FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# Assignment of bachelor's thesis

| | |
|---|---|
| **Title:** | AWS DeepRacer Controller Training Scenarios Exploration for Real-World Perfomance |
| **Student:** | Yelizaveta Tskhe |
| **Supervisor:** | Ing. Miroslav Čepek, Ph.D. |
| **Study program:** | Informatics |
| **Branch / specialization:** | Software Engineering 2021 |
| **Department:** | Department of Software Engineering |
| **Validity:** | until the end of summer semester 2024/2025 |

## Instructions

AWS DeepRacer cars are a platform for experimenting with autonomous driving using machine learning. This thesis aims to explore required steps to transfer the machine learning controller from simulated environment to real world car and make it reliably navigate a track even in visually cluttered environment. The challenge is in the discrepancy between clean and fully controlled simulated environment and very noisy real environment. The noise appears in the inputs - like moving objects in the visual field of the camera or different light conditions. The challenge is also in noise in actions - like different velocity and steering behaviour due to battery charge level. The thesis aims to find techniques and approaches to address the problems and demonstrate the improvement in the car driving in the real environment.

Steps:
1) Review literature on machine learning techniques and approaches for self-driving cars. Review past bachelor theses on AWS DeepRacer cars.
2) Propose and train a baseline controller and demonstrate its performance in real-world environment.
3) Propose improvements to the training procedure to improve car's performance in the real-world.
4) Test your improvements and demonstrate the impact.

---

*Electronically approved by Ing. Michal Valenta, Ph.D. on 15 February 2024 in Prague.*

Bachelor's thesis

# AWS DEEPRACER CONTROLLER TRAINING SCENARIOS EXPLORATION FOR REAL-WORLD PERFORMANCE

**Yelizaveta Tskhe**

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing.Miroslav Čepek,Ph.D.
May 13, 2024

Czech Technical University in Prague
Faculty of Information Technology


Citation of this thesis: Tskhe Yelizaveta. *AWS DeepRacer Controller Training Scenarios Exploration for Real-World Performance.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

# Contents

# List of Figures

# List of Tables



# List of code listings

*I am profoundly grateful to my supervisor, Miroslav Čepek, for his invaluable guidance, patience and support throughout this thesis. His mentorship and expertise helped me in navigating through the complexities of research. I would also like to express the deepest appreciation to my family for their love and encouragement. It has been a source of strength and motivation for me. Additionally, I would like to acknowledge the professors who have played a significant role in my academic development throughout my bachelor studies. Their passion for teaching and dedication to their students have been truly inspiring.*

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Praze on May 13, 2024

# Abstract

This thesis aims to investigate the steps required to transfer a machine learning controller from a simulated environment to the real-world vehicle, allowing it to navigate a track in a safe and reliable way despite the visual differences. The main challenge lies in bridging the gap between the ideal conditions in simulated environment and dynamic, noisy real-world scenario. The noise manifests in the lighting conditions, light reflections, presence of foreign objects in the vehicle's camera as well as uncertainty in the speed and steering behaviour due to the battery level.

I have conducted an exhaustive review of the existing literature on the techniques and strategies for autonomously driving vehicles in order to gain a deeper understanding of the domain. Afterwards, I have trained and evaluated a reinforcement learning model that served as a benchmark for further experiments. Based on that, I proposed the potential improvements: training with a domain randomization and using a deeper 5-layer neural network. The improvements have been applied to the models in order to enhance the vehicle's performance in the real-world setting. They have been thoroughly tested and, as a result, domain randomization proved to be helpful, while using a 5-layer neural network did not seem to bring significant improvement due to the number of reasons.



**Keywords** AWS DeepRacer, autonomous driving, reinforcement learning, neural network



# Abstrakt

Cílem této práce je prozkoumat kroky potřebné k přenosu řídicí jednotky se strojovým učením ze simulovaného prostředí do reálného vozidla, aby bylo možné bezpečně a spolehlivě navigovat po trati navzdory vizuálním rozdílům. Hlavní výzva spočívá v překlenutí rozdílu mezi ideálními podmínkami v simulovaném prostředí a dynamickým, hlučným scénářem reálného světa. Šum se projevuje ve světelných podmínkách, odrazech světla, přítomnosti cizích objektů v záběru kamery vozidla a také v nejistotě rychlosti a chování řízení v důsledku stavu baterie.

Provedla jsem vyčerpávající přehled stávající literatury o technikách a strategiích pro autonomně řízená vozidla, abych získala hlubší porozumění této oblasti. Poté jsem natrénovala a vyhodnotila model učení s posilováním, který sloužil jako měřítko pro další experimenty. Na jeho základě jsem navrhla možná vylepšení: trénování s náhodným výběrem domény a použití hlubší pětivrstvé neuronové sítě. Tato vylepšení byla aplikována na modely s cílem zvýšit výkonnost vozidla v reálném prostředí. Byla důkladně otestována a ve výsledku se ukázalo, že doménová randomizace je užitečná, zatímco použití 5vrstvé neuronové sítě zřejmě nepřineslo významné zlepšení z řady důvodů.



**Klíčová slova** AWS DeepRacer, autonomní řízení, posílení učení, neuronová síť

# Abbreviations

| | |
|---|---|
| ANN | Artificial Neural Network |
| AWS | Amazon Web Services |
| CNN | Convolutional Neural Network |
| FNN | Feed-forward Neural Network |
| ML | Machine Learning |
| NN | Neural Network |
| PPO | Proximal Policy Optimization |
| ReLU | Rectified Linear Unit |
| RL | Reinforcement Learning |
| RNN | Recurrent Neural Network |
| ROS | Robot Operating System |
| SAC | Soft Actor-Critic |

# Introduction

In recent years, autonomous driving technology has significantly improved with the promising potential to reshape the mobility field. The central challenge is to develop reliable, robust, and safe systems with the capability to navigate complex, unpredictable real-world environments efficiently. AWS DeepRacer platform offers a space for developers to train, test, and deploy reinforcement learning models, that can be further used in self-driving vehicles in real-world scenarios.

This thesis focuses on the comprehensive exploration of the AWS DeepRacer training techniques and strategies with the goal of enhancing its performance in real-world conditions. The fundamental difficulty is the transfer of the model from the simulation environment with ideal conditions to the realistic noisy environment with varying conditions.

My goal is to review related works on the same topic, analyze the utilized methods and approaches, and compare them. Subsequently, I will create a reinforcement learning model and train it in the simulated environment. The evaluations will be conducted not only in the simulation environment but also in the real-world setting on the printed "A to Z Speedway" track under different conditions. After analyzing the model's performance, I will propose improvements to the training process that could possibly enhance the model's behavior in the real-world scenario.

# Chapter 1

# Foundational Concepts

*In this chapter, I describe the general concept of autonomous driving, including levels of autonomy, radars, sensors and cameras. Moreover, I explain different types of machine learning: supervised, unsupervised and reinforcement learning. I also discuss ethics and safety of autonomous driving.*

## 1.1 Definition of autonomous driving

There is no universal definition of autonomous driving. The word originates from Greek *autos* (independent) and *nomos* (human orders) in a sense that a human orders the vehicle to move by programming its behavior. The vehicle, in turn, autonomously continues driving obeying the rules and constraints that were considered while programming its behavior. It involves the use of complex systems, sensors and algorithms in order to control all processes, including steering, accelerating and braking.

### 1.1.1 Levels of automation

The Society of Automotive Engineers has defined five levels of automation.

- **Level zero** means no automation at all; a human must control the vehicle at all times and perform operations like steering, accelerating, decelerating, braking, observing and evaluating driving environment [1].
- **Level one** (driver assistance) allows a primitive system for driver assistance, anti-lock braking systems as well as stability control start. The driver assisting system can either control steering or acceleration and baking, however driver has to be fully aware of the environment and execute all operations [1].
- **Level two** (partial automation) automation uses more advanced assistance systems, including emergency brake and collision avoidance. The system can perform both steering and accelerating, braking. The vehicle still requires full attention and partial control of the driver [1].
- **Level three** (conditional automation) means the vehicle is driving autonomously but the driver must stay alert and take over the control at any time [1].
- **Level four** (high automation) is where the human attention is not needed anymore. The vehicle's system executes all operations and does not require driver's control [1].

- **Level five** (full automation) means the vehicle is able to drive autonomously in any road network and independent of weather conditions with unlimited driving modes [1]

. Nowadays, no vehicle is capable of full automation (level 5) and as stated by Toyota Research Institute, no one in the industry is capable of producing a vehicle with level 5 automation [1].

### 1.1.2 Sensors

Sensors and cameras are needed for the perception of environment. One of the most important features is recognising static and dynamic objects in the environment.

- **Monocular cameras** – 2D cameras with a passive sensor (does not emit any signal for measurements) that distinguish colors, which is important for traffic lights detection. However, this type of cameras lacks the depth perception in order to obtain more accurate information from the image [2].
- **Event cameras** – cameras producing output based on the changes in brightness. The data is recorded asynchronously for individual pixels with respect to visual stimuli. [2]
- **Radar (Radio Detection and Ranging)** – a sensor with the help of which 3D information, such as depth information, can be measured. It is an active sensor that emits radio waves that bounce back from the objects and measures the time of each bounce. The result from the radar is presented as a set of 2D coordinates that show the center of the object associated with the velocity at that point. There are different types of radar depending on the range: long-range radar (up to hundreds of meters), medium-range radar (up to fifty meters), short-range radar (up to few meters) as well as multi-modal radar that can operate at different ranges. Radars are cheaper than LiDARs and work at a greater distance but are less accurate. The weather and light conditions do not affect radars, while other types of sensors might struggle during rain or fog [2].
- **LiDAR (Light Detection and Ranging)** – works in the same principle as a radar but operates with light waves instead of radio waves. LiDARs are larger than radars, which might be a disadvantage when a smaller size required but are more accurate than radar on distance under 200 meters. The data acquired from a LiDAR is often used for the creation of 3D maps [2].

The *multisensor setup* can be also used for the vehicle as a combination of radar and LiDAR to achieve multiple goals and fault-free systems.

## 1.2 AWS Deepracer

AWS DeepRacer is a fully autonomous 1/18th scale car. The DeepRacer car is trained in the simulation by AWS Robomaker and the neural networks are updated by AWS Sagemaker [3].

The agent has a function that is used for approximation of the policy. The state is taken from the image from the car's cameras, which then defines the action taken by the agent. If the vehicle performs favorable actions, it receives a positive reward, otherwise, negative reward. One episode is the time period starting when the vehicle is placed on the track and ending either with the vehicle going off track or the vehicle completing the whole lap around the track while staying on the track at all times [4].

As it was stated in the AWS DeepRacer Developer Guide, the environment state might refer to everything connected to the problem. It can be either the vehicle's physical location on the track and the track itself or the image acquired from the vehicle's front cameras. The latter one does not capture the environment state fully, therefore the environment is considered partially observed and the state might be referred to as an observation [5].

AWS DeepRacer car is equipped with Wi-Fi capabilities. It has 4 MP cameras, a 360-degree 12-meter scanning radius LiDAR sensor, and an integrated accelerometer. The physical car can drive itself on the track or can be controlled manually. Autonomous driving is executed in the car's compute module, which uses images captured from the vehicle cameras[3].

**Figure 1.1** AWS Deepracer car [3]

The AWS DeepRacer is equipped with various sensors that allow it to perceive its environment and make independent decisions. These sensors usually consist of a front-facing camera, which acts as the main tool for visual perception, along with additional options like LiDAR and ultrasonic sensors for gauging depth and avoiding obstacles. The camera takes pictures of the surroundings, and onboard software processes these images to gather important details about the track, obstacles, and nearby objects.

| **Component** | **Specification** |
|---|---|
| Car | 1/18 Scale 4WD with Monster Truck Chassis |
| CPU | Intel Atom™ Processor |
| Memory | 4GB RAM |
| Storage | 32GB (expandable) |
| Wi-Fi | 802.11ac |
| Camera | Stereo 4MP Cameras with MJPEG (second camera is optional) |
| LIDAR Sensor | 360 Degree 12 Meters Scanning Radius LIDAR Sensor (optional) |
| Software | Ubuntu OS 16.04.3 LTS, Intel® OpenVINO™ toolkit, ROS Kinetic |
| Drive Battery | 7.4V/1100mAh Lithium Polymer |
| Compute Battery | 13600mAh USB-C PD |
| Ports | 4x USB-A, 1x USB-C, 1x Micro-USB, 1x HDMI |
| Sensors | Integrated Accelerometer and Gyroscope |

**Table 1.1** Specifications of the physical vehicle [3]

## 1.3 Introduction to machine learning

**Machine learning (ML)**, a subclass of artificial intelligence, is a method of learning from data and, based on that, making predictions or decisions. It is categorized to supervised machine learning, unsupervised machine learning, and reinforcement learning [6].

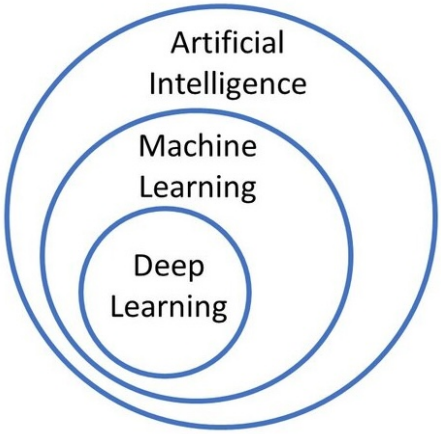



**Figure 1.2** Hierarchy [7]

### 1.3.1 Supervised Machine Learning

**Supervised machine learning** is a subclass of ML, where learning is made with labeled data. The data has already been divided into predetermined classes and new data has to be mapped to desired outputs. The input dataset is divided into train and test datasets. It is also referred to as a classification problem, where the machine finds patterns in multiple examples of mappings of input to output and, consequently, maps the given input. Other supervised machine learning algorithms are decision trees, Naive Bayes, and support vector machines [6].

### 1.3.2 Unsupervised Machine Learning

**Unsupervised machine learning** does not present the data divided into classes, it analyzes and clusters unlabelled datasets. It extracts some features from the analyzed dataset and uses them to classify the newly introduced data. An example of unsupervised machine learning algorithms might be a k-means clustering [6].

### 1.3.3 Reinforcement Learning

**Reinforcement learning (RL)** is a method of machine learning where the agent is trained in a dynamic environment through trial-and-error interactions. At each time step t, the agent receives a state st. in a state space S and selects an action at from an action space A, following a policy, which represents an agent's behavior (a mapping from state st to actions at, receives a scalar reward rt, and transitions to the next state st+1, according to environment dynamics, or model, for reward function R(s, a) and state transition probability P(st+1—st, at) respectively [6].

## 1.4 Ethics for autonomous driving

With all the benefits autonomous driving brings to humanity, there are also issues raised by it such as ethical challenges. The questions arise in the case when an autonomous vehicle crashes into another vehicle - *how the autonomous vehicle interacts with the human driver and how the responsibility is assigned.* Accidents on the road are inevitable, but the difference is that human drivers have limited control in such situations, whereas autonomous vehicles might have the opportunity to make decisions about that.

It is still questionable if so-called *"trolley cases"* are useful in such situations or not. **"Trolley case"** is a scenario where a trolley can result in some number of deaths if it goes one way or some other number of deaths if it chooses to go the other way. [8]. Similarly, an autonomous vehicle driving down a street can either crash into multiple pedestrians who ran onto the road or drive sideways to kill one pedestrian walking on the sidewalk [9]. There is another version of the *"trolley case"*, where an alternative to hitting a group of pedestrians is to be stopped by another vehicle that is pushed in front of the autonomous vehicle.

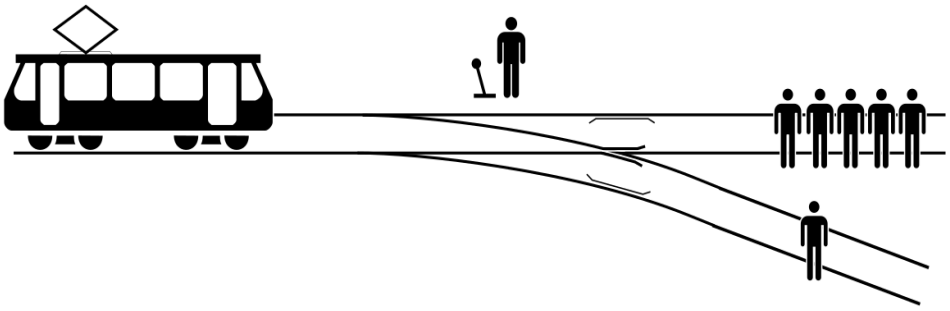


**Figure 1.3** Trolley case scenario [10]

It is believed by some people that the *"trolley cases"* have to be programmed, so in the case of a potential car crash, the autonomous vehicle behaves in a way pre-defined earlier [11].

On the other hand, **Trolley Pessimism** is the belief that it is incorrect to use *"trolley cases"* as a basis for programming autonomous vehicles' behavior in accident scenarios [8]. One of the reasons is the idea that imaginary experiments are far from realistic ones, so it is impossible to program the behavior of the vehicle in a future imaginary scenario. Another reason is the uncertainty of the autonomous vehicle's behavior in a particular situation: even though the vehicle's behavior is programmed with certainty, there is no absolute certainty in whether the expected outcome will be generated [12]. Autonomous vehicles fully depend on machine learning algorithms, so the training set has to be made of all possible situations with a desired input-output par for it. However, the decision about the proportion of daily driving scenarios and accident scenarios in the training set is made by the programmer and can differ [13].

It is clearly an ethical choice to make as one programmer might decide to not include any accident scenarios, because it is better to focus on the most possible daily driving situations, while the other one can dedicate half of the training set to accident scenarios. The goal of both programmers is to maximize the number of lives saved, but they plan to achieve it in different ways.

There is no certain answer to which way is better, but the conclusion is that programmers and ethicists should carefully evaluate the decisions they make considering the ethical principles.

## 1.5 Safety for autonomous vehicles

One of the key challenges of autonomous driving is the **safety of autonomous vehicles**. Even though autonomous driving has had a significant improvement over the past years, there are still no fully autonomous vehicles ready for use yet. A safe vehicle must be able to obey traffic laws and avoid road hazards effectively. There have been huge improvements in sensors in the 21st century, mainly in sonar, LiDAR, radar, vision sensors, and global navigation satellite systems [14]. The data for the decision-making of autonomous vehicles comes not only from sensors but also from communication with the infrastructure, other vehicles, the Internet, and the cloud. However, multiple errors cause accidents on the road.

One of them is a **perception error**. It happens on the *perception layer* which has the main function of acquiring data from sensors in order to perceive the environment for real-time decision-making. This type of error is mainly caused by failures in hardware (sensors) that create confusion in the decision system and result in dangerous driving behavior. It can be also caused by errors in the software when it misleads the other two layers (action and decision layers) causing safety problems [14].

Another type of error is a **decision error**. The *decision layer* acquires information from the perception layer, makes decisions, and passes the information to the action layer. The sources of such errors are system and human factors. The efficient autonomous vehicle must be able to detect road hazards and inform the driver to take over. This should happen immediately, however, it can take some time for a driver to respond and start the control of the vehicle [14].

Lastly, the **action error** happens on the last layer - *action layer*, which is responsible for controlling the vehicle using the information acquired by perception and decision layers. Its functions include steering the wheel, braking, accelerating, decelerating, and changing the direction. The main reasons for such errors might be a malfunction of the controlling system, the heat management system, or failure of actuators [14].

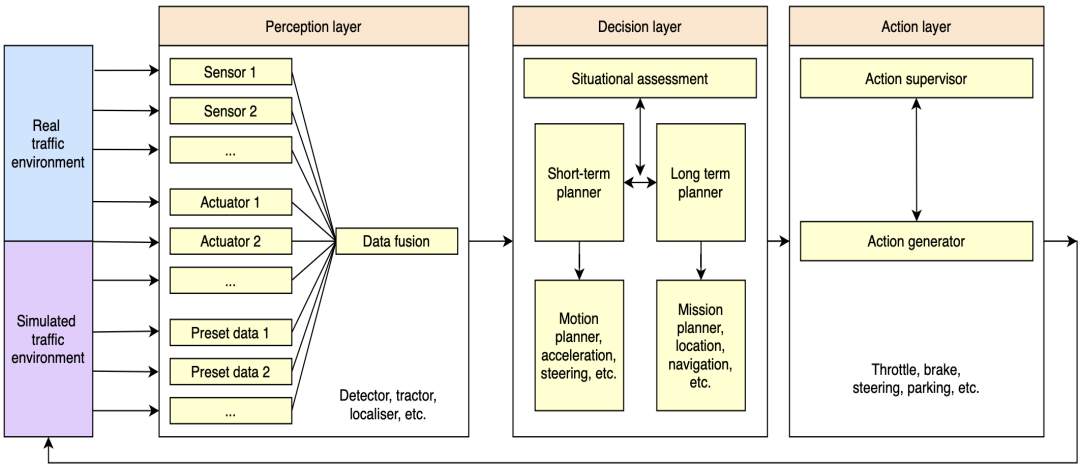



**Figure 1.4** Vehicle system architecture [14]

To minimize the errors of autonomous vehicles the objects in the environment must be detected, localized, and categorized effectively; the system must immediately respond to the changes in the environment with the help of a thoroughly tested, efficient decision-making system; the autonomous vehicle must wirelessly communicate with other vehicles (vehicular cloud) and road facilities.

## 1.6 AWS Cloud for Robotics

### 1.6.1 AWS Robomaker

**Amazon Robomaker** is a platform for developing, testing, and deploying robotic applications. It offers a cloud-based environment, in which developers can test their models under various conditions and random domains before transferring the model to the physical robot. Robomaker supports the Robot Operating System and its simulation environment replicates the real-world conditions accurately, allowing developers to test and adjust their models efficiently [15].

### 1.6.2 AWS Sagemaker

**Amazon Sagemaker** is a machine learning service for building, training, and deploying machine learning models. Even though it was not designed specifically for robotics, its functionalities are relevant for the development of robotics applications. It is possible to utilize the built-in algorithms or use custom models with support for frameworks such as TensorFlow and PyTorch. Sagemaker enables developers to go through the whole machine learning flow, including data labeling, model training, and deployment [16].

### 1.6.3 DeepRacer-for-Cloud

The platform where these 2 services integrate is a **DeepRacer-for-Cloud** environment which offers a comprehensive set of tools for the development and training of autonomous driving models in the AWS cloud. It has been used to train and fine-tune the models prior to their deployment to the physical vehicle [17].

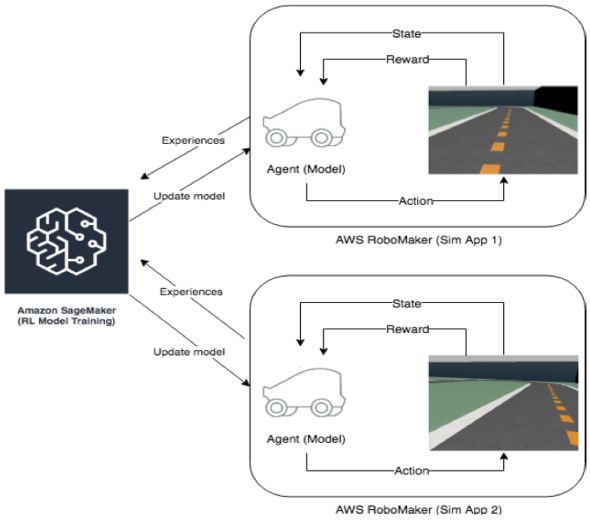



**Figure 1.5** AWS Sagemaker scheme [5]

## 1.7 Related Research

In recent years, the field of autonomous driving, along with significant advancements in Artificial Intelligence, has gained a surge of interest and innovation. AWS DeepRacer is one of the main platforms for the exploration and testing of RL models for autonomous driving. Having

conducted a comprehensive review of research and studies on this topic, I aimed to identify key strategies and methodologies as well as areas for further improvement in the field of autonomous driving with AWS DeepRacer.

One of the algorithms proposed is the hybrid-state A* path search algorithm. The environment is represented as a graph, in which nodes are specific locations and edges are connections between locations. There is a start state and an end state that the vehicle is aiming to achieve. However, each grid cell is associated with a continuous 3D state of the vehicle [18].

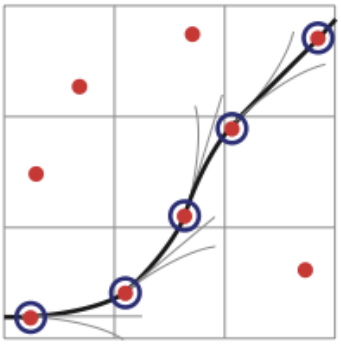


**Figure 1.6** Hybrid-state A* [18]

One of the most popular approaches is the Optimum Race Line algorithm has been implemented. The goal was to calculate the optimum race line path for the particular track in order to motivate the vehicle to follow that trajectory [19].

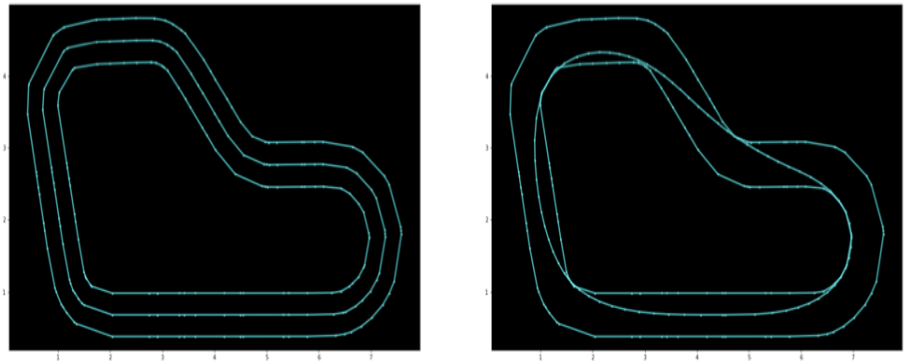


**Figure 1.7** Optimum Race Line trajectory [19]

However, the main disadvantage of this algorithm is that it is focused on one particular track only. Hence, it is not universal as the vehicle will perform well on the training track but poorly on the unknown tracks. Therefore, the more generalized reward function has been used, considering steering angle, speed, and vehicle staying within the borders of the track.

A relatively new race type: head-to-bot has been explored along with the comparison of different reward functions [4]. In another paper, the DeepRacer model has been transformed into the model for the full-sized autonomous vehicle and evaluated it in the real-world setting [20]. Moreover, the simulation of the real-world transfer has been researched and the filtering of training data has been performed in order to increase the robustness of the model in real-world conditions [21]. Furthermore, the human-machine co-driving has been compared to the fully autonomous driving [22].

# Chapter 2

# Theoretical background

*In this chapter, I describe reinforcement learning and its elements: agent, environment and reward function. Furthermore, I define what deep learning is and how it is used in autonomous driving. The multi-armed bandit problem is also outlined in the chapter.*

## 2.1 Reinforcement learning

**Reinforcement learning** is a training method of ML, where an agent interacts with the environment in a way to maximizes the numerical reward. The agent is not explicitly told which actions to perform, instead, it has to discover and explore different actions in order to find out which brings the biggest reward. The two most important features of RL are *trial-and-error search* and *delayed reward*. The agent must have a goal related to the environment, which can be achieved through interactions with the environment [6].

In comparison with supervised and unsupervised machine learning, reinforcement learning does not get to train on the labeled dataset or find some hidden structure, instead, it learns on its own through trial-and-error. Moreover, reinforcement learning focuses on the whole problem of an agent interacting with the environment to achieve a particular goal, while other types of machine learning tend to focus on subproblems without specifying how the solution to the subproblem will be finally used as a solution to the whole problem [23]. Modern reinforcement learning is strongly connected to psychology and neuroscience, as it is the most similar type of learning that human beings and animals do.

Reinforcement learning can be model-based or model-free.

- The *model-based type* of learning occurs when there is an environment (model) accessible, so the agent learns the police or a value function by interacting with it, while in the latter one, the agent has to find out the optimal policy through trial-and-error interactions with the environment.
- The *model-free reinforcement learning* can be divided into policy-based and value-based reinforcement learning. The policy-based approach (e.g. REINFORCE) learns the policy, which is a mapping of states to actions, while the value-based approach (e.g. Q-learning) focuses on predicting the value of actions in different states in the environment [23].

### 2.1.1 Elements of Reinforcement Learning

The main elements of reinforcement learning, besides agent and environment, are policy, reward signal, value function, and model of the environment as an optional element [23].

A **policy** determines the agent's behavior at a given time. Policy can be a simple function mapping perceived states of the environment to the actions that the agent can perform as well as more complex systems. The policy is a vital component of reinforcement learning, it is enough to define the agent's behavior [23].

A **reward signal** is a goal of reinforcement learning. After each action taken by the agent, the environment sends a reward, as a number, back to the agent. The agent's main objective is to maximize the goal in the long run, so the acquired low reward might contribute to the change of the policy to not select the same action in the consequent runs [23].

A **value function** specifies the reward that can be accumulated by the agent in the long run, starting from the given state. Reward immediately informs the agent if the action that has been just taken was good or not, while the value function considers the following steps and the rewards that will be acquired in the long run. There might be a case where the next state brings a lower reward but all subsequent states bring a big reward, resulting in the maximized total reward, or the reverse [23].

Rewards are considered primary and values are secondary in the sense that values are based on rewards. However, the values have a higher importance when an agent makes a decision.

A model of environment predicts the behavior of the environment, mainly the reward and next state, given a current state and action that was taken by the agent. Models are useful to plan the future actions to be performed by the agent.

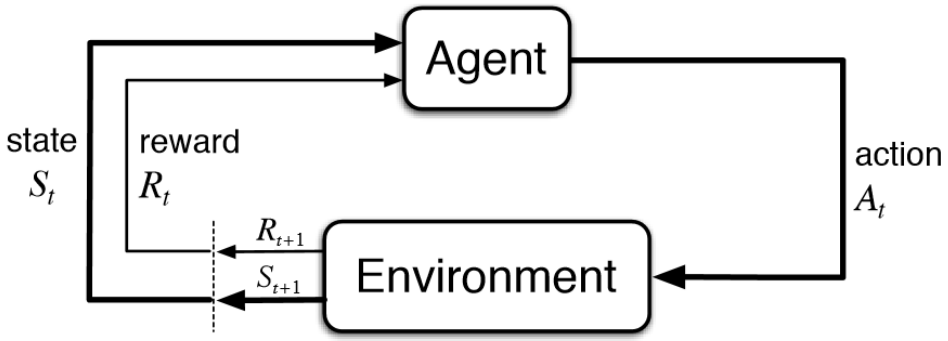



**Figure 2.1** RL elements [24]

## 2.2 Deep learning

**Deep learning** is a subclass of machine learning that exploits multiple layers of representations in order to model complex relations between data. This kind of learning usually uses neural networks. In autonomous driving deep learning models can be used for the detection of road signs and pedestrians. Deep learning methods significantly improved speech recognition, visual object recognition, object detection, and many other fields. It uses the back-propagation algorithm to discover the structure in large datasets and get a machine to change its parameters in order to compute the representation in each layer [25].

### 2.2.1 Components of a Deep Learning network

There are 3 main components of a deep learning network: an **input layer** which consists of nodes that input data into the artificial neural network, a **hidden layer** acquires information from the input layer and analyzes it from several different perspectives (as there might be multiple hidden layers), and an **output layer** consisting of nodes that output the data, where each node represents one answer [26].

Deep learning methods have multiple advantages over traditional machine learning methods, such as they do not require manual feature extraction which leads to the more efficient processing of unstructured data and they can analyze vast amounts of data and find hidden patterns.

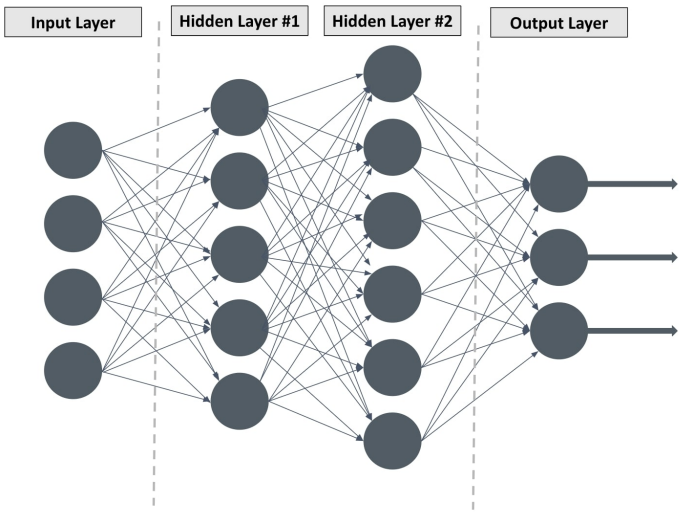



**Figure 2.2** Deep Learning network architecture[24]

## 2.3 Deep Reinforcement Learning in autonomous driving

There are multiple tasks where reinforcement learning could be applied in autonomous driving, such as *path planning, motion planning, trajectory optimization, development of driving policies for navigation tasks, scenario-based policy for highways, intersections, merges, and splits.* Path planning is one of the main problems in autonomous driving, mainly because of the dynamic environment in which vehicle has to pass through the intersection or merge into the highway. Another important feature is motion planning which ensures the existence of a path between start and end points [27].

The reward function for a deep reinforcement learning agent is a complex task that has to consider multiple factors, such as the speed of the vehicle, distance traveled towards the endpoint, interactions with the sidewalks, keeping the vehicle in lane and avoiding extreme steering, acceleration, and braking [27].

There are multiple challenges in using deep reinforcement learning for autonomous driving. For example, the sample efficiency - an autonomous vehicle requires a large number of samples to learn a reasonable policy, safety in autonomous driving - even the well-trained vehicle might be dangerous to deploy in the real environment, and autonomous driving being a multi-agent task - besides the autonomous vehicle, there will be multiple other actors, such as other vehicles, pedestrians, and cyclists [27].

## 2.4 Neural Networks

**Artificial Neural Networks (ANN)** are processing systems comprised of multiple interconnected computational nodes, also known as neurons, that learn from the input data [28].

### 2.4.1 Convolutional Neural Networks

**Convolutional Neural Networks (CNN)** are similar to the traditional ANNs in a way that they also consist of neurons, each of which receives the input and executes an operation. The main difference is that CNNs are commonly used for pattern detection within images. The neurons in the layers of CNN are organized into 3 dimensions: height, width, and depth. The height dimension is the number of rows in the input data, where each row is a horizontal line of pixels. The width corresponds to the number of columns in the input (vertical lines of pixels). The depth refers to the number of channels in the input data or each layer of CNN. The channels can be color channels or feature maps produced by a specific filter. For example, in the case of the image of 5 colors, the depth would be 5 (one channel for each color) [28].

CNNs are composed of 3 main types of layers: *convolutional, pooling* and *fully-connected layers*. The process starts in the **input layer** storing the pixel values of the input image [28].

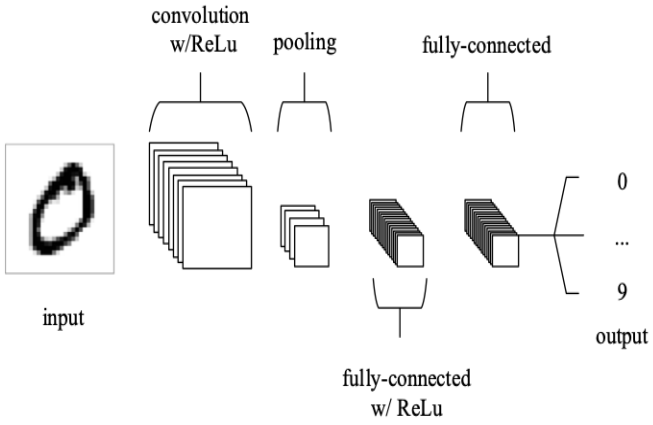



**Figure 2.3** CNN architecture [28]

Next, the **convolutional layer**, the core building block of CNN, extracts features. Each filter in the convolutional layer performs a convolution operation to the region of input and produces a feature map that represents a particular pattern or feature [28].

After that, the **pooling layer** executes the downsampling of the input, reducing the dimensionality of the input but preserving valuable features and patterns. This layer operates over each activation map in the input data and performs the max pooling operation, in which the maximum value in each region is preserved, while other values are discarded [28].

The **fully connected layers**, also known as dense layers, interpret high-level features extracted from previous layers and make decisions based on those patterns. Fully connected layers are usually used for tasks like classification or regression, where the learned features need to be mapped to specific output classes or values [28].

### 2.4.2 Feed-forward Neural Networks

**Feed-forward neural networks (FNN)** are one of the simplest types of ANNs. The information flows in one direction from the input layer to the output layer without any loops. FNNs consist of 3 layers: *input layer, hidden layer, and output layer*. The input and output layers are similar to the ones in CNNs, while the hidden layer is concealed between the input and output layers and serves as a computational core of the NN. This type of layer applies an activation function on input from the previous layer and transfers the output to the next layer. The NN may include 0 or more hidden layers [29].

There are 2 phases: *feedforward phase* and *backpropagation phase*. In the first phase, the input is fed into the FNN and it goes through hidden layers to the output layer, where the decision is made. The second phase starts after the result is obtained and the error, the difference between the prediction and real output, is calculated. This error is fed back to the NN and the weights used are adjusted in order to minimize the error [29].

### 2.4.3 Recurrent Neural Networks

**Recurrent Neural Network (RNN)** is a network containing a feedback connection, where activations go through in the loop. It is usually used on sequential data, such as text, numerical series, or handwriting, in order to detect features and patterns. One of the main differences is the way data goes through the NN. In comparison to FNN, data goes through cycles and feeds back [30].

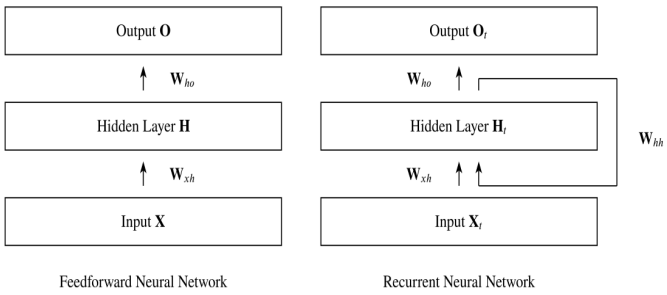



**Figure 2.4** FNN and RNN comparison [30]

The RNNs are difficult to train owing to its nonlinear iterative structure. Therefore, a small change can result in a huge effect multiple iterations later [31].

### 2.4.4 Overfitting

**Overfitting** refers to the model not being able to learn effectively anymore. It can happen for multiple reasons. The model can perform flawlessly on the training set but fails on the testing set. Overfitted models can learn the noise in the training set instead of extracting more meaningful features behind the input data. It is important to stop the training at the point when the model is neither underfitted nor overfitted. One of the ways to find out the point to stop training is computing the accuracy at the end of each iteration and stopping training at the point when accuracy does not improve anymore [32].

## 2.5 Multi-armed bandit problem

The **multi-armed bandit problem** was introduced in 1952 and it is directly connected to the exploration vs exploitation dilemma. There is a bandit with K arms (K actions to choose from) and T rounds. Every round one hand is chosen and the reward from that hand is collected. Hence, the reward on the other hand is unknown. The exploration in this case is that it is necessary to try different arms to explore, however, the already acquired information can be exploited and possibly lead to a greater reward [33].

There is no universal solution to whether exploration or exploitation is preferred in a dynamic environment. *Exploration* happens when a machine discovers new policies with the goal of improving performance. *Exploitation*, on the other hand, is using already discovered policies

that are known to be beneficial. Systems that choose to use exploration are likely to suffer the costs of experimentation without it bringing benefits, while machines engaging in exploitation will find themselves stuck in the same state without any improvements. Therefore, it is necessary to keep the balance between exploration and exploitation.

# Chapter 3

# AWS DeepRacer

*In this chapter, I present general information about AWS DeepRacer as well as its architecture. Additionally, the physical vehicle is described. Regarding the training process, information about virtual and real environments, training algorithms, reward function, action space and parameters is presented.*

## 3.1 Software architecture

The software architecture of AWS DeepRacer encompasses a set of elements working together to facilitate training, simulation, and deployment of RL models for autonomous driving.

An integral component of the DeepRacer software structure is the **simulation environment** which offers a virtual space for training and evaluation. The platform allows developers to use various tracks, race types, driving scenarios, and lighting conditions for the model to train under different real-world settings. Therefore, it makes it possible to iterate effectively and fine-tune the models before transferring them to the physical car.

The **perception component** within the software framework is responsible for processing the input data from sensors in order to perceive the environment. Its main function is to analyze images from the front-facing camera and recognize obstacles, track lanes, and other features required for driving.

The fundamental role is played by **reinforcement learning**, which facilitates the agent's learning process by interacting with the environment. It associates specific actions with rewards or penalties, gradually discovering efficient driving techniques.

## 3.2 Race modes

There are 3 race types available for AWS DeepRacer: time trial, object avoidance, and head-to-bot [5]. In the current thesis, I focus mainly on the time trial type.

- The **time trial** race focuses on optimizing lap times around a given track. The aim is to complete as many laps as possible within the time limit [5].
- The **object avoidance** race type requires the model to navigate through tracks while avoiding the obstacles placed along the track. It is necessary to use the reward function that enables models to detect obstacles, learn from mistakes, and make timely decisions in order to steer away from them [5].

- In **head-to-bot** races, the agent competes against ”bots” that represent a particular level of racing expertise. The model has to outperform the benchmark models by completing more laps within the given time or by completing the lap faster [5].

## 3.3 Training algorithms

There are 2 possible policy optimization algorithms for the AWS Deepracer: Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). These algorithms optimize the policy in order for the vehicle to execute actions that get the maximum possible reward [34].

### 3.3.1 Proximal Policy Optimization

The **PPO** algorithm is an on-policy algorithm that helps the agent learn a value function by using the observations made by the current policy. It works in both discrete and continuous action spaces. It uses trial and error to collect data about the environment while taking actions based on the initial policy. The policy is changed based on the reward acquired after the particular action by the agent, however, the change in the policy must be small which makes the PPO stable algorithm. It uses the clipping function to make sure that the new policy is not completely different from the old policy [34].

### 3.3.2 Soft Actor-Critic

The **SAC** is an off-policy algorithm that works only in continuous action space. It uses observations acquired from previous policies to learn its value function with the goal of maximizing reward as well the entropy [34].

*Entropy* measures how confident the policy is at choosing the action for the state given. Low entropy means the policy is confident in the action chosen, while high entropy means the policy is uncertain.

For training the models, I used the PPO algorithm, which is also a default algorithm for AWS Deepracer. It is well-suited for the environments with discrete action space, in which I trained the models in. The PPO is known for its stability during training as well as the sample efficiency. Hence, it required fewer training sets in order to achieve a good performance compared to other more advanced algorithms like SAC.

## 3.4 AWS DeepRacer Community

The **AWS DeepRacer Community** is the community created by enthusiasts willing to share their ideas and questions in order to expand their knowledge. It was founded after the AWS London Summit in 2019 and the initial purpose of it was to be a place for the top 10 winners to have insightful discussions, but it quickly evolved into a thriving hub for developers of all levels.

It used to be hosted on Slack, but recently it has been moved to Discord. Nowadays, there are about 500 enthusiasts, developers, and professionals gathered to discuss topics related to machine learning, reinforcement learning, and autonomous driving using AWS DeepRacer. There are various channels dedicated to different aspects such as virtual racing, physical racing, model training, and race announcements. The community members regularly organize racing events and tournaments, where participants can test their models and learn from others [35].

## 3.5 Robot Operating System implementation

**The Robot Operating System (ROS)** is an open-source system consisting of software libraries and utilities for the development of robot applications [36]. The AWS DeepRacer had been built as a **ROS 2 Foxy Fitzroy** architecture and application. It is considered to be the most secure and reliable ROS release [37].

ROS 2 has a layered architecture and distinguishes the ROS Client Layer (RCL) from the ROS Middleware Layer (RML). The former serves as the interface for developers, while the latter facilitates compatibility with various low-level communication protocols. Built upon the Data Distribution Service (DDS), the RMW enhances the robustness of ROS 2. This architecture effectively abstracts the details of low-level protocol from the application layer, enabling developers to focus on algorithms without having any concerns about the underlying structure [38].

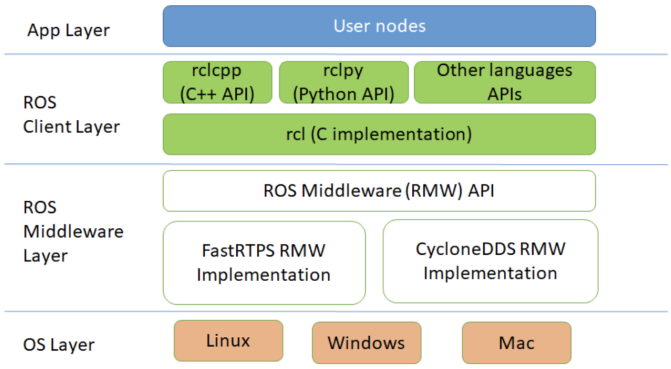



**Figure 3.1** ROS architecture [38]

Moreover, ROS provides a universal interface for the interactions with sensors of the AWS DeepRacer vehicle in order to acquire sensor data efficiently.

## 3.6 Simulation environment

The simulation environment used for training and evaluation was the **Deepracer-for-cloud** environment. It uses AWS Sagemaker and Robomaker, which run as docker containers during the training and evaluation processes. The environment allows developers to train reinforcement learning models in the simulation without the need for additional hardware.

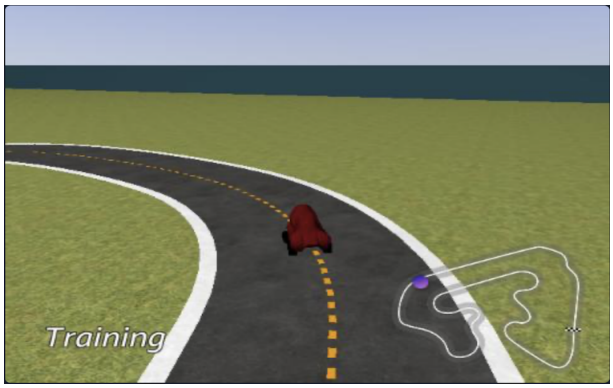



**Figure 3.2** Training in simulation

The environment is based on Unity, a game development engine, which ensures high-quality 3D graphics and realistic conditions [39]. It is possible to choose a track for the training and evaluation from the variety of them with different layouts and obstacles. Moreover, the virtual agent is equipped with sensors, such as a front-facing camera and optionally, a LiDAR or radar. These sensors provide inputs to the RL model, enabling it to perceive the environment and make decisions based on it. It is also possible to adjust the action space (discrete or continuous) along with the speed and the steering angle.

# Chapter 4
# Training

*In this chapter, I discuss the training process, the reward function and tracks I used for it. Besides that, I describe the results of training, the overall reward achieved and the routes that were taken on the training tracks.*

## 4.1 Problem Statement and Objective

Training and evaluating the vehicle in the simulated environment can perform flawlessly until the model is transferred to the physical vehicle, where its performance might dramatically drop. Transitioning from controlled simulations to real-world conditions presents a challenge.

The focus of this thesis is the **gap between simulated environments and real-world driving scenarios**. In particular, the inherent noise and variability in the real world. The noise manifests in various forms, such as dynamic objects in the visual field, unstable lighting conditions, and other variations, for example, connected to the battery charge levels.

The primary objective of this research is *to explore and develop techniques to bridge the sim2real gap, allowing the models trained in simulated environments to navigate real-world tracks safely.*

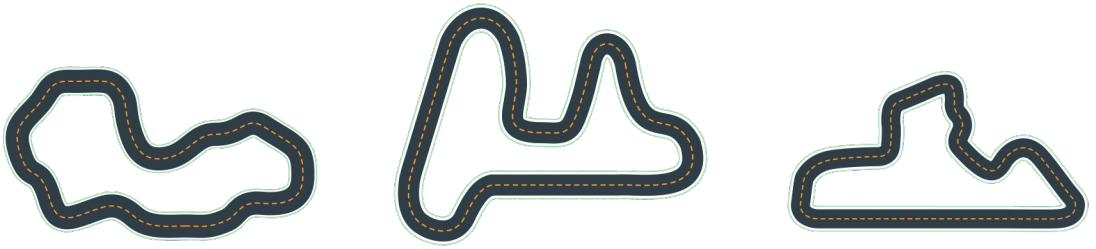
## 4.2 General training information

In order to achieve the objective of a car driving on unknown tracks, the DeepRacer car has been trained in a simulation environment Deepracer-for-Cloud on a **selection of tracks** of different difficulty and curvature, including sharp turns and straight roads, for the car to handle complex unexpected trajectories. There have been chosen *6 distinct tracks* to ensure the comprehensive training of the model. The models were trained locally without AWS Console based on the whitepaper about the local training configuration [40].

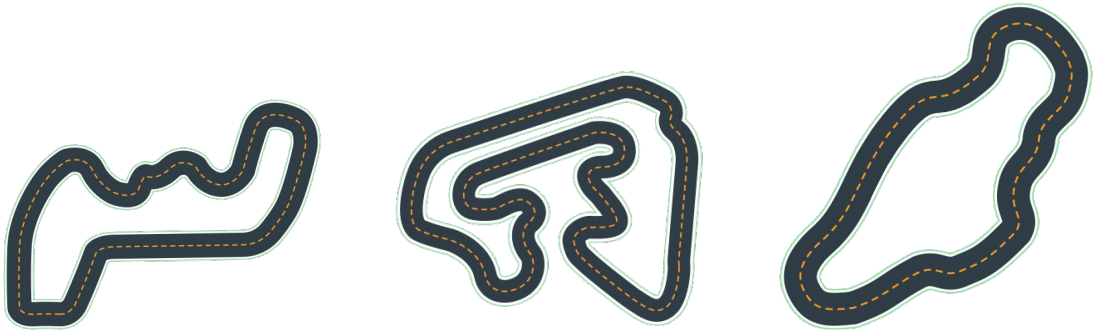
### 4.2.1 Tracks selection

The first track is *Penbay Open*, which has a pretty straightforward layout without any sharp turns. It helps the car learn the basic maneuvers and navigate through similar simple tracks. The next *Albert* track is more challenging than the Penbay Open, as it includes more tight turns and bends. The next 3 tracks chosen were *Thunder Hill Open, Dubai Open, and Canada Training* which represent a mixture of tight and gentle turns, resembling the real urban driving environment. The last one and the most complex one is *Red Star Pro* track. It has multiple tight

turns that require precise control of steering, braking, and navigating through the track, which makes this track suitable for training.



**(a)** Albert **(b)** Penbay Open **(c)** Dubai Open



**(a)** Thunder Hill Open **(b)** Red Star Pro **(c)** Canada Training

**Figure 4.2** Track layouts

## 4.3 Reward function

The agent learns the value function which uses the **reward function** to make a judgment on how good or bad the action taken was. It helps the agent learn the most optimal policies that lead to a greater reward. The optimal policy is considered the one that balances the exploration and exploitation time.

The reward function is one of the most important components of reinforcement learning, as it shapes the behavior of the agent, encouraging the vehicle to stay on track and maintain optimal speed.

I have experimented with multiple reward functions in the training process in order to find the right balance between exploration and exploitation as well as the right weights for the penalties and rewards. An example of the failed reward function is in 4.1. The model trained with that reward function has never converged. In contrast, the reward function in 4.2 proved to be effective and has been used as a main reward function in the training process.

### 4.3.1 Failed reward function

```
def reward_function(params):
    all_wheels_on_track = params['all_wheels_on_track']
    speed = params['speed']
    distance_from_center = params['distance_from_center']

    reward = 0

    SPEED_THRESHOLD = 2.0
    MIN_SPEED = 1.0
    if not all_wheels_on_track:
        reward -= distance_from_center

    if speed > SPEED_THRESHOLD:
        reward += 1
    elif speed < MIN_SPEED:
        reward -= speed * 10

    return float(reward)
```

**Code listing 4.1** Unsuccessful reward function

The model trained with the reward function in 4.1 has never converged even after numerous training iterations on multiple tracks. The initial goal was for the model to stay on track and maintain an optimal speed. However, the ways I defined penalization and rewards were too harsh. The penalty for going off the track and driving at a lower speed was too large compared to the reward that was given for having at least the optimal speed.

The reward function did not motivate the agent to explore and learn new patterns, as it struggled to discover what could bring the higher reward, it could not define the policy correctly. Therefore, the agent had difficulties associating rewards with any particular action, and over the iterations, the reward was declining (see 4.3).

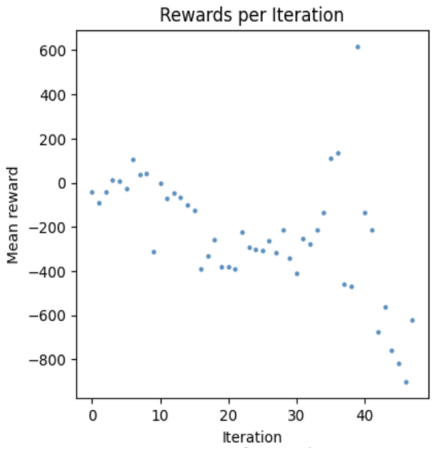



**Figure 4.3** Rewards per iteration for the unsuccessful model

The conclusion I made from this reward function was that penalization and rewards had to be balanced in a way that the agent could deduce which actions are favorable and which are

not. Hence, I had to adjust the weights for the parameters that I used in the reward function. Furthermore, I understood that in order to balance exploration and exploitation, it was essential to give the agent more frequent rewards for intermediate actions, so that it is encouraged to explore new actions and states.

### 4.3.2 Effective reward function

```
def reward_function(params):
    all_wheels_on_track = params['all_wheels_on_track']
    speed = params['speed']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']

    reward = 1e-3

    if all_wheels_on_track:
        reward -= distance_from_center / (0.5 * track_width)
        reward += 2 * speed
    else:
        reward = 0.1

    return float(reward)
```

**Code listing 4.2** Effective reward function

The reward function in 4.2 is the main reward function that has been used for training. Initially, the reward is 1e-3 as a base reward to establish a baseline. The main goal is to train the vehicle in a way that it can swiftly drive through the track while staying within the borders of the track. It motivates the agent to stay on the track while decreasing the penalty as it gets closer to the center of the track. The penalty for not being in the center of the track is scaled by the track width which ensures the penalty correlates to the track size.

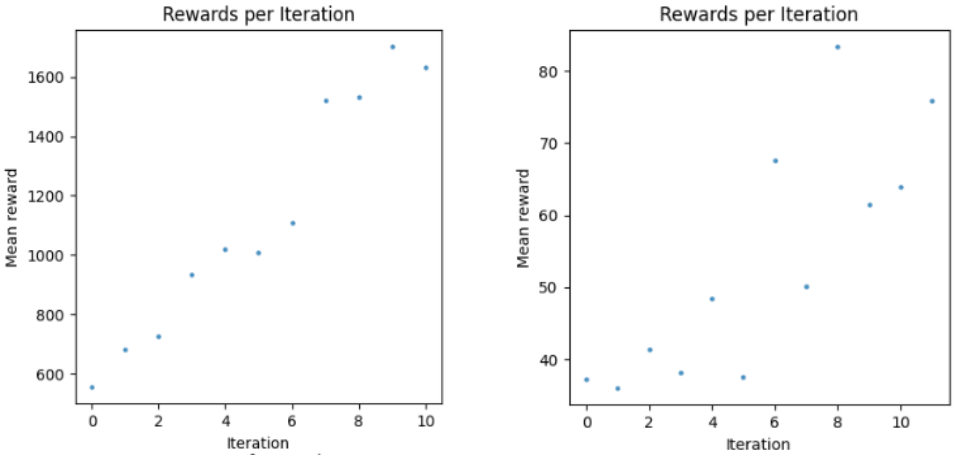
### 4.3.3 Rewards comparison for 3-layer and 5-layer CNN

As it can be seen from the graphs in 4.4, the total reward received by the agent shows a significant continuous improvement across multiple episodes. This trend proves the fact that the agent uses the reinforcement learning algorithm effectively to learn from its past experiences and to fine-tune the decision-making process over time.

In the initial stages of training, the reward stays relatively low which can be explained by the *agent exploring the environment* and understanding the track in order to create the driving strategy.

After the main exploration phase is finished and the agent acquired a decent amount of knowledge about the environment, the *exploitation phase starts*, in which the vehicle navigates through the track yielding higher rewards.

During the whole process of training, the agent keeps adjusting its driving policy using the trial and error method. Hence, the fluctuations in the graph can be observed in the points, where the agent explores new driving strategies and faces challenges. However, the overall trend remains uprising and the agent continuously improves the driving policy.




**(a)** 3-layer CNN **(b)** 5-layer CNN

**Figure 4.4** Rewards per iteration

The graphs in 4.4 demonstrate a comparison in rewards obtained by 3-layer CNN and 5-layer CNN models. The findings show the striking difference in the reward of 1600 by the 10th iteration obtained by the 3-layer CNN model and the reward of 80 acquired by the 5-layer CNN model by the same iteration. One of the main factors contributing to the performance of the models is the complexity of neural network architecture. Deeper neural networks require more training and hyperparameters fine-tuning, so it is possible that the 5-layer model could not achieve a higher reward within the same timeframe as the 3-layer one achieved. Moreover, the deep neural networks are more prone to overfitting, which could additionally contribute to the gap in the performance between the models.

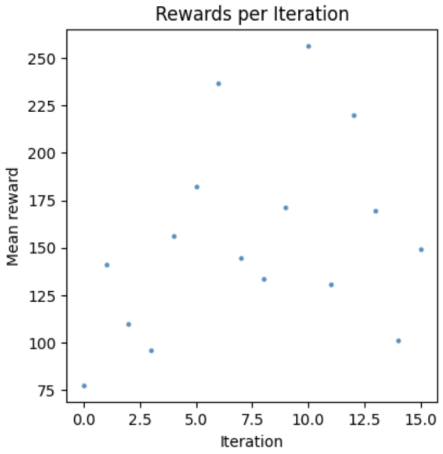
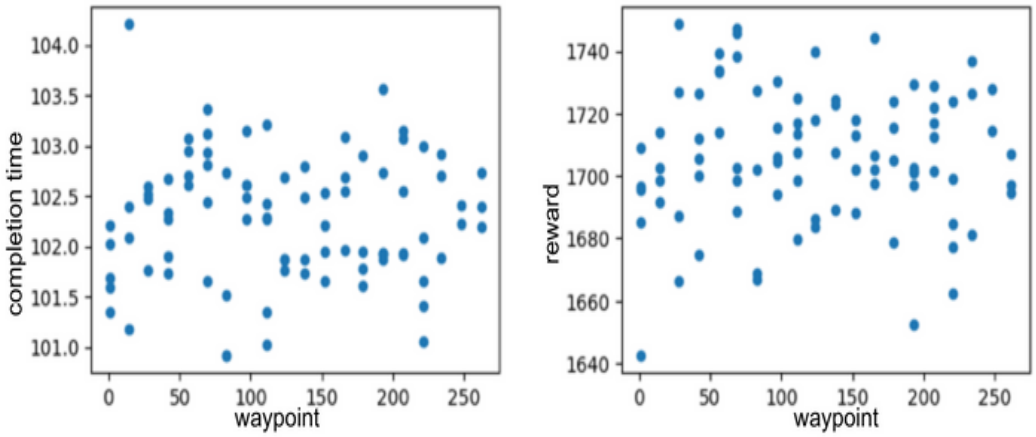



**Figure 4.5** Rewards per iteration for a 5-layer CNN model trained for 10 more iterations

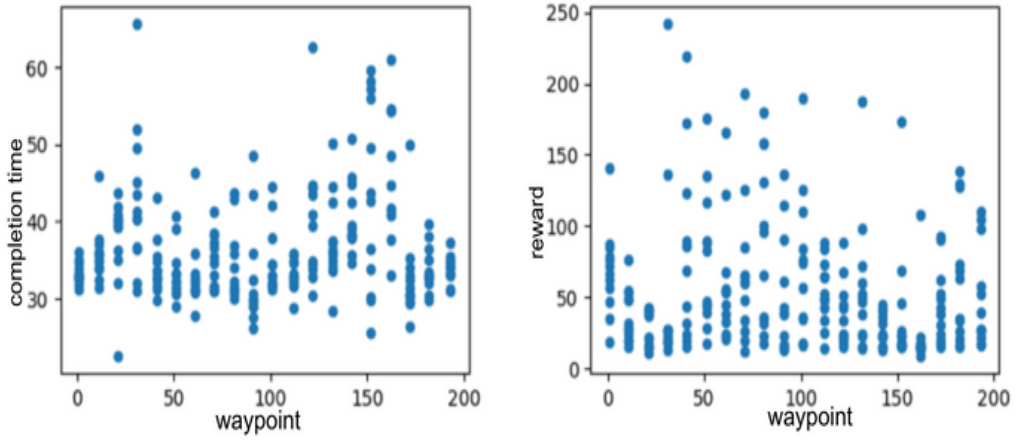
In order to comprehend the 5-layer model extensively, I incremented the initial 5-layer CNN model (the graph of it is 4.4b) and trained it for 15 more iterations to see its behavior. As a result, it is visible on graph 4.5 that its performance has not become equal to the performance of the 3-layer CNN model (on graph 4.4a) even after 15 additional iterations. It achieved the reward of about 260 by the 25th iteration of overall training in comparison to the reward of 1600 obtained by the 3-layer CNN by the 10th iteration. Moreover, the performance started declining after the 12th iteration, which could be due to overfitting and meant that the training had to be

stopped at that point.

### 4.3.4 Relationship between lap completion time and reward




**(a)** 3-layer CNN




**(b)** 5-layer CNN

**Figure 4.6** Completion time and reward obtained

In the graphs above 4.6, the **relationship between the lap completion time and reward starting at different waypoints** obtained can be observed.

For the 3-layer CNN model (on graph 4.6a), the lower lap time corresponds to the higher reward, which means that the model effectively learned to drive through the track. The model's decision-making process is correctly aimed to minimize the lap time, which led to the correct correlation between the reward and lap time.

However, for the 5-layer CNN model (on graph 4.6b), some differences can be found. In some cases, the model manages to correlate the lower lap time to the higher reward, but in other cases, it does not do so. It has some difficulty in optimizing the strategy in order to minimize the lap completion time while maximizing the obtained reward. It could be due to various factors, such as the complexity of the NN architecture, not optimal hyperparameters, or insufficient training

time and data.

## 4.4 Action space

**Action space** is a set of all valid actions that can be taken by the agent. The AWS DeepRacer can be trained in either discrete or continuous action space.

- **Discrete action space** represents a finite set of agent's actions. Therefore, the vehicle's neural network must choose direction and speed according to the inputs of its cameras and LiDAR sensor. The options set is limited. For example, the vehicle driving close to a turn might choose to accelerate, brake, turn left, right, or continue straight. The options, enumerated 0-9, are combinations of the steering angle and speed.
- **Continuous action space** is an infinite set of possible actions. The neural network chooses speed and direction, as it does in discrete action space, but there is an unlimited range of options. The speed can range from 0.75 m/s to 4 m/s and the steering angle differs from -20 to 20 degrees.

| Action number | Steering | Speed |
|---|---|---|
| 0 | -30 degrees | 0.4 m/s |
| 1 | -30 degrees | 0.8 m/s |
| 2 | -15 degrees | 0.4 m/s |
| 3 | -15 degrees | 0.8 m/s |
| 4 | 0 degrees | 0.4 m/s |
| 5 | 0 degrees | 0.8 m/s |
| 6 | 15 degrees | 0.4 m/s |
| 7 | 15 degrees | 0.8 m/s |
| 8 | 30 degrees | 0.4 m/s |
| 9 | 30 degrees | 0.8 m/s |

**Table 4.1** Discrete action space [5]

### 4.4.1 Action space for 3-layer CNN model

```
{
    "action_space": [
        {
            "steering_angle": -30,
            "speed": 0.6
        },
        {
            "steering_angle": -15,
            "speed": 0.6
        },
        {
            "steering_angle": 0,
            "speed": 0.6
        },
        {
            "steering_angle": 15,
            "speed": 0.6
        },
        {
            "steering_angle": 30,
            "speed": 0.6
        }
    ]
}
```

**Code listing 4.3** Model's action space for 3-layer CNN

The action space chosen for the training with 3-layer CNN was the discrete one with *5 distinct actions*, each differing in steering angle and speed. It is quite straightforward and simple for the model to understand, which can facilitate the training of the RL model. It includes a range of steering angles, allowing the model to gradually adjust its steering behavior based on the surrounding environment. The *speed has been chosen to be constant* across all actions in order for the model to focus on the steering adjustments without adding the complexity of speed control. The discrete action space promotes the exploration of different steering angles while ensuring the exploitation of known effective actions for the particular scenario.

### 4.4.2 Action space ofr 5-layer CNN model

```
{
    "action_space": [
        {
            "steering_angle": -30,
            "speed": 0.5
        },
        {
            "steering_angle": -15,
            "speed": 0.5
        },
        {
            "steering_angle": 0,
            "speed": 0.5
        },
        {
            "steering_angle": 15,
            "speed": 0.5
        },
        {
            "steering_angle": 30,
            "speed": 0.5
        },
        {
            "steering_angle": -30,
            "speed": 0.8
        },
        {
            "steering_angle": -15,
            "speed": 0.8
        },
        {
            "steering_angle": 0,
            "speed": 0.8
        },
        {
            "steering_angle": 15,
            "speed": 0.8
        },
        {
            "steering_angle": 30,
            "speed": 0.8
        }
    ]
}
```

**Code listing 4.4** Model's action space for 5-layer CNN

The action space chosen for the 5-layer CNN is more complex, consisting of *10 discrete actions*. The steering angles stayed the same in order to allow the vehicle to explore and learn different steering tactics effectively. However, the speed settings have been changed. There are 2 different options: *0.5 and 0.8*, that allow the model to learn the speed control along with the steering angle, enabling it to adapt to different track layouts and conditions.

## 4.5 Hyperparameters

**Hyperparameters** can help a model to have a more effective training process. For example, in order for it to have a good mix of exploration and exploitation, multiple variables have to be tuned, such as the number of episodes, the learning rate, and entropy. Moreover, to speed up the training process, other variables have to be taken into account. They include batch size, number of epochs, and discount factor. A detailed description of hyperparameters can be found in Appendix A.

### 4.5.1 Hyperparameters for 3-layer CNN model

The hyperparameters chosen for the 3-layer CNN ensure stability during training and balance between exploration and exploitation. For example, the beta entropy 0.01 is a relatively low value reflecting a medium emphasis on exploration, while balancing it with a certain level of exploitation. The discount factor of 0.99 makes the model consider the future rewards which helps in the long-term planning.

```
{
    "batch_size": 64,
    "beta_entropy": 0.01,
    "discount_factor": 0.99,
    "e_greedy_value": 0.05,
    "epsilon_steps": 10000,
    "exploration_type": "categorical",
    "loss_type": "huber",
    "lr": 0.0003,
    "num_episodes_between_training": 20,
    "num_epochs": 5,
    "stack_size": 1,
    "term_cond_avg_score": 350.0,
    "term_cond_max_episodes": 1000,
    "sac_alpha": 0.2
}
```

**Code listing 4.5** Hyperparameters for 3-layer CNN

### 4.5.2 Hyperparameters for 5-layer CNN model

Some of the hyperparameters have been adjusted for the 5-layer CNN. The batch size has been increased to 128, which should lead to faster training but also requires more memory. Next, the beta entropy has been increased from 0.01 to 0.05 indicating a higher emphasis on the exploration. A slight change has been made to the discount factor increasing it to 0.999 and resulting in even higher emphasis on future rewards and long-term planning. Moreover, the learning rate has been changed to 0.0005 meaning the model might converge faster but with less stability during the training.

```
{
    "batch_size": 128,
    "beta_entropy": 0.05,
    "discount_factor": 0.999,
    "e_greedy_value": 0.05,
    "epsilon_steps": 10000,
    "exploration_type": "categorical",
    "loss_type": "huber",
    "lr": 0.0005,
    "num_episodes_between_training": 20,
    "num_epochs": 5,
    "stack_size": 1,
    "term_cond_avg_score": 350.0,
    "term_cond_max_episodes": 1000,
    "sac_alpha": 0.2
}
```

**Code listing 4.6** Hyperparameters for 5-layer CNN

## 4.6 Domain randomization

**Domain randomization** is a technique used to train models that introduces variability into the training environment in order to make the model more robust and easily adaptable to diverse real-world scenarios.

In training the AWS DeepRacer, domain randomization plays a crucial role as the model has to be ready to encounter different conditions while driving on the track. Exposing the model to different lighting conditions helps *prevent overfitting* the model to specific conditions and encourages the model to learn features that can be applied across various settings.

Moreover, domain randomization helps *decrease the gap between simulation and reality.* While training in a simulated environment makes the model learn in a specific setting, it usually lacks the complexity of real-world scenarios.

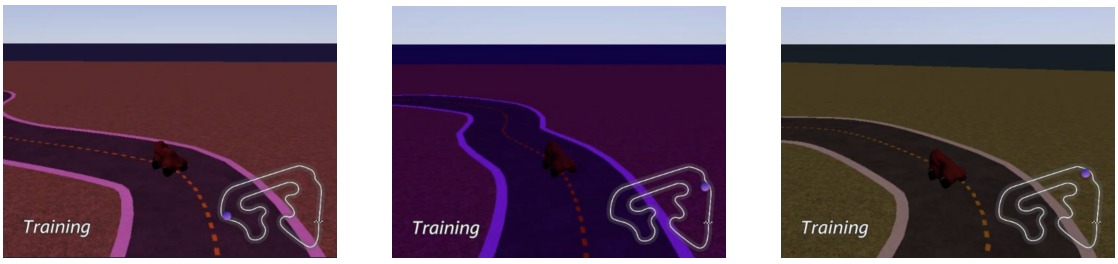



**Figure 4.7** Training with domain randomization

## 4.7 Neural network architecture

The AWS DeepRacer's neural network architecture involves a Convolutional Neural Network (CNN) to process the visual input from the vehicle's camera. The CNN architecture is designed to perform analysis of the raw pixel data from the camera and extract meaningful features, relevant to autonomous driving, such as other vehicles on the track, lane lines, and obstacles. The process involves multiple layers of operations that allow the network to learn the representations of the images from the camera.

There are **3 available neural networks** for AWS DeepRacer: 3-layer CNN, 5 layer-CNN, and 6-layer CNN. In this thesis, I will focus on the *3-layer and 5-layer networks*, as they are also the ones that are offered in the AWS Console.

### 4.7.1 3-layer CNN

**The 3-layer CNN** consists of 3 layers, where the first layer has 32 filters, an 8x8 kernel size, and a stride of 4, the second layer has 64 filters, a 4x4 kernel size, and a stride of 2 and the third layer has 64 filters, a 3x3 kernel size, and a stride of 1. There is no dense layer - no fully connected middleware.

*Filters* are 2D matrices that are applied to the input data and responsible for detecting a particular feature in the input. For example, the layer having 32 filters will identify 32 different patterns or features in the input. *Kernel size* is the dimension of the filter or the size of the grid of values. During the convolution operation, the filter will slide over the input data, and perform multiplication and summation of values in order to produce a single value in the output. *Stride value* is how much the filter slides over the input after each convolution operation.

The activation function used is **Rectified Linear Unit (ReLU)**, which is defined as

$$f(x) = max(0, x)$$

It means that it returns x for the positive input and 0 for any negative input.

### 4.7.2 5-layer CNN

**The 5-layer CNN** is more complicated and takes more time to train a model, but it can extract more complex features out of the input in comparison to the 3-layer CNN. There are 4 convolutional layers: the first one consists of 32 filters with a kernel size of 5x5 and a stride of 2, the second layer is similar with the only difference in the size of the stride which is 1 in this case, and the third and fourth layer have 64 filters with the kernel size of 3x3 and a strides of 2 and 1 respectively. After the convolutional layers, there is a *dense (fully connected) layer* with 64 neurons. The dense layer processes the flattened output from the previous convolutional layer and performs classification tasks. The activation function used in the 5-layer CNN is the **hyperbolic tangent activation function**. Mathematically it is defined as

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The tanh function outputs values in the range [-1,1] which results in the output being centered around 0.

The 3-layer networks, compared to the deeper 5-layer networks, have a simpler shallower architecture, which means faster training times and lower computational power required for the training, enabling developers to experiment more. Having fewer layers, the 3-layer CNN decreases the risk of overfitting and leaves more space for the models to be robust and generalize on the unknown tracks efficiently.

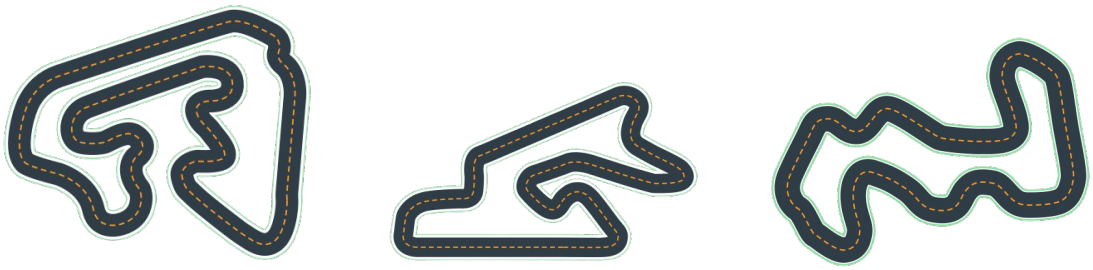
# Chapter 5
# Evaluation

*In this chapter, I discuss the evaluation process of the model, both virtual and in real world. I describe the differences in performance of different models in different conditions.*

## 5.1 Evaluation information

The evaluation in the simulation environment Deepracer-for-Cloud of the autonomous driving model trained on AWS DeepRacer was conducted across three distinct tracks, differing from those utilized during the training phase. This approach was essential to assess the generalization capability of the model and its adaptability to novel environments. The selected tracks varied in complexity, encompassing diverse features such as sharp turns, straight segments, and varying road conditions.

### 5.1.1 Evaluation tracks selection



**(a)** Red Star Pro **(b)** Dubai Pro **(c)** Singapore

**Figure 5.1** Evaluation track layouts

The tracks chosen were **Red Star Pro, Dubai Pro** and **Singapore**. The first track is a challenging circuit with a combination of light turns and sharp corners which demands precise navigation and agile maneuvering. Throughout the course, the agent has to maintain optimal speed and careful braking. The second track Dubai Pro is a blend of long straights and sweeping curves. The long straightaways allow the vehicle to reach top speeds before encountering sharp turns, in which the vehicle has to adjust speed to ensure stability and control. The last track is called Singapore and its layout is challenging due to the tight corners throughout the whole track.

The selection of the Red Star Pro, Dubai Pro, and Singapore tracks for the evaluation of the autonomous driving model was deliberate and strategic. Each track offers a unique set of challenges, ranging from tight technical sections to high-speed straights. By evaluating the model across these diverse tracks, we aimed to assess its generalization capability and adaptability to varied racing scenarios.

### 5.1.2 Evaluation in simulation environment

The evaluation in simulation environment consisted of 3 laps in the time trial race type. The videos are present in attachments.

| Model | No. of training tracks | Completion time | Resets |
|---|---|---|---|
| 3-layer CNN, initial model | 6 | 5:08 | 2 |
| 3-layer CNN, with domain randomization | 6 | 5:05 | 0 |
| 3-layer CNN | 7 | 5:06 | 0 |
| 3-layer CNN | 8 | 5:05 | 0 |
| 3-layer CNN | 9 | 5:06 | 0 |
| 5-layer CNN | 6 | 4:51 | 4 |
| 5-layer CNN, improved | 9 | 4:52 | 4 |

**Table 5.1** Evaluation on Red Star Pro track

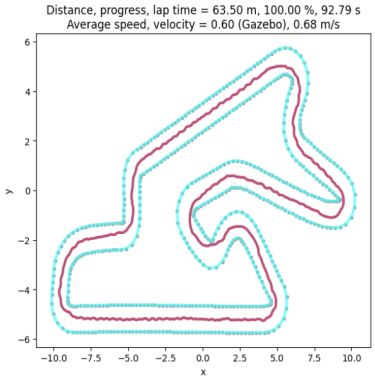
| Model | No. of training tracks | Completion time | Resets |
|---|---|---|---|
| 3-layer CNN, initial model | 6 | 4:49 | 1 |
| 3-layer CNN, with domain randomization | 6 | 4:49 | 0 |
| 3-layer CNN | 7 | 4:47 | 0 |
| 3-layer CNN | 8 | 4:49 | 0 |
| 3-layer CNN | 9 | 4:49 | 0 |
| 5-layer CNN | 6 | 4:39 | 3 |
| 5-layer CNN, improved | 9 | 4:54 | 7 |

**Table 5.2** Evaluation on Dubai Pro track

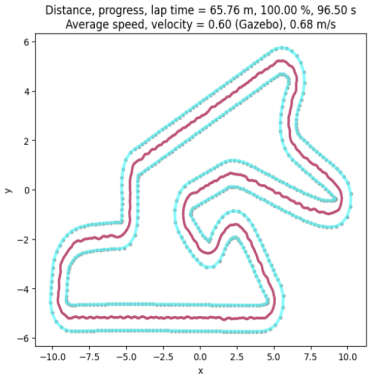
| Model | No. of training tracks | Completion time | Resets |
|---|---|---|---|
| 3-layer CNN, initial model | 6 | 4:49 | 0 |
| 3-layer CNN, with domain randomization | 6 | 4:54 | 0 |
| 3-layer CNN | 7 | 4:33 | 0 |
| 3-layer CNN | 8 | 4:37 | 0 |
| 3-layer CNN | 9 | 4:44 | 0 |
| 5-layer CNN | 6 | 5:46 | 15 |
| 5-layer CNN, improved | 9 | 4:16 | 2 |

**Table 5.3** Evaluation on Singapore track

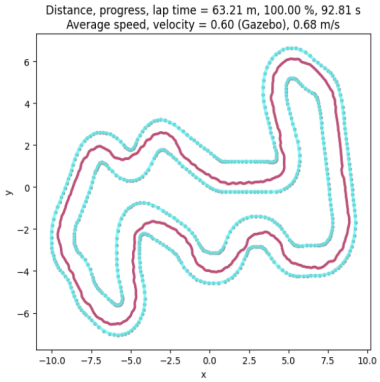
### 5.1.3 Comparison of 3-layer CNN models



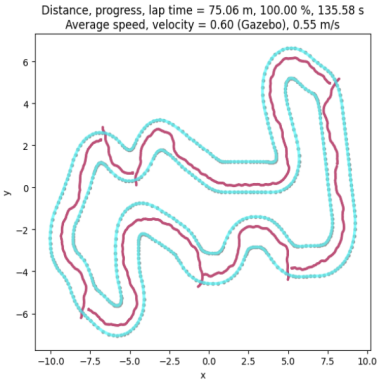

**(a)** Dubai Pro (smaller dataset)



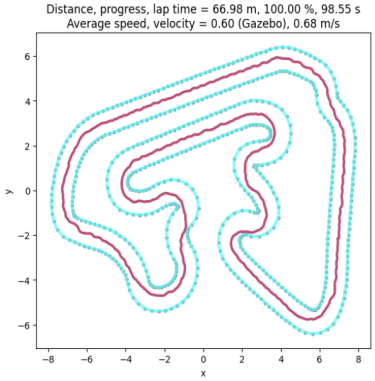

**(b)** Dubai Pro (bigger dataset)



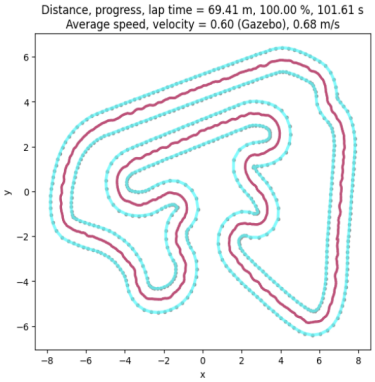

**(c)** Singapore (smaller dataset)




**(d)** Singapore (bigger dataset)




**(e)** Red Star Pro (smaller dataset)




**(f)** Red Star Pro (bigger dataset)

**Figure 5.2** Evaluation tracks of 3-layer CNN. *Smaller dataset* refers to 3 training tracks. *Bigger dataset* refers to 6 training tracks.

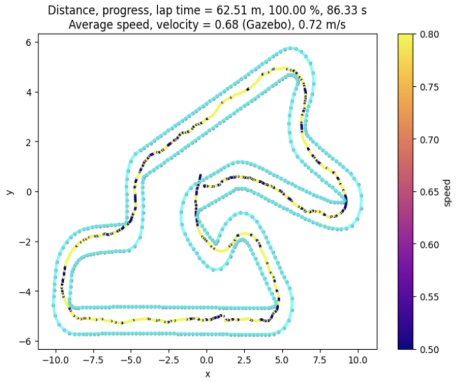
Above each graph in 5.2 there is a distance driven by the vehicle on that particular track, the progress (percentage of track completed), and the lap time (time that the vehicle took to complete the lap). Moreover, there is an average speed and velocity values. The average speed is always 0.6, which was defined as the only speed option in the discrete action space. Gazebo is an open-source robotic simulation engine, which is supported by a Robomaker environment and

provides a realistic environment for model testing [41]. The velocity is an instantaneous measure and it depends on multiple factors, such as direction changes and acceleration.

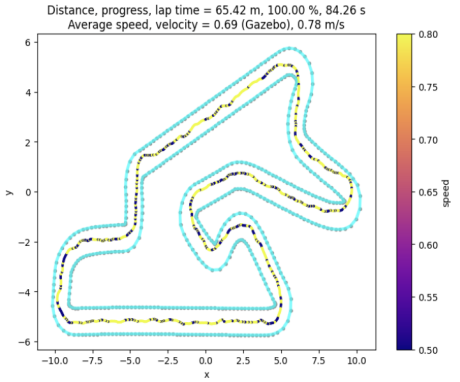
I compared the performance of 2 models: one that has been trained on *3 different tracks (the left column)* and another one that has been trained on *6 different tracks (the right column)*. For the Dubai Pro and Red Star Pro tracks, both models demonstrated good performance and improvements after having been trained on 6 tracks. It is noticeable that the vehicle's driving on the sharp turns has become significantly more precise and clean.

However, the model's performance on the Red Star Pro track has not followed the same pattern as it has on the 2 previous tracks. The model trained on the larger dataset of 6 tracks does not outperform its counterpart trained on 3 tracks. The model exhibits diminished performance due to its over-fitting on that specific track. It occurs when a model learns the details of the training data up to the point when it cannot generalize on the new unseen tracks. In this case, the model's extensive exposure to the diverse array of tracks leads to over-fitting. On the other hand, the model trained on the smaller dataset demonstrates a better, more robust performance, which indicates a better balance between learning from diverse environments and avoiding over-training.

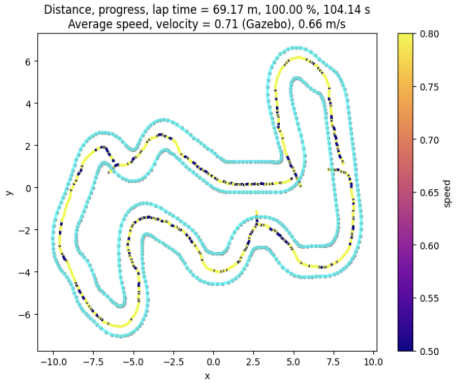
### 5.1.4 Comparison with 5-layer CNN models



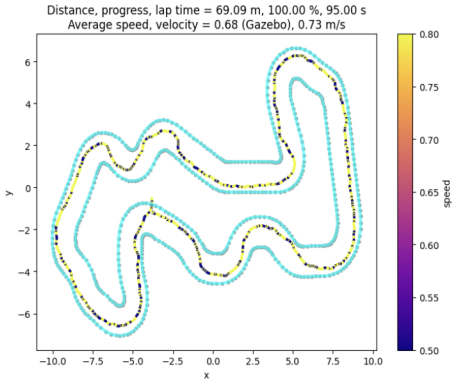

**(a)** Dubai Pro (smaller dataset)



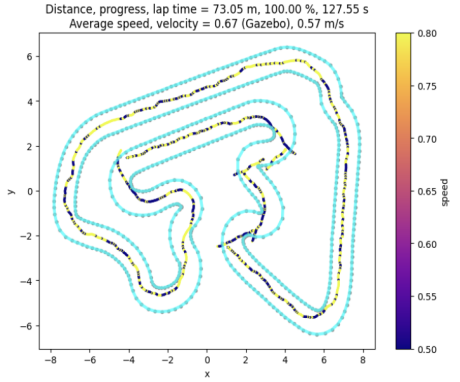

**(b)** Dubai Pro (bigger dataset)



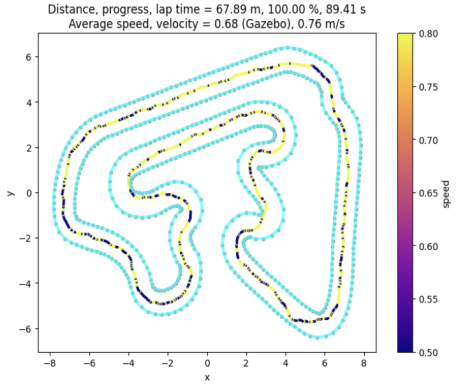

**(c)** Singapore (smaller dataset)




**(d)** Singapore (bigger dataset)



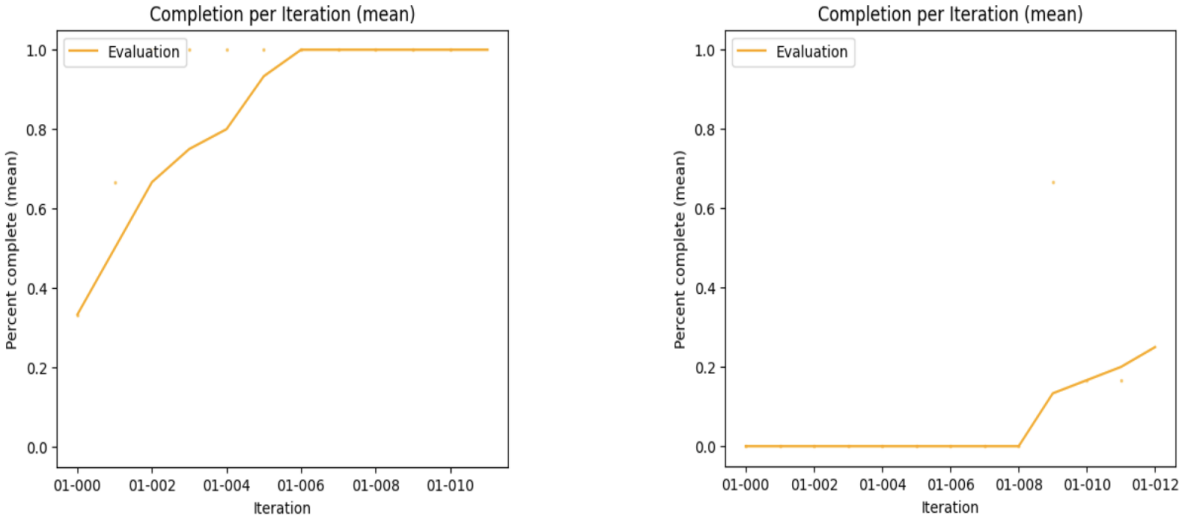

**(e)** Red Star Pro (smaller dataset)




**(f)** Red Star Pro (bigger dataset)

**Figure 5.3** Evaluation tracks of 5-layer CNN. *Smaller dataset* refers to 3 training tracks. *Bigger dataset* refers to 6 training tracks.

In comparison with the 3-layer models trained in discrete action space with constant speed, the 5-layer CNN models have been trained with various speed options in the discrete action space as well. The performance on all 3 tracks has improved after being trained on the bigger dataset of tracks for a longer time.




**(a)** 3-layer CNN

**(b)** 5-layer CNN

**Figure 5.4** Completion per iteration

The **completion per iteration graphs** 5.4 demonstrate models' abilities to effectively navigate the track over time.

The 3-layer CNN graph illustrated a rapid progression in completing the lap from the initial training episodes. The mean completion steadily increases over time, which demonstrates that the model is quickly learning to adapt to the track. After a short period of time, it achieves a 100% completion, meaning it mastered the track and could subsequently complete the lap.

On the other hand, the 5-layer CNN model shows a slower increase in the lap completion percentage. The slower delayed progress corresponds to the more complex, deeper architecture of the 5-layer CNN, which requires more time and training data to learn the track effectively.

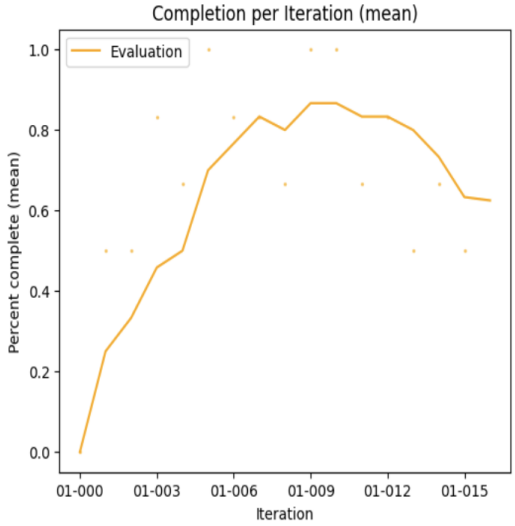



**Figure 5.5** Completion per iteration for the 5-layer CNN model trained for 10 more iterations

For the more extensive analysis of a 5-layer CNN model, I trained it for 10 additional iterations and its behavior has improved. On the 5th iteration in the additional training, it managed to

complete a full lap for the first time followed by 2 more full lap completions on the 9th and 10th iterations (see 5.5). On average, the maximum completion percentage was about 85%. However, after the 12th iteration, the completion percentage started declining (corresponds to the decline on 4.5), which could possibly happen due to the model's overfitting.

## 5.2 Transfer of the model to the vehicle

After uploading the trained model to the S3 bucket in the local Minio storage, it was available to see in the Minio UI. Having downloaded the zip archive of the model, some adjustments to the structure had to be made before uploading the model to the physical vehicle.

### 5.2.1 Model structure required for AWS DeepRacer vehicle

```
rl-deepracer-model/
├── worker_0.multi_agent_graph.main_level.main_level.agent_0.csv
├── model_metadata.json
├── worker_0.multi_agent_graph_0.json
└── agent/
    └── model.pb
```

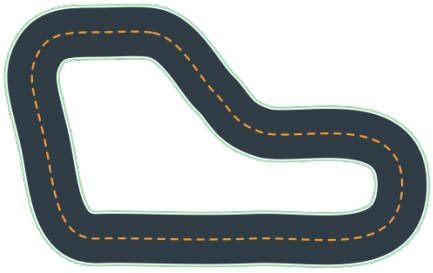
**Figure 5.6** The directory structure required for AWS DeepRacer vehicle

The files inside of the directory:

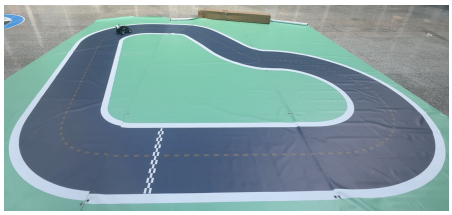
- **model.pb**: The file representing the trained model itself, stored in the Protocol Buffers Format. It encapsulates the neural network architecture, weights, and other parameters learned during the training process. It is the core component for the physical vehicle to make predictions and decisions.
- **model_metadata.json**: This file includes metadata about the trained model, such as the action space details, the training algorithm, the neural network, and the sensors used in the training.
- **worker_0.multi_agent_graph.main_level.main_level.agent_0.csv**: The CSV file contains data about the training process and performance metrics of an agent in the RL environment. Each row corresponds to one training episode, with columns describing various aspects of the training, including episode length, total reward, and total steps.
- **worker_0.multi_agent_graph_0.json**: This file encompasses the configuration for the RL training. It includes specifications, such as the number of training iterations and evaluation steps. Additionally, it presents the parameters related to the exploration strategies, neural network architecture, and optimization algorithms.

## 5.3 Evaluation on a physical vehicle

After the models have been transferred to the physical car, I started the evaluation process in the real world.



**(a)** A to Z Speedway track



**(b)** The printed track

During the evaluation phase of the models, I conducted comprehensive tests on the **A to Z Speedway track** in various environments in order to assess the model's robustness and adaptability to real-world scenarios. The evaluation consisted of multiple conditions, each designed to challenge the vehicle and test its abilities. One stereo camera has been used as a sensor on AWS Deepracer.

### 5.3.1 The initial evaluation

I started with the evaluation of the **first 3-layer CNN model** in the ideal lighting conditions on the half of the printed track and it could complete the half lap in 66.67% out of all cases. On the other hand, in the darker lighting conditions, it could complete the half-lap in 30% of the tests conducted.

### 5.3.2 Proposed improvements and evaluation

After analyzing the results of the first evaluation, I proposed the improvements that could be done to the model in order to enhance its performance on the track - **domain randomization and using a deeper 5-layer CNN.**

The approach that I used with the improved models was starting the car from 3 different waypoints, as can be seen in 5.8 and, additionally, in the reverse direction. This method challenges the vehicle and its adaptability to various driving scenarios.

**(a)** First waypoint

**(b)** First waypoint in reverse direction

**(c)** Second waypoint

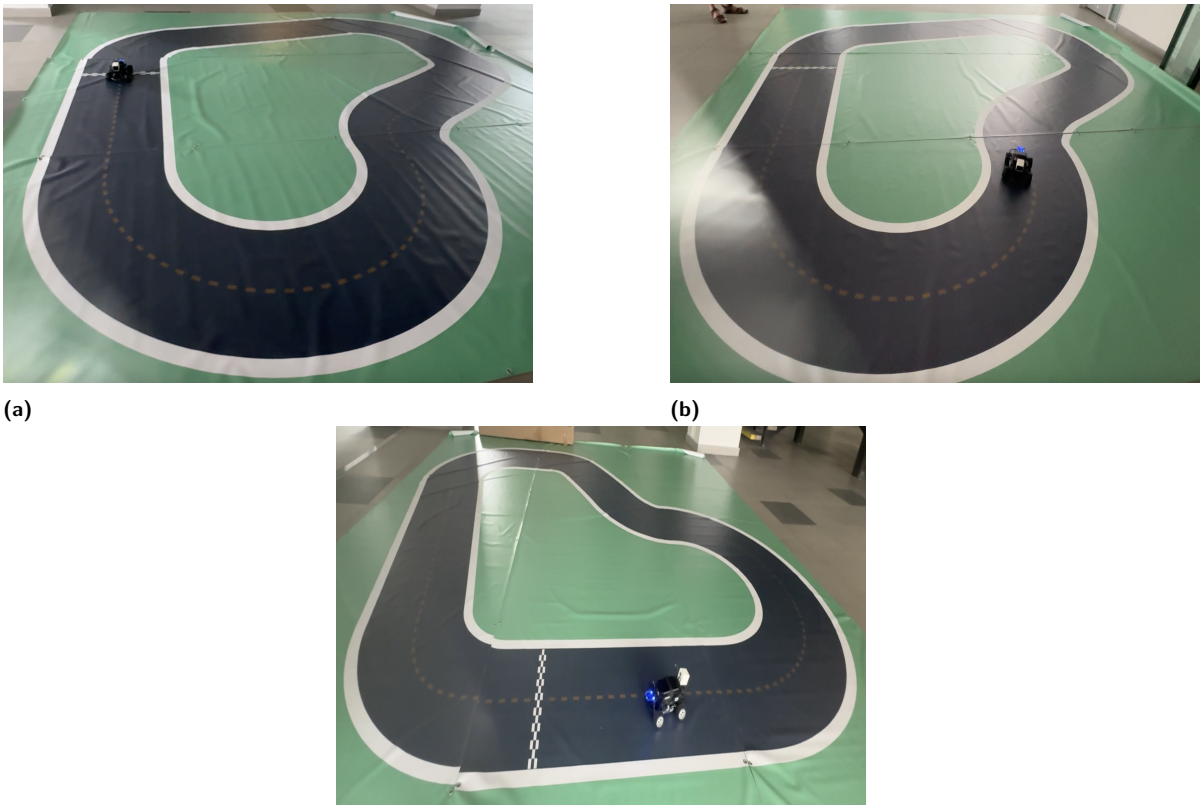
**(d)** Second waypoint in reverse direction

**(e)** Third waypoint

**(f)** Third waypoint in reverse direction

**Figure 5.8** Evaluation starting positions

The 3-layer CNN model, trained with domain randomization in various lighting conditions, completed the full lap in 88.89% in the ideal light, and in the worse lighting conditions (as on 5.9) it managed to complete the half lap in 100% of cases. The domain randomization during training proved to be helpful for the vehicle to navigate on the unknown track in the real-world scenario.



(a) (b) (c)

**Figure 5.9** Darker lighting conditions

The further improvement I came up with was to train the model on 3 more tracks with domain randomization and test the new models on the track. As a result, the model trained on 1 more additional track performed worse than the baseline model in the ideal lighting conditions, completing the full lap in 50% of drives. The model trained on 2 more additional tracks had the same performance as the one trained on 1 more additional track, resulting in 50% of success. However, the model trained on 3 more additional tracks, achieved 100% of lap completion in an ideal lighting environment. Regarding the model's performance in the darker light environment, it could complete the half lap in only 33.33% and the main reason for such poor performance is the presence of light reflections on the track (visible on 5.9b), which completely confused the vehicle and made the track unrecognizable from the car's camera.

| Model | No. of training tracks | Ideal light | | Dark light | |
|---|---|---|---|---|---|
| | | Half lap | Full lap | Half lap | Full lap |
| 3-layer CNN, initial model | 6 | 66.67% | 0% | 30% | 0% |
| 3-layer CNN, with domain randomization | 6 | 88.89% | 88.89% | 100% | 0% |
| 3-layer CNN | 7 | 50% | 50% | N/A | N/A |
| 3-layer CNN | 8 | 50% | 50% | N/A | N/A |
| 3-layer CNN | 9 | 100% | 100% | 33.33% | 33.33% |
| 5-layer CNN | 6 | 41.67% | 33% | 0% | 0% |

**Table 5.4** Evaluation results

The 5-layer CNN model in the ideal lighting conditions completed the full lap in 33.33% of the tests and half lap in 41.67% of the evaluations. In the darker lighting conditions, it did not manage to finish either a full lap or a half lap due to multiple reasons but mainly because of the light reflections on the track. The reason for the poorer performance of the 5-layer CNN model could be the model's complexity as the NN could be too complex for the task which led to the computational overhead and potential overfitting. While the 5-layer CNN has a bigger capacity to learn more hidden features, it might also be prone to overfitting, resulting in the failure of generalization. Compared to the 3-layer CNNs, 5-layer CNNs have to be trained on the larger dataset for optimal performance, and failing to provide a bigger training set for it could have impacted the model's ability to generalize on unknown tracks.

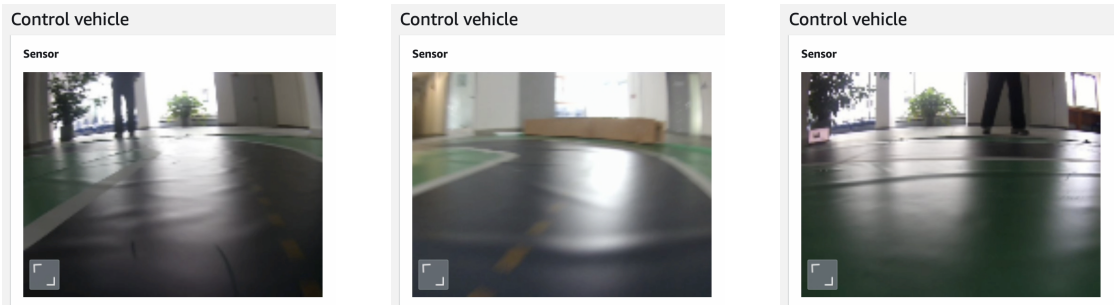



**Figure 5.10** Light reflections on the track

Furthermore, 3 evaluation runs of the 3-layer CNN model in ideal lighting conditions were conducted with the low battery and the lap was completed in 100% of cases.

# Chapter 6
# Conclusion

This thesis has been focused on bridging the gap in transferring the model from the simulation environment to real-world conditions in autonomous driving using AWS DeepRacer.

I have conducted a thorough review of existing literature on machine learning techniques and approaches for autonomously driving vehicles. It involved a wide array of articles, research papers, and past bachelor theses. By analyzing the methodologies applied and the issues encountered, I have gained valuable lessons and insights that have been reflected in the current thesis.

Moreover, I have trained 3-layer CNN models in the simulation environment on 6 tracks and those models have been tested in the real-world environment on the printed "A to Z Speedway" track. The models performed well in the ideal lighting conditions but failed to do so in the worse lighting with light reflections being on the track.

After the models' performance has been analyzed, I have come up with the potential improvements - using domain randomization in order to mimic the real-world conditions in the training environment, and using a deeper 5-layer neural network in the training.

The improved models have been tested again on the physical vehicle and the whole testing process has been recorded on video (videos can be found in the attached media). The domain randomization proved to be useful and resulted in the vehicle completing the full lap. However, the 5-layer CNN models have not demonstrated a great improvement which could be due to numerous factors, such as the network's complexity, overfitting, or insufficient training data.

One area for potential future research could be investigation and training using deeper neural networks that go even beyond the 5-layer CNNs used in this thesis. By using the more complex neural network with deeper architecture, it is possible to capture more detailed patterns and finer structures, that could lead to further improvements in the autonomous driving field.

# Appendix A

# Appendix

| Parameter | Type | Description |
|---|---|---|
| all_wheels_on_track | Boolean | Flag to indicate if the agent is on the track |
| x | float | Agent's x-coordinate in meters |
| y | float | Agent's y-coordinate in meters |
| closest_objects | [int, int] | Zero-based indices of the two closest objects to the agent's current position of (x, y) |
| closest_waypoints | [int, int] | Indices of the two nearest waypoints |
| distance_from_center | float | Distance in meters from the track center |
| is_crashed | Boolean | Boolean flag to indicate whether the agent has crashed |
| is_left_of_center | Boolean | Flag to indicate if the agent is on the left side to the track center or not |
| is_offtrack | Boolean | Boolean flag to indicate whether the agent has gone off track |
| is_reversed | Boolean | Flag to indicate if the agent is driving clockwise (True) or counter clockwise (False) |
| heading | float | Agent's yaw in degrees |
| objects_distance | [float, ] | List of the objects' distances in meters between 0 and track_length in relation to the starting line |
| objects_heading | [float, ] | List of the objects' headings in degrees between -180 and 180 |
| objects_left_of_center | [Boolean, ] | List of Boolean flags indicating whether elements' objects are left of the center (True) or not (False) |
| objects_location | [(float, float),] | List of object locations [(x,y), ...] |
| objects_speed | [float, ] | List of the objects' speeds in meters per second |
| progress | float | Percentage of track completed |
| speed | float | Agent's speed in meters per second (m/s) |
| steering_angle | float | Agent's steering angle in degrees |
| steps | int | Number steps completed |
| track_length | float | Track length in meters |
| track_width | float | Width of the track |
| waypoints | [(float, float), ] | List of (x,y) as milestones along the track center |

**Table A.1** Input parameters for reward function [5]

| Parameter | Description |
|---|---|
| Gradient Descent Batch Size | A subset of an experience buffer composed of images captured by the camera mounted on the AWS DeepRacer vehicle and actions taken by the vehicle. |
| Number of Epochs | The number of passes through the training data to update the neural network weights during gradient descent. |
| Learning Rate | Controls how much a gradient-descent (or ascent) update contributes to the network weights. |
| Entropy | Added uncertainty that helps the AWS DeepRacer vehicle explore the action space more broadly. |
| Discount Factor | A factor determining the importance of future rewards. A factor of 0 means the current state is independent of future steps, whereas a factor of 1 means that contributions from all future steps are included. |
| Loss Type | The type of objective function used to update the network weights. |
| Number of Experience Episodes Between Each Policy-Updating Iteration | The size of the experience buffer used to draw training data from for learning policy network weights. |

**Table A.2** Hyperparameters for AWS DeepRacer Training[5]

# Bibliography


1. BARABÁS, Istvan; TODORUŢ, Adrian; CORDOŞ, N; MOLEA, Andreia. Current challenges in autonomous driving. In: *IOP conference series: materials science and engineering*. IOP Publishing, 2017, vol. 252, p. 012096. No. 1. Available from DOI: 10.1088/1757-899X/252/1/012096.
2. YURTSEVER, Ekim; LAMBERT, Jacob; CARBALLO, Alexander; TAKEDA, Kazuya. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*. 2020, vol. 8, pp. 58443–58469. Available also from: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9046805.
3. AMAZON WEB SERVICES. *AWS DeepRacer - Racing Simulator Software* [online]. [N.d.]. [visited on 2024-05-05]. Available from: https://aws.amazon.com/deepracer/.
4. TIAN, Allen; JOHN, Eddy Guerra; YANG, Kecheng. Poster: Unraveling Reward Functions for Head-to-Head Autonomous Racing in AWS DeepRacer. In: *Proceedings of the Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. 2023, pp. 592–594. Available from DOI: 10.1145/3565287.3617987.
5. AMAZON WEB SERVICES. *AWS DeepRacer - Developer Guide* [online]. [N.d.]. [visited on 2024-05-05]. Available from: https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-get-started.html.
6. LI, Yuxi. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*. 2017. Available from DOI: 10.48550/arXiv.1701.07274.
7. HAZRATIFARD, Mehdi; GEBALI, Fayez; MAMUN, Mohammad. Using machine learning for dynamic authentication in telehealth: A tutorial. *Sensors*. 2022, vol. 22, no. 19, p. 7655.
8. BASL, John; BEHRENDS, Jeff. Why everyone has it wrong about the ethics of autonomous vehicles. In: *Frontiers of engineering: Reports on leading-edge engineering from the 2019 symposium*. National Academies Press, 2020, pp. 75–82. Available from DOI: 10.17226/25620.
9. NYHOLM, Sven; SMIDS, Jilles. The ethics of accident-algorithms for self-driving cars: An applied trolley problem? *Ethical theory and moral practice*. 2016, vol. 19, no. 5, pp. 1275–1289.
10. WIKIPEDIA CONTRIBUTORS. *Trolley problem — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Trolley_problem&oldid=1218726891. [Online; accessed 7-May-2024].

11. GEISSLINGER, Maximilian; POSZLER, Franziska; BETZ, Johannes; LÜTGE, Christoph; LIENKAMP, Markus. Autonomous driving ethics: From trolley problem to ethics of risk. *Philosophy & Technology*. 2021, vol. 34, no. 4, pp. 1033–1055.
12. HIMMELREICH, Johannes. Never mind the trolley: The ethics of autonomous vehicles in mundane situations. *Ethical Theory and Moral Practice*. 2018, vol. 21, no. 3, pp. 669–684.
13. WU, Stephen S. Autonomous vehicles, trolley problems, and the law. *Ethics and Information Technology*. 2020, vol. 22, no. 1, pp. 1–13.
14. WANG, Jun; ZHANG, Li; HUANG, Yanjun; ZHAO, Jian; BELLA, Francesco. Safety of autonomous vehicles. *Journal of advanced transportation*. 2020, vol. 2020, pp. 1–13.
15. AMAZON WEB SERVICES. *Robot Simulator - AWS Robomaker* [online]. [N.d.]. [visited on 2024-05-05]. Available from: `https://aws.amazon.com/robomaker/`.
16. AMAZON WEB SERVICES. *Amazon Sagemaker - AWS* [online]. [N.d.]. [visited on 2024-05-05]. Available from: `https://aws.amazon.com/pm/sagemaker/?trk=3ea5c9d1-0497-4ab3-92e6-c583f43ac2f9&sc_channel=ps&ef_id=CjwKCAjw3NyxBhBmEiwAyofDYXKLOTU3t-u7zKfsaCNivb3QxDiuPgN27YpGmUTEbCHd_NSLzdC-HhoCSyMQAvD_BwE:G:s&s_kwcid=AL!4422!3!645186192649!e!!g!!aws%20sagemaker!19571721771!146073031580&gclid=CjwKCAjw3NyxBhBmEiwAyofDYXKLOTU3t-u7zKfsaCNivb3QxDiuPgN27YpGmUTEbCHd_NSLzdC-HhoCSyMQAvD_BwE`.
17. AWS DEEPRACER COMMUNITY. *Deepracer-for-Cloud* [online]. [N.d.]. [visited on 2024-05-05]. Available from: `https://github.com/aws-deepracer-community/deepracer-for-cloud`.
18. DOLGOV, Dmitri; THRUN, Sebastian; MONTEMERLO, Michael; DIEBEL, James. Path planning for autonomous vehicles in unknown semi-structured environments. *The international journal of robotics research*. 2010, vol. 29, no. 5, pp. 485–501. Available from DOI: `10.1177/0278364909359210`.
19. GHIMIRE, Mukesh. A Study of Deep Reinforcement Learning in Autonomous Racing Using DeepRacer Car [online]. 2021.
20. NAVARRO, Anthony; GENC, Sahika; RANGARAJAN, Premkumar; KHALIL, Rana; GOBERVILLE, Nick; ROJAS, Johan Fanas; ASHER, Zachary. *Using Reinforcement Learning and Simulation to Develop Autonomous Vehicle Control Strategies*. 2020. Tech. rep. SAE Technical Paper. Available from DOI: `10.4271/2020-01-0737`.
21. NAIR, Unnikrishnan R; SHARMA, Sarthak; PARIHAR, Udit Singh; MENON, Midhun S; VIDAPANAKAL, Srikanth. Bridging Sim2Real Gap Using Image Gradients for the Task of End-to-End Autonomous Driving. *arXiv preprint arXiv:2205.07481*. 2022.
22. YE, Heng; LIU, Yanbo; XU, Jiahao; DU, Haikuo; SUN, Weiqi; XU, Wenchao; LI, Zhengyu; SHEN, Zanwei; LIU, Yan. Application of Human-Machine Co-driving in Multi-vehicle Formation of Deepracer Automatic Driving Platform. In: *Society of Automotive Engineers (SAE)-China Congress*. Springer, 2023, pp. 20–36.
23. SUTTON, Richard S; BARTO, Andrew G. *Reinforcement learning: An introduction*. MIT press, 2018.
24. HUANG, Honglan; HUANG, Jincai; FENG, Yanghe; ZHANG, Jiarui; LIU, Zhong; WANG, Qi; CHEN, Li. On the improvement of reinforcement active learning with the involvement of cross entropy to address one-shot learning problem. *PloS one*. 2019, vol. 14, no. 6, e0217408.
25. LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. *nature*. 2015, vol. 521, no. 7553, pp. 436–444. Available from DOI: `10.1038/nature14539`.
26. AMAZON WEB SERVICES. What is Deep Learning? [Online]. 2021. Available also from: `https://aws.amazon.com/what-is/deep-learning/#:~:text=Deep%20learning%20is%20a%20method,produce%20accurate%20insights%20and%20predictions`.

27. KIRAN, B Ravi; SOBH, Ibrahim; TALPAERT, Victor; MANNION, Patrick; AL SALLAB, Ahmad A; YOGAMANI, Senthil; PÉREZ, Patrick. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*. 2021, vol. 23, no. 6, pp. 4909–4926.

28. O'SHEA, Keiron; NASH, Ryan. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*. 2015. Available from DOI: 10.48550/arXiv.1511.08458.

29. SAZLI, Murat H. A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*. 2006, vol. 50, no. 01. Available from DOI: 10.1501/commua1-2_0000000026.

30. SCHMIDT, Robin M. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*. 2019. Available from DOI: 10.48550/arXiv.1912.05911.

31. SUTSKEVER, Ilya. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

32. YING, Xue. An overview of overfitting and its solutions. In: *Journal of physics: Conference series*. IOP Publishing, 2019, vol. 1168, p. 022022. Available from DOI: 10.1088/1742-6596/1168/2/022022.

33. EVEN-DAR, Eyal; MANNOR, Shie; MANSOUR, Yishay; MAHADEVAN, Sridhar. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of machine learning research*. 2006, vol. 7, no. 6.

34. PETRYSHYN, Bohdan; POSTUPAIEV, Serhii; BEN BARI, Soufiane; OSTREIKA, Armantas. Deep Reinforcement Learning for Autonomous Driving in Amazon Web Services DeepRacer. *Information*. 2024, vol. 15, no. 2, p. 113. Available from DOI: 10.3390/info15020113.

35. *AWS DeepRacer Community* [online]. [N.d.]. [visited on 2024-05-05]. Available from: https://deepracing.io/#about.

36. *ROS (Robot Operating System)* [online]. [N.d.]. Available also from: https://www.ros.org.

37. SERVICES, Amazon Web. *AWS DeepRacer is now open source and ready to hit the road with ROS 2* [online]. [N.d.]. Available also from: https://aws.amazon.com/blogs/opensource/aws-deepracer-is-now-open-source-and-ready-to-hit-the-road-with-ros-2/.

38. AMAZON WEB SERVICES. *ROS 2 Foxy Fitzroy: Robot Development* [online]. [N.d.]. Available also from: https://aws.amazon.com/blogs/robotics/ros2-foxy-fitzroy-robot-development/.

39. UNITY TECHNOLOGIES. *Unity Simulation Now on AWS* [online]. [N.d.]. Available also from: https://blog.unity.com/engine-platform/unity-simulation-now-on-aws.

40. JAKL, Vincent; KOSORIN, Peter; PROCHAZKA, Adam. AWS DeepRacer local training configuration. 2023. Available also from: https://apps.datalab.fit.cvut.cz/static/deepracer/deepracer_setup_whitepaper.pdf.

41. KOENIG, Nathan; HOWARD, Andrew. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*. Ieee, 2004, vol. 3, pp. 2149–2154.

# Contents of the attached media

- thesis
  - text
    - appendix.tex ..... training parameters appendix
    - bib-database.bib ..... bibliography database
    - text.tex ..... thesis text
  - images ..... images used in thesis
  - thesis.tex ..... thesis source code
  - tskheyel-assignment.pdf ..... thesis assignment
  - thesis.cls ..... thesis source code support file
- models
  - .tar.gz files ..... model files
- evaluation
  - real_world_evaluation ..... real world evaluation videos
  - simulation_environment_evaluation ..... simulation environment evaluation videos