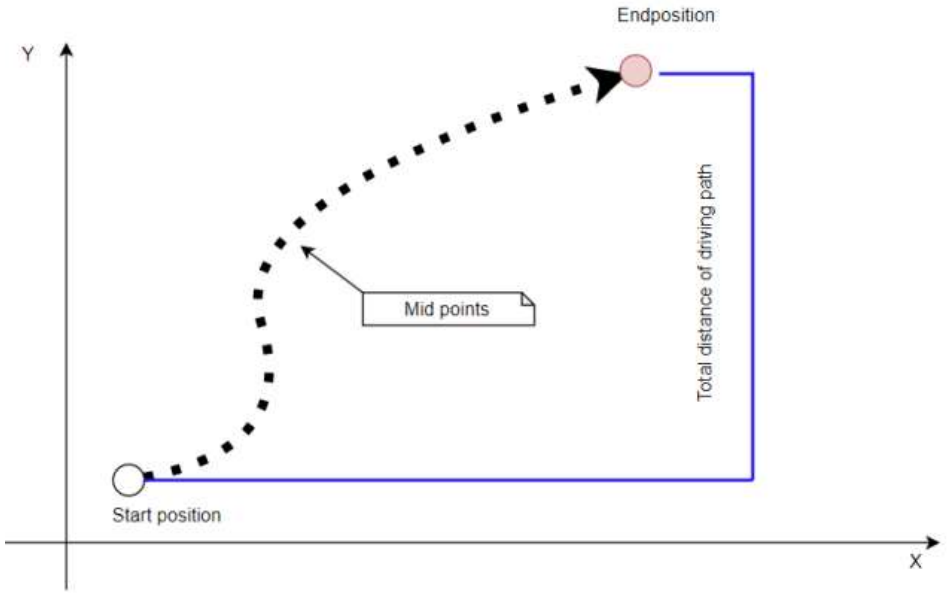

Figure 9: Estimating of path distance [16]

5. Implementation and Results

A. Environment

The assets used in the environment include car models that were downloaded from the Unity Asset Store. The wheel colliders were specifically added to the wheels of the car. These wheel colliders enable the car to have realistic driving capabilities, allowing it to interact with the environment more accurately. These car models consist of a body and four wheels with cylinders. The specific car models used are depicted in Figure 10.

Figure 10: Assets used for simulation [18]

B. Parking lot

The environment was built by creating a parking lot (shown in Figure 11), which is represented as a rectangular-shaped plane with walls on the sides. The parking simulation consists of other cars and a designated parking slot highlighted in red. The objective for the cab is to locate a suitable parking slot highlighted as green and safely park there without causing any damage to other vehicles. The reward function provides positive or negative rewards based on the cab's actions. If the cab collides with other cars, a highly negative reward is given, emphasizing the importance of avoiding collisions. Similarly, hitting random obstacles results in a slightly negative reward. On the other hand, successfully parking the cab in the designated slot leads to a high positive reward. The environment is based on [19].

Figure 11: Parking lot simulation

C. Components of parking lot

Car Agent: Controls the autonomous vehicle's actions (acceleration, braking, steering) using algorithms.
Max Step: Limits Car Agent's actions to avoid indefinite tasks, setting max instructions per episode.
Car Spawner: Generates interacting vehicles, specifying initial positions and parameters.
Spawn Radius: Defines area for generating interacting vehicles; reset if outside radius.
Target: Represents parking destination for Car Agent, guiding actions.
Box Collider and Rigid Body: Rigid body simulates car physics (e.g., acceleration, collision response); Colliders detect collisions with objects.
Sensors: Cameras capture visual data for input. Parameters like Width, Height, grayscale, and compression affect resolution and format. Sensors detect objects within 10 meters, aiding safe navigation and parking.

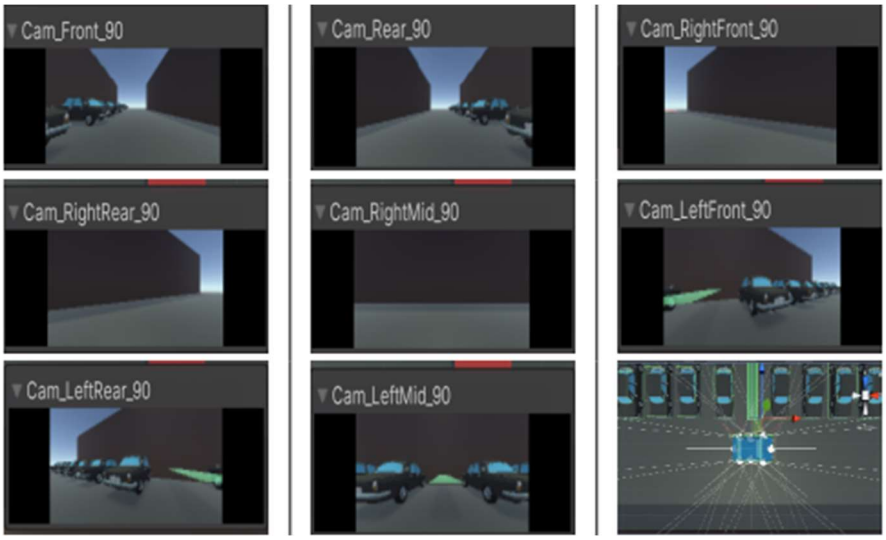

Figure 12: Pictured View from Cameras

In this project, the spawned cab is equipped with eight cameras, view of which depicted in Figure 12. These sensors are responsible for measuring distances to nearby obstacles and providing this information to the underlying AI system. The AI system utilizes reinforcement learning techniques and a reward function to guide the cab's actions. The integration of multiple sensors can overcome limitations and enhance the overall performance of the system. With the sensor information, the cab must learn to navigate the environment, avoid obstacles, and autonomously locate the appropriate parking slot. The reinforcement learning algorithm optimizes the reward function, enabling the cab to improve its performance over time.

D. Behaviour options

Behaviour Name: Defines hyperparameters for system behaviour, distinguishing configurations.
Vector Observations: Sets array size for environment inputs, like camera data.
Vector Action: Specifies agent actions—float array for continuous or int array for discrete.
Model: Trained model for agent decisions during non-training phases.
Behaviour Type: Sets agent behavior—Default for training and inference, heuristic for manual testing.
Decision Requester: Controls action update rate from learning model.
Decision Period: Sets interval for action requests—1 for direct script actions.

E. Training process

To enhance the training process, two additional components were incorporated: extrinsic rewards and curiosity-driven exploration. After that, GAIL was added to learn the parking from recorded demo files.

Extrinsic Reward

An extrinsic reward system is used to guide the agent's behavior in the parking lot. The agent receives rewards based on its performance in completing the parking task. The extrinsic rewards serve as a measure of success and encourage the agent to learn optimal parking strategies.

Behaviour Driven

Behaviour-driven approaches involve shaping the agent's behavior by incorporating specific rules or expert knowledge into the learning process.

Training with Generative Adversarial Imitation Learning (GAIL)

To provide the AI with an understanding of the environment from the early stages, 100 demonstration instances were recorded. These demonstrations are stored in the demo's folder. By incorporating these demos into the training process, the AI can learn from the expertise demonstrated in the recordings. The path to the recorded model is added to the PPO hyperparameters during training, allowing the AI to leverage the knowledge gained from the demonstrations. The combination of extrinsic reward, behaviour-driven approaches, and GAIL was used to enhance the training and performance of an autonomous parking system.

By providing extrinsic rewards, the agent receives explicit feedback on its parking performance, enabling it to learn the desired behaviour more efficiently. Incorporating behaviour-driven techniques allows for the incorporation of specific parking rules and expert knowledge, further guiding the agent's actions toward safe and efficient parking. Additionally, GAIL can be used to leverage expert demonstrations, training the agent to imitate the behaviour of skilled human drivers. This combination can lead to improved parking performance, smoother manoeuvres, and better adherence to parking regulations.

F. Vehicle model

During the parking manoeuvre, the vehicle operates at a relatively low velocity. To facilitate motion planning, a kinematic vehicle model, as depicted in Figure 13, is utilized. This model allows us to represent the vehicle's posture in the parking coordinate system using variables (x, y, θ), where (x, y) represents the vehicle's position, and (θ), denotes its orientation. Φ stands for the steering angle of the front wheel, and v stands for the velocity at the center of the rear axle.

G. Kinematic vehicle model

In the low-speed motion scenario, it can be assumed that the car's wheels only roll without any slipping, and the lateral dynamics of the tires can be disregarded. Under these assumptions, a car kinematics state change, including angle and position will be done using the formula below [15].

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= v\tan(\phi)/L \end{aligned} \tag{5}$$

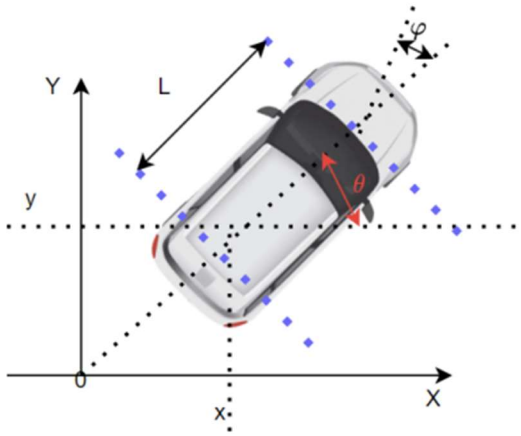

Figure 13: Kinematic model of vehicle [15]

H. Reward function

The reward function in this experiment is designed to evaluate the agent's actions at each step. Negative rewards are given to actions that the agent should avoid, such as collisions, with the accumulation of negative rewards reflecting their impact on safety. The agent receives a sparse reward only when the distance between the agent and the target decreases by 10%. To gradually communicate progress, the agent is given a modest positive Dense Reward and Checkpoints reward at each stage. When the agent completes checkpoints, which act as intermediate goals, they also get an extra reward. The cumulative reward is the sum of the benefits the agent has received for each episode [20, 21]. This approach encourages the model to strive for higher rewards from successful parking.

I. Metrics

I. Cumulative Reward:

The cumulative reward chart typically shows the total reward accumulated by the agent over time or episodes. As the agent investigates the world and picks up new skills, the chart may initially display a random or low reward. As the agent investigates the world and picks up new skills, the chart may initially display a

random or low reward. As the agent learns and improves its policy, the chart should generally show an upward trend, indicating increasing rewards.

II. Episode Length:

The episode length chart illustrates the duration or length of each episode during training. At the beginning of training, the episode length may be relatively long as the agent explores and takes time to learn optimal actions. As the agent learns and becomes more efficient, the episode length should generally decrease.

III. GAIL Loss:

The GAIL loss chart represents the loss incurred during the training of the discriminator network in the GAIL algorithm. Initially, the loss may be high as the discriminator struggles to differentiate between expert and policy actions. As the discriminator learns, the loss should decrease, indicating an improvement in its ability to distinguish expert actions from policy actions.

IV. Value Loss:

The value loss chart shows the loss incurred during the training of the value function or value estimator.

V. Pre-training Loss:

The pre-training loss chart represents the loss incurred during the pre-training phase of the model.

VI. Policy Loss:

The policy loss chart shows the loss incurred during the training of the policy network. For the value, pre-training, and policy loss at the beginning of training, the loss may be high, but as the model improves, the loss should decrease, indicating a better alignment between predicted values and observed rewards.

The metrics collected after training are shown in Figure 14.

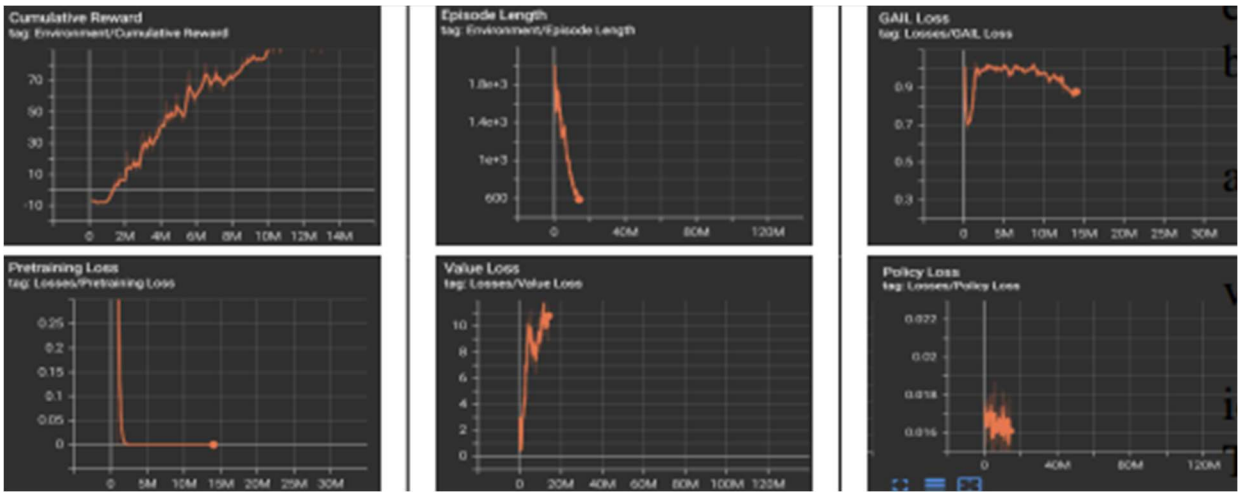

Figure 14: Metrics Visualized with Tensor Board

J. Implementation in Unity

The class diagram for the proposed environment is shown in Figure 15.

Description of some of the classes:

1. The MoveToGoalAgent class represents an agent that learns to navigate a car to reach a goal in a Unity ML-Agents environment. It uses reinforcement learning to train the agent's behavior.
2. The CarStripSpawner class is responsible for spawning a strip of cars in a Unity scene.
3. The CarSpawnerclass is responsible for spawning individual cars in a Unity scene.
4. The Car class represents a car in the Unity scene. It typically includes a car model, colliders, and other components. The exact functionality and behavior of the car differ based on the particular implementation.
5. The CarController class is responsible for controlling the movement and behavior of a car in a Unity scene. It typically includes functions for steering, accelerating, braking, and applying physics forces to the car's rigidbody component. The specific implementation of the car controller can vary depending on the desired car physics and behavior.

6. The SimpleCarController class is responsible for controlling a car in a simple driving simulation. It provides functionality for steering and applying motor torque to the wheels of the car.

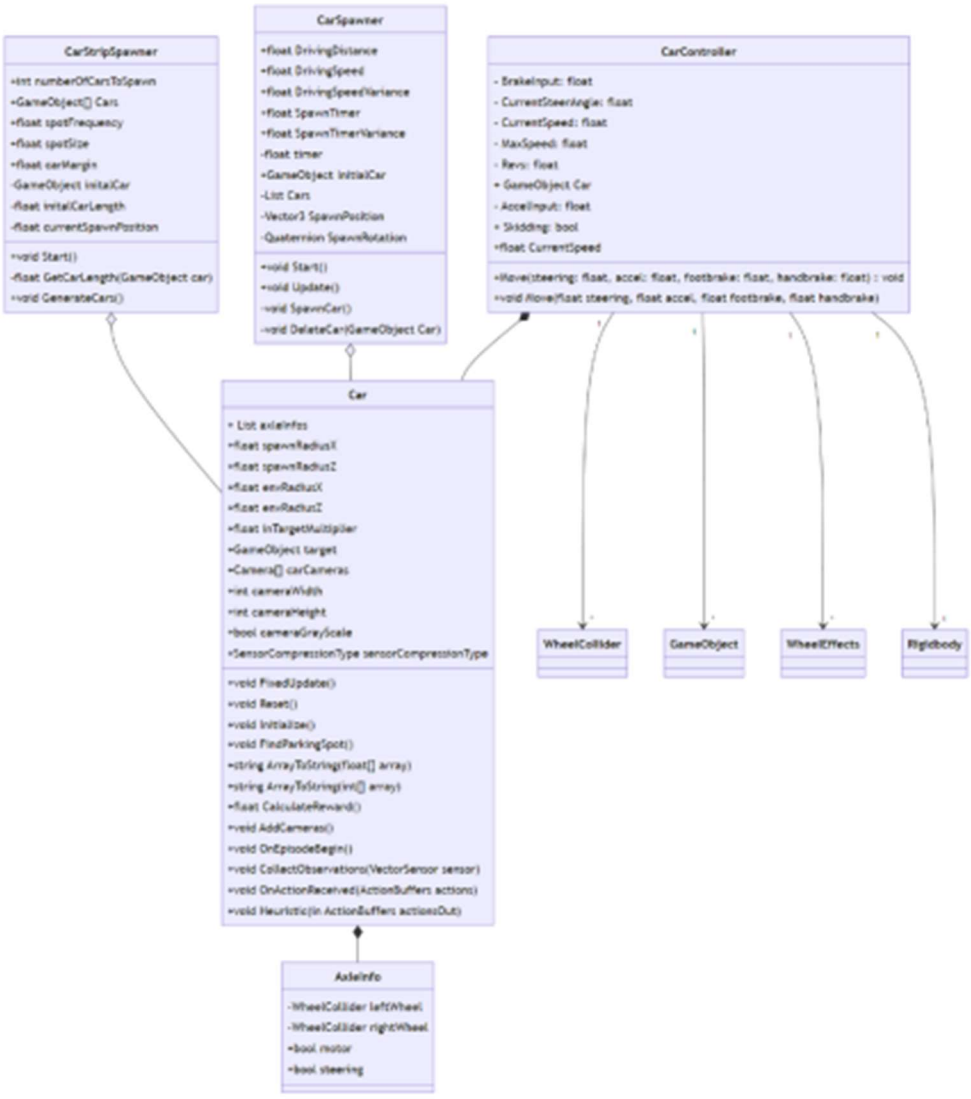

Figure 15: Class Diagram of classes used for simulation

K. Optimization of PPOs hyperparameters

The main hyperparameters used are shown in Table 1. There are a lot of recommendations on how to improve the PPO algorithm by adjusting the hyperparameters [8] [5]. Grid searching represents one of the potential techniques for optimizing a set of hyperparameters in PPO. It involves sampling a subset of values for each parameter, and then every combination of these values is evaluated. By exhaustively trying all combinations, the best-performing configuration can be identified. This method allows for systematic exploration of the hyperparameter space to find the optimal values for PPO training. This method will be used to adjust parameters for the PPO [19].

The algorithm steps are described below and shown in Figure 16:

1. Generate all possible combinations of the parameters being grid-searched.
2. For each combination, create a separate PPO YAML configuration file by injecting the parameter values into a base YAML file.

3. Create individual .slurm files for each parameter combination. These files are used to call the ”magnets-learn” command with the corresponding YAML files.
4. Execute the jobs on the compute cluster of the department. This is done by using the ”srun” command with each of the .slurm files created in the previous step.
5. Once the training is complete, automatically run ”analysis.py” on all the trained models. This is achieved by executing ”analysis.py” after ”magnets-learn” in the .slurm files.
6. The result of the analysis can be imported to a spreadsheet and further visualized to gain comprehensive insights.
7. To compare the output models with the grid-searched parameters, step 5 passes each grid-searched parameter to ”analysis.py”. This ensures that the parameter values appear as columns in the resulting CSV output.

Table 1: PPO Hyperparameters

Parameter	Value	Explanation
Trainer_type	PPO	Type of trainer (PPO)
Batch_size	1024	Experiences per batch
Buffer_size	5120	Experience replay buffer size
Learning_rate	0.00035	Neural network learning rate
beta	0.0025	Entropy regularization strength
epsilon	0.3	PPO clipping parameter
lambd	0.95	Generalized advantage estimation parameter
Num_epoch	5	Optimization epochs per batch
normalize	TRUE	Input data normalization
Hidden_units	264	Hidden units per layer
Num_layers	3	Number of layers
gamma(extrinsic)	0.95	Extrinsic reward discount factor
strength (extrinsic)	0.99	Extrinsic reward strength
strength (GAIL)	0.3	GAIL reward strength

6. Conclusions

The research implemented an autonomous parking system in Unity’s simulated environment. The goal was safe parking in a designated green slot without collisions. To enhance training, extrinsic rewards, behavior-driven methods, and GAIL were used. Extrinsic rewards guided agent behavior. Positive rewards for successful parking, negative for collisions. Agent aimed to maximize cumulative reward by learning parking strategies. This approach has driven the agent to maximize cumulative rewards and develop advanced parking strategies. Behavior-driven methods shaped agent behavior with rules and expert knowledge, improving adherence to parking regulations. The introduction of GAIL has significantly improved the system’s performance. Leveraging 100 expert demonstrations, the AI system learned from these recordings, resulting in enhanced parking precision, maneuvering, and rule adherence. During the process of training and simulation, metrics like Stimulative Reward, Episode Length, and GAIL loss evaluated system performance during training. Additionally, using the grid-search method, various value combinations were evaluated. It led to faster optimization of PPO algorithm hyperparameters. As a result of this research, the autonomous parking system was implemented in a simulated environment using the Unity engine. The objective for the autonomous vehicle, represented by the car agent, was to locate a suitable parking slot highlighted in green and safely park there without causing any damage to other vehicles. To summarize, the research contributes to efficient autonomous parking. Insights on optimizing Proximal Policy Optimization parameters using grid search. The proposed simulation framework supports exploring various aspects of autonomous parking performance under diverse scenarios.

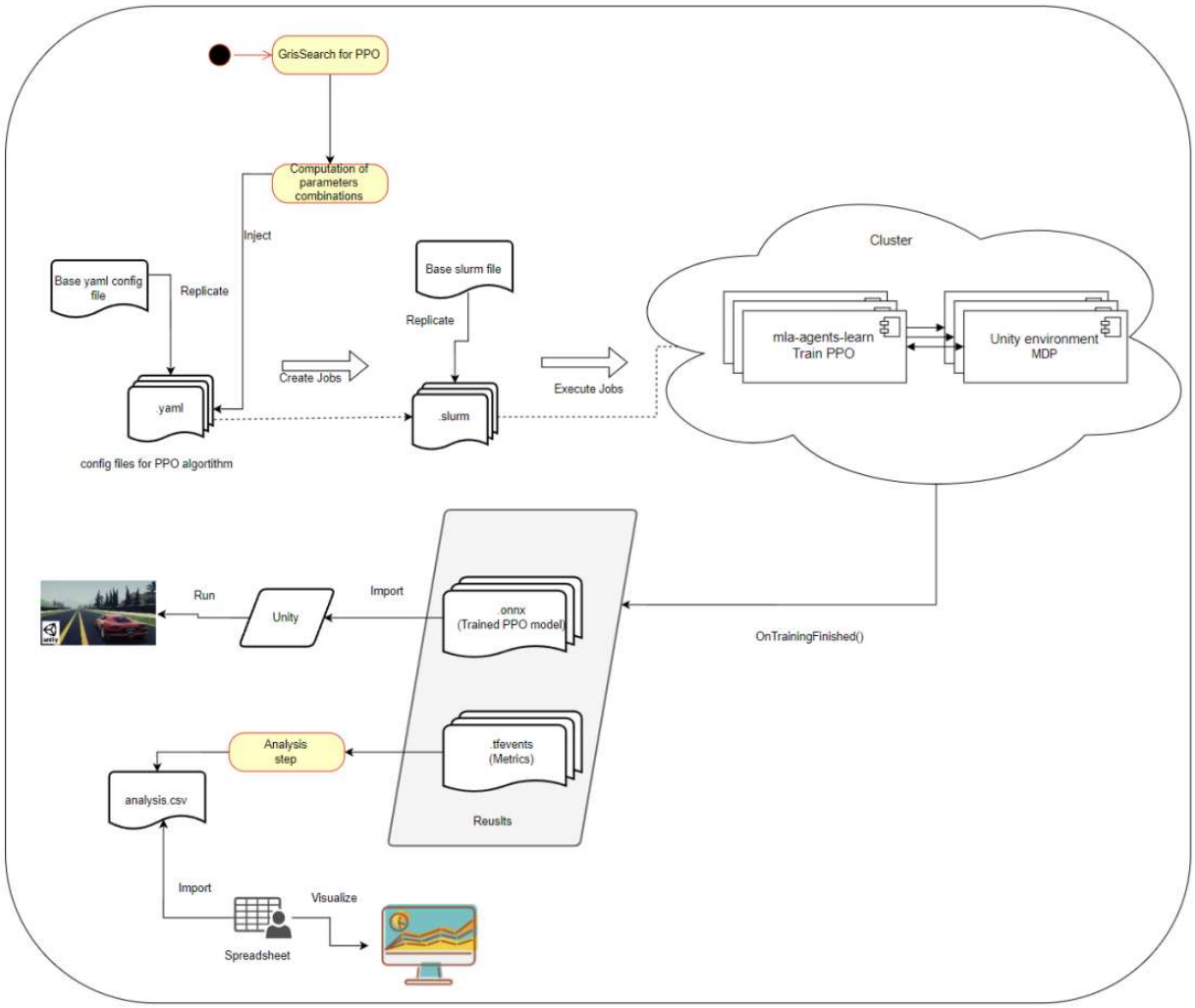

Figure 16: Proposed algorithm for PPO's hyperparameters improvement

References

1. Tiong, Teckchai, Ismail Saad, Kenneth Tze Kin Teo, and Herwansyah Bin Lago. "Autonomous Valet Parking with Asynchronous Advantage Actor-Critic Proximal Policy Optimization." In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0334-0340. IEEE, 2022.
2. Shahi, Saugat, and Heoncheol Lee. "Autonomous Rear Parking via Rapidly Exploring Random-Tree-Based Reinforcement Learning." *Sensors* 22, no. 17 (2022): 6655.
3. Feng, Ziyue, Shitao Chen, Yu Chen, and Nanning Zheng. "Model-based decision making with imagination for autonomous parking." In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2216-2223. IEEE, 2018.
4. Albilani, Mohamad, and Amel Bouzeghoub. "Dynamic Adjustment of Reward Function for Proximal Policy Optimization with Imitation Learning: Application to Automated Parking Systems." In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1400-1408. IEEE, 2022.
5. Zhang, Peizhi, Lu Xiong, Zhuoping Yu, Peiyuan Fang, Senwei Yan, Jie Yao, and Yi Zhou. "Reinforcement learning-based end-to-end parking for automatic parking system." *Sensors* 19, no. 18 (2019): 3996.
6. Tanner, Omar. "Multi-Agent Car Parking using Reinforcement Learning." *arXiv preprint arXiv:2206.13338* (2022).
7. Thunyapoo, Baramee, Chatree Ratchadakorntham, Punnarai Siricharoen, and Wittawin Susutti. "Self-parking car simulation using reinforcement learning approach for moderate complexity parking

scenario." In *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 576-579. IEEE, 2020.
8. Urmanov, Marat, and Madina Alimanova. "Training a single Machine Learning Agent using Reinforcement Learning and Imitation Learning methods in Unity environment." *Suleyman Demirel University Bulletin: Natural and Technical Sciences* 52, no. 1 (2020).
9. P. Pandita. Evaluation of soft actor critic in diverse parking environments. Master's thesis.
10. Urmanov, Marat, and Madina Alimanova. "Training a single Machine Learning Agent using Reinforcement Learning and Imitation Learning methods in Unity environment." *Suleyman Demirel University Bulletin: Natural and Technical Sciences* 52, no. 1 (2020).
11. Quang Tran, Duy, and Sang-Hoon Bae. "Proximal policy optimization through a deep reinforcement learning framework for multiple autonomous vehicles at a non-signalized intersection." *Applied Sciences* 10, no. 16 (2020): 5722.
12. D. Liu. Learning to imitate: using gail to imitate ppo, 2021.
13. Kiran, B. Ravi, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. "Deep reinforcement learning for autonomous driving: A survey." *IEEE Transactions on Intelligent Transportation Systems* 23, no. 6 (2021): 4909-4926.
14. Juliani, Arthur, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy et al. "Unity: A general platform for intelligent agents." *arXiv preprint arXiv:1809.02627* (2018).
15. Majumder, Abhilash. *Deep Reinforcement Learning in Unity: With Unity ML Toolkit*. Apress, 2021.
16. Shahi, Saugat, and Heoncheol Lee. "Autonomous Rear Parking via Rapidly Exploring Random-Tree-Based Reinforcement Learning." *Sensors* 22, no. 17 (2022): 6655.
17. Anwar, Muhammad, Changlong Wang, Frits De Nijs, and Hao Wang. "Proximal policy optimization based reinforcement learning for joint bidding in energy and frequency regulation markets." In *2022 IEEE Power & Energy Society General Meeting (PESGM)*, pp. 1-5. IEEE, 2022.
18. https://assetstore.unity.com/packages/3d/vehicles/ukraine-free-cars-191822
19. Thomas Van Iseghem. Ai-parking-unity, 2021.
20. https://medium.com/xrpractices/autonomous-car-parking-using-ml-agents-d780a366fe46
21. Saha, Sumit. "A comprehensive guide to convolutional neural networks—the ELI5 way." *Towards data science* 15 (2018): 15.