Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Evaluation of Soft Actor Critic in Diverse Parking Environments

Pratush Pandita

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Computer Science (Data Science)

Supervisor : Prof Vinny Cahill

August 19, 2022

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Signed: _____

Abstract

Parking vehicles inside a parking lot is an intricately complex process. Firstly the vehicle needs to find empty slots and choose the most appropriate one from them. Next, after detecting the spot, the vehicle needs to plan a trajectory from its current position to the empty slot. After finalizing the path, it should maneuver safely without colliding with other parked(or moving) vehicles and pedestrians. On the other hand, the layout of the parking layout can vary a lot. For instance, a parking lot inside a building has constrained space as compared to an open parking lot. Additionally, the parking slot can be at different angles, for example, vertical, diagonal, or parallel. A vertical slot is the most common form of slot that we come across because of its efficiency to pack more cars in an area. While on the other hand diagonal slots promote one-way traffic and reduce the time it takes to park. Parallel parking is used places to save space width-wise.
In this paper, we evaluate the performance of SAC algorithm in a continual learning environment to park a vehicle in parking lots with different lane orientations. The evaluation shows that the agent is not able to incrementally learn to park over multiple phases ranging from simple to more complex layouts as the agent is susceptible to catastrophic forgetting. HER has no effect in mitigating the catastrophic forgetting when the learning is done in a phased manner. The agent can be seen to perform the best if it learns in a mix of environments with different variations. Additionally, if the variation in the lane orientation is large, it takes more episodes to train as compared to small variations. We simulate the car parking in an OpenGym-based environment with multiple scenarios and assess the performance for the same.

Keywords: Autonomous Cars Parking, Lifelong Reinforcement Learning, Deep Reinforcement Learning, OpenGym AI

Acknowledgements

First of all, I would like to thank my mentor, Prof. Vinny Cahill for his invaluable suggestions and patience during my thesis. I am extremely grateful for his constant support throughout the dissertation period. His immense knowledge and expertise in the field of Reinforcement Learning motivated me to stay focused on the problem at hand . I am also thankful to Computer Science and statistics department of Trinity College Dublin to provide me with all the necessary IT tools and infrastructure without which it would have been difficult to work on my thesis.

I am also grateful to my classmates for their editing help, feedback and moral support. Lastly, I would like to express my gratitude to my parents, my sister , and my partner. Without their emotional and mental support, it would be impossible for me to complete my study.

Contents

List of Figures

List of Tables

Nomenclature

CL	Continual Learning
MDP	Markov Decision Process
MRP	Markov Reward Processs
PPO	Proximal Policy Optimization
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
SAC	Soft Actor-Critic
HER	Hindsight Experience Replay
DDPG	Deep Deterministic Policy Gradient
RRT	Random-exploring Random Tree
DDQN	Double Deep Q-Network
SUMO	Simulation of Urban Mobility
SARSA	State Action Reward State Action

1 Introduction

1.1 Context

Parking these days have become one of the biggest concerns for privately owned vehicles because of the increasing number of vehicles that are coming on road every year.In urban areas, it is now become increasingly difficult to get a parking slot because of the high density of vehicles. The energy costs associated from finding a suitable parking space to actually park the car is very high. These high energy costs do not only create a huge carbon footprint in the environment but also impacts the individual's productivity. If a person going to the office spends time looking for a parking, that is a waste of valuable time which can be directed to other priorities. Furthermore,majority of state governments in the United states as well in the EU have started adopting policies which discourage vehicle ownership and promote public transport. For example,in New York the public parking costs are very high to curb the traffic congestion problems in the city. Another example is in Ireland, buying a car has additional taxes introduced to promote public transport services.

With the advancement in the areas of Artificial Intelligence, Image Processing and 5G networks, we are looking at a fast paced development in vehicles automation. The automated vehicles are now even more connected as they can exchange information with other vehicles and receive important Over The Air(OTA) updates in a seamless manner. This means that the vehicles are now not only enabled to receive new information, but 'how to act' when it receives it. Many companies are heavily investing in car automation for example, Tesla, Waymo and Uber to name a few. Even though most of these automated vehicles have lane assistance, lane changing and adaptive cruise control,self parking is somewhat less explored.

When it comes to parking, it is a cumbersome process.The vehicle first needs to figure out the most convenient slot. Choosing the most appropriate slot can subjective and depends on the situation. For example,the slot nearest to the entrance will reduce time to park but it may increase time to exit if it is a big parking lot while on the other hand if we park closer to the exit it will have an inverse effect. Once a slot is detected it becomes essential to figure out a path to the slot and then maneuvering the vehicle safely to the slot without

colliding with any other cars or pedestrians. Also, the parking layouts vary a lot from place to place. For instance, it can range from a closed parking layout inside the building with restricted space for the vehicle to move to large open layouts outside the buildings which has more room for the car. Additionally, the parking slots can have different orientations . The most common ones are vertical slots which are efficient to accommodate more cars in a given area. Then there are diagonal slots which promote one way traffic for fast park and exit from the parking space. Then there are parallel parking slots which save the space width wise and are mostly used for a curb side parking. In this thesis we want to explore if learning in one particular orientation can significantly help in parking in the other by using RL.Furthermore, we want to analyze if learning continually between different lane orientations is impacting the overall performance of the model to execute each of the task independently. This can mean the model is susceptible to catastrophic forgetting.So we want to analyze if using the replay buffer can help us mitigate that problem.This can form a basis for developing efficient models to self park the car in different parking lots.

1.2 Motivation

The methodology for comparison between the independent learning strategy vs incremental learning strategy of the RL agent in multiple lane orientations can form a solid foundation to choose the right approach for both learning and deployment of the model in variable parking environments. Through the exploration of the agent in this environment can be a first step in building a baseline for more complex scenarios of multiple moving agents.

Solving the problem of parking in autonomous vehicles can help in creating a smaller carbon footprint on the environment and save the drivers and passengers from exhaustion.Since the manual parking suffers from human error, the parking space can be efficiently utilized as the agent parks the vehicle in the most optimal way possible. Although the analysis is restricted to parking environments, however the results of this analysis can give some insights which can be beneficial in somewhat similar settings as well. For instance, a robot transporting the objects from source to destination inside a packaging centre. Another similar example can be a robot moving raw materials on a construction site.

1.3 Problem Definition

We have a parking environment with a vehicle which is at a fixed position on the east side facing in the west direction. We have a parking slot which is our goal and we want to manoeuvre the ego vehicle in such a way so that it is correctly parked in the right slot with the right orientation without colliding with any other parked vehicle or obstacle on it's path. The environment will be challenging as we will be exposing our agent to parking slots of

variable orientation , i.e. vertical, diagonal or parallel as shown in 1.1.

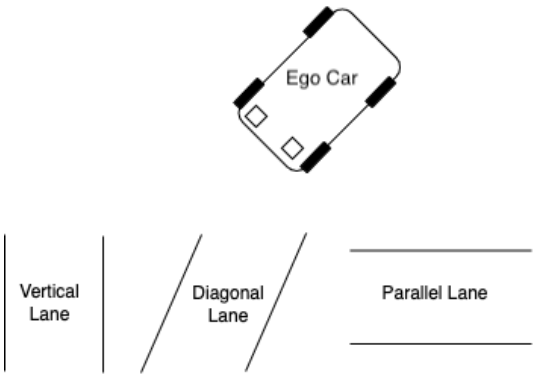


Figure 1.1: Lane orientation

1.4 Objectives & Approach

1.4.1 Objectives

The main objective of this thesis is to evaluate the learning of the RL agent to park the car in multiple lane orientations, i.e vertical, diagonal and parallel parking. These multiple orientations can be considered as different goals because of the variation in the angle. We want to analyze how effectively the agent is able to do the task of parking in simple and complicated environments. We want to do a performance comparison between SAC model with HER, default buffer and no buffer to park the car in the slot within the simulation. We also want to analyze if using HER can potentially mitigate the catastrophic forgetting to some extent.

1.4.2 Methodology

We will achieve the above objectives by first developing and customizing a simple testing environment where we can run our experiments as our testing bed. As a next step we will simulate the continual learning process in the form of a phased training strategy by exposing our agent from simple to complex layouts with different lane orientations.We will then compare the performance of the phased setup with different combinations of the environment having different variations.

The RL agent is able to learn through a suitable reward model where it understands which manoeuvres lead to crashes and which one leads to successful parking. In order to see the impact of HER in mitigating catastrophic forgetting, we will run the model once with HER and then without HER to compare the results.

1.5 Document Structure

The structure of this thesis has been organized into 6 chapters. Chapter 1 has 5 subsections out of which the first three sets the context, motivation and problem definition of the problem we are addressing. Additionally, the last 2 subsections talk about the objectives and methodologies and the document structure of the thesis. Chapter 2 gives a deep dive analysis of the latest state-of-the-art in parking with Autonomous Cars. The first two subsections in this chapter give the overview of RL and Lifelong RL, and in 3rd sections we give the overview of the Autonomous vehicles and techniques used for parking. Chapter 3 outlines the approach we have proposed to the problem of parking. In Chapter 4, firstly we test several tools by assessing the pros and cons and decide our choice of the framework.Chapter 5 explains the evaluation process and an analysis of the results. Lastly, Chapter 6 concludes the dissertation and lays down the scope for future works.

2 Background

2.1 Reinforcement Learning

RL algorithms are based on the concept of having a reward system where the agent (or model) is rewarded if the expected action was correct and penalized for incorrect actions.It is different from supervised learning as the data is not labeled for training. It is also different from unsupervised learning as we are not trying to find any patterns by clustering.

In a RL paradigm, we have an agent which learns by continuously interacting with its environment by performing a certain set of actions to transition to an end/terminal state from an initial state. The feedback/reward for each action is not instantaneous and can be delayed. The agent's action taken at a particular step will be closely correlated to the actions in the next and it's current action will determine the type of subsequent data it receives. Even though the environment can be fully or partially observable, we only care about the subsequent observations and rewards that come from the environment from the agent's perspective.

An Agent's state in simple terms represents the internal representation of the model which can be used to determine what to do next. History of an agent represents a sequence of all observations, actions and rewards up to a point in time.The information state of an agent has Markov property which says that once the current state is known, the history is not needed to determine the next action. In a fully observable environment, the environment's state is the same as the agent's state and we use the MDP which represents the state. On the other hand, when we have a partially observable environment, we use POMDP which is used to construct the state from history and a set of beliefs (probabilities of what the state is).

(5) There are 3 Components of an RL agent namely policy, value function and model. The policy of an agent is the behaviour which means the mapping of action from a state. It can be deterministic or stochastic. Value function tells about the goodness/badness of the current state and in turn gives a prediction of future rewards. Model is the representation of the environment from the perspective of an agent.

A Markov Process is just a state transition matrix which contains the probabilities of going from the current state to the next state .The MRP is a Markov Process which now has reward value associated with each state. This reward value is based on the expected value (mean) of G_t (return) along all the paths. G_t in simple terms is the discounted reward from time step t towards the terminal state. Formally, with the Bellman Equation, we can calculate the value function of a state s by adding the reward we got to reach state s plus discount factor times sum of value functions of all states we can reach from s. MDP is a MRP with Actions states where a policy defines the current behaviour of the agent. A simple MDP structure is defined by <S, A, r, p, γ > where S is a set of states, A is a set of Actions, r is a set of rewards which is defined for each state action pair, p is the transition matrix and $\gamma \epsilon$ [0,1].

There are 2 broad categories of RL algorithms namely Model-Based and Model-Free. As the name suggests, in Model-Based algorithms the agent constructs or has the complete knowledge of the environment. It means it has the complete model with state transitions which can then be used for searching for the next appropriate action to be taken. A well known model-based algorithm is the AlphaZero (6). On the other hand, Model-free algorithms are those which do not need a model, rather they directly optimize the policy based upon a value assigned to each state which keeps on improving over multiple episodes. Some of the Model-Free examples are DDPG, SAC, PPO etc as shown in 2.1

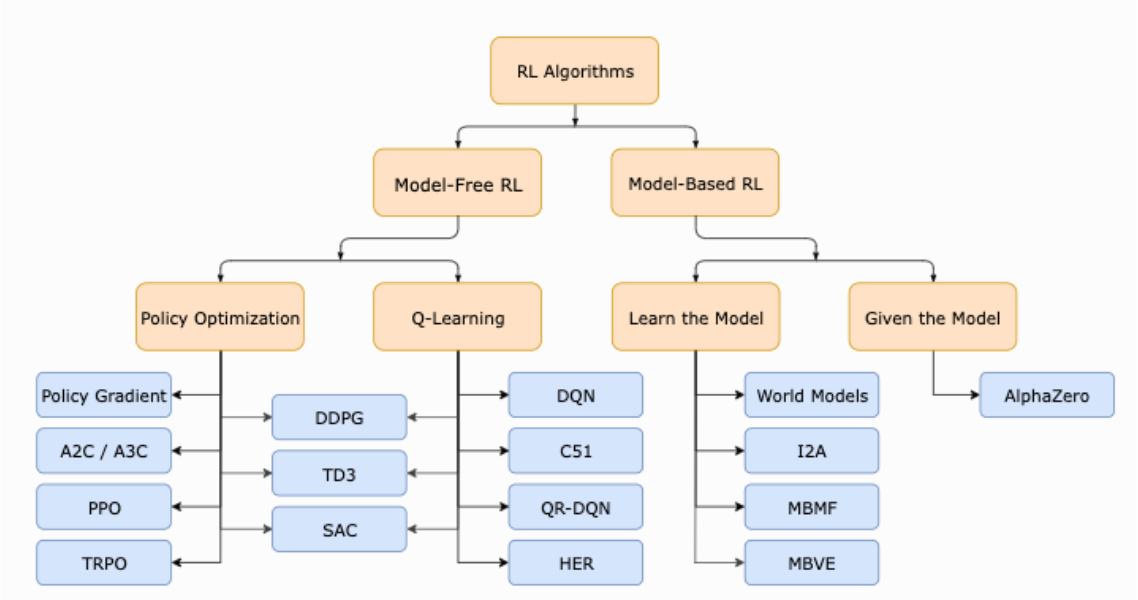


Figure 2.1: Different Model-Based and Model-Free Algorithms (1)

RL algorithms can also be divided on the basis of whether they are off-policy or on-policy. As the names suggest, on-policy means that the algorithm utilizes the same policy learned for their actions in order to optimize the policy. Popular examples of on-policy algorithms are PPO, TRPO, SARSA to name a few. On the other hand, Off-policy means that instead

of using the same policy for convergence, we use 2 different policies for learning namely, the target policy and the behaviour policy. The target policy focuses on the task at hand and tries to maximize the rewards to achieve that task while the behaviour policy can try new unexplored paths. Some examples of Off-policy are DDPG, SAC, Q-Learning etc.

2.2 Lifelong Reinforcement Learning

Lifelong RL comes under the broad category of CL. CL is the process where we take the previous experiences as our foundations to build complex behaviours and apply them to unseen environments. As the name Continual suggests, we have an agent which takes continuous streams of inputs and should have the following characteristics as defined in (7).Firstly, the agent should have an incremental learning without being dependent on a fixed training set. Secondly, it should be able to quickly adapt to variations in the environment over a duration and recover quickly.Most importantly,it should not forget past behaviours when it learns new tasks which is known as catastrophic forgetting and inference. This is part of a broader issue of stability-plasticity dilemma where we need to strive a fine balance between the past and present experience to determine the future. According to (8), a set of tasks means a set of problems which we want to our model to solve in an incremental fashion. While continually learning new tasks, we don't want to forget the experiences we have gathered while solving older tasks . An abstract representation of a CL mode is shown in 2.2.

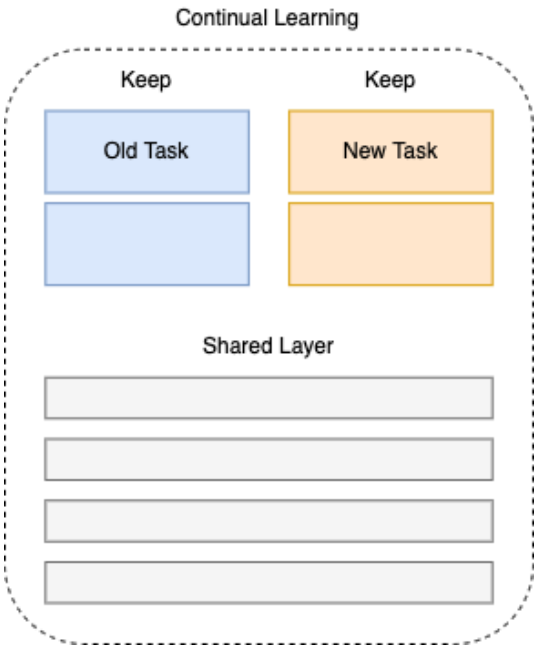


Figure 2.2: Abstract representation of CL taken from (2)

Catastrophic forgetting can occur due to various reasons for example agent's memory

limitation due to hardware, the new environment having completely different states from the previous environment, environment state explosion etc. When we use the traditional RL approach on a episodic setting in a continual episodic environment, it is shown to produce biases. What it means is that the agent will give more significance to the more recent episodes and will completely disregard the previous episodes.

For a CL model, there are some features of the system according to (9) that we need to understand. We want the system to not forget previous tasks when learning new ones. We want the system to be scalable because of limited memory and compute resources.The architecture should support forward and backward transfer, i.e experience gained during previous tasks should help learning new tasks and vice versa. All of the above characteristics are in contrast to each other as remembering everything is not possible with having fixed capacity or enable forward/backward transfer. Hence, we need to be able to assess the right trade-offs between various properties of the system first.

In a RL setting, we also need to identify the areas where Non-stationarity is present. The components which can have a time based distribution can be the transition function p, reward set r, observation function x or actions space A. This is called the degrees of non-stationarity. Next, we then look for the drivers of Non-stationarity. It can either be stationary, passive, active or hybrid. In passive, the agent's behaviour does not affect the non-stationarity of the environment, while in the active setting it does influence the behaviour of the environment. On the other hand, in hybrid case, it is mixture of active and passive setting.

2.3 Deep Reinforcement Learning

(10) Deep Learning is the technique of combining RL with Deep Learning. Since some MDPs have very large state spaces, it is not feasible to use conventional RL techiniques. Therefore by using deep learning we can makes use of neural networks to mitigate the issue. For example, using by using the neural network as an approximator instead of maintaining large state-action value tables. Another example would be to have a actor critic model such as SAC which uses 2 neural networks as explained below.

2.3.1 Soft Actor Critic Model

This is an off-policy and model-free RL algorithm which has 2 components, namely the actor and the critic. The critic estimates the value function which can be a state value or an action value. The actor updates the policy in the direction of the critic's value estimation. The SAC not only maximizes the future rewards, but also the maximizes the future entropy of the policy which encourages exploration as opposed to only exploitation. This additional

entropy in calculations prevent the agents to mitigate the problems of being trapped in local minima and not converging to an optimum policy. SAC is a hybrid of learning by q-value as well as the policy optimization. SAC algorithm is shown in 2.3.

2.3.2 Hindsight Experience Replay

Often times we are in specific environments where the rewards are either very delayed or sparse. It becomes extremely important for an RL agent to fetch these rewards in a more uniform manner to converge on a good policy. Consider a situation where the agent is learning to play football. The agent shoots the ball which hits the edge of the goal post and gets diverted away from the goal. What a standard RL would see is that the sequence of actions, which is hitting the ball, did not produce a desired outcome. Therefore the reward structure is not only about giving positive reward to the agent for the desired task, but it is also about shaping the reward function in such a way which promotes policy optimization. Wherever we have a binary signals of successful task completion, it becomes very difficult to engineer the right reward model. On the other hand, consider where there are multiple tasks in sequence which need to be completed in a random or a predefined order. We now not only want an agent to do learn a single task efficiently, but we also want it to remember old tasks, as it starts to learn the new ones. Hence we need an efficient sampling strategy from a storage which can replay episodes from the past experiences. Hindsight Experience Replay(HER) (11) is able to address the challenges faced in these types of environments. 2.4 shows an abstract representation of HER.

The essence of HER is that after the agent completes an episode from $S_1, S_2..S_n$, the transition is stored in the replay buffer not only with the original goal used for the current episode but with a subset of other arbitrary goals. We generally use HER in addition to Off-policy algorithms like DQN(Deep Q-Network), DDPG, SAC etc.

2.4 Autonomous Cars

2.4.1 Overview

There are about 6 levels of Autonomy defined for a vehicle as per (12).Level 0, as the level suggests has no level of automation and is completely dependent on the driver's inputs. Level 1 and Level 2 provides partial assistance to the driver for example Adaptive Cruise control to regulate the speed on a highway or Lane Keeping Assistance which keeps the car inside a lane. They require attention of the driver which should be able to override controls if the Automation system goes haywire. Level 3 systems require minimal assistance from the driver and it can take critical decisions such as lane changing, speed regulation etc. For example, Tesla Cars are equipped with level 3 automation system. Level 4 and 5 require

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence

Figure 2.3: Pseudocode taken from (3)

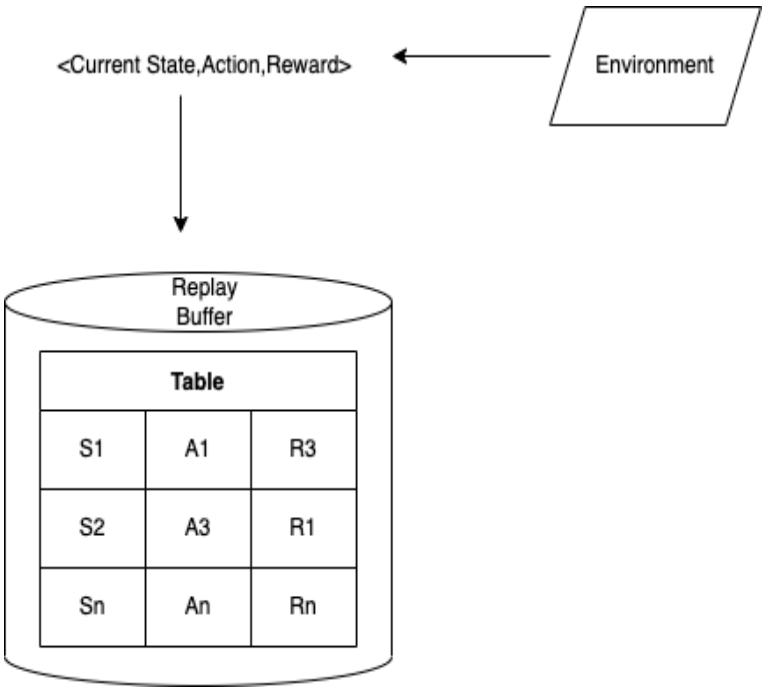


Figure 2.4: Abstract representation of HER from (4)

little to no assistance from outside and no monitoring of the vehicle is required.

Autonomous cars broadly have 3 systems, namely the perception system, the decision & planning system and the execution system. The perception system is mainly concerned with inputs like mapping location, obstacle detection etc. On the other hand, the execution system consists of subsystems which control the motions of actuators and other motor devices of the vehicle. The Decision and planning system is where the core processing of the input from the perception system happens. It includes things like route planning, next best action, trajectory planning etc. (13) is review which provides an exhaustive taxonomy of self driving tasks and explores DRL methods to address challenges faced in real world deployment of autonomous driving agents.

A lot of work (14) has already been done to solve the problem of decision making using the conventional supervised learning model where the ego vehicle takes in input (stream of images) from the cameras and labelling the actions such as steering angle, acceleration etc for the purpose of training data. Although, the agent seems to perform decently as it is able to associate the pixel data with action, however, the results have shown that generalising a model for a scenario which the agent has not seen previously can easily become challenging. (15) uses a combination of supervised learning and reinforcement learning to follow a path. (16) uses Reinforcement Learning and Imitation Learning to navigate in complex urban scenarios.

2.4.2 Parking

With increasing number of vehicles coming on road each year, the demand for parking has also increased exponentially.Parking is a time and energy consuming task especially in densely populated urban areas. The cars can take the sensory input, for instance using a camera feed, LIDAR sensors etc, to decide the best course of action to park in an available parking slot. Since it is very hard to take into account every use case which a car can encounter during parking, it becomes an obvious choice for using Reinforcement Learning.

When parking lot gets more complex in (17) as compared to the one in (18) because of multiple lanes, PPO performance degrades when the parking slot is farther away from the entry point. Both use Unity as their choice of frameworks to simulate multiple scenarios. (19) uses RL to train a model to follow a pre-generated path from the initial position to the parking slot inside a parking lot. It compares the performance of DDPG,TD3 and SAC algorithms to follow a complex path that includes cusps and overlaps towards the parking space. The path is pre-generated before the learning stage using RRT and Reeds-Shepp path creation algorithm.The simulation framework is based on (20) used in this dissertation as well. (21) use the DDQN to train the ego vehicle to successfully and safely drive in a multi-lane highway environment which is based on SUMO simulation framework. (22) uses multi agent environment RL to park upto 7 cars in a parking environment with 98.1% success rate.

3 Problem Approach

3.1 Vehicle Attributes

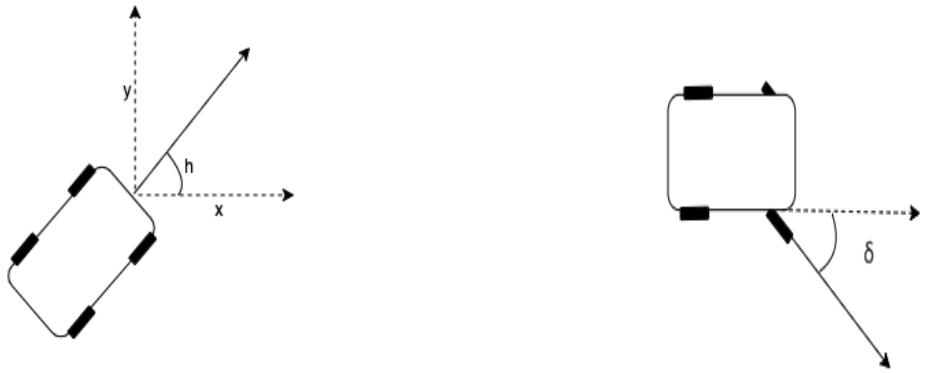
Our Environment is a simple parking lot with 1 ego vehicle and several parked vehicles. The vehicle is about 5 meters in length and about 2 meters in width. It is represented by the (x,y) in the coordinate system along with an heading angle h (angle between car head and x axis). The front wheel's steering angle of the vehicle can turn 45 degrees in either direction and the acceleration is capped at 5 meters / s^2 in both forward and backward directions. The vehicle needs these two inputs from the agents to move, namely the steering angle δ and acceleration α.

Duration of each episode is 150 seconds which gives the vehicle enough time to park in a complex environments. A fine balance needs to be achieved when it comes to choosing the duration or steps in each episode. For instance if we set the duration to very low, the agent might increase the speed or end up crashing the vehicle into the nearby parked vehicles. On the other hand, if we set the duration to very high, the training time overall will increase. To keep the experiments simple, parking nose-out in the goal slot has been scoped out and only parking head-in is considered.

3.2 State Space

As explained briefly in the previous section, the state of an agent is characterized by the equation [1] . x, y 3.1a and δ 3.1b will be the x coordinate, y coordinate and angle of steering respectively. v_x, v_y are the speeds in the x and y directions and $cos(h)$, $sin(h)$ are the cosine and sin of angle of heading respectively. Each observation that we get back from the environment Obs_{step} is shown in [2]. $S_{vehicle\_current}$ is the state of the vehicle before taking any action, $S_{vehicle\_achieved}$ is the state of the vehicle after taking the action and $S_{vehicle\_desired}$ is the desired state where the vehicle is parked inside the slot.

$$S_{vehicle} =< x, y, v_x, v_y, cos(h), sin(h), \delta) \tag{1}$$



(a) Car coordinates with heading angle [x,y,h] (b) Car with steering angle δ

Figure 3.1: Car coordinate with heading angle (left) and steering angle(right)

$$Obs_{step} =< S_{vehicle\_current}, S_{vehicle\_achieved}, S_{vehicle\_desired} > \quad (2)$$

3.3 Action Space

The agent can send inputs in the form of steering angle δ within the range of [-π/4,π/4] radians and acceleration α within the range of [-5, 5] meters/s^2. The Speed of the vehicle that should be allowed depends on the scenario for example the requirements inside a parking lot will be different from the ones required on a highway. We will look at how we can control speed by using an appropriate reward discussed in the next section.

3.4 Reward Function

The agent is rewarded on the basis of it's proximity to the goal, i.e the parking space. At each step a weighted p-norm is used to get the distance from the current location to the goal and computed as shown in 3.2. The default value of p is chosen as 2, which makes it an euclidean distance (Euclidean Norm). The term inside the square root is calculated as shown in equation 3 which is the dot product of the difference between state of the vehicle achieved and desired with a weighted rewards.

The vehicle is penalized for speeds above 5 meters/s^2.The agent is also penalized if during maneuvering, there is a collision expected with an obstacle or a parked vehicle. A summary of the reward model is shown in 4

$$\sum_{i=1}^{n} |x_i|^p = (S_{vehicle\_achieved} - S_{vehicle\_desired}) * (reward_{weights}) \quad (3)$$

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Figure 3.2: P-norm

$$Reward_{step} = \begin{cases} -4 & \text{if collision with car/obstacle} \\ -||x||_2 & \text{euclidean distance to goal} \end{cases} \tag{4}$$

3.5 Algorithm

The choice of algorithm for comparing the performance of the agent in various different scenarios will be SAC. In addition to the off policy algorithm, we will be running experiments for all the scenarios with and without HER.

We will be using a neural network of 3 hidden layers which consists of 512 neurons each with a default learning rate of 0.001. The complete hyperparameters used are shown in 3.1. The maximum duration of an episode has been set to 150 steps which gives the agent enough time to figure out complicated maneuvers. If increase the time too much, it might lead unexpected results where agents tends to explore a longer path or get away from the slot.On the other hand, if we give the agent less time it might not able to completely figure out the path and just try to park as near as possible to the parking spot. The replay buffer size is being set to 10^6 to give it enough memory to store the past episodes. All the other hyperparameters are being run in default setting.

Parameter	Value
Environment	Highway-Env
Batch size	1024
Polyak Update	0.05
Learning Rate	0.001
Discount factor	0.95
Hidden Layers	3
Neurons count per layer	512
Episode Duration	150 steps
Replay Buffer Size	10^6

Table 3.1: SAC Hyperparams

4 Simulation

In this Simulation Tools section, we will first discuss Unity and Open AI's gym, the two most widely used tools these days for performing RL experiments. Then, in the Test Setup section we will describe the environment and overall architecture that has been built to train the RL agent.

4.1 Simulation Tools

Unity is a game engine primarily designed for developing interactive 2D/3D games. Unity ML agents is a plugin which is a popular choice among RL enthusiasts to run experiments because it provides a rich high level visualization and in built support for parallel training of agents in an environment. To perform our experiments this (23) project was explored to see the ease of use for modifying and training. After considerable amount of time, these are some of the observations that were made:

- The unity platform has a high learning curve in itself which takes time to understand.
- Opening the project had lot of compatibility issues with current versions.
- ML Agent Plugin has a limited list of algorithms that can be used inside the environment which makes it more like a black box.
- Even though 3D car environment makes the visualizations great, however it needs heavy GPU machines to run simulations and perform training on neural network.

Since a lot of time was being spent on the environment itself rather than the actual problem, it was decided to explore a simpler tool which has these standard set of features:

- Ease of installation.
- APIs for interaction between RL Agents and it's environment.
- Pluggable Agent Algorithms and environments.
- A simple 2D environment which can be tweaked with ease.

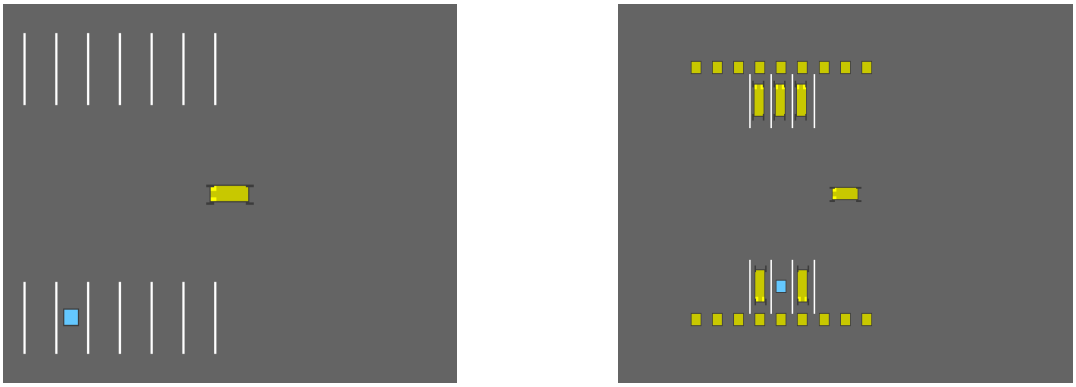
. We came across Open Gym AI ,an open source python library, which checked on all of the points above. It was decided to move forward with this tool to use as our training and testing framework.

4.2 Test Setup

To build our environment the following framework was used for this dissertation:

4.2.1 Environment

The gym library Highway-Env (20) is used as our base project for developing the simulations. This is built on top of Open Gym AI and therefore it extends the same API functionalities for communications between the agent and it's environment. This library has a lot of different environments ranging from high speed highway, intersections and racetrack environments to small confined parking environments. We select the parking environment which is a simple 2D environment having straight parking lanes as the one shown in 4.1a. We have 1 ego vehicle which gets initialized in a global coordinate system (x,y) at particular angle of heading *h* (angle between the car head and the x axis. The vehicle inputs that can be provided are the steering angle δ and acceleration α as an action in each step will result in an environment observation which is the movement of the car according to the Kinematics Model (24). Certain tweaks to this environment has been made to fit our use case as shown in 4.1b starting with making the car position, car heading, lane slots length/width, lane angle, goal lanes configurable. In addition to these, we have added added dummy cars in other lane slots to make the environment more restrictive in movement.



(a) Parking Environment Original (b) Parking Environment Customized

Figure 4.1: Original and customized Parking Env

The table A1.1 shows RL environment specific configurations for adding variations

4.2.2 Algorithm and Architecture

We will be using Stable Baselines (25) as the the Algorithm Library which consists of an up-to-date collection of most widely used algorithms to perform training and analysis. Since it is based on Open AI Baselines (26), this makes it an obvious choice for our training model.

An overall architecture of our simulation environment is shown in 4.2. The parking environment code has been modified to accomodate csv writer module which will record metrics during each step/episode and write it to a csv file. After the experiment has been conducted, the graphs will be generated using this csv file for generating insights.

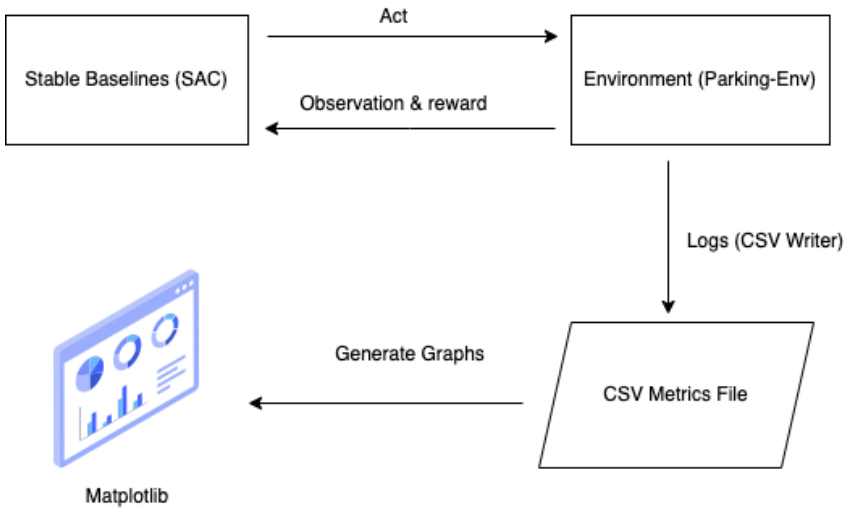


Figure 4.2: Architecture

4.2.3 Data Gathering Strategy

To collect the data during training and testing we will be using the Tensor-board as our tool to track metrics such as average rewards, policy loss etc. To gather additional metrics, we will be using an in-built CSV module to write to the files.

4.2.4 Evaluation Metrics

After the completion of training and testing, the following 2 metrics are chosen to assess the performance of the algorithms:

- **Total Reward** : This is an obvious choice of metric because we want the agent to maximize the rewards it has collected over the training and testing phase. If we see an increase in the total reward collection it means that the policy is overall improving.

- **Success Rate** : This metric gives us an average of the number of times the agent was able to complete the task, i.e park the itself in the lane correctly without colliding with other vehicles.
- **Vehicle Velocity** : This will be an important metric which will help us understand if the velocity is going outside the desired range of the ego vehicle's behaviour. This will also help us further fine tune our parameters.

4.3 Implementation Classes

The project (20) has a collection of different environment which a autonomous vehicles face in various situations. We will not be going over the entire codebase, but just over some important classes to understand how were the changes added and why.

4.3.1 AbstractEnv and GoalEnv Class

AbstractEnv To begin with we have an AbstractEnv class which is a generic environment for various tasks involving a vehicle driving on a road. It contains a road populated with vehicles, and a controlled ego-vehicle. The action space is fixed, but the observation space and reward function must be defined in the environment implementations as shown in 4.3.

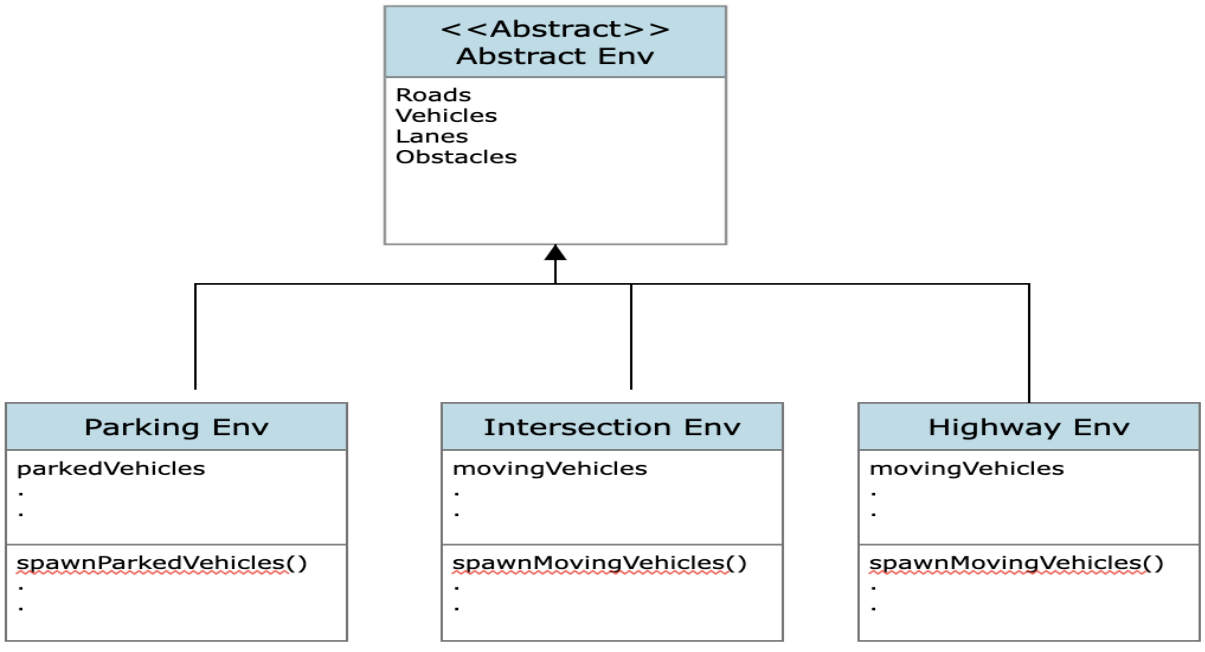


Figure 4.3: Abstract Environment

GoalEnv is the interface that enforces the structure of the observation shown in equation 2.It functions just as any regular OpenAI Gym environment but it imposes a required

structure on the observation_space. More concretely, the observation space is required to contain at least three elements, namely observation, desired_goal, and achieved_goal. Here, desired_goal specifies the goal that the agent should attempt to achieve.achieved_goal is the goal that it currently achieved instead. 'observation' contains the actual observations of the environment.

4.3.2 ParkingEnv class

This is the concrete implementation of the AbstractEnv class which contains the implementation of the code for the default parking environment. This class is instantiated whenever we call 1 in our code which according to the OpenGym APIs, configures an environment and returns an environment object.We will be specifically building on top of this environment. Since this class has the concrete implementations of various environment related configurations, we have added the customizations such as -

- making angle of lane as a configuration of the environment.
- Adding other parked vehicles inside the empty lanes.
- Adding obstacles to either end of the parking rows.
- making empty corridor width between two lanes configurable
- Integrating csv module to write logs in a file for analysis.

$$env = gym.make("parking\text{-}v0") \tag{1}$$

StraightLane class We have customized the Straight Lane class to accommodate the diagonal slots with configurable angles. Additionally, some extra changes have been added to make the parallel slots.
Vehicle and Obstacle classes We have added stationary vehicles to make the parking environment complex. We have also added obstacles to the either side of the parking slots.

4.3.3 Parking Driver Class

The file park_model.py is a single python script which was added for simulating different learning and deployment modes for our experiments. The reason behind making a single driver script is :

- changing environment modes by using commandline arguments.
- running in a headless mode on local desktop machine or on cloud for instance Kaggle or Colab

- Integrating matplotlib module for generating graphs.

4.4 shows a snapshot of the notebook run on kaggle which was used to train a model. With less than 6 lines of code, we are able to run our training process anywhere. The complete list of script arguments are given in A1.2.

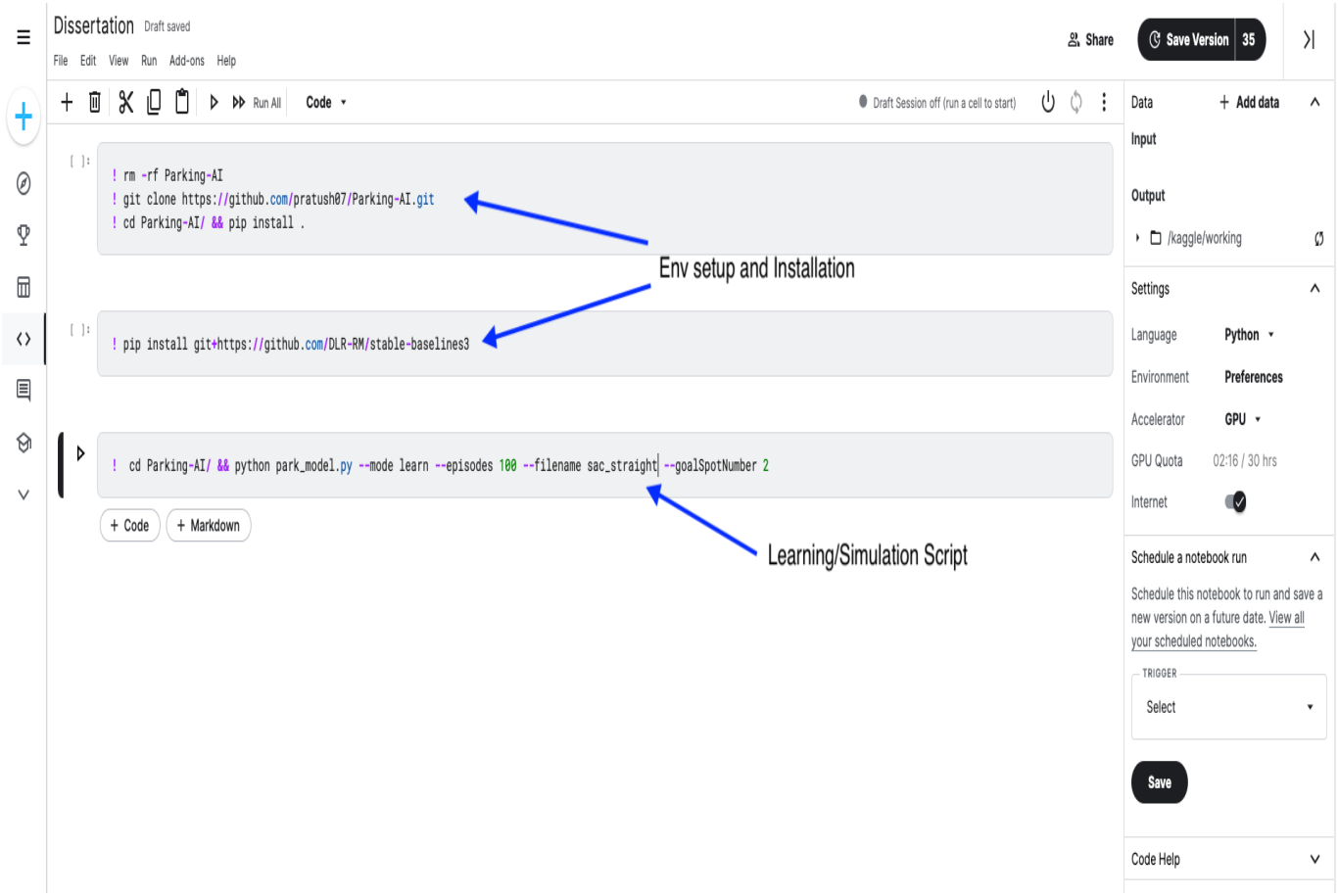


Figure 4.4: Kaggle notebook for learning 4.8

4.4 Training Strategy

We will perform training using a series of multiple approaches as described below. Firstly we will train and test our agents in multiple independent scenarios.For instance if we train an agent in scenario 1, then we will test the agent by deploying them in the scenario 1.Similarly the same will be done for other scenarios as well.

Next we will see the performance of the agent when it is deployed in an environment which it has not seen before. For example, if the agent is trained in scenario 1, it will be deployed in other scenarios not previously seen during learning.

Now we will test the phased approach where the agent will be exposed to various environments in a phased manner. For instance, if there are a total of 500 episodes, the

agent will learn scenario 1 for the 1st 250 episodes and then it will see a variation of the environment for the remaining 250 episodes. This creates an environment similar to CL where the tasks differ in the terms of parking the car in different slots.

Lastly, we will expose our agent to all variations in the environment during the learning and deployment stage. This means that the agent can see any type of scenario uniformly randomly both during training and deployment in a single simulation.

All of the above experiments create a suitable environment for testing the Catastrophic Forgetting that happens during the Continual Learning and will help us analyze which approach has the least forgetting. A list of variations in the environment are described below in detail.

4.4.1 Scenario 1

In this scenario, the agent is initialized at a fixed position heading in the west direction. We have a parking lot with just 2 straight slots (lane angle is 90) and we have our goal lane randomly placed between two during the training as shown in 4.5. Here the agent has no prior experience in manoeuvring the vehicle in a parking lot so it is building it's experience right from the start.

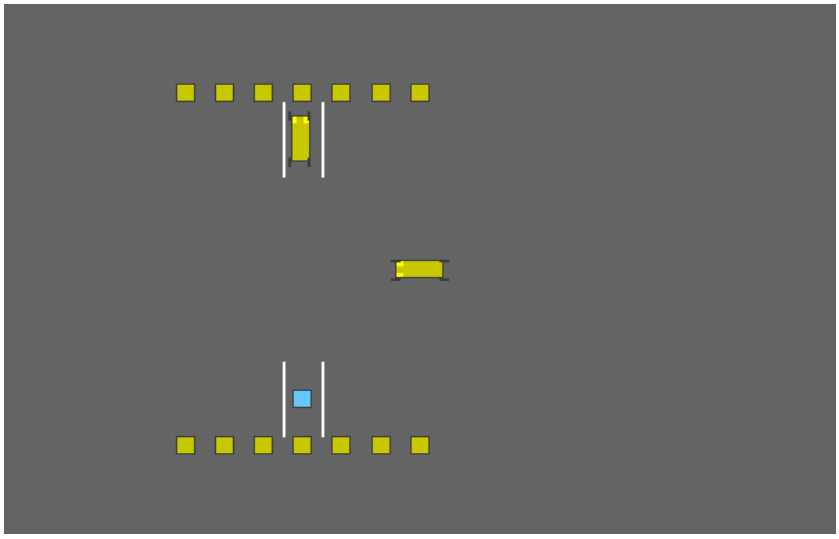


Figure 4.5: Scenario 1

4.4.2 Scenario 2

In this scenario, the agent and the environment setting would be identical to Scenario 1, but now we will increase the complexity slightly by adding angles to the lanes as shown in 4.6. Here the agent has prior experience in manoeuvring the vehicle in a parking lot from the previous scenario, so it will build it's experience on top of what it has already learned.

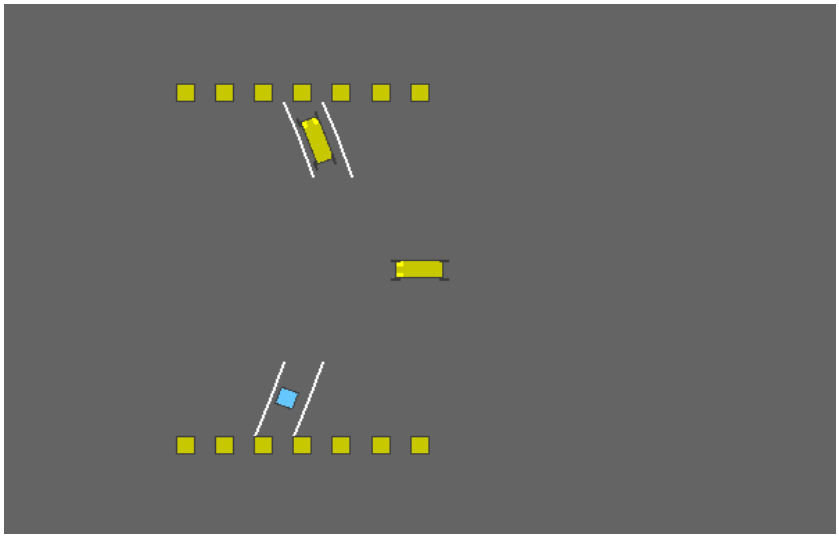


Figure 4.6: Scenario 2

4.4.3 Scenario 3

In this scenario, the agent and the environment setting would be identical to Scenario 2, but now we will be adding angles to the lanes to make it a parallel parking as shown in 4.7 .

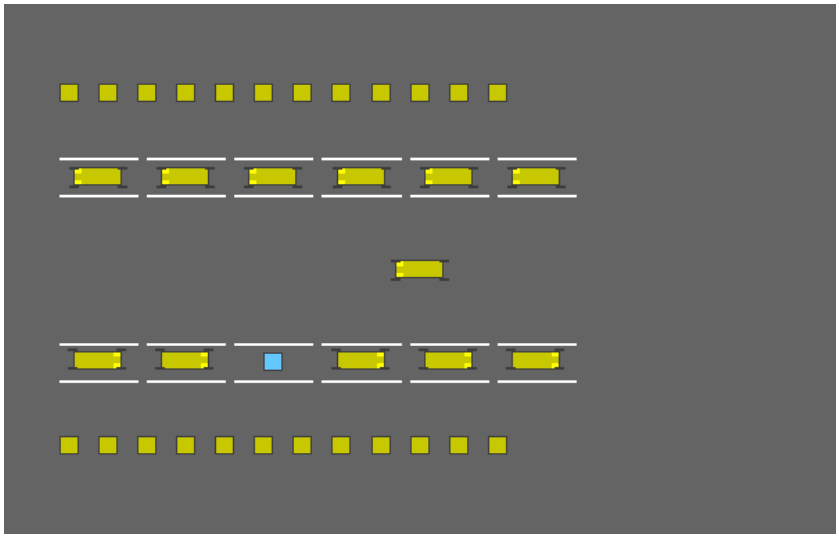


Figure 4.7: Scenario 3

4.4.4 Scenario 4 and 5

In this scenario, we would be adding further complexity to our environment as shown in 4.8 and 4.9 by adding more lanes at varied angles with parked vehicles and also fixing the goal lanes in the middle slots so as to make it more challenging to park. These scenarios will be somewhat close to actual parking lots close to real life parking lots.

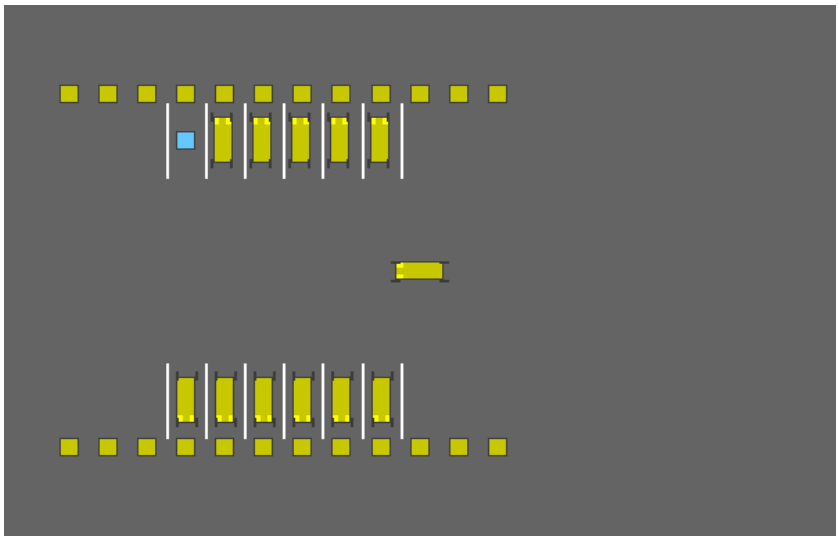


Figure 4.8: Scenario 4

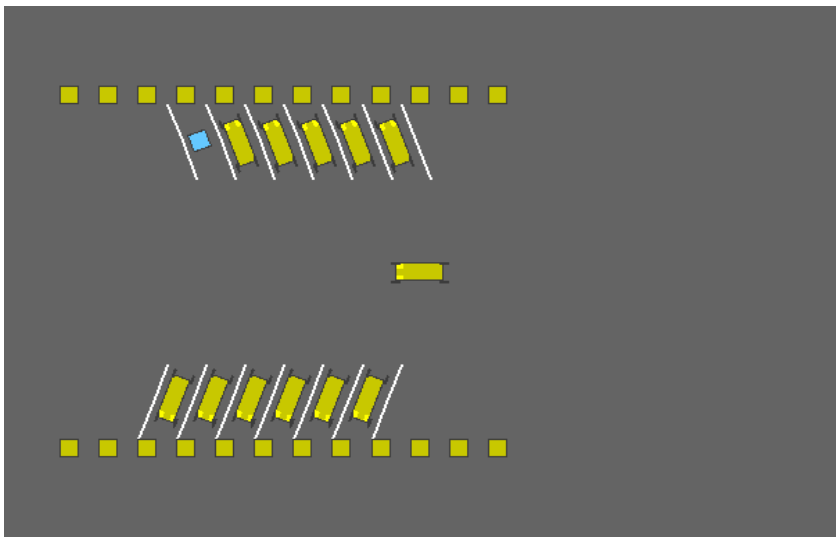


Figure 4.9: Scenario 5

5 Evaluation

We performed a series of experiments as described below. In each of the experiments, we run the agent in the learning mode first and collect the metrics in a csv file as explained in the 4.2. At the end of training, we run the agent in a deployment mode where the agent only performs action in the environment. We know if a particular episode is successful if the total reward accumulated in an episode is greater then or equal to success goal reward (-0.10) defined in A1.1. We will finally do an overall comparison using the results.

5.0.1 Experiments

We will be running different experiments which have been described below on a very high level.

Isolated Environments :In this experiment we will be running the training and testing within the same lane orientation to assess how is agent able to perform on already seen lane orientations. This means that the agent will not be deployed in unseen lane orientations. This is the simplest form of testing.

Unseen Environments :In this experiment we will be running the training on vertical slots and testing/deployment on diagonal slots to assess how is agent able to perform on unseen lane orientations.

Phased Environments :In this experiment we will be exposing our agent to vertical and diagonal slots in sequence. This means that if we start the training for 300 episodes, it will see 150 episodes of vertical slots and then 150 episodes of diagonal slots.

Interleaved Environments :In this experiment we will be exposing our agent to vertical and diagonal slots randomly. This means that if we start the training for 300 episodes, it will see any of the 2 scenarios uniformly randomly.

5.0.2 SAC with no buffer

Firstly in the experiment, the agent will be trained to park the car in a vertical slot 4.8. We begin learning with the model without using any buffer by making it's size 0. We run the

training for over a 1000 episodes. To train, the script is called in the following way:

```
 python park_model.py --mode learn --episodes 1000 --filename
sac_straight --goalSpotNumber 2 --buffer 0
```

5.1 shows that initially the model brings the car quite close to the parking slot and the episode ends(Elapsed). It then crashes even after successfully parking once early in the episode.This means that having a 0 buffer size makes SAC perform poorly. We will now go to the next section where we will use a default replay buffer to store the previous experiences.

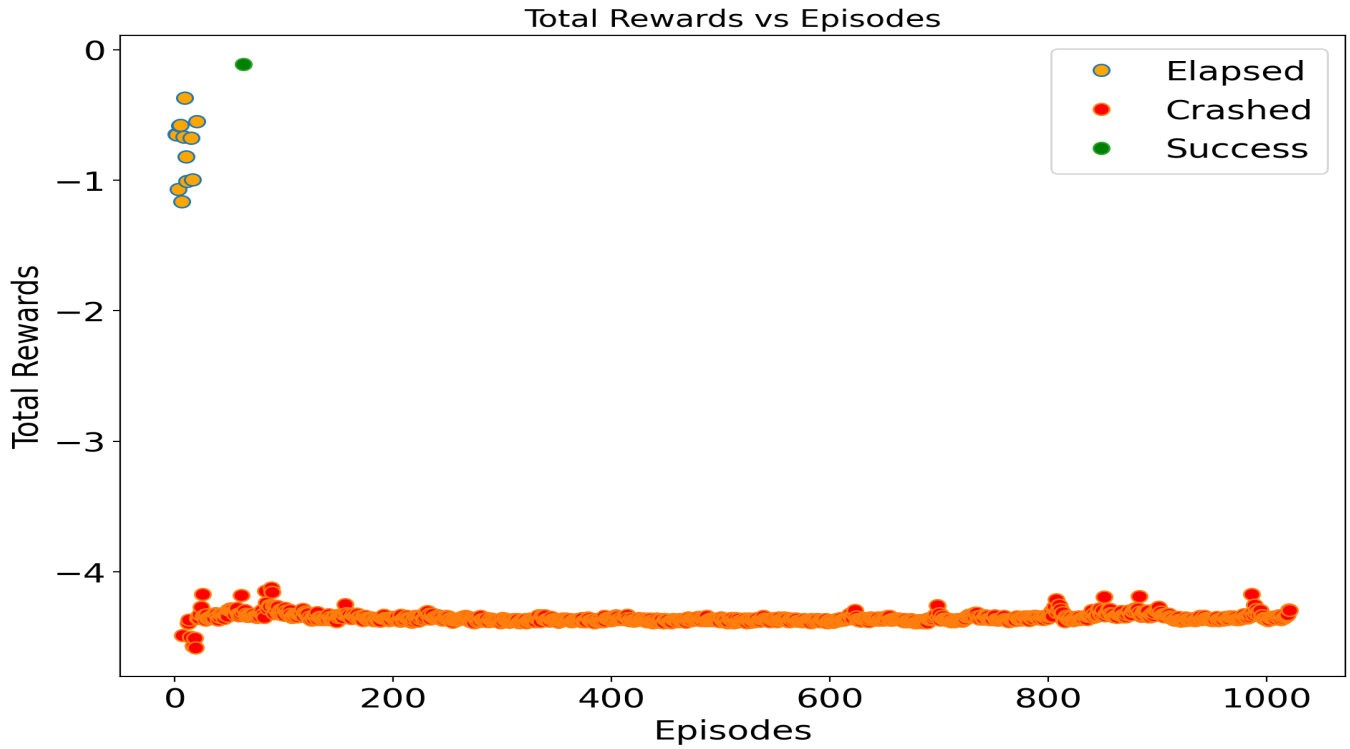


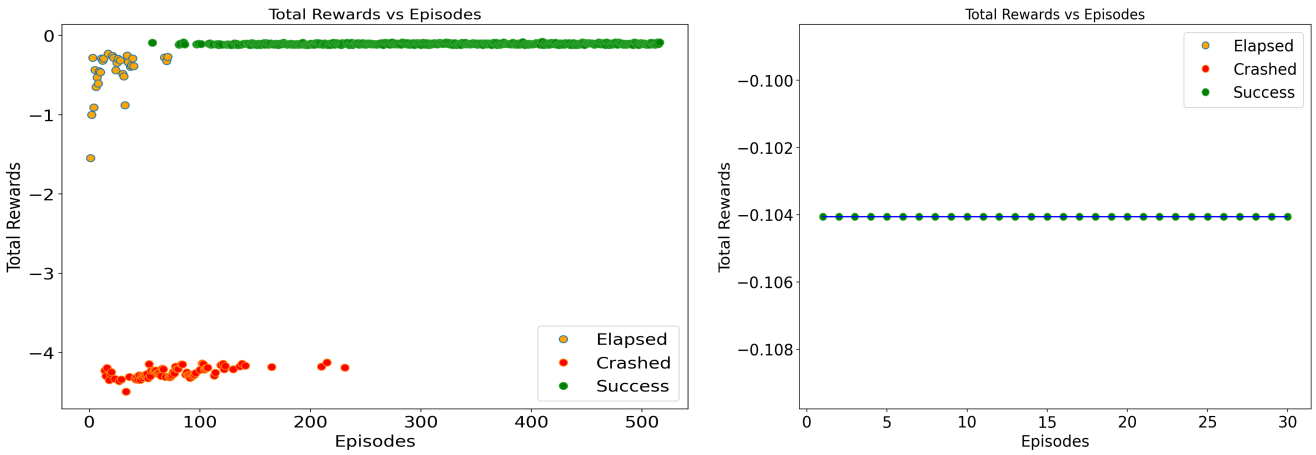
Figure 5.1: Learning with zero buffer

5.0.3 SAC with default buffer

Isolated Environments In this experiment, the agent is trained in environment with vertical parking slots and positioned at the east side facing in the west direction as shown 4.8 using a default replay buffer. It is then deployed in the same environment.

During Learning, the agent is able to learn to park successfully after 100^{th} episode. To train, the script is called in the following way:

```
 python park_model.py --mode learn --episodes 100 --filename sac_straight
--goalSpotNumber 2 --buffer 1
```



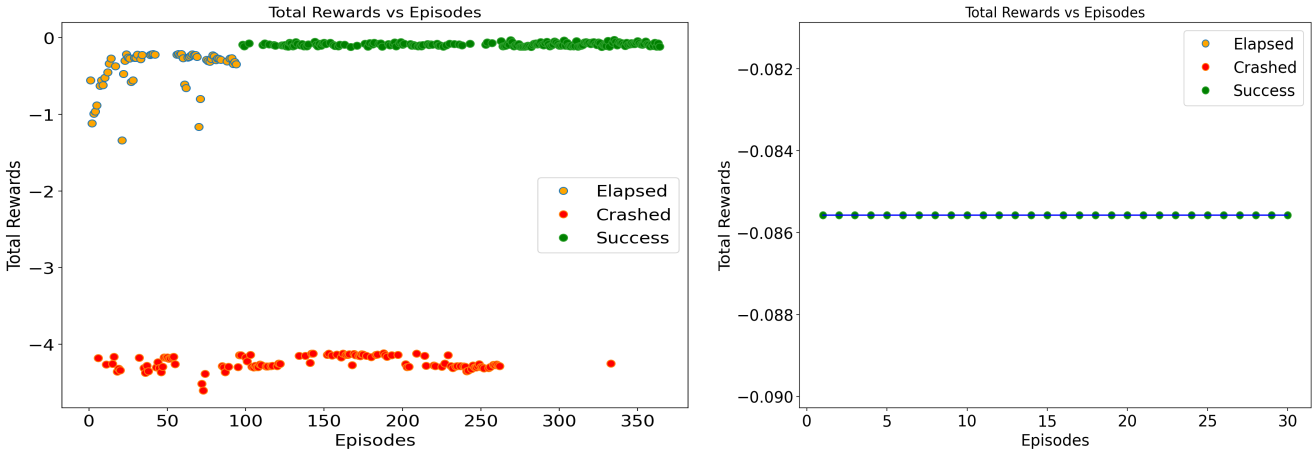
(a) Vertical Slots Learn using default buffer

(b) Scenario 4 Deploy using default buffer

Figure 5.2: Isolated Environment Vertical Slot

When the model is deployed in the same scenario, 5.2b we see a success rate of 1 as the agent is able to park in all the 30 episodes. Similarly for Diagonal slots 5.3a,the success rate starts to increase around 100^{th} episode onward and 5.3b shows successful parking in all episodes.

From the above experiments, we can conclude that the agent is able to learn for both vertical and diagonal slots in isolated setting very quickly.



(a) Diagonal Slot using default buffer Learn

(b) Scenario 5 using default buffer Deploy

Figure 5.3: Isolated Environment Diagonal Slot

Unseen Environment The agent trained on vertical slot from the previous experiment is now deployed in the diagonal slots 4.9 environment.Deploying the agent in an environment with a small variation in the slot angle will help in assessing the performance of the model in unseen environment. In 5.4, as expected we see a success rate of 0 as the agent makes the vehicle crash in all the 30 episodes. 5.5 shows the snapshot of the simulation environment

during the end of an episode.

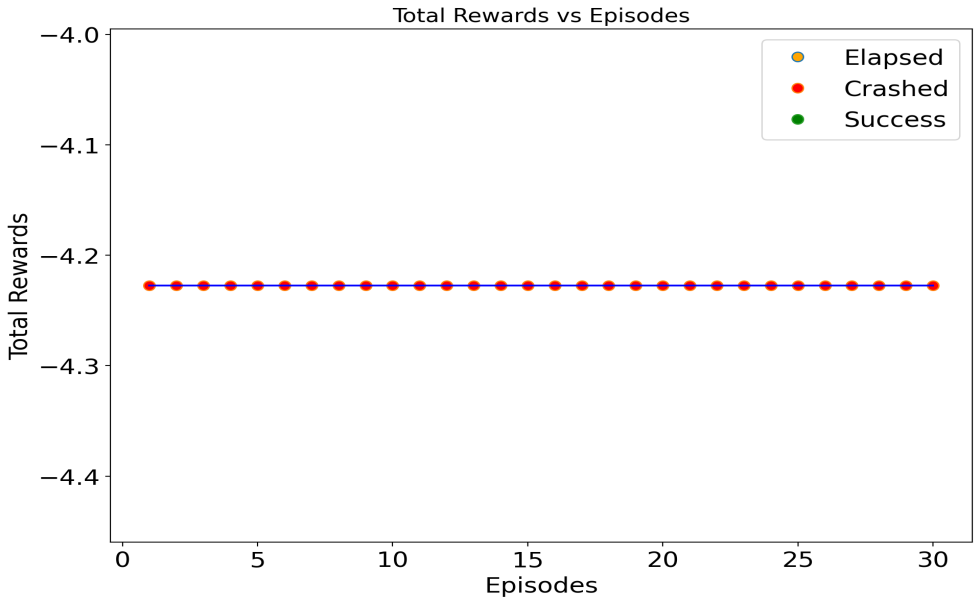


Figure 5.4: Unseen Environments

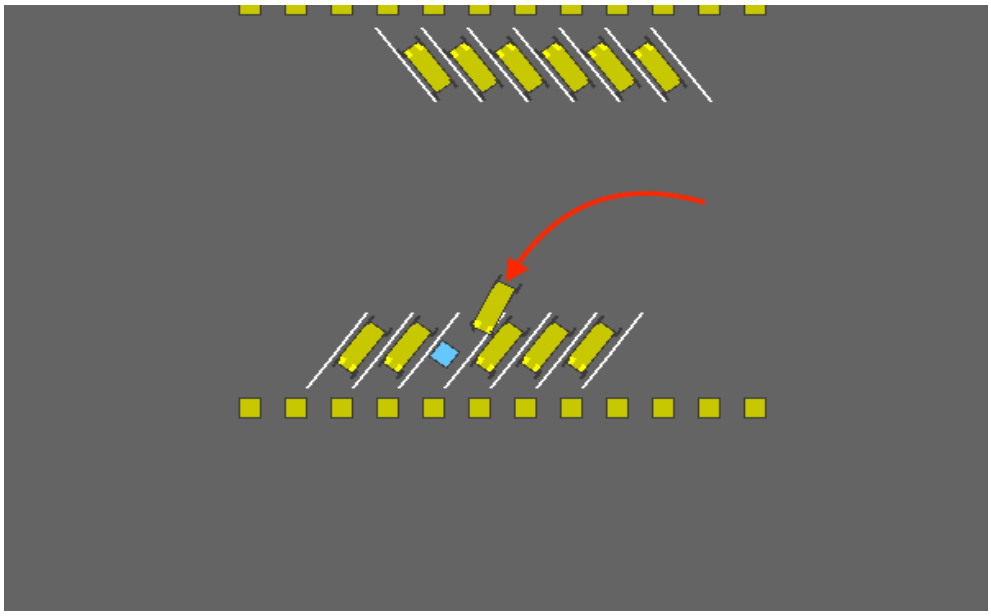
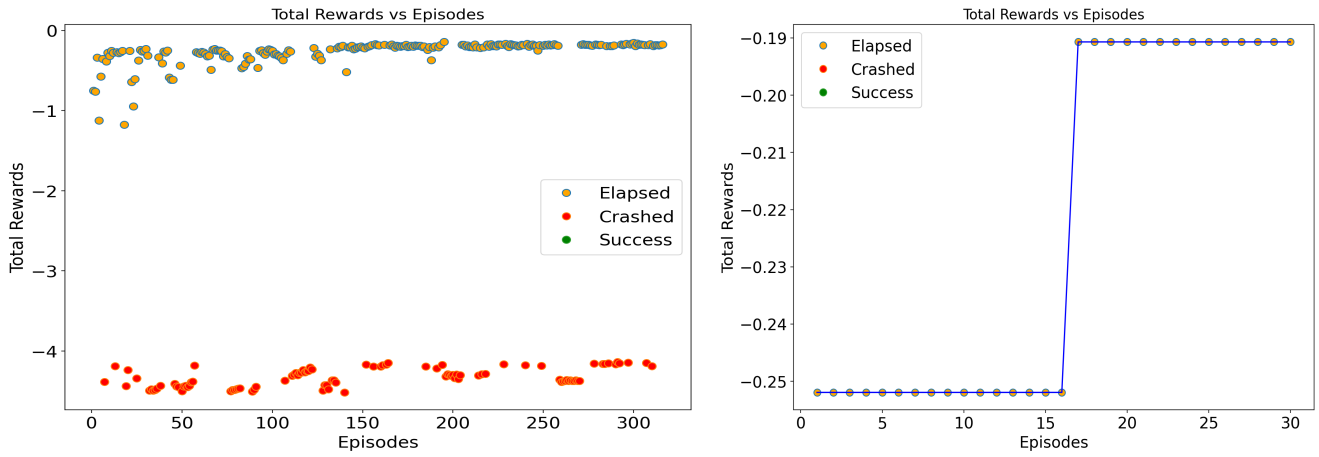


Figure 5.5: Unseen Environments Simulation crash

Phased Environments In this experiment, we want the agent to learn in a phased manner for a total of 300 episodes where the ego vehicle will be exposed to the vertical slots 4.8 for the 1st half of the simulations and the other half will have diagonal slots 4.9 (150 episodes for each). During learning 5.6a we observe that all of the episodes are either elapsing or getting crashed. One of the reasons behind getting elapsed episodes can be that agent gets quite close to the goal slot and is stuck. To confirm this, we will look at the velocity vs step graph in 5.7 during the training. The graph shows that towards the end of episode, the vehicle is stuck in a loop by moving back and forth near the goal lane. This means that the

policy is not able to converge in 300 episodes in either of the environments. 5.6b shows that during the deployment phase in both the environments, the agent comes very close to the parking but gets stuck in all the 30 episodes. The results are in contrast to training in the isolated environments experiment because we do not see any success in this case.



(a) Phased Learn

(b) Phased Deploy

Figure 5.6: Phased Environment

Interleaved Environments with 2 variations In this experiment the agent is trained in an interleaved environment of vertical 4.8 and diagonal slots 4.9. This means that during the learning, the agent can be in any of the 2 environments with a probability of 0.5.

In 5.8a we observe that we start to see success rate a little closer to 150^{th} Episode along with elapsed and crashed episodes. After 1000^{th} Episode, we only see successful episodes which means the policy has converged for both vertical and diagonal slots. 5.8b shows that deployment in a random environment of vertical and diagonal slots, we are getting a success rate of 1. This confirms that the agent is know able to park in both the environments successfully.

We will now rerun the experiments described above and use HER as our choice of replay buffer and analyze if we can get any improvement in our performance in the next section.

5.0.4 SAC with HER

Isolated Environments The agent is trained on the vertical slots 4.8 and then deployed in the same environment to see the performance. To train, the script is called in the following way:

```
python park_model.py --mode learn --episodes 100 --filename sac_straight
--goalSpotNumber 2
```

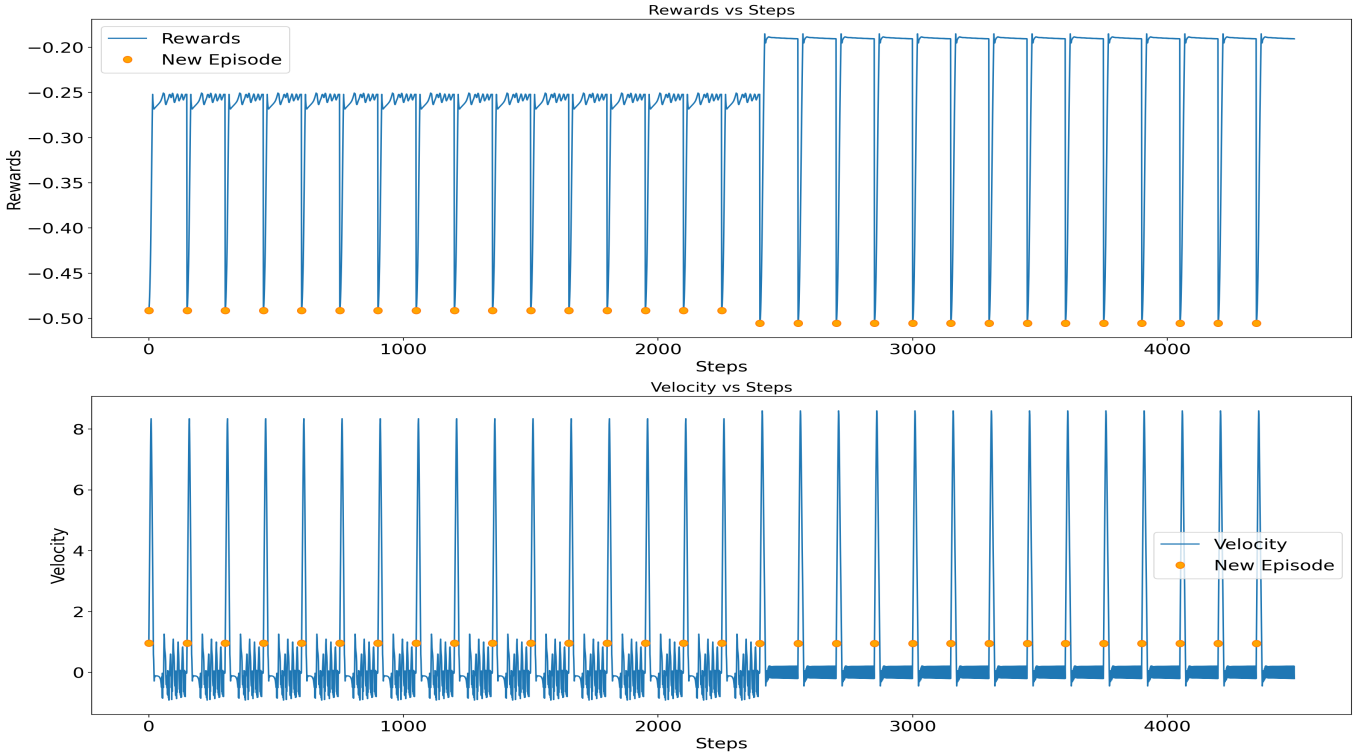
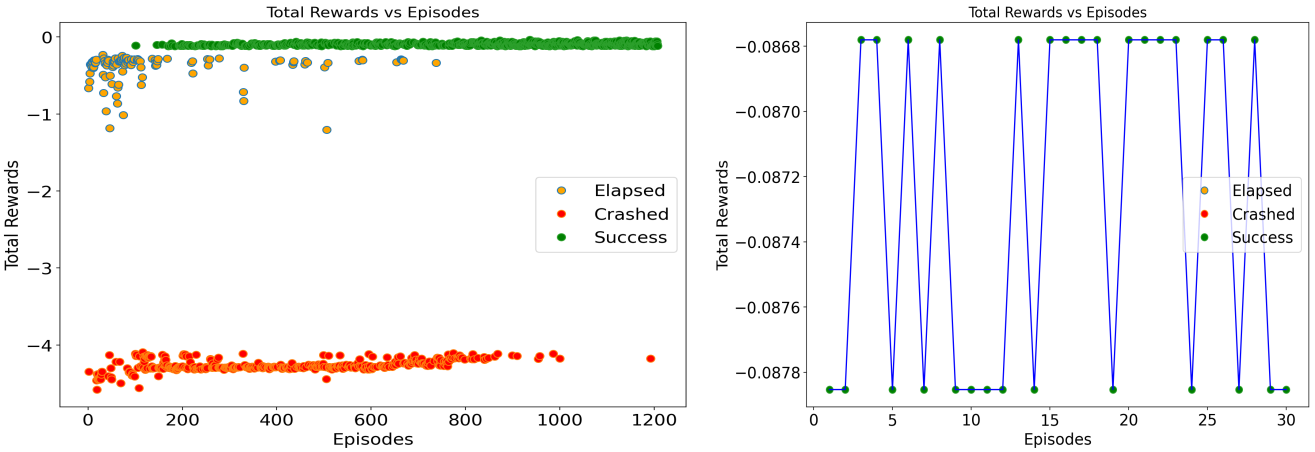


Figure 5.7: Rewards vs Steps(Up) and Velocity vs Steps(Down)



(a) Random Learn

(b) Random Deploy

Figure 5.8: Interleaved Environment with 2 variations

During Learning, 5.9a we can see that the car is either unable to reach the the parking slot or it is crashing in the first 50 episodes. At approximately, 70^{th} episode, we can start to see some success in parking rate and it keeps on getting better there after. Even though, the policy is able to converge fairly quickly after 70^{th} episode, we do observe a lot of crashes as well. This means that our algorithm is not over-estimating the rewards and is exploring other paths. This is expected because SAC maximizes entropy in addition to rewards which leads to more exploration as compared to exploitation. It is observed that as compared to the 5.3a where we use the default buffer, learning with HER is slightly more efficient. Please note that even towards the end of the episode when the success rate is very high, we see occasional crashes which means model is still trying to explore but however the frequency is less. When the model is deployed in the same scenario, 5.9b we see a success rate of 1 as the agent is able to park in all the 30 episodes.5.10 shows the path followed by the trained agent.To deploy, the script is called in the following way:

```
python park_model.py --mode run --episodes 100 --filename sac_straight
--goalSpotNumber 2
```

We perform this experiment on 4.9 with diagonal slots.To train, the script is called in the following way:

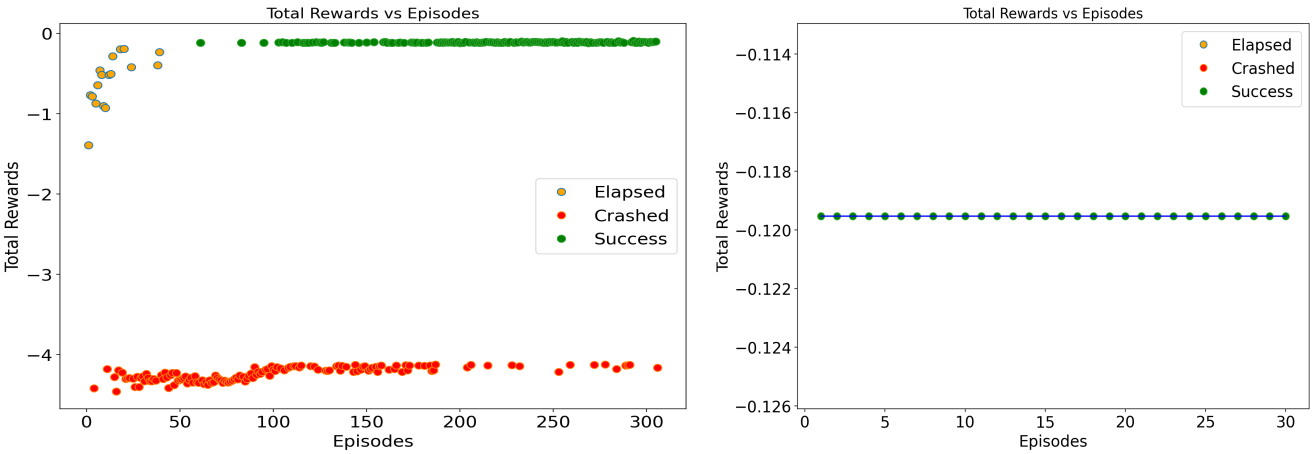
```
python park_model.py --mode learn --episodes 100 --filename sac_diagonal
--goalSpotNumber 2 --diagonalShift 6
```

We see that the observations for diagonal slots, 5.11a and 5.11b, are similar to the vertical slots. 5.12 shows the path followed by the trained agent. To deploy, the script is called in the following way:

```
python park_model.py --mode run --episodes 100 --filename sac_diagonal
--goalSpotNumber 2 --diagonalShift 6
```

Unseen Environments The trained agent on vertical slots 4.8 environment from the previous experiment is deployed in the 4.9 diagonal slots environment to see the impact of the performance with a small variation in the slot angle. To deploy, the script is called in the following way:

```
python park_model.py --mode run --episodes 30 --filename sac_straight
--goalSpotNumber 2 --diagonalShift 6
```



(a) Vertical Slot Learn

(b) Scenario 4 Deploy

Figure 5.9: Isolated Environment Vertical Slot

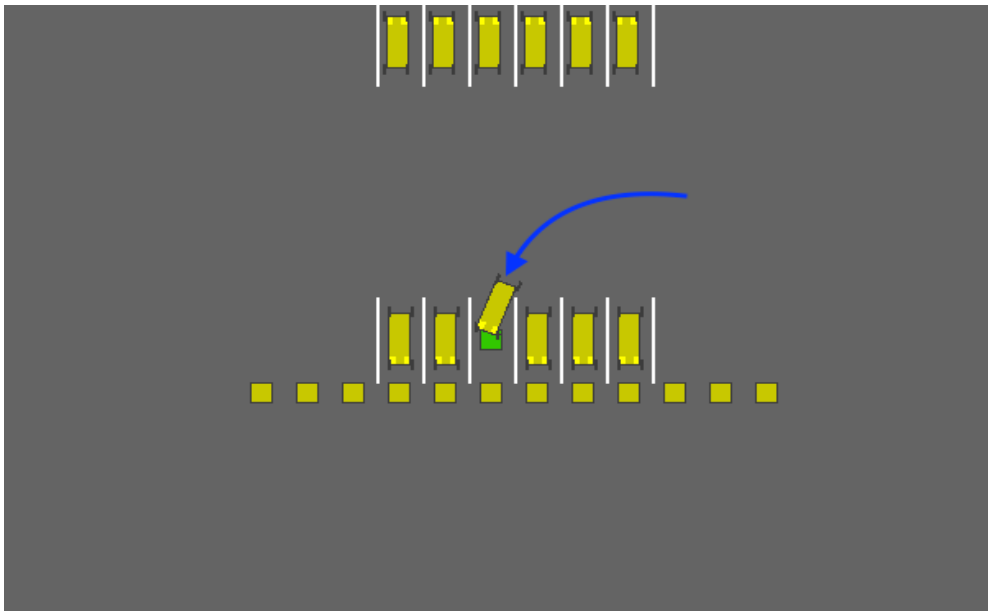
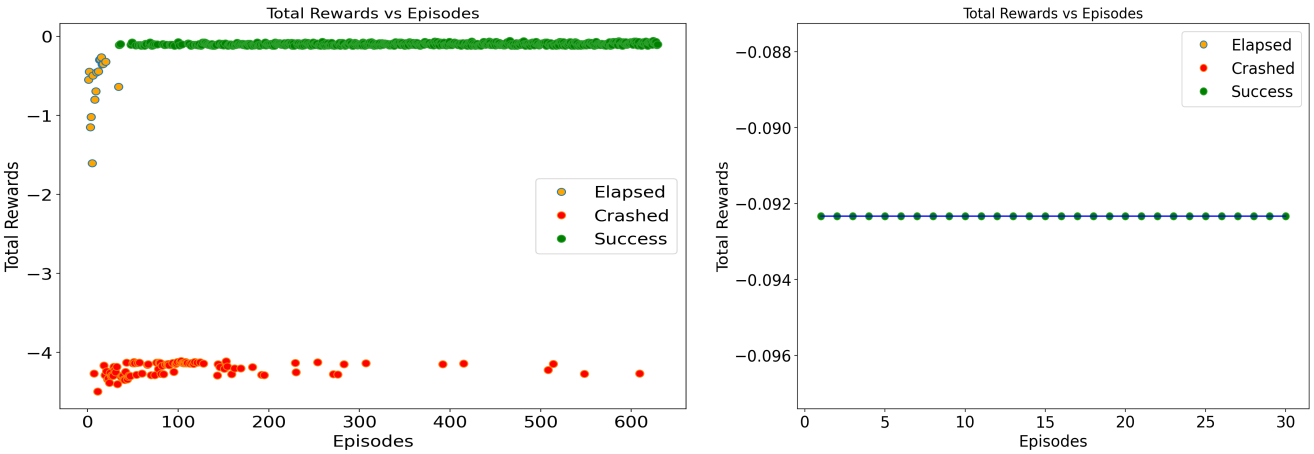


Figure 5.10: Vertical slot successful park



(a) Scenario 5 Learn

(b) Scenario 5 Deploy

Figure 5.11: Isolated Environment Diagonal Slot

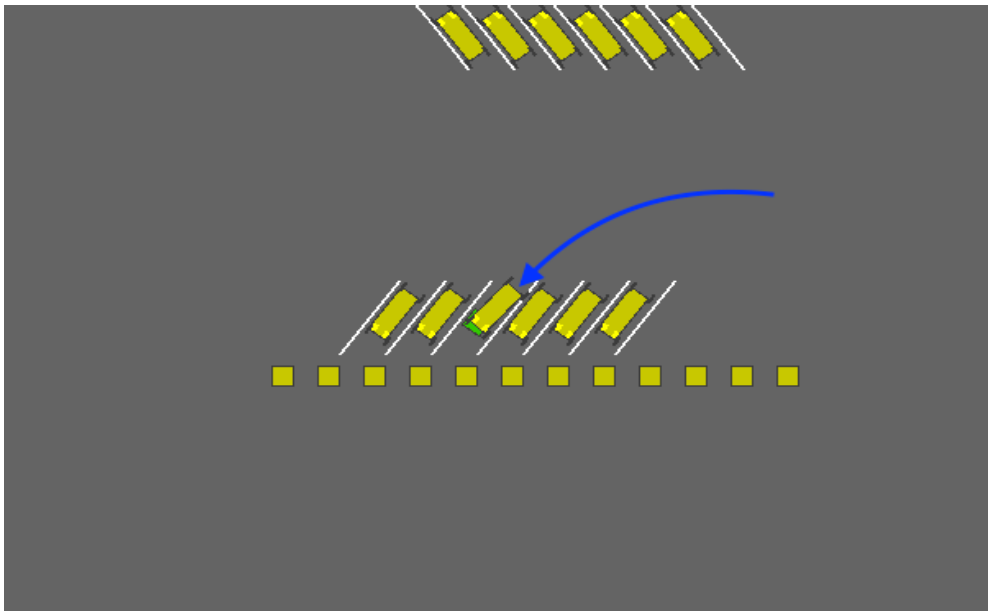


Figure 5.12: Diagonal slot successful park

In 5.13 as expected, we see a success rate of 0 as the agent makes the vehicle crash in all the 30 episodes. This means that even after using the HER, we do not get any change in generalizing performance in unseen environments.

5.14 shows the snapshot of the simulation during the end of the episode.

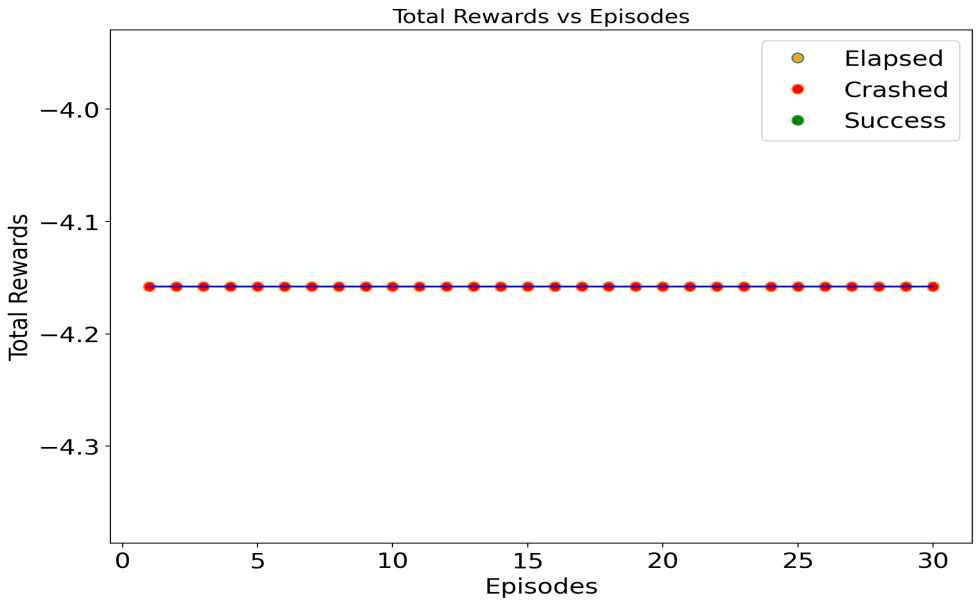


Figure 5.13: Unseen Environment

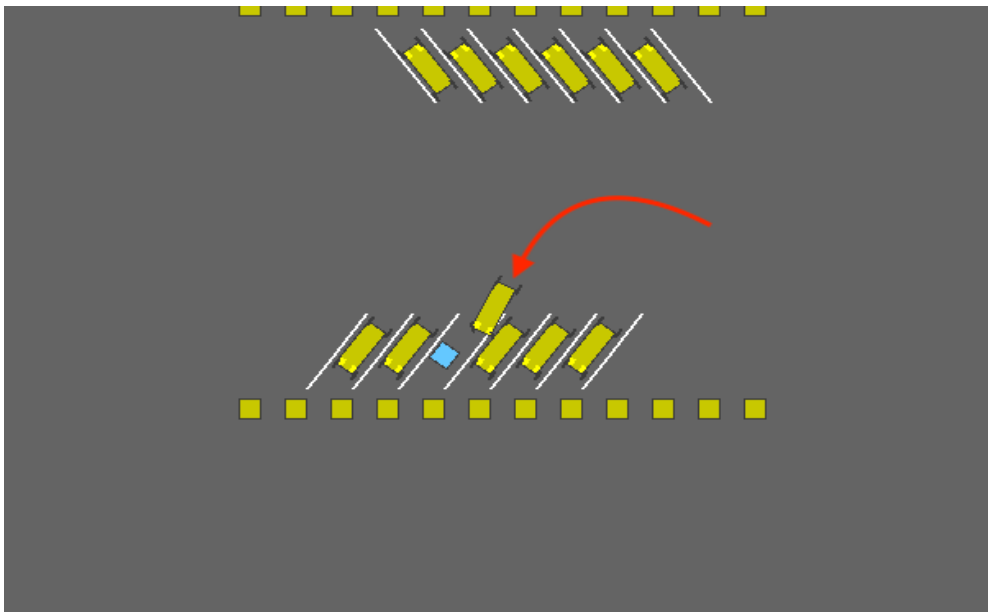


Figure 5.14: Agent crashing in diagonal slot

Phased Environments : The agent is now trained in a phased manner starting from vertical slots 4.8 and then diagonal slots 4.9. One thing to note here is that during the phased learning, if the number of episode is 200, the slots will be vertical for the 1st 100 episodes and then it will switch to diagonal slots for the remaining 100 episodes. Up to 100

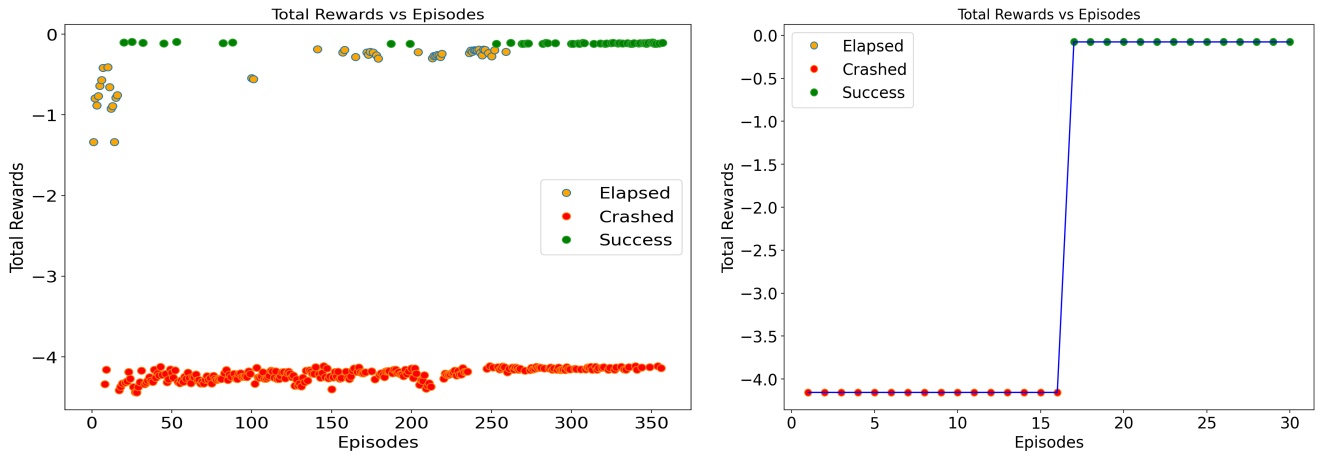
episodes in 5.15a the agent learns how to park in vertical slots, but as soon as it detects diagonal slot after 100^{th} episode, we see the agent performance deteriorates and it takes time to relearn in the new environment. After we switch to the diagonal slot, we observe that there are still a some elapsed episodes with a lot of crashes which means that agent learns the new environment from scratch and does not reuse any of it's learned experience. To learn, we call the script is called in the following way:

```
python park_model.py --mode phasedLearn --episodes 250 --filename
sac_phased --goalSpotNumber 2 --diagonalShift 6
```

To deploy, the script is called in the following way:

```
python park_model.py --mode phasedRun --episodes 30 --filename
sac_phased --goalSpotNumber 2 --diagonalShift 6
```

When the trained model is now deployed 5.15b for both the scenarios, we see that the model perform perfectly in 4.9 (Episode 16-30) but crashes the vehicle in 4.8 Episode(1-15).The simulation can be seen for a crash in 5.16a and a successful parking in 5.16b. This means as the agent is forgetting what it learned once it has been exposed to a variation in the environment.



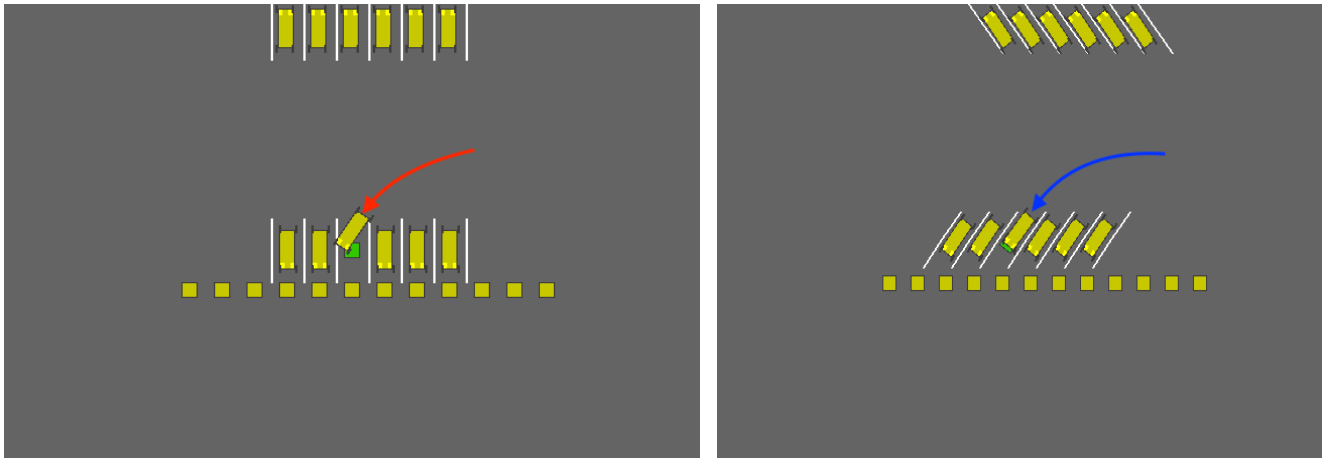
(a) Phased Learn

(b) Phased Deploy

Figure 5.15: Phased Environment

Interleaved Environments with 2 variations The agent is now trained in an interleaved environment of 4.8 and 4.9. This means that agent can be in any of the 2 environments with a probability of 0.5. To learn, the script is called is called in the following way:

```
python park_model.py --mode randomLearn --episodes 250 --filename
```



(a) Vertical slot crashed (b) Diagonal slot successful park

Figure 5.16: Phased Environment Simulation

```
sac_random --goalSpotNumber 2 --diagonalShift 6
```

5.17a shows the training of the agent in this random environment. We notice a lot of crashes in the 1st 100 episodes along with episode elapse without parking. We start to see successful parking from 70^{th} episode onwards. We notice a small number of episodes where time is elapsed without parking around 200th episodes.

The agent is now deployed in the environment where it will again see 4.8 and 4.9 scenarios with a probability of 0.5. To deploy, the script is called in the following way:

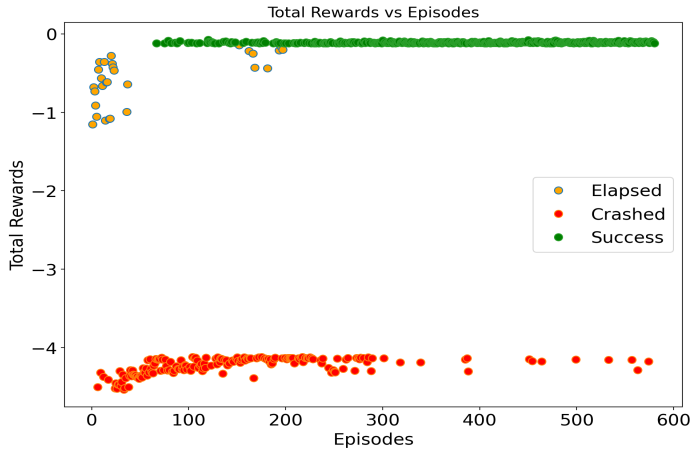
```
 python park_model.py --mode randomRun --episodes 30 --filename
sac_random --goalSpotNumber 2 --diagonalShift 6
```

The 5.17b shows that the agent is able to successfully park the car in both the environments with a success rate of 1 in all the 30 episodes.5.18a and 5.18b show the successful parking in these 2 environments.

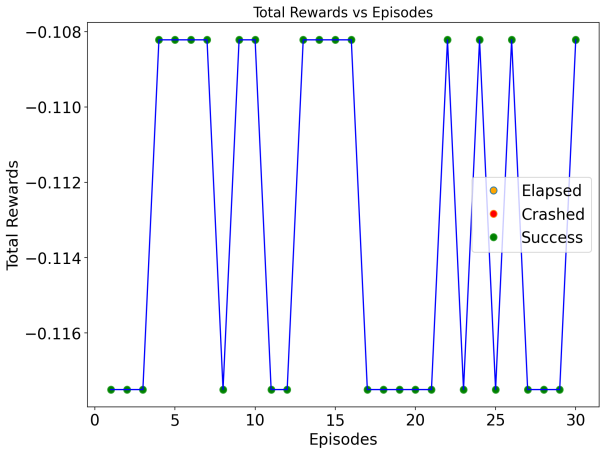
Interleaved Environments with 3 variations The agent is now trained in an interleaved environment of vertical slot 4.7 , diagonal slot 4.8 and parallel slot 4.9. This means that the agent can see any of the scenarios uniformly randomly during training and deployment both. To learn, the script is called in the following way:

```
 python park_model.py --mode randomLearn --episodes 1200 --filename
sac_random_3env --goalSpotNumber 2 --diagonalShift 6 --parallelParking
1
```

5.19a shows the training of the agent in this random environment. Here we can observe that our agent starts to learning the scenarios in 1st 500 episodes

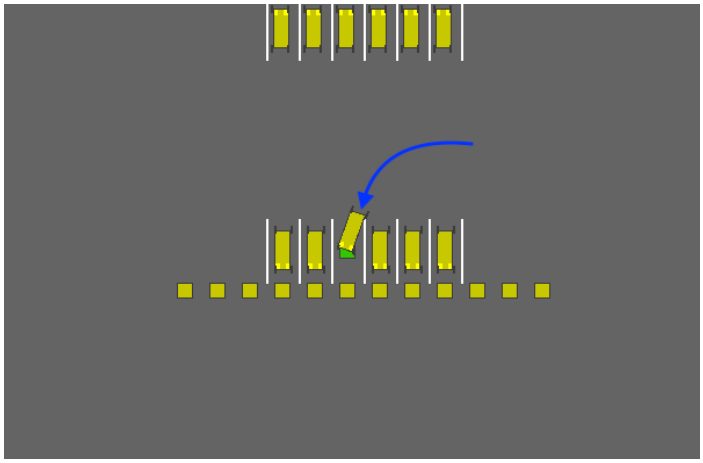


(a) Random Learn

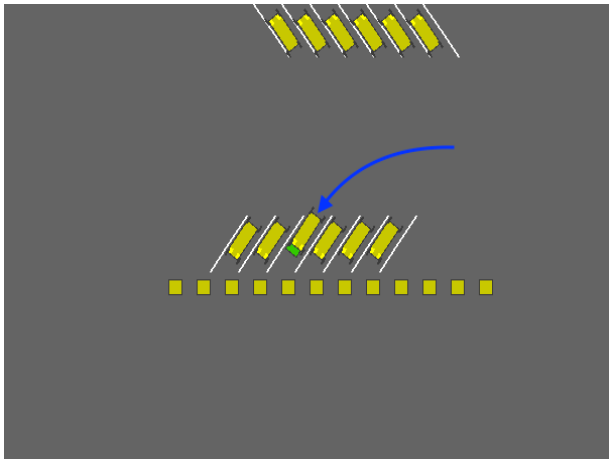


(b) Random Deploy

Figure 5.17: Interleaved Environment with 2 variations

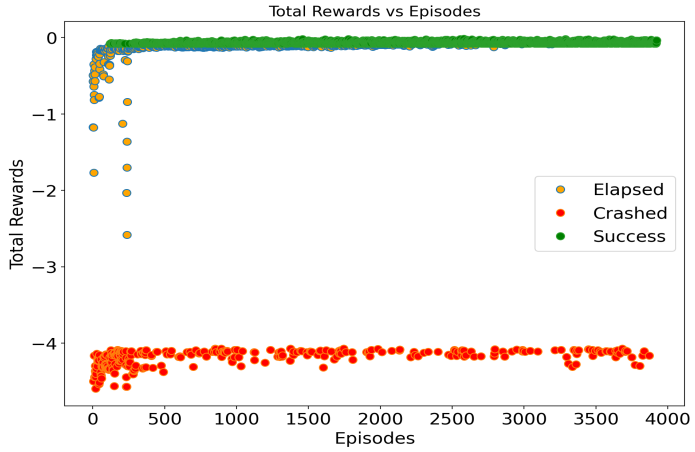


(a) Vertical slot successful park

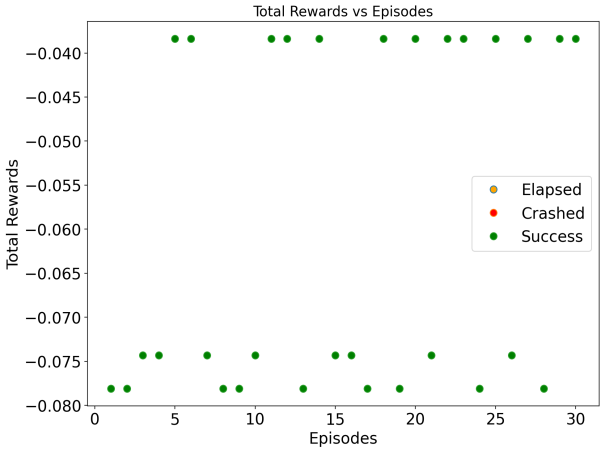


(b) Diagonal slot successful park

Figure 5.18: Interleaved Environment Simulation



(a) Random Learn



(b) Random Deploy

Figure 5.19: Interleaved Environment with 3 variations

To deploy, the script is called in the following way:

```
 python park_model.py --mode randomRun --episodes 30 --filename
sac_random_3env --goalSpotNumber 2 --diagonalShift 6 --parallelParking
1
```

The 5.19b shows that the agent is able to successfully park the car in all three environments with a success rate of 1 in all the 30 episodes.5.20 show the successful parking in for parallel parking.

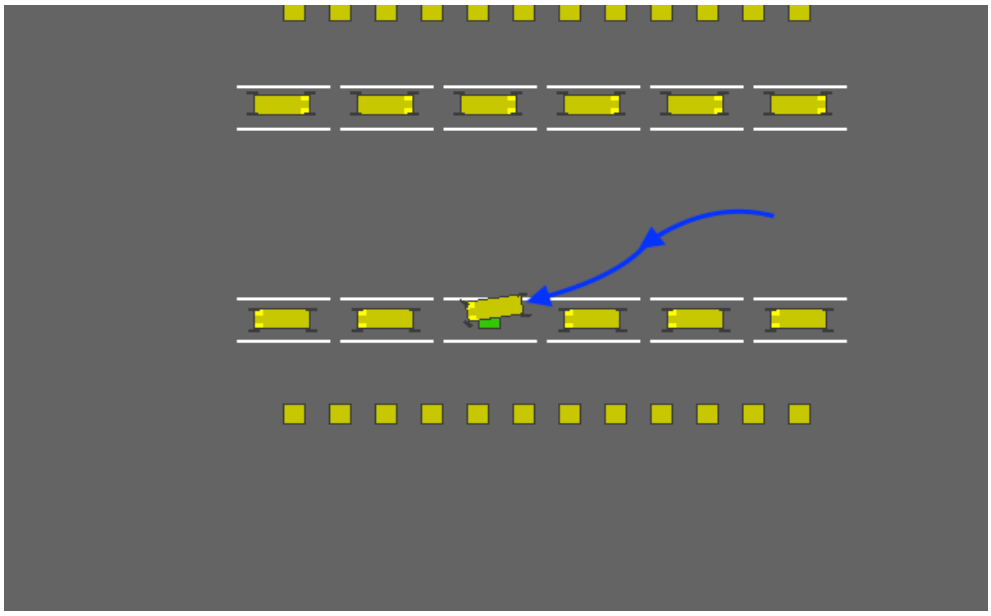


Figure 5.20: Agent parallel parking successfully

6 Conclusion

After running the above experiments, the following subsection presents the final conclusions

No Buffer Using no buffer at all makes training performance very poor. Over a period of 1000 episodes, we just see a single successful parking with few elapsed episodes in the beginning and then all the episodes end up in a crash.

Default Buffer We perform similar experiments with a default buffer. The agent learns to park in isolated environments for both vertical and diagonal slots quickly. There is no change in performance in Unseen environments where it crashes in all 30 episodes. However,for phased environment for vertical and diagonal slots we observe that the agent comes quite close to the parking slot but gets stuck in a loop moving back and forth between the same position. This means that with default buffer does not have a good sampling efficiency and makes training hard. The agent is performs well in an interleaved environment of vertical and diagonal slots where at the end of the training it is able to park the vehicle efficiently in both the lane orientations.

Using HER During the isolated environment of both vertical and diagonal, the policy converges fairly quickly when the learning and deployment environment is the same, however the agents performance quickly deteriorates when it is deployed in an environment with a slight variation to the one it is trained on as shown in experiment 2. On the other hand, when a phased approach for 2 environments is used as shown in phased environment, we observe that the learning places a higher bias on environment seen more recently and tends to 'forget' the past environment which is known as catastrophic forgetting. The agent is not able to generalize well using the phased approach even when the 2 scenarios only differ by a few degrees of lane angle. We get the best results when the agent is exposed to both the environment in a random fashion as described in the interleaved environment with 2 variations. The deployment in this interleaved environment shows that the agent is able to generalize well and perform successful parking in both vertical and angled slots with a high accuracy. The interleaved environment with 3 variations is an extension of the previous experiment where now we have parallel slot added on top of vertical and diagonal slots. From the learning we can however observe that it takes much more time for the agent to

generalize over all 3 variations as compared to experiment 4. For instance, the agent takes more time to generalize over all three orientations as shown in 5.19a because we see episodes being elapsed or crashed even at 2500_{th} episode.

From the above observations, we can conclude that HER makes the sampling efficient during training. The use of phased approach leads to catastrophic forgetting even when we use HER. The best way for an agent to learn is to expose it to all the variations in a uniformly random fashion during the training. The more degree of variation is present between the environments, the agent will take more time to converge to the optimal policy which works for all variations in the environment.

6.0.1 Future Work

This dissertation sets a basic framework for evaluating the agent's training and deployment performance as we introduce variations such as changing parking lane orientation in the environment. With respect to the future work, there are some additional experiments that can be performed which are described below.

Non-stationary vehicles We can introduce non-stationary vehicles in our environment which is quite close to real world scenario. Since the only non-stationary part of the current environment is through the changing parking lane orientation, adding moving vehicles into the mix will add more complexity to the learning and deployment process.

Multi lane and Multi level layout Another experiment would be how the agent will work in a parking layout much different to what it has seen before. Since our current experiments are limited to 2 rows of lanes, we can extend the experiment to multiple rows of slots stacked together. To add further complexity to this environment, another interesting experiment would be to see the agent's performance when the parking slot extends to multiple levels or floors in a building.

Varying Position and Speed Since our current experiments have an ego vehicle positioned in the east having a head angle pointing towards west, further experiments can be conducted where we can vary the initial position, the velocity and angle of heading in various combinations and see the effect on the learning and deployment performance.

Varying Position and Speed Since our current experiments have an ego vehicle positioned in the east having a head angle pointing towards west, further experiments can be conducted where we can vary the initial position, the velocity and angle of heading in various combinations and see the effect on the learning and deployment performance.

Using On-Policy Since our current experiments are done on SAC,an Off policy algorithm, we can experiment with using on policy algorithms like SARSA, TRPO etc

Other CL techniques In our experiments since the environment is customized to change the lane orientation, it is suitable for applying other CL learning techniques for example Gradient Episodic Memory (GEM).

6.0.2 Reflection

All of the experiments run in our thesis are a simulation of the parking lots and any analysis done are based on various assumptions described in Chapter 3. .As can be observed from the chapter 5, for learning in isolated environments, the agent is able to learn quickly in less than 100 episodes, but when agent is placed in small variation of environment they have seen before, they perform poorly. When agent is exposed to the variations in a phase wise manner, our agent is susceptible to catastrophic forgetting. When the agent is exposed to all the variations in a environment uniformly randomly, the agent tends to learn faster.

Since the learning process is very resource intensive and will takes a lot of time to train the model, deployment of the model on vehicles can be a challenge in the real world. On top of that, RL algorithms are susceptible to taking actions which might be incorrect which can be overcome by having emergency systems in place. For example a vehicle equipped with sensors can automatically enable brakes when there is a high chance of collision. The impact of a wrong action will be relatively less as compared to a fast moving highway environments because the speed can always be capped in parking environment.

Bibliography

[1] Open AI. Rl taxonomy. URL https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[2] Davide Abati. Introduction to continual learning. URL https://drive.google.com/file/d/11A3zxBAOYAGl8yELE0YcbaIlEt_wUYgX/view.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

[4] Ketan Doshi. Policy gradient, 2021. URL https://towardsdatascience.com/reinforcement-learning-explained-visually-part-6-policy-gradients-step-by-step-f

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[6] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. URL https://arxiv.org/abs/1712.01815.

[7] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives, 2020. URL https://arxiv.org/abs/2012.13490.

[8] Davide Abati, Jakub Tomczak, Tijmen Blankevoort, Simone Calderara, Rita Cucchiara, and Babak Ehteshami Bejnordi. Conditional channel gated networks for task-aware continual learning, 2020. URL https://arxiv.org/abs/2004.00070.

[9] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress amp; compress: A scalable framework for continual learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of

Proceedings of Machine Learning Research, pages 4528–4537. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/schwarz18a.html.

[10] Wikipedia contributors. Deep reinforcement learning — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Deep_reinforcement_learning&oldid=1100606157. [Online; accessed 11-August-2022].

[11] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2017. URL https://arxiv.org/abs/1707.01495.

[12] Gokhan Egilmez and Omer Tatari. A dynamic modeling approach to highway sustainability: Strategies to reduce overall impact. *Transportation Research Part A: Policy and Practice*, 46:1086–1096, 08 2012. doi: 10.1016/j.tra.2012.04.011.

[13] Bangalore Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad Sallab, Senthil Yogamani, and Patrick Perez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, PP:1–18, 02 2021. doi: 10.1109/TITS.2021.3054625.

[14] Myoung-jae Lee and Young-guk Ha. Autonomous driving control using end-to-end deep learning. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 470–473, 2020. doi: 10.1109/BigComp48618.2020.00-23.

[15] Wen-Yi Gu, Xin Xu, and Jian Yang. Path following with supervised deep reinforcement learning. In *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 448–452, 2017. doi: 10.1109/ACPR.2017.30.

[16] Zeyu Li. A hierarchical autonomous driving framework combining reinforcement learning and imitation learning. In *2021 International Conference on Computer Engineering and Application (ICCEA)*, pages 395–400, 2021. doi: 10.1109/ICCEA53728.2021.00084.

[17] Baramee Thunyapoo, 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)*, pages 576–579, 2020. doi: 10.1109/ECTI-CON49241.2020.9158298.

[18] Rikuya Takehara and Tad Gonsalves. Autonomous car parking system using deep reinforcement learning. In *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, pages 85–89, 2021. doi: 10.1109/ICITech50181.2021.9590169.

[19] Dinis Moreira. Deep reinforcement learning for automated parking, 2021. URL https://repositorio-aberto.up.pt/bitstream/10216/136074/2/494682.pdf.

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

[21] Junwu Zhao, Ting Qu, and Xu Fang. A deep reinforcement learning approach for autonomous highway driving. *IFAC-PapersOnLine*, 53:542–546, 01 2020. doi: 10.1016/j.ifacol.2021.04.142.

[22] Omar Tanner. Multi-agent car parking using reinforcement learning, 2022. URL https://arxiv.org/abs/2206.13338.

[23] Thomas Van Iseghem. A rl project focussed on autonomous parking. URL https://github.com/VanIseghemThomas/AI-Parking-Unity.

[24] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, 2015. doi: 10.1109/IVS.2015.7225830.

[25] Maximilian Ernestus Adam Gleave Anssi Kanervisto Rene Traore Prafulla Dhariwal Christopher Hesse Oleg Klimov Alex Nichol Matthias Plappert Alec Radford John Schulman Szymon Sidor Ashley Hill, Antonin Raffin and Yuhuai Wu. Stable baselines, 2018. URL https://github.com/hill-a/stable-baselines.

[26] Oleg Klimov Alex Nichol Matthias Plappert Alec Radford John Schulman Szymon Sidor Yuhuai Wu Prafulla Dhariwal, Christopher Hesse and Peter Zhokhov. Openai baselines, 2017. URL https://github.com/openai/baselines.

A1 Appendices

Parameter	Value	Definition
Success Goal Reward	-0.09	The total reward threshold which makes the episode successful
Grid Size	6	Number of slots on both sides of the parking lot
Collision Reward	-4	how much reward to give when vehicle collides
Goal Slot Number	2	Fixing Goal slot. If None, it will be picked at random.
Diagonal Shift	0	Diagonal displacement to change the angle of the slot.0 is vertical
Corridor Width	10	maneuvering empty area in between 2 parking rows
Slot Width	4	slot width
Slot Length	8	length of slot
Obstacle box	1	if 1, adds obstacles to either ends of the parking lot.
Other vehicles	1	if 1, adds parked to all the slots except the goal slot.
Steering Range	$[-\pi/4, \pi/4]$	maximum steering in either direction
Initial Position	[10,0]	Initial Position of the ego vehicle in [x,y] coordinates
Initial Heading	0.5	Initial direction of the ego vehicle. 0.5 means heading towards west.

Table A1.1: RL Environment specific parameters

Argument	Value	Definition
mode	learn	will run the script in a learning mode for a single environment
mode	phasedLearn	will run the script in a learning mode for 2 environments. 1st enivronment will run for half of the episodes and then 2nd environment will run for the other half.
mode	randomLearn	will run the script in a learning mode for 2 environments. 1st enivronment and 2nd environment will be picked up uniformly randomly.
mode	run	will run the script in simulation/deployment mode for a single environment.
mode	phasedRun	will run the script in a simulation/deployment mode for 2 environments. 1st enivronment will run for half of the episodes and then 2nd environment will run for the other half
mode	randomRun	will run the script in a simulation/deployment mode for 2 environments. 1st enivronment and 2nd environment will be picked up uniformly randomly.
episode	100	will run the script for 100 episodes
filename	sac_straight	after the learning/training is complete, it will save the model as sac_straight_timestamp.zip
her	0	if 0, will not use the HER during training otherwise will use HER.
saveGraphs	0	if 0, will not save the graphs to the disk otherwise will save.
timeDelay	1	will introduce a time delay of 1 second during deployment/simulation between each environment rendering step.
gridSizeX	6	there will be 6 slots on either side of the parking lot. Total 12 slots will be present.
disagonalShift	6	will displace the bottom part of the parking slot by a magnitude of 6 for giving an angle. 0 means vertical slot and any value above 0 will be in the order of increasing angle.
goalSpotNumber	2	will make the parking slot 2 as the goal slot. the bottom left slot will indexed at 0 and top right slot will be indexed at gridSizeX - 1.
duration	150	150 steps will mark the end of an episode

Table A1.2: Commandline Arguments