Autonomous Navigation in Parking Lots using TD3 and Option-Critic

Aatmaj Amol Salunke, Mukesh Kumar Javvaji
{salunke.aa, javvaji.m}@northeastern.edu
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115 USA

Abstract

This project addresses the problem of autonomous car parking, a critical task for autonomous vehicle systems. We propose a solution using advanced reinforcement learning (RL) algorithms, specifically Twin Delayed Deep Deterministic Policy Gradient (TD3) and Options Critic, to enable efficient and accurate parking. A realistic parking lot environment was designed in Unity, incorporating sensors and physics-based interactions to simulate real-world conditions. The Unity environment was connected to Python for training via the ML-Agents library. The results demonstrate that advanced RL algorithms significantly outperform entry-level methods, achieving faster and more stable learning. Within just a few episodes, the agent exhibited remarkable progress in parking tasks, whereas simpler algorithms required up to ten times more episodes to reach a comparable level of performance.

Introduction

Autonomous vehicles represent the future of transportation, with parking being one of the most challenging and essential tasks they must perform. Traditional rule-based or heuristic approaches to parking often lack adaptability and fail in dynamic environments. Reinforcement learning (RL) offers a promising alternative by enabling agents to learn optimal strategies through trial and error in simulated environments. This project explores the application of state-of-the-art RL algorithms, TD3 and Options Critic, to autonomous car parking. These algorithms are designed to handle continuous action spaces effectively, making them well-suited for tasks involving precise control, such as parking.

To simulate the complexity of real-world parking, we designed a realistic parking lot environment in Unity. This environment includes sensor data and physics-based object interactions to provide an accurate representation of the challenges faced by autonomous vehicles in real-world scenarios. The Unity ML-Agents library was used to connect the simulation to Python, facilitating the training of RL agents. Our results demonstrate that advanced RL methods not only accelerate the learning process but also achieve superior performance compared to traditional approaches.

Background

Reinforcement learning is a machine learning paradigm where agents learn to make decisions by interacting with an environment. Traditional RL algorithms like Q-learning and SARSA have been successful in discrete action spaces but struggle with continuous control problems due to scalability and exploration challenges. Advanced algorithms like TD3 and Options Critic have been developed to address these issues.

TD3 extends the Deep Deterministic Policy Gradient (DDPG) algorithm by addressing overestimation bias and improving policy stability. Options Critic leverages hierarchical RL by learning high-level "options" or macro-actions, enabling agents to plan more efficiently in complex environments. These methods are particularly effective for tasks that require precise control, such as autonomous parking.

The Unity ML-Agents toolkit bridges the gap between Unity-based simulations and Python-based RL frameworks, enabling researchers to train agents in highly realistic environments. By combining Unity's physics engine and ML-Agents, this project creates a robust platform for testing RL algorithms in a simulated parking lot.

Related Work

Prior research on autonomous parking has explored various approaches, from traditional control theory to machine learning techniques. Rule-based methods lack adaptability, while supervised learning struggles with generalization.

Reinforcement learning offers a promising data-driven approach. Earlier studies using Q-learning performed poorly in continuous control tasks. Recent algorithms like DDPG and PPO have limitations in convergence and overfitting. Advanced algorithms such as TD3 and Options Critic have improved policy stability and hierarchical planning.

The development of autonomous parking systems has seen significant advancements through the application of deep reinforcement learning (DRL) techniques, with many studies focusing on enhancing performance, robustness, and adaptability. Chan et al. [11] addressed the robustness of the Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm, demonstrating that using the Huber Loss function achieves faster convergence and reduces Q-value overestimation, a key challenge in reinforcement learning. Similarly, Yang et al. [12] highlighted TD3's adaptability and

resilience in diverse parking scenarios, showcasing its superior performance in complex environments with success rates up to 0.93 when combined with convolutional neural networks (CNNs).

The integration of advanced techniques has further improved reinforcement learning applications in parking. For example, Ertekin et al. [13] investigated the use of Hindsight Experience Replay (HER) to enhance the performance of TD3, DDPG, and SAC algorithms, enabling precise steering and throttle control in parking tasks. Du et al. [15], on the other hand, demonstrated that while both TD3 and DDPG are effective for high-precision parking path planning, DDPG excels under controlled conditions, emphasizing the importance of algorithm selection for specific use cases.

Innovative hybrid approaches have also emerged, such as the work by Shi et al. [14], which combined Model Predictive Control (MPC) for trajectory tracking and Proximal Policy Optimization (PPO) for reinforcement learning. This method achieved significantly faster convergence and reduced training times compared to TD3 and DDPG, offering a practical solution to the challenges of long training durations and slow convergence in trajectory planning.

Expanding beyond parking, Elsayed et al. [16] integrated TD3 with sensor fusion using Nvidia CNNs for local path planning, leveraging imitation learning to enhance safety and rule adherence. This approach demonstrates the versatility of TD3 in broader autonomous vehicle navigation tasks. Additionally, Soliman et al. [17] proposed a cooperative system combining TD3 for Adaptive Cruise Control (ACC) and DQN for Lane Keeping Assist (LKA), achieving effective velocity control and lane centering, showcasing the potential of multi-agent reinforcement learning in integrated systems.

These studies collectively underscore the versatility and efficiency of TD3 in various autonomous driving applications. The insights from these works, particularly the focus on improving convergence rates, robustness in complex environments, and hybrid methods, directly inform the design of our project, which aims to develop an autonomous car parking system using TD3 in Unity. By leveraging findings such as the effectiveness of Huber Loss [11] and the adaptability of TD3 in complex scenarios [12], our approach seeks to address key challenges in autonomous parking, enhancing precision and efficiency in real-world-like simulations.

Project Description

Environment Design

A custom parking lot environment was created in Unity, featuring realistic physics, sensor data, and environmental elements like other parked vehicles and obstacles. The Unity ML-Agents library was used to facilitate communication between Unity and Python, allowing the RL agents to train within this simulation. The environment provides continuous observations, including the car's position, Torque, and orientation.

Additionally, the vehicle is equipped with a ray perception sensor mounted on top, simulating a 360° LiDAR-like sensor. This sensor emits n equally spaced rays within its detection range, capable of identifying obstacles restricted to 20 units away. Each ray returns a scalar value representing the distance to the nearest detected obstacle. These sensor values allow the agent to perceive its surroundings and make decisions accordingly, enabling it to navigate around obstacles and parked vehicles within the parking lot.

The parking lot consists of 12 designated parking spots, with one spot kept empty at random in each episode, managed by a spawner script to simulate a dynamic environment. Additionally, the car agent is spawned at a random position and with a random orientation within a predefined area of the environment. This setup enhances the learning process by providing varied initial conditions for the agent, promoting better generalization of its learned policies.

State, Action and Reward Representation

1. **State Space**
   The state space S is a multidimensional space that captures the agent's perception and physical attributes. At each time step t, the state $s_t \in S$ is defined as:

   $$s_t = [r_1, r_2, \ldots, r_n, x, y, \tau, \theta, \phi]$$

   where

   - r_i (for i = 1, 2, …, n): Ray perception sensor float values, representing distances to the nearest obstacles detected by n sensors. These values are constrained to the range:

     $$r_i \in [0.0, 20.0], \forall \text{ i} \in \{1, 2, \ldots, \text{n}\}$$

   - (x, y): The position coordinates of the agent in a 2D plane, representing its location within the environment.
   - τ: The torque applied to the agent's motor or control system, restricted to the range:

     $$\tau \in [-30.0, 30.0]$$

   - θ: The orientation of the agent, represented as an angle in degrees:

     $$\theta \in [0, 360^o]$$

   - ϕ: The current steer angle of the agent, represented as an angle in degrees:

     $$\phi \in [-20.0, 20.0]$$

2. **Action Space**
   The action space A is a continuous space that defines the agent's control inputs at each time step t. The action $a_t \in$ A is represented as a two-dimensional vector:

   $$a_t = [\tau, \phi]$$

3. **Rewards**

(a) **TD3**

The reward structure for the RL agent in the autonomous car parking environment is designed to guide the agent towards desired behaviors while penalizing undesired actions.

- **Distance-based Reward:** The agent receives a small reward for moving closer to its target, with a controlling factor to prevent overshooting. A penalty is applied if it moves away, encouraging a steady approach.
- **Alignment Reward:** An alignment reward is based on the angle between the agent's forward direction and its direction towards the target. A lower angle earns a higher reward, guiding the agent to move straight towards its goal.
- **Smooth Driving Reward:** The agent receives a slight reward for smooth driving behavior, specifically when the steering angle is minimal and throttle is moderately applied. This promotes smoother turns and efficient driving maneuvers, reducing unnecessary steering corrections.
- **Velocity and Stationary Behavior:** A stationary penalty is applied if the agent remains in one place for too long. This incentivizes the agent to keep moving towards its goal rather than staying idle.
- **Obstacle and Ray Perception Rewards:** The agent receives a penalty for being too close to obstacles detected by a 360° LiDAR-like sensor. The closer the obstacle, the larger the penalty, encouraging effective navigation around obstacles. Penalty intensity increases with proximity to obstacles like walls or other cars.
- **Collision Penalties:** Collisions with walls or other cars result in a significant penalty, which is sustained during the collision and slightly decreased if the collision continues. These penalties discourage collisions and promote careful navigation around the environment.

This structured reward mechanism, combining positive rewards and penalties, guides the RL agent towards efficient and safe navigation, leading to successful parking behaviors within the custom parking lot environment.

(b) **Option-Critic**

The reward structure in the Options Critic framework is designed to encourage the agent to achieve sub goals corresponding to each option. At each time step t, the agent selects an option o $\in \{o_1, o_2, o_3, o_4\}$ and a specific reward function $R_{o_i}(s_t, a_t, o_i)$ is applied based on the chosen option and the transition. The reward design for each option is as follows:

- **Option 1** - Encourages the agent to explore the environment to find the Entrance of the Parking Lot:
  - A penalty is applied when the agent remains stationary or applies zero torque, discouraging inactivity.
  - A positive reward is given for applying positive torque, incentivizing movement and exploration.

- **Option 2** - Encourages the agent to move towards the entrance:
  - Reward is provided based on the agent's Euclidean distance to the entrance and its angular rotation towards the entrance, encouraging both proximity and proper orientation.
  - A positive reward is given for applying positive torque, promoting movement toward the entrance.

- **Option 3** - Encourages the agent to explore the parking lot to find an empty parking spot:
  - A positive reward is given for movement, encouraging the agent to explore the environment and search for an empty parking spot.
  - A penalty is applied for being stationary or applying zero torque, discouraging inactivity and promoting continuous exploration.

- **Option 4** - Encourages the agent to park into the empty spot.
  - A positive reward is given for reducing the Euclidean distance to the parking spot and aligning the agent's orientation with the parking spot.

Rewards Aggregation

At each time step, only one option is active, and the corresponding reward is computed based on the specific behavior associated with that option. The reward for the active option is calculated by summing the individual reward components for that option.

Only the reward from the active option is passed to the agent at each time step, ensuring that the agent receives feedback specific to the option it is executing. This approach allows the agent to focus on optimizing the behavior associated with the active option.

4. **State Transition**

The environment operates deterministically, meaning that the next state is fully determined by the current state and the action taken by the agent. At each time step, the agent performs an action, and based on this action, the environment transitions to the next state. The next state s_{t+1} is represented by the observations collected at that time step, which include the sensor readings, agent's position, torque, orientation, and steer angle.

$$s_{t+1} = T(s_t, a_t)$$

where

- s_t is the current state at time step t.
- a_t is the action taken at time step t.
- T represents deterministic transition function.

Algorithms

Two RL algorithms, TD3 and Options Critic, were implemented:

1. **TD3**
   - Utilizes two Q-networks to reduce overestimation bias.
   - Incorporates delayed updates to the policy and target networks for improved stability.
   - Adds noise to target actions to promote exploration.
2. **Options Critic**
   - Implements hierarchical RL by learning both high-level options and low-level policies.
   - Encourages temporal abstraction, allowing the agent to execute complex maneuvers efficiently.

Architecture

1. **TD3**
   The Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm forms the backbone of the autonomous parking agent's control system. TD3 is an enhancement of the Deep Deterministic Policy Gradient (DDPG) method, addressing its limitations through a series of improvements: employing two Q-networks for more accurate Q-value estimation, introducing a delayed policy update to stabilize training, and incorporating noise in the action space to encourage better exploration. These refinements make TD3 particularly well-suited for continuous action spaces, such as those required for autonomous parking.

   The architectural components of the TD3 framework are detailed below:

   Actor Network The Actor network is responsible for mapping the current state of the environment to continuous throttle and steering commands. It is a feedforward neural network with three fully connected layers:

   - **Input Layer**: Accepts the state vector, consisting of environment features such as the vehicle's position, orientation, velocities, and ray perception sensor data.
   - **Hidden Layers**: Two fully connected layers with 400 and 300 neurons, respectively, each followed by a ReLU activation function to introduce non-linearity.
   - **Output Layer**: Produces a vector representing the continuous actions (throttle and steering). The outputs are passed through a `tanh` activation function to ensure that the actions are bounded within the specified range. Scaling by `max_action_T` (maximum throttle) and `max_action_S` (maximum steering angle) maps the normalized outputs to real-world limits.

   Critic Network The Critic networks evaluate the Q-values for given state-action pairs, guiding the policy learning. TD3 employs two Critic networks to mitigate overestimation bias, with their predictions averaged during policy updates. Each Critic network comprises:

   - **Input Layer**: Combines the state vector and action vector into a single tensor for processing.
   - **Hidden Layers**: Two fully connected layers with 400 and 300 neurons, respectively, using ReLU activations.
   - **Output Layer**: A single neuron outputting the Q-value estimate.

   Target Network Updates To stabilize training, TD3 employs target networks for both the Actor and Critic. These target networks are soft-updated periodically using a weighted average of the main network parameters, controlled by the hyperparameter τ. This update mechanism reduces training instability caused by rapid changes in target values.

Algorithm 1: Target Network Update

Input: current network parameters θ, target network parameters θ', soft update parameter τ
Output: updated target network parameters θ'
$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
Return: θ'

Exploration Strategy TD3 introduces action noise for effective exploration. Gaussian noise is added to the actor's actions during training, promoting diverse experiences in the replay buffer.

Algorithm 2: Action Noise Generation

Input: mean μ, standard deviation σ
Output: noisy action a'
noise $\sim \mathcal{N}(\mu, \sigma)$
$a' \leftarrow a +$ noise
Return: a'

Policy Update (Delayed Actor Update) The Policy Update mechanism in TD3 introduces a strategic delay in updating the actor network relative to the critic networks. This approach aims to stabilize the learning process by reducing the frequency of updates to the actor policy, allowing the critic networks to converge more accurately before influencing the actor.

Algorithm 3: Policy Update (Delayed Actor Update)

Input: current actor parameters θ_{actor}, target actor parameters θ'_{actor}, critic parameters θ_{critic}, policy delay n_{policy}
Parameters: θ'_{actor}, θ_{critic}, n_{policy}
Output: updated actor parameters θ_{actor}
if current step number % n_{policy} == 0 **then**
  $\theta_{\text{actor}} \leftarrow \text{optimizer}_{\text{actor}}.\text{step}(\nabla_{\theta_{\text{actor}}} J(\theta_{\text{critic}}, \theta_{\text{actor}}))$
end if
Return: θ_{actor}

This architecture enabled the TD3 agent to exhibit robust performance, demonstrating effective learning and convergence in a complex autonomous parking environment.

The inclusion of dual Q-networks and noise-enhanced exploration significantly improved the agent's adaptability and stability, making it highly suitable for real-world deployment in autonomous vehicle systems.

2. **Option-Critic**
The architecture of the Options Critic framework consists of five key neural networks, each serving a distinct purpose to enable the agent to effectively choose, execute, and switch between options. These networks include: the Policy over Options Network, Intra-Option Policy Network, Termination Policy Network, State-Option Function Network, and State-Option-Action Function Network. Below is a detailed explanation of the purpose and structure of each network:

- **Policy Over Options:**
This network learns the high-level policy that determines which option (temporally extended action) the agent should select at each time step. This network determines which option the agent should execute, given a simplified set of input features. Unlike the original implementation in the Option Critic Architecture, this design separates the input to the Policy over Options network from the state input to other networks, reducing model complexity.

  Architecture:

  - **Input Layer:** The input comprises of three boolean values.
    - **IsEntranceFound**: Indicates whether the agent has identified the entrance to the parking lot.
    - **IsSpotFound:** Indicates whether the agent has identified an empty parking spot.
    - **IsInsideTheLot:** Indicates whether the agent is inside the parking lot.
  - **Hidden Layer:** A hidden layer with 8 neurons and ReLU activation function.
  - **Output Layer:** A softmax layer that outputs a probability distribution over the available options.

  Rationale for Design Change:
  - This modification simplifies the decision-making process at the level of option selection by excluding unnecessary information such as ray perception values, torque, or orientation.
  - The approach aligns with the observation that the high-level decision of which option to choose depends primarily on the agent's current task (e.g., locating the entrance or finding a parking spot), not on the detailed sensor values or control actions.
  - Reducing the input dimensionality decreases computational overhead and accelerates convergence during training, while still enabling the agent to make effective decisions.

- **Intra-Option Policy:**
This network learns the policy for each option, determining the actions the agent should take while executing that option.

  Architecture:
  - **Input Layer:** Takes the state and one-hot encoded option as input.
  - **Hidden Layer:** Two hidden layers with 64 and 16 neurons respectively.
  - **Output Layer:** A tanH layer that outputs two values representing torque and steer angle respectively. These values are scaled to their respective ranges before sending to the environment.

  To encourage exploration during training, Gaussian noise is added to the outputs of the intra-option policy network.

- **Termination Policy:**
This network learns when an option should terminate. It provides the termination probability for the selected option, determining whether the agent should continue or switch to another option.

  Architecture:
  - **Input Layer:** The input consists of the simplified state representation as described in the policy over options and the one-hot encoded option.
  - **Hidden Layer:** One hidden layer of 16 neurons with ReLU activation function.
  - **Output Layer:** A sigmoid layer that outputs one probability representing the likelihood of current option being terminated.

- **State-Option function(Q_Ω):**
This network estimates the value of each option in a given state, representing the expected cumulative return of selecting each option at a specific state. It is used by the critic to evaluate options.

  Architecture:
  - **Input Layer:** The input consists of the simplified state representation as described in policy over options and one hot encoded option.
  - **Hidden Layer:** One hidden layer with 16 neurons and ReLU activation function.
  - **Output Layer:** A single scalar value representing the Q-value of the current state and option pair.

- **State-Option-Action function(Q_U):**
This network estimates the expected cumulative return for a specific option and action pair, allowing the critic to guide the agent's learning on how well each action within an option contributes to long-term success.

  Architecture:

- **Input Layer:** The input consists of the state, one hot encoded option and the action.
- **Hidden Layer:** Two ReLU hidden layers with 128 and 64 neurons respectively.
- **Output Layer:** A single scalar value representing the Q-value of the current state-option-action pair.

Environment Dynamics

- **Episode Initialization:**
  - At the start of each episode, the agent is spawned at a random position and orientation within the environment.
  - The parking lot contains 12 parking spots, all of which are randomly occupied except for one designated empty spot.
- **Objective:** The agent's goal is to navigate to the parking lot, locate the empty parking spot, and park within it as efficiently as possible.
- **Episode Termination:** An episode ends when
  - The agent successfully parks in the designated empty spot.
  - A maximum time step limit of 1500 is reached, resulting in a failure.

Training Process

The training process leverages the Unity ML-Agents (ML-Agents) toolkit to facilitate communication between the Unity simulation environment and the reinforcement learning (RL) algorithms implemented in Python. The training was conducted in a headless mode to optimize computational resources, allowing the simulation to run without rendering the environment visually.

- **Action Selection**:
  Uses the current state to generate actions. The Actor processes the state to produce continuous actions, which are then scaled by their respective maximum values.
- **Learning Rate**:
  A learning rate of 1×10^{-3} is used for all networks in TD3 and 1×10^{-6} is used for all networks in Option-Critic, allowing a good balance between convergence speed and stability of the training process.
- **Delayed Updates**:
  - **TD3**
    The target networks are updated less frequently (every 2 steps) to prevent the critics from following the actor too closely, which can destabilize learning. This practice encourages more exploration during training.
  - **Option-Critic**
    The policy over options, Intra-option policy and termination policy are updated less frequently than the State-Option function and State-Option-Action function.
- **Policy Noise and Clipping**:
  Added noise to the actions during exploration (policy noise) helps the agent to explore more diverse actions and avoid falling into local minima. The noise is clipped to prevent it from becoming too large, which could destabilize the learning process.

- **Exploration Strategy**:
  To encourage exploration during training, action noise is added to the policy's actions:

$$a_t = \pi(s_t) + N(\mu, \sigma)$$

  where $N(\mu, \sigma)$ represents Gaussian noise with mean μ and standard deviation σ, enabling the agent to explore the action space effectively.
- **Loss and Gradient Flow**:
  - The back propagation process computes gradients for all the network's losses. These gradients are then used to update the respective networks.
  - For the critics, the gradient computation involves back propagating the loss with respect to the Q-values.

$$L_{\text{Critic}} = \frac{1}{2}\left[(Q(s,a) - y)^2\right]$$

  - For the networks other than critics, the gradient is derived from the Q-value's negative gradient with respect to the network's parameters to ensure actions improve the Q-values.

$$L_{\text{Actor}} = -E_s\left[\min Q'(s,a')\right]$$

- **End of Training**:
  - The training process continues for a set number of episodes. For each episode, the cumulative reward is calculated and stored, giving insight into the agent's learning progress over time. The loss values of the critic and actor, along with the rewards, are recorded for analysis and plotting purposes, allowing the visualization of learning curves and training stability.
  - After completing the training, the learned network is exported to .pt format. This format is useful for deployment in other environments or for integrating with other systems.

1. **TD3**
   - **Training Loop**:
     - The train method is invoked repeatedly during the training phase:
       * **Sampling from Replay Buffer**: A batch of experiences (states, actions, rewards, next states, done flags) is sampled from the replay buffer.
       * **Critic Loss Calculation**:
         · For each batch, noise is added to the actions of the next states (policy_noise) to simulate exploration. The noise is then clamped (noise_clip) to avoid extreme deviations.
         · The target actions are computed using the target Actor network on the next states, combined with the added noise.

· The targets (target_q1 and target_q2) from the critics are computed using these noisy actions and then combined (`torch.min`(target_q1, target_q2)) to provide a single target Q-value.

· The target Q-value for updating the critics is calculated using:

$$y = r + (1 - d) \times \gamma \times \min(Q'(s', a'))$$

where r is the reward, d is the done signal, γ is the discount factor, Q' is the target critic network, s' is the next state, and a' is the action sampled from the target actor.

· The critic loss is computed using Mean Squared Error (MSE) between the current Q-values (current_q1 and current_q2) and the computed target Q-values:

∗ **Target Networks Update**:

· Every few steps (as defined by target_update_interval), the target networks (actor_target, critic_1_target, critic_2_target) are updated using a soft update mechanism:

$$\theta_{\text{target}} \leftarrow \tau\theta_{\text{current}} + (1 - \tau)\theta_{\text{target}}$$

where θ_{current} are the current network parameters, θ_{target} are the target network parameters, and τ is a soft update parameter.

· This delayed policy update helps stabilize the training process by updating the target networks less frequently:

$$a_{t+1} = \pi(s_{t+1}) + N(\mu, \sigma)$$

where $N(\mu, \sigma)$ is noise sampled from a Gaussian distribution added to the action a_{t+1}, ensuring continuous exploration.

2. **Option-Critic**

- **Policy Optimization:**

– **Policy Over Options**
This policy determines which option to activate at a given state, using observations described earlier. It is trained to maximize the long-term discounted reward by encouraging the selection of the most appropriate option for the task.

– **Intra-Option Policy:**
This policy dictate the low-level actions (e.g., torque and steering angle) to be executed under each option. They are trained using gradient-based methods to maximize rewards within the option.

- **Termination Optimization:**
The termination policy learns when to terminate the current option. It is trained using the advantage function, which captures the value difference between continuing with the current option and switching to a new one.
- **Value Function Training:**

– he state-option function estimates the value of executing an option in a given state.

– The state-option-action function estimates the value of taking a specific action within an option.

– Both are optimized using Temporal Difference (TD) error to improve value estimation accuracy over time.

- **Loss Functions:**
The loss calculations for policy optimization, termination optimization, and value function updates are derived directly from the original Option-Critic architecture. These losses ensure that all components of the framework are jointly optimized.

This training methodology ensures that the agent learns both high-level strategies (e.g., choosing the right option) and low-level controls (e.g., precise maneuvering), enabling it to solve complex hierarchical tasks effectively.

Experiments

1. **TD3**

In this section, we discuss two distinct approaches that were employed to evaluate and compare the efficacy of reinforcement learning (RL) for the autonomous parking task using Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm. Each approach utilized different sets of observations and reward structures to control and guide the agent towards its goal of parking autonomously in a predefined environment.

Approach 1: Minimal Observations and Heuristic-based Reward Shaping

For the first approach, the RL agent was provided with a limited set of observations and no external guidance other than the immediate feedback from its environment. The observations consisted solely of:

- Agent's Normalized Local Position (X-axis)
- Agent's Normalized Local Position (Z-axis)
- Agent's Normalized Yaw Angle
- Agent's Normalized Speed
- Normalized Relative Position (X-component)
- Normalized Relative Position (Z-component)
- Normalized Relative Position Magnitude (Distance to Parking Spot)
- Agent's Forward Direction vector X-component
- Agent's Forward Direction vector Z-component

With these inputs, the agent's decisions were entirely based on its local spatial perception and immediate speed, steering angle, and relative position to the goal, with no external cues to aid its navigation.

- **Training Results:**

– Actor Loss plot shows:

∗ High variance

∗ Erratic spikes

– Reward plot indicates:

∗ Significant variance

∗ Difficulty accumulating consistent rewards

- **Learning Progression:**

- Frequent crashes into obstacles
- Poor navigation towards parking spot
- Low parking success rate
- Successful parking only in final training episodes

- **Key Insight:** Minimal observations and unstructured reward mechanism severely hindered effective learning.

Plots:

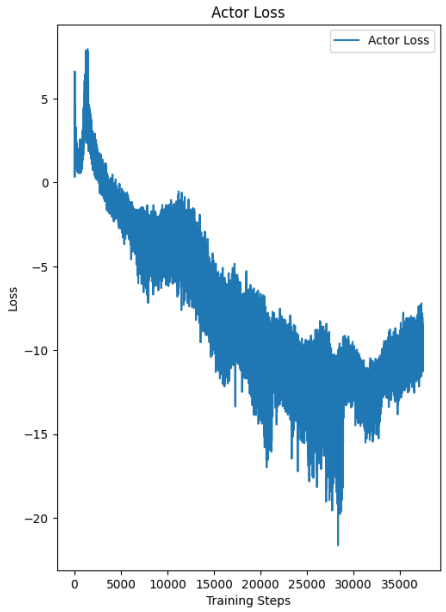

Figure 1: Actor Loss

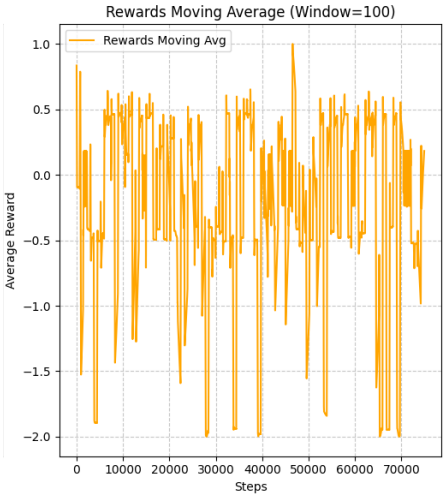

Figure 2: Average Rewards over episodes

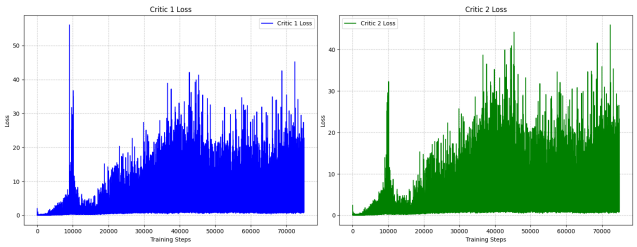

Figure 3: Critic 1 and 2 Losses

Approach 2: Enhanced Observations with Ray Perception and Targeted Reward Shaping

In the second approach, the agent was provided with a significantly richer set of observations and a more structured reward mechanism. A Ray Perception Sensor was integrated into the vehicle, providing 50 ray-based distance readings, simulating a 360° LiDAR-like sensor that detected obstacles within the environment. In addition to the basic observations used in the first approach, the agent also had access to:

- Agent's Normalized Local Position (X-axis)
- Agent's Normalized Local Position (Z-axis)
- Agent's Normalized Yaw Angle
- Agent's Normalized Speed
- Agent's Normalized Steer Angle

- **Reward Structure:**
  - Penalties for proximity to obstacles
  - Substantial reward for navigating to parking lot entrance and empty spot
- **Training Results:**
  - Actor Loss plot showed smooth, consistent downward trend
  - Episode rewards plot demonstrated clear upward trajectory
  - Agent successfully learned to:
    - Navigate to parking lot entrance
    - Reach empty parking spot
    - Avoid obstacles and parked vehicles
- **Learning Progression:**
  - Initial obstacle crashes were brief
  - Quick adaptation to safer navigation routes
  - Later training stages showed consistent successful parking
- **Key Insight:** Structured observations and well-defined reward mechanism significantly improved agent's learning effectiveness

Plots:

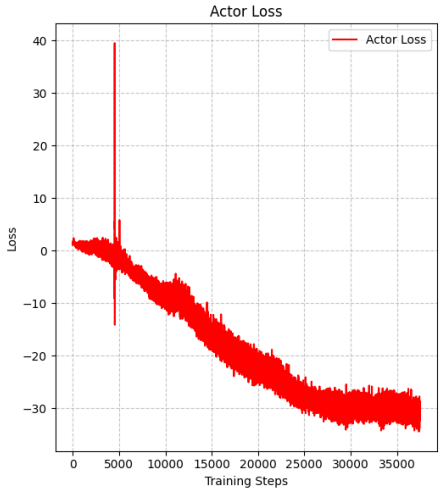

Figure 4: Actor Loss

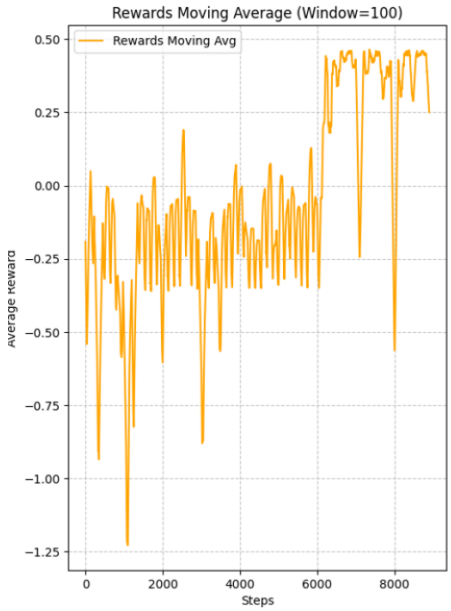

Figure 5: Average Rewards over episodes

Comparative Analysis

A comparative analysis between the two approaches reveals several key insights:

– **Approach 1**: It relied solely on local observations and did not incorporate any form of heuristic or reward shaping. The lack of guidance led to an unstable and ineffective learning process, resulting in minimal success in parking.

– **Approach 2**: By integrating Ray Perception Sensor data and targeted reward shaping, the agent's learning process became more structured. This approach allowed the agent to understand its environment better, adapt to obstacles, and follow a more informed path towards the parking spot. The inclusion of targeted rewards significantly improved the agent's success rate in parking, making it far more effective than the first approach.

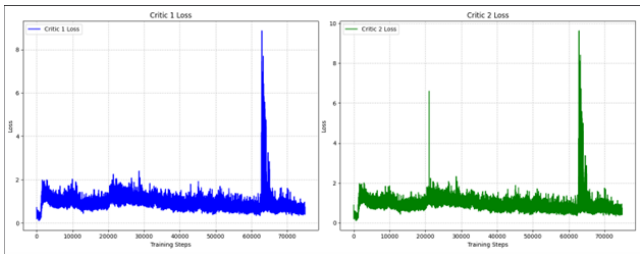

Figure 6: Critic 1 and 2 losses

– **Failure Circumstance**: Approach 1 failed consistently throughout the majority of the training episodes, with the agent often crashing into obstacles and not learning to avoid them until the final few episodes. Approach 2, while initially struggling with navigation due to tight environments, quickly adapted and succeeded in parking by focusing on avoiding obstacles and aligning with the parking spot. However, it occasionally faced challenges when the agent got stuck in tight situations, highlighting a need for more complex reward shaping and exploration strategies to handle such edge cases effectively.

Limitations and Edge Cases

Despite the improved performance of the second approach, several limitations were observed:

– **Partial Parking:** In some instances, the agent would only partially align with the parking spot, prematurely concluding the parking task.

– **Navigational Constraints:** The agent occasionally encountered difficulties when trapped in tight spaces, demonstrating limitations in complex navigational scenarios.

- **Option-Critic**

  The experiments aimed to assess the effectiveness of different reward strategies in training the Option-Critic architecture.

  – **Approach 1: Aggregating Rewards**

    * **Description:**
      · In the first approach, a single aggregated reward signal was used, combining the feedback from the policy over options, intra-option policy, and termination policy. This aggregated reward was intended to guide the learning of all components of the Option-Critic architecture simultaneously.
      · The reward signal for the agent was computed by summing the individual rewards from each component at each time step. This approach aimed

to provide a unified feedback mechanism to the agent, simplifying the reward structure.

* **Challenges:**
  - **Exploding Gradients:**
    The primary issue encountered during training was the occurrence of exploding gradients. This instability was seen when the gradients of the loss functions became excessively large during back propagation, leading to unstable updates to the neural networks and preventing effective learning.

  - **Difficulty in Back propagating:**
    Since all five networks were updated using the same reward signal, the gradients had to flow through all of them simultaneously. This setup resulted in difficulty in learning the individual functions of the Option-Critic architecture. Specifically, the model could not effectively disentangle the different learning signals required for each network, leading to insufficient training for certain components.

* **Trials:**
  - **Decreasing the Learning Rate:** To mitigate the gradient explosion, the learning rate was reduced, hoping that smaller updates would prevent the gradients from growing too large.

  - **Gradient Clipping:** Gradient clipping was employed to cap the maximum gradient size at a certain threshold, thereby preventing the gradients from becoming excessively large and causing instability.

* **Result:**
  - Despite these attempts, the exploding gradient problem was not fully resolved. The aggregated reward was not sufficiently informative to guide each of the networks effectively.

  - A single reward signal was not able to provide enough information to train each component of the model. Different components of the framework, such as the termination policy and the intra-option policy, required different types of feedback, and the single reward signal could not effectively guide them all.

  - The training became unstable, and the networks failed to converge to an optimal solution.

* **Plots:**

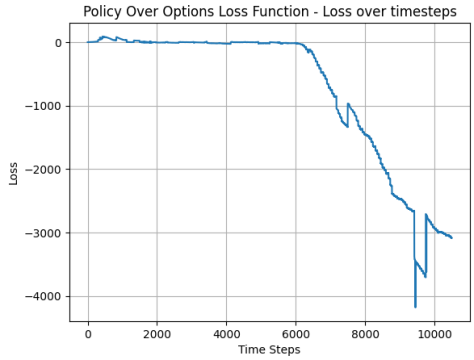

Figure 7: Policy Over Options - Loss over Time steps

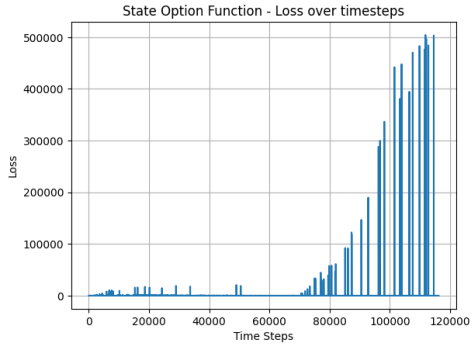

Figure 8: State-Option function - Loss over Time steps

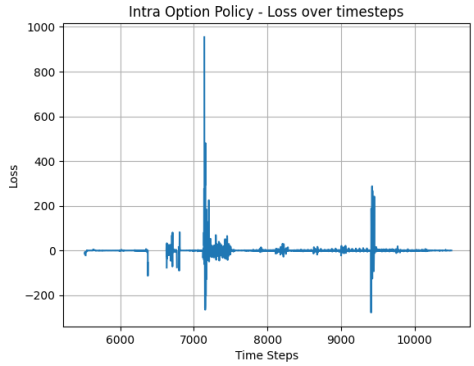

Figure 9: Intra-Option Policy - Loss over Time steps

– **Approach 2: Multiple Rewards**

* **Description:**
  - In the second approach, separate reward signals were used for each component of the Option-Critic framework. Specifically, the rewards were designed as follows:
  - **Policy Over Options:** This component received a reward signal based on how well it selected appropriate options for the agent at each time step.

- **Intra-Option Policy:** A separate reward was used to guide the low-level control of the agent, specifically the torque and steering angle actions.
- **Termination Policy:** A third reward signal was assigned to train the termination policy, encouraging the agent to determine the appropriate time to end an option and transition to another action or option.
- This approach aimed to provide more precise and targeted feedback to each component of the framework, improving the training process's overall stability and efficiency.

* **Challenges:**
  - Initially, there was concern that the additional reward signals could increase the complexity of the model, but this approach allowed for better control over the individual learning processes of each neural network.
  - The separation of rewards for each network mitigated the exploding gradient problem, as the gradient updates could now focus on specific tasks and were less likely to become unstable when passing through all the networks.
* **Result:**
  - This approach successfully resolved the exploding gradient problem. With separate rewards, each neural network could be trained independently, receiving feedback that was tailored to its specific function.
  - The individual rewards led to smoother and more stable training. The networks were able to converge more effectively, and the agent learned to perform the task (e.g., parking in the empty spot) more efficiently.
  - The training process showed consistent improvement in convergence rates and performance metrics, with the agent achieving better task completion over time.
* **Plots:**

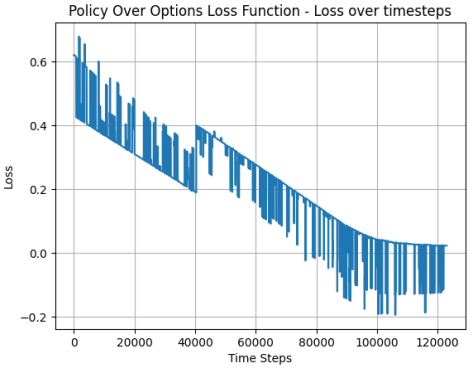

Figure 10: Policy Over Options - Loss over Time steps

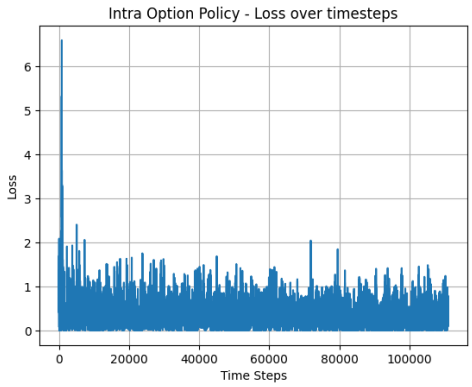

Figure 11: Intra-Option Policy - Loss over Time steps

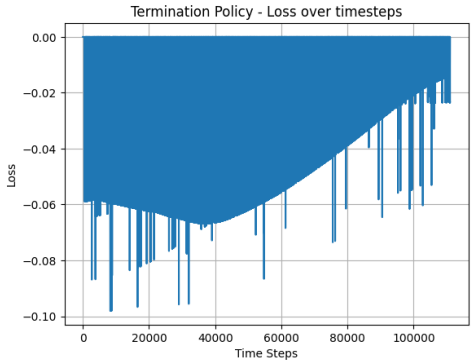

Figure 12: Termination Policy - Loss over Time steps

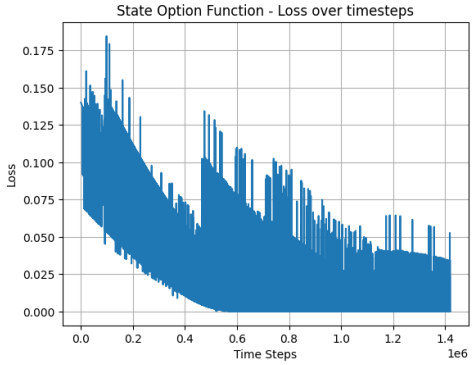

Figure 13: State-Option Function - Loss over Time steps

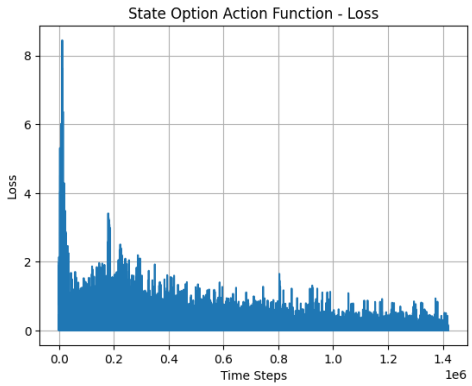

Figure 14: State-Option-Action Function - Loss over Time steps

- **Comparative Analysis**
  In Approach 1, a single aggregated reward signal was used, combining all components of the agent's actions. However, this approach had the issue of providing insufficient feedback for the individual networks, which hindered the learning process. A key problem was the occurrence of exploding gradients, which resulted in unstable training, despite efforts to mitigate this by decreasing the learning rate and clipping the gradients. This instability caused the agent to struggle in completing the task effectively, leading to poor convergence and oscillating loss curves.

  In contrast, Approach 2 used separate reward signals for the policy over options, intra-option policy, and termination policy. This strategy provided targeted feedback to each component, which helped eliminate the gradient instability observed in Approach 1. As a result, training was more stable, with smoother loss reduction and faster convergence. The agent demonstrated improved performance, reliably completing the task by parking in the empty spot, leading to higher task completion rates.

  While Approach 1 was simpler in terms of reward structure, it required additional steps to handle the gradient instability, which increased training time and complexity. Approach 2, although involving more complex reward management, resulted in better overall performance, with faster convergence and more reliable behavior from the agent.

  In conclusion, Approach 1 failed due to the insufficient reward feedback and unstable gradients, while Approach 2 proved to be successful, providing better stability, convergence, and performance.

- **Limitations:**
  * **Reward Shaping Complexity:**
    In Approach 2, where separate rewards are assigned for different components (policy over options, intra-option policy, and termination policy), the complexity of designing and tuning these individual reward signals increases. Care must be taken to balance these rewards so that they guide the agent effectively without introducing unintended biases.
  * **Scalability:**
    As the number of options or actions increases, Approach 2 may become harder to manage, especially if more separate rewards are needed. This can lead to greater computational overhead in terms of both training time and memory usage, as each policy and termination function may need to be computed independently.
  * **Task-Specific Generalization:**
    The reward structures in Approach 2 are designed specifically for the parking lot environment. While they work well in this context, they might not generalize well to other types of environments where task-specific rewards are harder to define.

Comparative Analysis of TD3 and Options Critic Approaches

TD3 and Options Critic offer distinct methods for tackling reinforcement learning tasks, particularly in environments like parking lot navigation, which require sequential decision-making.

- **Learning Strategy:** TD3 directly optimizes policies using continuous action spaces. In Approach 1, TD3 struggles with minimal observations and heuristic-based rewards, leading to high variance and slow learning. Approach 2 improves learning with enhanced observations and structured rewards, resulting in smoother convergence. In contrast, Options Critic uses temporal abstraction with options (high-level actions), allowing the agent to focus on long-term goals, making it more effective for complex tasks.
- **Reward Mechanism:** TD3 relies on immediate rewards, which in Approach 1 leads to sparse feedback and inconsistent training. Approach 2 improves this with additional sensors and structured rewards. Options Critic benefits from multiple reward signals for different decision-making layers (policy over options, intra-option, and termination), leading to more stable learning and better performance in complex environments.
- **Training Efficiency:** TD3 faces challenges like exploding gradients, especially with sparse reward signals, and requires careful tuning. Approach 1 suffers from poor learning efficiency, while Approach 2 shows improved convergence. Options Critic is more efficient in learning long-term tasks due to its hierarchical structure, which allows the agent to learn complex sequences more effectively.
- **Exploration and Exploitation:** TD3 explores via noise in the action space, but struggles with sparse

observations in Approach 1. Approach 2 improves exploration with richer inputs. Options Critic provides better exploration by using multiple policies and temporal abstraction, enabling more efficient handling of exploration-exploitation trade-offs.

– **Generalization and Adaptability:** TD3 requires significant data and careful tuning to generalize across tasks. Options Critic handles generalization better due to its hierarchical task decomposition, enabling it to adapt more easily to changes in the environment.

Conclusion

The research on autonomous parking using reinforcement learning revealed critical insights into the importance of observation design and reward mechanisms. By comparing two distinct approaches—one with minimal observations and an unstructured reward system, and another with enhanced ray perception and targeted reward shaping—we demonstrated that the complexity and specificity of the learning environment significantly impact an agent's ability to successfully complete autonomous navigation tasks. The first approach, which relied solely on local observations without external guidance, resulted in unstable learning, frequent crashes, and minimal parking success. In contrast, the second approach, which integrated a 360-degree ray perception sensor and implemented a more nuanced reward structure, enabled the agent to learn effective navigation strategies, avoid obstacles, and consistently park in designated spots.

Furthermore, our exploration of the Option-Critic architecture highlighted the critical role of reward design in machine learning. By initially using a single aggregated reward signal, we encountered significant challenges such as exploding gradients and insufficient learning feedback across different network components. Transitioning to an approach with separate reward signals for policy over options, intra-option policy, and termination policy proved transformative, leading to smoother training, faster convergence, and more reliable agent performance. While this method introduced increased complexity in reward shaping and potential scalability challenges, it demonstrated the profound impact of carefully designed reward mechanisms on reinforcement learning outcomes. The project underscored the nuanced nature of training autonomous systems, emphasizing that success depends not just on the algorithm's architecture, but on the thoughtful construction of the learning environment and reward signals.

GitHub Links

- TD3 Implementation: Autonomous Car Parking RL - TD3
- Option-Critic Implementation: Autonomous Car Parking RL - Option-Critic

References

1. Takehara, R., Gonsalves, T. (2021, September). Autonomous car parking system using deep reinforcement learning. In 2021 2nd International Conference on Innovative and Creative Information Technology (ICITech) (pp. 85-89). IEEE.
2. Maravall, D., Patricio, M. Á., de Lope, J. (2003). Automatic car parking: a reinforcement learning approach. In Computational Methods in Neural Modeling: 7th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2003 Maó, Menorca, Spain, June 3–6, 2003 Proceedings, Part I 7 (pp. 214-221). Springer Berlin Heidelberg.
3. Folkers, A., Rick, M., Büskens, C. (2019, June). Controlling an autonomous vehicle with deep reinforcement learning. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 2025-2031). IEEE.
4. Thunyapoo, B., Ratchadakorntham, C., Siricharoen, P., Susutti, W. (2020, June). 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.
5. Zhang, J., Chen, H., Song, S., Hu, F. (2020). Reinforcement learning-based motion planning for automatic parking system. IEEE Access, 8, 154485-154501.
6. Song, S., Chen, H., Sun, H., Liu, M. (2020). Data efficient reinforcement learning for integrated lateral planning and control in automated parking system. Sensors, 20(24), 7297.
7. Khalid, M., Wang, L., Wang, K., Aslam, N., Pan, C., Cao, Y. (2023). Deep reinforcement learning-based long-range autonomous valet parking for smart cities. Sustainable Cities and Society, 89, 104311.
8. You, C., Lu, J., Filev, D., Tsiotras, P. (2019). Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. Robotics and Autonomous Systems, 114, 1-18.
9. Awaisi, K. S., Abbas, A., Khattak, H. A., Ahmad, A., Ali, M., Khalid, A. (2023). Deep reinforcement learning approach towards a smart parking architecture. Cluster Computing, 26(1), 255-266.
10. Junzuo, L., Qiang, L. (2021, April). An automatic parking model based on deep reinforcement learning. In Journal of Physics: Conference Series (Vol. 1883, No. 1, p. 012111). IOP Publishing.
11. Chan, K. H., Mustapha, A., Jubair, M. A. (2024). Comparative Analysis of Loss Functions in TD3 for Autonomous Parking. *Journal of Soft Computing and Data Mining, 5*(1), 1-14.
12. Yang, Z., Tang, J., Cai, L. (2024, August). Multi-scenario Automatic Parking Based on Deep Reinforcement Learning. In *International Conference on*

Traffic and Transportation Studies (pp. 481-488). Singapore: Springer Nature Singapore.

13. Ertekin, M., Önder Efe, M. (2021, October). Autonomous Parking with Continuous Reinforcement Learning. In *The International Conference on Artificial Intelligence and Applied Mathematics in Engineering* (pp. 25-37). Cham: Springer International Publishing.
14. Shi, J., Li, K., Piao, C., Gao, J., Chen, L. (2023). Model-Based predictive control and reinforcement learning for planning vehicle-parking trajectories for vertical parking spaces. *Sensors, 23*(16), 7124.
15. Du, M. (2022, October). Research on the parking planning algorithm based on DDPG and TD3. In *International Conference on Cloud Computing, Performance Computing, and Deep Learning (CCPCDL 2022)* (Vol. 12287, pp. 130-136). SPIE.
16. Elsayed, M. A. M. (2024). Navigating the Rules: Integrating TD3 and Sensor Fusion for Traffic-Aware Autonomous Vehicle Path Planning.
17. Soliman, T. M., Elshenawy, A., Tantawy, H. (2024). Adaptive Cruise Control with Lane Keeping Assist Using Reinforcement Learning. *Journal of Al-Azhar University Engineering Sector, 19*(73), 1349-1368.
18. Bacon, P. L., Harb, J., Precup, D. (2017, February). The option-critic architecture. In Proceedings of the AAAI conference on artificial intelligence (Vol. 31, No. 1).
19. Pateria, S., Subagdja, B., Tan, A. H., Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. ACM Computing Surveys (CSUR), 54(5), 1-35.
20. Botvinick, M. M. (2012). Hierarchical reinforcement learning and decision making. Current opinion in neurobiology, 22(6), 956-962.
21. Barto, A. G., Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. Discrete event dynamic systems, 13, 341-379.
22. Dann, C., Mansour, Y., Mohri, M. (2023, July). Reinforcement learning can be more efficient with multiple rewards. In International Conference on Machine Learning (pp. 6948-6967). PMLR.
23. Shelton, C. (2000). Balancing multiple sources of reward in reinforcement learning. Advances in Neural Information Processing Systems, 13.
24. Agarwal, M., Aggarwal, V. (2023). Reinforcement learning for joint optimization of multiple rewards. Journal of Machine Learning Research, 24(49), 1-41.