Deep Reinforcement Learning on a Parking Environment

CS 5180 Final Project Report

Atharva Jamsandekar
Northeastern University
Boston, MA
jamsandekar.a@northeastern.edu

Aditya Aspat
Northeastern University
Boston,MA
aspat.a@northeastern.edu

***Abstract*—Autonomous parking systems have garnered significant attention in recent years due to their potential to enhance driving safety, convenience, and efficiency. Deep Reinforcement Learning (Deep RL) has emerged as a promising approach for training agents to navigate complex environments and make optimal decisions in real-time. This report provides a comprehensive review of how Deep RL techniques can be utilized for autonomous parking. The report begins by outlining the fundamental challenges of autonomous parking, including continuous state space, decision-making, and continuous action space. It then presents an overview of Deep RL, highlighting its ability to learn complex behaviors through interaction with the environment and reward feedback. Various Deep RL architectures, including Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC), are discussed in the context of autonomous parking.**

Keywords *- (Reinforcement Learning, Deep RL, continuous spaces)*

I. INTRODUCTION AND MOTIVATION

A. *Background*

Parking tasks pose a significant challenge for autonomous vehicles, requiring precise navigation to designated spots. Reinforcement learning (RL) provides a promising avenue for training ego-vehicles to master parking maneuvers autonomously. In RL-based approaches, ego-vehicles learn parking policies through interaction with the environment, leveraging algorithms like Deep Deterministic Policy Gradient (DDPG) or Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC) to navigate to specified spots with the correct orientation. RL enables vehicles to adapt to diverse parking scenarios and environmental conditions, offering flexibility in reward design and exploration strategies. Ultimately, RL empowers autonomous vehicles to achieve efficient and safe parking in urban environments, contributing to the advancement of autonomous driving technology.

B. *Problem statement*

The objective is to design and implement a goal-conditioned continuous control task for an ego-vehicle, requiring it to autonomously navigate and park within a predefined space while aligning with the specified heading.

C. *Introduction to DDPG*

Deep Deterministic Policy Gradient (DDPG) combines elements of both deep learning and policy gradients. It is designed specifically for environments with continuous action spaces, making it suitable for a wide range of tasks including robotics, control systems, and games.

Architecture: The architecture of DDPG consists of two main components: an actor network and a critic network. The actor network takes the current state as input and outputs a continuous action, representing the agent's policy. The critic network, on the other hand, takes both the state and the action as input and predicts the expected return, which is used to evaluate the action taken by the actor.

Key Features: One of the key features of DDPG is its use of target networks. These are copies of the actor and critic networks periodically updated with the main networks' parameters. This helps stabilize training by providing more consistent target values for the critic and reducing the likelihood of overestimation.

Exploration: Exploration in DDPG is achieved through adding noise to the actions selected by the actor network. This noise encourages the agent to explore different actions, allowing it to discover potentially better policies. Commonly used noise sources include Ornstein-Uhlenbeck noise or simple random noise sampled from a distribution.

D. *Introduction to PPO*

Proximal Policy Optimization (PPO) is a versatile algorithm tailored to seamlessly navigate continuous action spaces, rendering it indispensable across a spectrum of domains including robotics, control systems, and gaming scenarios. PPO differs from SAC and DDPG as it is an on-policy algorithm, while the other two are off-policy algorithms.

Architecture: PPO's architecture revolves around a pivotal actor-critic framework, comprising an actor network and a critic network. The actor network interprets the current state to produce actions, effectively shaping the agent's policy. Simultaneously, the critic network evaluates these actions in the context of the state, offering insights into the expected return and guiding the actor's decision-making process.

Key Features: PPO has the distinctive advantage in balancing exploration and exploitation through clipped objectives. This approach ensures stable and efficient training by constraining policy updates within a proximity threshold, preventing drastic policy changes that could hinder learning progress. Moreover, PPO integrates an adaptive learning rate mechanism, dynamically adjusting learning rates to suit the

training dynamics, thereby enhancing convergence speed and robustness.

Exploration: PPO apart from the clipping feature doesn't have a direct way to promote exploration. The usual method of promoting exploration involves perturbing the actor's policy through additive noise or stochastic policies, encouraging the agent to explore alternative strategies and robustly adapt to various environmental dynamics. These exploration techniques enable PPO to effectively navigate complex and continuous action spaces, facilitating efficient learning and adaptation.

*E. **Introduction to SAC***

The Soft Actor-Critic (SAC) algorithm stands out as a versatile approach tailored for continuous action spaces, ideal for this domain involving continuous action space of steering & acceleration. SAC uses a dual-critic architecture that allows to control variance in the updates to the policy.

Architecture: SAC is comprised of two pivotal components: an actor network and two distinct critic networks. The actor network interprets the current state to produce continuous actions, effectively shaping the agent's policy. Meanwhile, the critic networks evaluate the state-action pairs, considering the expected return, guiding the actor towards optimal decision-making.

Key Features: SAC also uses target networks to enhance stability during updates like DDPG. These target networks are synchronized with the primary ones using the soft update technique, contributing to training stability by furnishing consistent target values for the critics. It helps mitigate overestimation concerns, ensuring smoother convergence and more reliable performance.

Exploration: SAC's exploration strategy is mainly based on leveraging entropy maximization. Entropy maximization fosters exploration while optimizing policy entropy, bolstering adaptability and robustness. The Entropy term is part of the Loss function of SAC that promotes exploration. Along with Entropy maximization, there are flavors of SAC that include additive noise in action selection.

*F. **Feature Association***

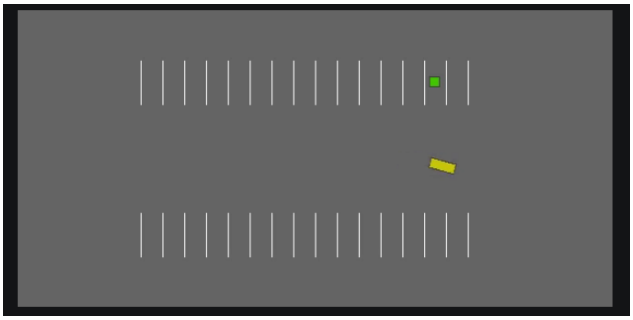
Feature association using Semi-Gradient SARSA is a reinforcement learning technique aimed at learning value functions in large state spaces with limited computational resources. It is particularly useful when dealing with high-dimensional input spaces by using feature representations instead of the raw state space. Feature association refers to the association of features of the state space with their corresponding values. In Semi-Gradient SARSA, this is achieved through a linear function approximation, where each feature is associated with a weight that represents its importance in predicting the value of a state-action pair.

Features: The architecture of Semi-Gradient SARSA involves defining a set of features that capture relevant information about the state space. These features can be handcrafted based on domain knowledge or learned automatically from the data using techniques such as neural networks or kernel methods. Once the feature representation is established, Semi-Gradient SARSA uses a linear function approximation to estimate the value function. This approximation takes the form of a weighted sum of the features, where the weights are updated iteratively through the learning process.

Exploration: Typically achieved through epsilon-greedy action selection, where with probability epsilon, a random action is chosen to encourage exploration, and with probability 1 - epsilon, the action with the highest estimated value is selected.

II. SIMULATION ENVIRONMENT

The simulation environment used for this task is the Parking Environment that is a part of OpenAI Gymnasium. The task at hand is to attain a desired position and orientation of the vehicle in the parking lot.



The observation space is of type: "KinematicsGoal". In this case, it suggests that the observations represent kinematic information related to achieving a specific goal (the parking lot goal position).

The state features are: ['x', 'y', 'vx', 'vy', 'cos_h', 'sin_h']

These are the features or variables included in each observation. Each feature provides specific information about the state of the environment. Here's a breakdown:

'x': The x-coordinate position.

'y': The y-coordinate position.

'vx': The velocity along the x-axis.

'vy': The velocity along the y-axis.

'cos_h': The cosine of the heading direction.

'sin_h': The sine of the heading direction.

In reward calculation, a weighted Lp norm measures the distance between achieved and desired goal. The achieved and desired goals are of the state vector format. The rewards are weighted according to a tunable "reward_weights" parameter.

The simulation environment has the following additional features:

- Adding walls between the lots.
- Adding parked vehicles.

III. PROPOSED SOLUTION

A. Implementation of DDPG

ARCHITECTURE:

Actor Model: The actor neural network generates actions based on observed states. It is made up of three completely connected layers: an input layer with the size of the state space, two hidden levels with 512 and 256 units each, and an output layer with the size of the action space. The hidden layers use ReLU activation functions to introduce non-linearity, while the output layer is activated by a hyperbolic tangent (tanh) function scaled by the maximal action value.

Critic Model: The critic neural network measures the quality of state-action pairs by estimating Q-values. It takes both the state and the action as input, concatenates them, and feeds them through three fully connected layers: the input layer, which is the size of the concatenated state and action spaces, followed by two hidden layers of 512 and 256 units, and the output layer, which is a single unit representing the Q-value. The hidden layers use ReLU activations.

Target Networks: Two sets of target networks, one for the actor and one for the critic, are implemented to stabilize training. These target networks are copies of the main networks and are periodically updated with soft updates to their parameters.

FEATURES:

Replay Buffer: The replay buffer holds the experiences observed by the agent when interacting with the environment. It is constructed as a deque with a defined capacity, allowing for efficient random sampling of events during training. This allows the agent to break down correlations between consecutive encounters, facilitating off-policy learning.

Ornstein-Uhlenbeck Noise: Ornstein-Uhlenbeck noise is added to the agent's actions to facilitate exploration. This noise mechanism generates temporal correlations between consecutive actions, allowing for smoother exploration of the action space. It is parameterized by scale, mean (mu), theta, and sigma, which allows for greater control over the exploration process.

B. Implementation of PPO

ARCHITECTURE:

Actor Model: The actor neural network generates actions based on observed states. It is made up of three completely connected layers: an input layer with the size of the state space, two hidden levels with 512 and 256 units each, and an output layer with the size of the action space. The hidden layers use hyperbolic activation functions to introduce non-linearity, also the output layer is activated by a hyperbolic tangent (tanh) function scaled by the maximal action value.

Critic Model: The critic neural network measures the quality of state-action pairs by estimating Q-values. It takes both the state and the action as input, concatenates them, and feeds them through three fully connected layers: the input layer, which is the size of the concatenated state and action spaces, followed by two hidden layers of 512 and 256 units, and the output layer, which is a single unit representing the Q-value. The hidden layers use hyperbolic tangent (tanh) activations.

FEATURES:

Replay Buffer: The replay buffer holds the experiences observed by the agent when interacting with the environment. It is constructed as a deque with a defined capacity, allowing for efficient random sampling of events during training. This allows the agent to break down correlations between consecutive encounters, facilitating off-policy learning.

Clipping Objective: The updates according to the Loss Objective function are clipped using the parameter Epsilon to avoid large updates that are unstable and sub-optimal. The Advantage value is calculated according to the Monte-Carlo return of the timestep and is multiplied by the ratio of log-odds of the probability of the action before and after update. The update is then clipped with epsilon which eliminates high-variance updates.

C. Implementation of SAC

ARCHITECTURE:

Actor Model: The actor neural network generates actions based on observed states. It is made up of three completely connected layers: an input layer with the size of the state space, two hidden levels with 256 units each, and an output layer with the size of the action space. The hidden layers do not use activation, while the output layer is activated by a hyperbolic tangent (tanh) function scaled by the maximal action value.

Critic Model: The critic neural network measures the quality of state-action pairs by estimating Q-values. SAC uses a bi-critic architecture to increase stability in the updates. The update is calculated according to the Critic network which has the lowest Loss Objective Value. The Critic models take both the state and the action as input, concatenates them, and feeds them through three fully connected layers: the input layer, which is the size of the concatenated state and action spaces, followed by two hidden layers 256 units each, and the output layer, which is a single unit representing the Q-value. The hidden layers use ReLU activations.

Target Networks: Three sets of target networks, one for the actor and two for the two critics, are implemented to stabilize training. These target networks are copies of the main networks and are periodically updated with soft updates to their parameters according to the hyperparameter ‘tau’.

FEATURES:

Replay Buffer: The replay buffer holds the experiences observed by the agent when interacting with the environment. It is constructed as a deque with a defined capacity, allowing for efficient random sampling of events during training. This

allows the agent to break down correlations between consecutive encounters, facilitating off-policy learning.

Entropy Maximization: It serves as a clever strategy to inject randomness into decision-making. By maximizing entropy, SAC encourages exploration by making its decisions less predictable, leading to a more diverse range of actions tried out. SAC's temperature parameter governs the degree of randomness injected into its policy, allowing for fine-tuning of exploration versus exploitation. The parameter alpha controls the temperature.

D. Implementation of Feature Association

ARCHITECTURE

Q-Value Calculation (get_q_value): This function calculates the Q-value for a given state-action pair using linear function approximation. It takes the feature representation of the state, weight vector corresponding to the action, and state-action pair as inputs. The Q-value is computed as the dot product of the weight vector and the feature representation.

Action Selection (car_e_greedy): The car_e_greedy function implements an epsilon-greedy policy for action selection. It selects a random action with probability epsilon and the action with the highest estimated Q-value otherwise. This strategy balances exploration and exploitation during the learning process.

State Discretization (discretize_state): This function discretizes the continuous state space into a finite set of states. It maps each dimension of the state to its corresponding bin index, ensuring that the state falls within the valid range of bins.

Feature Extraction (feature_x): The feature_x function extracts features from the discretized state representation. It generates a one-hot encoded feature vector based on the index of the discretized state, representing the presence of a state in the state space.

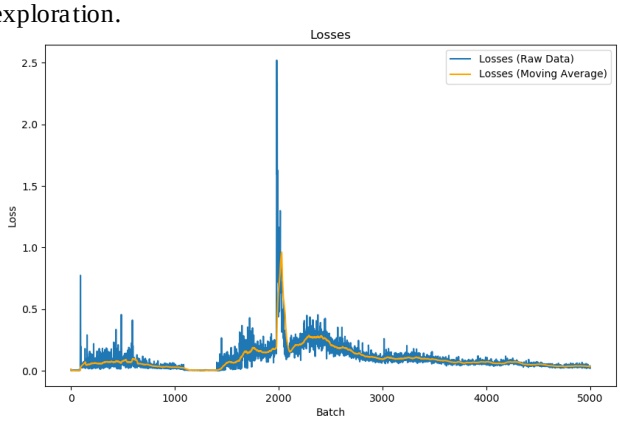
Action Step (stepper): The stepper function updates the agent's current speed based on the selected action. It increments or decrements the agent's velocity along the x and y axes according to the chosen action, ensuring that the speed remains within the valid range [-1, 1].

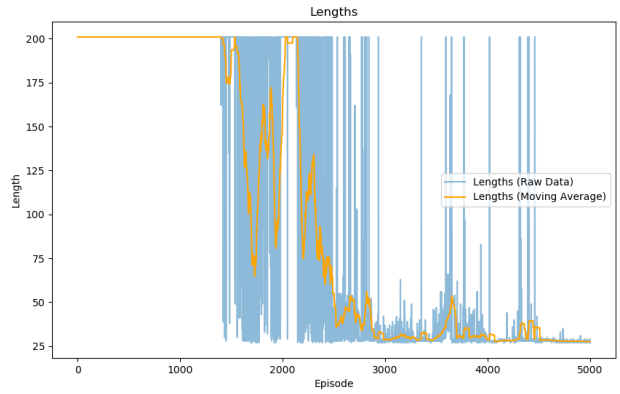
IV. RESULTS

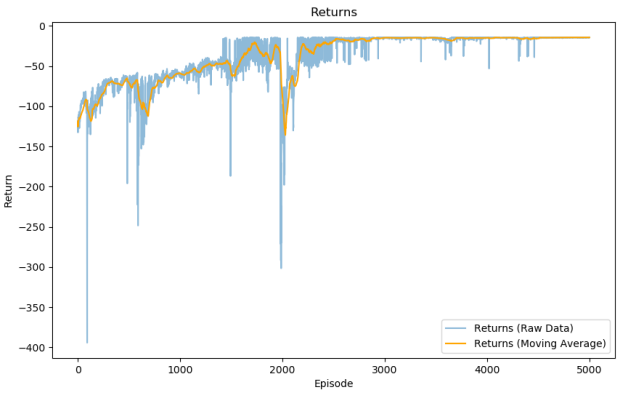
The Results of the discussed algorithms are compared in terms of their losses, returns and episode lengths during their training. All the algorithms are trained for 5000 episodes with a timeout of 200 timesteps. The episode ends if the goal is reached or due to timeout. Section VII has the video links which show the compilation of the training frames every 500 episodes (i.e. 10 episodes in total). The video provides a more intuitive way to analyze the agent's progress throughout the training.

A. DDPG

DDPG shows the best performance amongst all the other algorithms as it makes drastic progress around the 1500-episode mark and learns the optimal policy around the 2400-episode mark. The slight variance in the plots after learning optimal policy, are due to the inclusion of the OU Noise for exploration.

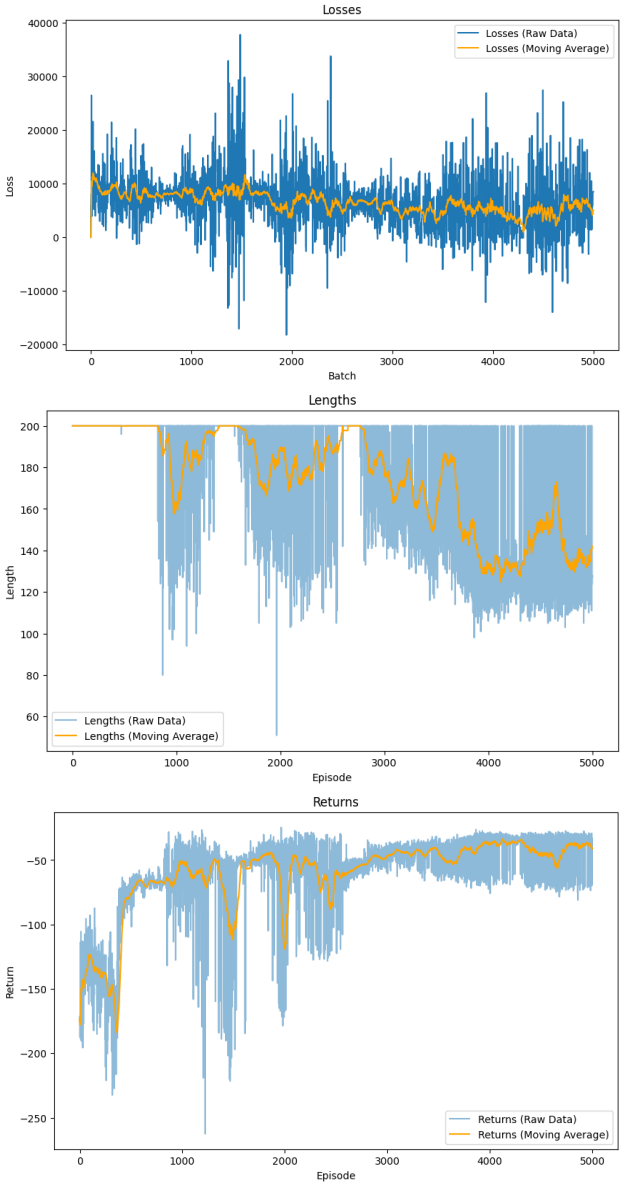


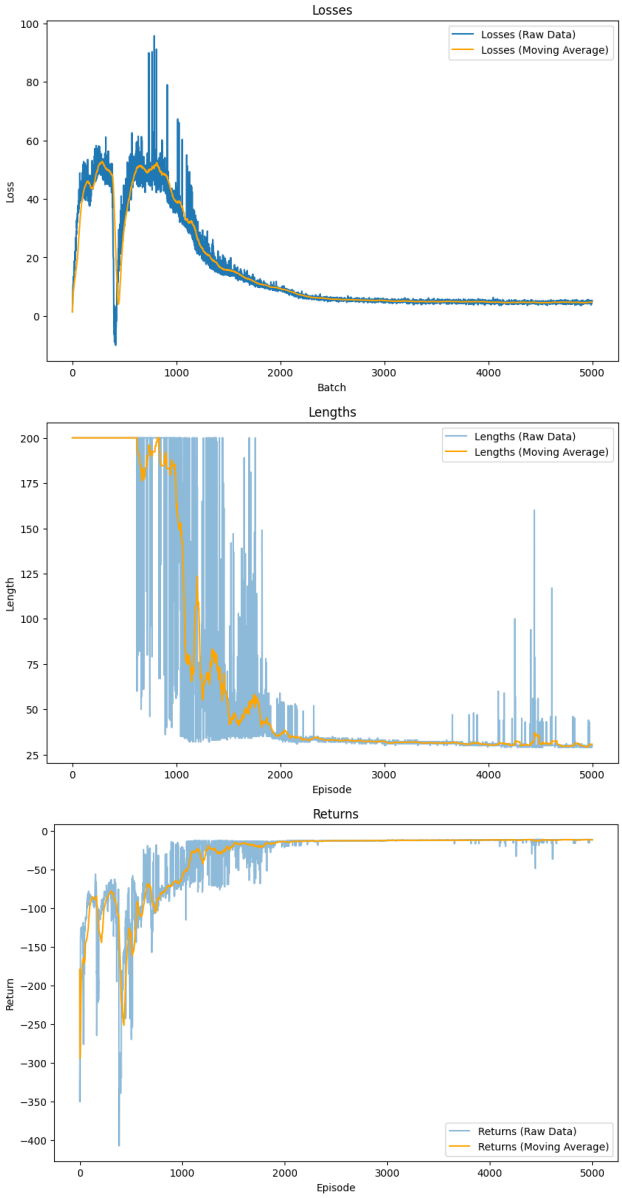




B. PPO

The performance of PPO is mediocre compared to DDPG as it has high fluctuations in its Loss function values making it highly unstable. It is believed that this is due to a sub-optimal clipping hyperparameter selection (Epsilon) that leads to the high fluctuations. The PPO algorithm is rather conservative in its final policy which is why it accumulates a higher negative reward and runs for a longer episode length. It can be concluded that the final policy is near-optimal and training it for more episodes may lead to better results.



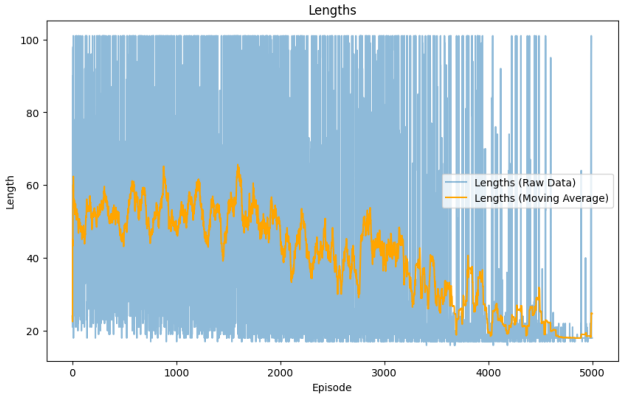


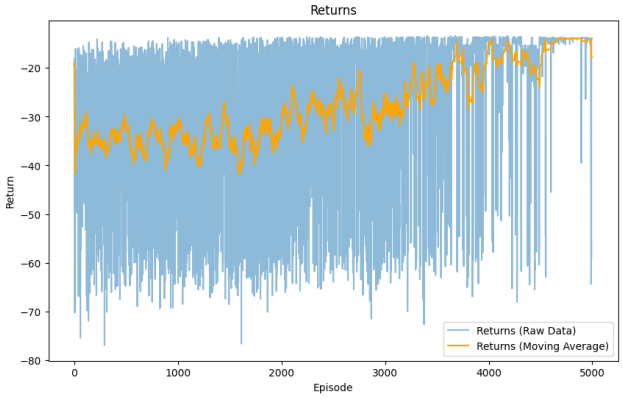
C. SAC

SAC shows promising performance with better results than PPO but still not as good as DDPG. Like DDPG, the agent learns the optimal policy around the 2000-episode mark. Contrary to DDPG, it shows less variance after learning the optimal policy. The SAC and DDPG agent show the best performance compared to the other two agents and are differentiated by a large margin.

D. Feature Association

Feature Association fails to learn a policy that leads it towards the goal. In this case, walls were added to give it initiative through rewards to stay in the parking lot. The final policy it learnt after 5000-epsiodes was to keep circling around the plot to avoid accruing the high collision penalty. The reason for the poor learning curve was the lack of orientation and velocity information that was encoded while discretizing the features. This led to it learning to only stay within the perimeter of the lot.





V. CHALLENGES FACED

Attempting feature association by discretizing continuous state and action space posed significant challenges, primarily due to the vast number of resulting states. Discretization led to a state space explosion, exacerbating the curse of dimensionality and resulting in a loss of precision. The agent struggled with sparse rewards and limited generalization, hindering its ability to learn robust policies. While discretization may be suitable for low-dimensional environments, it proved impractical for high-dimensional and complex domains. Alternative methods, such as function approximation using neural networks or kernel methods, are needed to effectively handle continuous state and action spaces in reinforcement learning.

VI. CONCLUSION AND FUTURE WORK

The implemented algorithms show a varying level of performance with DDPG performing qualitatively the best, while the Feature Association lacks the diversity in features to navigate through diverse orientations. The Deep RL approaches show good performance for this domain involving continuous observation and action space.

The future work in the Parking environment domain should include the inclusion of additional configurations like parking walls and parked vehicles to add an additional layer of complexity to the problem solved in the scope of this current project. Naively applying the current approaches doesn't provide desirable results. Considerable work is needed to encode additional information about possible collisions in the model's observation space, possibly through parking sensors.

The work in this project forms the basis for extending the work in the Autonomous vehicle research domain. Some interesting problems for the team members include:

- Learning Racing Maneuvers like Blocking, Nudging and Overtaking for Autonomous Racing.
- Exploration Algorithms for a Reconnaissance Robot.
- Behavior Planning for Lane Switching and Obstacle Avoidance.

The above-mentioned problems are not exhaustive but have similar interesting features of continuous observation and action space.

VII. LINKS

Codes:

DDPG
PPO
SAC
Feature Association

Videos:

DDPG
PPO
SAC
Feature Association

VIII. REFERENCES

https://spinningup.openai.com/en/latest/algorithms/ddpg.html
https://openai.com/research/openai-baselines-ppo
https://spinningup.openai.com/en/latest/algorithms/sac.html
https://michaeloneill.github.io/RL-tutorial.html
https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b
https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b
https://github.com/denisyarats/pytorch_sac