HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik
Ausbildungsschwerpunkt Machine Learning

DIPLOMARBEIT

Racing with Reinforcement Learning

Ausgeführt im Schuljahr 2020/21 von:

– Sebastian ROHRER 5BHIF-18

– Florian SCHWARZL 5BHIF-21

Betreuer / Betreuerin:

Dipl.-Ing. Harald Haberstroh

Wiener Neustadt, am 23rd February, 2021

Abgabevermerk: Übernommen von:

Eidesstattliche Erklärung

Wiener Neustadt, am 23rd February, 2021

Verfasser / Verfasserinnen:

Sebastian ROHRER

Florian SCHWARZL

Contents

Preface

This is **Version 2018/07/01** of our diploma thesis created in LaTex, based on the template provided by Wolfgang Schermann. During the creation of this document we were able to take a glimpse at the immense possibilities of machine learning. Artificial intelligence and robotics are, as seen over the last years, topics of immense importance, more so now than ever. This rise in importance and usage creates the necessity to be informed about state of the art technology and have some form of basic understanding of the principles applied in these technologies. Artificial intelligence is nothing new and has been used by our robotics teams, primarily for image and pattern recognition. The lack of proper self-driving vehicles was mainly due to the technological and financial requirements.

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDES- LEHR- UND VERSUCHSANSTALT WIENER NEUSTADT** Fachrichtung: Informatik Ausbildungsschwerpunkt: Maschineningenieurwesen

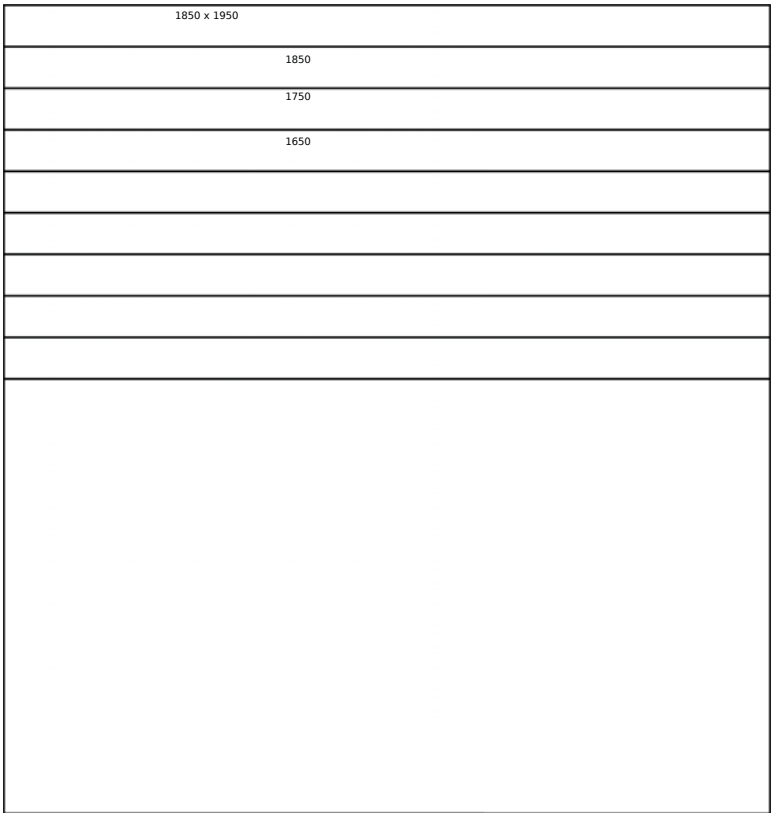
Diplomarbeit Dokumentation

Namen der Verfasser/innen	Name 1 Name 2 Name 3 Name 4 Name 5 Dieses PDF ausfüllen und als PDF drucken. Das gedruckte PDF in den Ordner pdf speichern.
Jahrgang Schuljahr	5AHIF 2015 / 16
Thema der Diplomarbeit	Entwurf eines Versuchstandes für Kreiselpumpen
Kooperationspartner	Irgendeine Firma wenn vorhanden

Aufgabenstellung	Hier steht die Aufgabenstellung in einigen Sätzen.

Realisierung	Beschreibung der Realisierung.

Ergebnisse	Beschreibung der Ergebnisse.

Typische Grafik, Foto etc. (mit Erläuterung)	Beschreibung des unten eingefügten Bildes. Unten klicken um das Bild zu wählen. Das Bild muss als PDF vorliegen (z.B. mit GIMP als PDF speichern). 

Teilnahme an Wettbewerben, Auszeichnungen	1. Preis in Irgendeinem Wettbewerb

Möglichkeiten der Einsichtnahme in die Arbeit	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt

Approbation	Prüfer	Abteilungsvorstand
(Datum, Unterschrift)	Mag. Max Mustermann	AV Mag. Max Mustermann

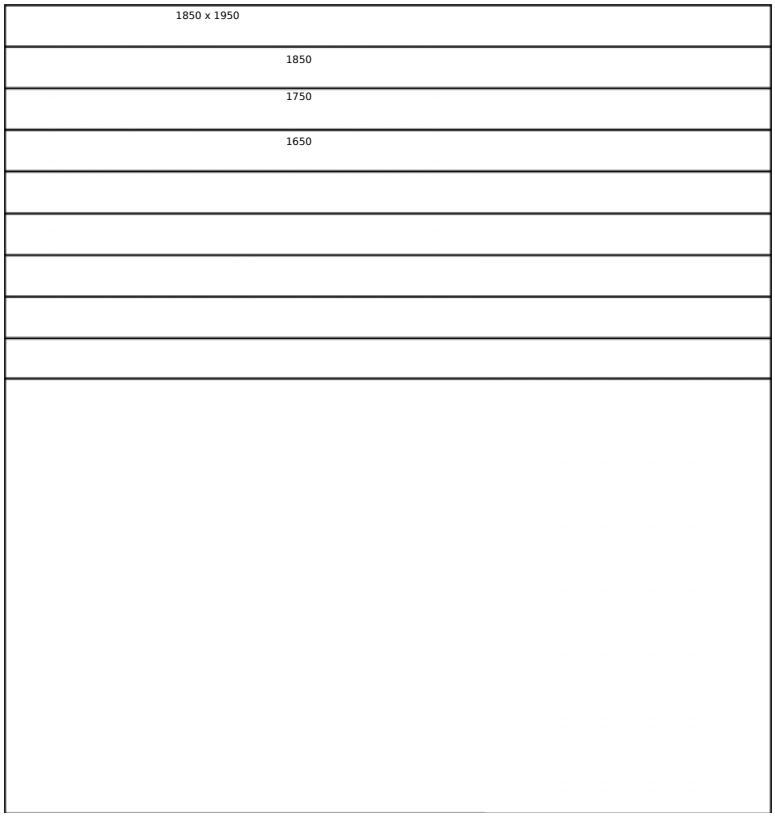
Diploma Thesis Documentation

Authors	Name 1 Name 2 Name 3 Name 4 Name 5
Form Academic Year	5AHIF 2015 / 16
Topic	Titel englisch
Co-operation partners	Irgendeine Firma wenn vorhanden

Assignment of tasks	Hier steht die Aufgabenstellung in englisch in einigen Sätzen.

Realization	Beschreibung der Realisierung auf englisch.

Results	Beschreibung der Ergebnisse auf englisch.

Illustrative graph, photo (incl. explanation)	Englische Beschreibung des unten eingefügten Bildes. Unten klicken um das Bild zu wählen. 

Participation in competitions, Awards	1. Preis in Irgendeinem Wettbewerb

Accessibility of diploma thesis	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt

Approval	Examiner	Head of Department
(Date, Sign)	Mag. Max Mustermann	AV Mag. Max Mustermann

Kurzfassung

After our school received two Amazon AWS DeepRacer vehicles, those vehicles had to be put to good use. With this thesis we paved the way for future students to learn about machine learning in an easy and understandable way. More experienced students will be more interested in the ability to change the training algorithm and other parameters, thanks to the possibility of local training.

Abstract

This should be a 1-page (maximum) summary of your work in English.

Chapter 1

Introduction

The goal of this document is to provide a solid foundation for others wanting to work with the AWS DeepRacer. In addition to that this document also provides information about other possible applications of the AWS DeepRacer, how it can be used in other situations than those intended by Amazon and the limits of these vehicles. Artificial intelligence has been used extensively in robotics, mainly in the form of image and pattern recognition. This allows robots to follow predefined paths or drive to certain areas. While these forms of artificial intelligence have been in use in our robotics lab for some time, the approach which is provided by reinforcement learning has not seen any application. This was mainly due to a lack of fitting equipment and usable software. The recently acquired AWS DeepRacers offer an opportunity to explore the options and applications of reinforcement learning. As there is currently no suitable environment to put them to good use, we were tasked with creating a suitable foundation for others to learn about reinforcement learning. The main part

Chapter 2

Reinforcement Learning

Author: Florian Schwarzl

2.1 RL-Baiscs

Chapter 3

AWS Implementation of Reinforcement Learning

Author: Florian Schwarzl

With DeepRacer, Amazon has created their own implementation of reinforcement learning, geared towards developers, with focus on learning about machine learning concepts. Compared to other robotics competitions, the DeepRacer League is more proprietary, as it is not just intended as a way of leaning about Machine Learning, but also a way for Amazon to promote its services. As a result, Amazon has tailored the reinforcement learning environment to suit their services.

3.1 Concept

Reinforcement learning as a whole is a complex topic that can seem overwhelming. In order to simplify the learning process and create an even playing field for all participants, only some parts of the entire learning process can be modified. When training via the AWS cloud, Amazon allows only the configuration of certain hyperparameters, the action space and the creation of a custom reward function. This reward function, whose purpose was explained earlier, represents the core of their implementation. It has by far the largest impact on the performance of the car both in simulations and real world scenarios. Apart from those two modifications, Amazon offers very little in terms of customisation. This simplification has multiple reasons, the first one being that, as mentioned earlier, it offers programmers with less experience in reinforcement learning the option to participate in the races. Amazon themselves label the DeepRacer as a way to learn about machine learning.The second reason behind this simplification is that it provides a common basis for the races, which, like in any sport, is required in order to provide a fair competition.

3.2 Reward Function

The reward function determines the reward the agent receives for taking a certain action. Therefor the reward has to depend on the environment and the situation in which the agent is currently in, which is called the state. As it is not possible to capture all aspects of a situation, the actual state is abstracted into a predefined set of parameters. These parameters are the foundation which the entire reward process is based upon.

The function itself is written in python and has to return a floating point number, which will from here on be called reward. The reward represents the immediate feedback

the agent receives when moving along the track. This feedback acts as an incentive plan an what action the agent will take. If an incentive plan is not considered carefully, it can lead to unintended consequences of opposite effect. This is the case because an immediate reward alone is not enough to determine if an action has a positive effect, as this affect can come up delayed. This requires not only the current action but also subsequent actions to earn a positive reward for the agent in order for it to consider the initial action preferable.[1]

One example of such unintended consequences of opposite effect occurred during one of the first training sessions. Using a simple reward function which took only the speed and offset from the centre line into consideration, the agent began to drive in a zig-zag pattern on the centre line. The reason behind this was that the reward function gave a significant reward for staying close to the centre of the track. Following this concept, the agent maximised the received reward by diving in said pattern. However due to this behaviour the vehicle performed poorly in terms of speed and was prone to going off track, especially when tested on a physical track in a real world environment. Avoiding scenarios like the one described above is crucial to creating and training functioning models.

3.2.1 Parameter Reference

The reward function receives a predetermined set of parameters, supplied via a Python dictionary. These parameters represent the abstracted information available to the agent during training and, subsequently, during testing. They are abstracted from the image provided by the single camera. This abstraction causes a natural loss in accuracy, as it is neither feasible nor possible to capture a complete picture of the environment in which the agent takes its actions. Taking into account this gap in precision and modifying training behaviour accordingly represents a major part in creating working models.

Not every parameter given has to be used in order for the reward function to be usable. Simpler functions might only rely on the speed and distance to the centre line and still manage to consistently complete laps. The following subsections represent a list parameters which we used in our reward functions, in the order in which we first made use of them. Reference is taken from the official DeepRacer documentation.[2]

Parameter: track width

This parameter is one of the simplest, yet also one of the most important. It represents the width of the current track in meters as a float value. The width of the track is used to give other parameters, like the distance between the car and the centre line enough context to be useful. Without this reference, the relative position of the vehicle on the track can not be determined. This is why this parameter is often used in tandem with the distance from the centre line, as described in the following subsection.

Parameter: distance from centre

Similar to the previous parameter, this one is used primarily to determine the relative position of the car on the track. Combining these values, the track width and the distance from the centre line, is sufficient for creating a simple reward function that will teach the model to follow the centre line. The following code listing shows a basic reward function utilising these two parameters. The track is divided into three sections on either side of the

[1] AWS, *AWS DeepRacer: Developer Guide*, p. 31 f.
[2] AWS, *AWS DeepRacer: Developer Guide*, p. 59 ff.

centre line. The reward is based on the sections, with those further away from the centre giving less reward.

Listing 3.1: Reward function using track width and distance from centre

```
def reward_function(params):
    '''
    Example of rewarding the agent to follow centre line
    '''

    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3

    return float(reward)
```

Parameter: speed

This parameter provides us with information about the current speed of the agent. This information is supplied in the form of a float value ranging from 0 to 5 meters per second. Although this is one of, if not the simplest parameter provided, it is non the less one of the most important values. After all, the concept of the DeepRacer project is to race cars controlled solely by machine learning algorithms. Therefor speed is in the end the deciding factor. One might think that the usage of this parameter is simple, increase the reward the agent receives when it is driving faster. This will definitely lead to the agent going fast, too fast in fact.

As it is in the real world, driving through a turn at high speed might result in the car losing contact with the road and subsequently going off said road. Most notably during the early stages of training, the agent was incited to drive at a slower, more constant speed. By default, the agent tries to move as fast as possible, which inevitably leads to the car going off track during testing on real tracks. Keeping this default behaviour in mind, performance will increase notably during early training sessions if the agent receives a reward for staying at a medium speed rather than going full throttle.

Parameter: steps

The step parameter, or rather the step count is an integer which count the action taken by the agent during a lap. As said, one step corresponds to one action taken by the agent according to the current policy. This parameter might not seem like an important value to consider, but depending on the reward function used, it is essential to creating a model capable of driving on a real-world track. Let us take the reward function displayed in 3.1 as an example. In this the agent receives an increased reward for staying near the centre line. In order to achieve this, the agent corrects its movement should it divert too much

from the line. This diversion can be caused my several things, for example an uneven track, different traction on the wheels or simply by inaccuracy when steering, all of which can occur frequently in a real world scenario. In the simulated environment however, such discrepancies have to be manually inserted, as seen in Section 3.5. Without the manually added noise, the agent will learn to constantly correct its movement, staying as close to the middle as possible. Such a behaviour could be observed during our first training sessions. A model trained this way will perform poorly during testing on real tracks, since it is permanently trying to steer.

Penalising the agent for taking far more steps than required is one way to prevent it from driving in a zig-zag pattern.

Parameter: Coordinates x, y

Although this pair of parameters is rarely used in the reward function, the coordinate system is useful to know if other parameters, namely the waypoints, want to be used. This parameter provides a list of tuples, which contain a pair of coordinates. These coordinates describe the current position of the vehicle relative to the bottom left corner of the simulated world. It is important to note that the point of origin is not the border of the track but rather the border of the predefined, simulated world. Taking this into consideration, it is questionable how accurately these coordinates will be when used during deployment on a real track.

Parameter: waypoints

This parameter is, together with others related to the waypoint system, one of the option for navigating along the track. Each simulation world has a predefined set of waypoints distributed along the centre line of the track. These waypoints are stored in this parameter, which is a list of tuples. Every tuple represents a single waypoint in the form of its coordinates along the x-axis and y-axis, respectively. The frequency and total number of waypoints differs greatly between tracks. More so, there is no official documentation about where these

3.3 Action Space

3.4 Simulation Worlds

3.5 Background Noise

Chapter 4

Vehicle

Author: Florian Schwarzl

The DeepRacer itself represents a core part of this thesis, as it is used to test the trained models in a real-world environment. Our school acquired two of these cars. one of which we are using for this thesis. Below is a list of all parts included with one DeepRacer vehicle.

1. Vehicle chassis
2. Vehicle body cover
3. Compute module battery
4. Power cable and power adapter
5. Vehicle battery

The vehicle itself is separated into two parts. The upper part houses the compute-module and its battery. On top of the car are four small contraptions for holding the cove in place. The cover is a simple plastic sheet meant to serve as visual cover. Other than hiding the interior, this hood serves no purpose. Therefor we removed it during most of our training sessions. Metal clips are used to keep the cover in place if used during training.

4.1 Chassis and Accessories

The vehicle itself consists primarily of a four wheel drive chassis which holds all other components. The chassis can be further separated into a lower part, which contains the brushed electric motor, and an upper part that carries the compute module and a power bank to supply it. The entire car is build on a scale of 1/18 to a real car, meaning proportions like distance between wheels were kept realistic. This is especially apparent while driving, as one would expect better manoeuvrability and smaller turning radius.[1]

At the front there are three USB ports used to mount the camera and other equipment like keyboards. As a crucial part of any self-driving car, the camera provided with the DeepRacer provides a 4-megapixel image directly to the compute module. Since we are working with the first edition of the DeepRacer vehicle and not with the newer DeepRacer Evo, which has stereo cameras and a LiDAR[2] sensor, object avoidance and head-to-head racing are not supported by default. It is however possible to purchase an upgrade kit for 249,00 US$, which includes an additional stereo camera and the LiDAR sensor.

[1] AWS, *AWS DeepRacer: Developer Guide*, p. 77.
[2] Light detection and ranging

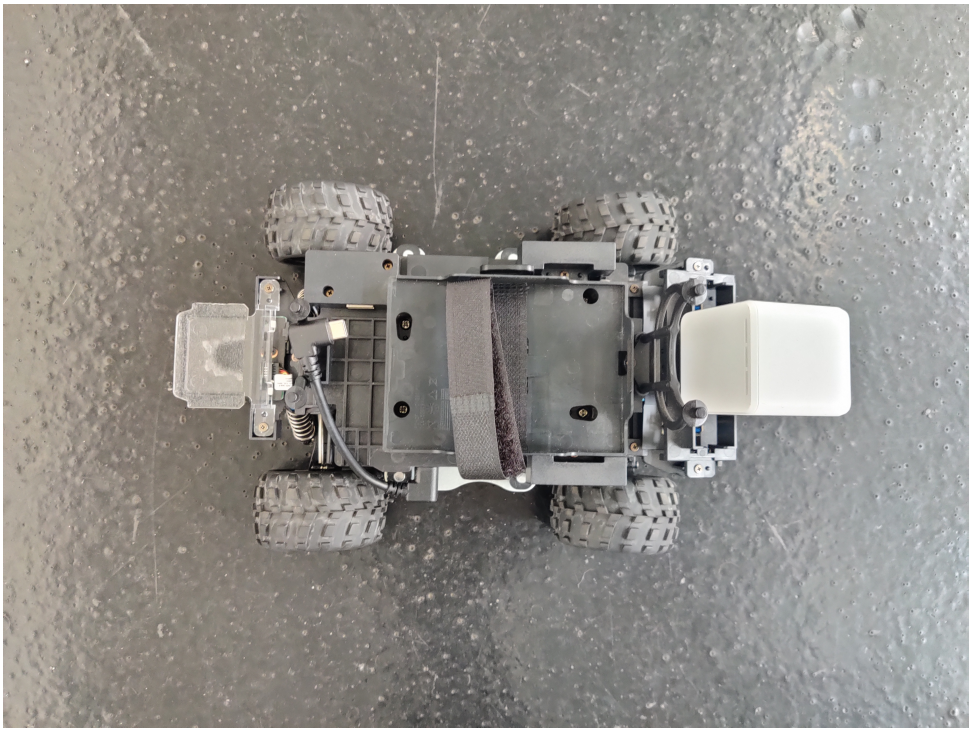


Figure 4.1: AWS DeepRacer Vehicle top view

4.1.1 Vehicle Steering

The car uses what is called Ackermann front-wheel steering. The front wheels are mounted on suspensions which are connected with a second axis. This second axis, located behind the primary one is moved to the left or right and the wheels follow this movement. The steering pivots point inwards on the back of the front axle. They are connected via a bar called a track rod. By moving this bar to the left or right, the vehicles wheels are turned. The length of the track rod is slightly shorter than that of the axle. This leads to the wheels seemingly being misaligned, but this condition is intended.[3]

Advantage of Ackermann Steering

The purpose of this steering mechanism is to minimize or in the best case completely eliminate wheel slip. The wheel alignment around a common turning point for both front wheels enables them to trace circles with different radii. Put differently, since the front wheels have different distances to the centre of the curve, in order to properly follow the intended path, they must turn at different angles.

Steering angle configuration

What the agent considers to be the centre steering angle as well as the maximum steering angles can be configured via the web interface. This is necessary so that the car can drive in a straight line. Incorrectly configured steering can lead to severe hits in performance as the model has to compensate for the drift caused by the misaligned wheels. In order to achieve the best results, the values defined in the action space before training should match the actions possible in real world scenarios. Alternatively, the steering configuration

[3]Norris, *Modern steam road wagons*, p. 63.

can be customised to fit the one defined in the action space. While fitting the simulation to the real world actions is likely to yield better results, it is often simpler to approximately configure the car's steering angle to match those defined in the action space.[4]

4.1.2 Vehicle and Compute-Module Batteries

4.2 Compute Module

The upper component houses the "brain" of our car. The compute module consists of an Intel Atom processor, 4 Gigabyte of RAM and 32 Gigabyte of memory, which can be expanded upon via a SD card. This hardware is running Linux Ubuntu 16.04.3 with Intel OpenVINO™ and ROS[5] Kinetic. Apart from that the chassis offers 4 USB type A ports, 1 USB type C port, 1 Micro-USB port and one HDMI port. The USB type C port is used to supply the compute module with power, while the HDMI port offers the ability to connect a display and directly access the modules operating system.

As seen in figure 4.2, facing the left side of the car are three small LED lights. These lights are used to display the status of the power supply and the wireless LAN-connection. The first light indicates various states of the compute module, more precisely the states of the operating system running on the compute module. The following list shows the possible stages that this indicator can display.[6]

- **Off** signals that the computer is turned off or is not supplied with sufficient power.
- **Blinking yellow** indicates the BIOS and the operating system being loaded after pressing the power button.
- **Steady yellow** means that the operating system is loaded and ready to use in a short amount of time.
- **Steady blue** shows that there is currently an application running on the compute module. This light will also display while automated driving is activated.
- **Blinking blue** indicated that a software update is in progress.
- **Steady red** signals that an error occurred either during startup or while running an application.

Web-Interface

The compute module hosts its own web interface, from which the vehicle can be monitored, remotely controlled and configured. This interface is meant to be the main access point when working with the car. The website can be retrieved by accessing the IP-address of the car with a web browser. The address as well as the password which is required in order to access the configuration interface can be found on a label on the bottom of the car. The primary purpose of the website is configuration. This is indeed very useful as it removes the need to access the compute modules operating system directly. If not for the web-interface, for every change it would be necessary to directly access the operating system by connecting a display, mouse and keyboard.

[4] AWS, *AWS DeepRacer: Developer Guide*, p. 96.
[5] Robot Operating System
[6] AWS, *AWS DeepRacer: Developer Guide*, p. 82 f.

Figure 4.2: AWS DeepRacer Vehicle side view

4.2.1 Configuration

4.2.2 Sensors

In order to acquire the necessary information from the environment, the agent is outfitted with at least one sensor. Although we are only using the camera provided as a default sensor, this subsection covers other sensors available to purchase and use.

As mentioned, the default sensor provided is a single front-facing camera, connected via an USB-port at the front of the vehicle. The image provided by this part is the only information which the agent receives from its environment. This information is sufficient for the agent to traverse a marked track without obstacles or other vehicles. The only other information at our disposal is the sensor data from the electric motor and steering control. This provides us with the information about the current speed and steering angle. Apart from that, all other information comes solely from the camera. Considering this, it is impressive that with a simple camera image, an entire vehicle can be controlled autonomously.

4.3 Remote Control

Chapter 5

The physical Track

Author: Florian Schwarzl

Being able to drive along a simulation of the track is only a partial success. In order to prove successful, the trained algorithm is required to complete multiple runs on a real track using our DeepRacer vehicle. When training an autonomous vehcile with reinforcement learning for a real-world scenario, it is logical to create the simulation so that is represents the real environment as precisely as possible. However, since this is a leaning environment and the environment is set by Amazon, we are required to achieve the exact opposite. We need to create a real-world track that fits those from the training simulation. How accurately the physical course represents the simulated one will have a direct impact on the performance of the vehicle.

We decided on rebuilding a simple loop with a length of 180" and a width of 140" and a 24" wide road. As mentioned later on, these dimensions could not be used.

Before we considered building the track, we drew it on the floor in our robotics laboratory using duct tape. This was possible due to the floor being solid black, just like the road on the track. This variant has several drawbacks compared to a properly built track. Built like this the track is stationary and can not be removed without destroying parts of it. During removal it might even happen that the adhesive will leave stains on the floor, which then again need to be removed. The alternative, interlocking foam pieces, are trans-

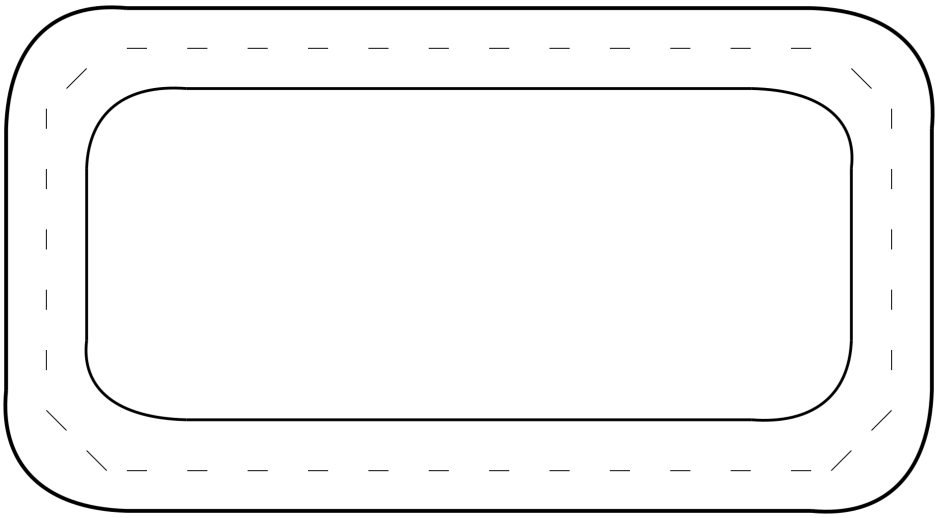


Figure 5.1: AWS re:Invent 2018 track

portable and more durable, as they can be easily removed and rebuild as needed. Their major drawback is the costs. While using duct tape as markings on the floor is cheaper and faster, using foam puzzles will be more efficient in the long run.

5.1 The Material

Choosing the right material for the track proved to be straightforward. At first we thought of interlocking foam pieces. Foam pieces provide many advantages compared to other materials. They are lightweight and easy to transport. Durability and thickness are not as important, as the track is not meant to be stepped on and our car does not cause much wear and tear. Far more important is the ability of the pieces to stick together. The tiles must not under any circumstances break away from each other, this could cause serious damage to the track and vehicle. The only downside was the price. As seen in the table below, laying out the entire loop would cost around 300 €. Our second option was to use duct tape as markings on the black floor in our laboratory. While it is not nearly as durable as the foam tiles, this method is far cheaper. In the end the decision fell on using duct tape markings on the floor for our first test runs. Therefor white and yellow duct tape was bought at a nearby hardware store.

5.1.1 Track and Field Colour

The material is not the only factor we need to consider when building the track. Since our model is trained in a simulation with a specific colour scheme, we went to recreate this colour set in order to achieve the best performance. The road is the easiest to paint, as its colour is solid black with orange markings in the middle. The green area which covers most of the track is far more difficult to recreate. This leads to two options:

5.2 Building the track

After the decision on the material was final, the process of building the track began. For multiple reasons we were required to increase the size of our track, which lead to the track being 204" long and 162" wide. The width of the road was reduced to 22". This, combined with the uneven floor, made accurate measurements difficult. Therefor the track borders have an average derivation of 0.8". In contrary to our expectations the duct tape proved to be durable enough. Other than the tape on the floor representing the road markings, no other materials were used. This means there were no vertical borders around the track. This lead to multiple occasions where the vehicle went off track and ran into an obstacle before it could stop.

5.2.1 Side borders

Our first course did not have any railings or borders on the side which would have prevented the car from going off track too far. The main purpose of these borders would have been to keep the vehicle from driving into obstacles around the track. Although the car is supposed to stop should it recognise that it went off the track, it happens that due to its speed it is not able to slow down in time. The barriers on the track limits serve the purpose to stop the car without damaging it, should such a scenario as described above occur.

The secondary purpose of the barriers is to provide a solid colour background, preferably in contrast to the road. Similar side covers can be seen during official tournaments. Recognising that it drove outside of the track limits is simpler and more consistent.

Chapter 6

Local Training

6.1 Local Training vs. Cloud Training

One of the major drawbacks of using the DeepRacer in a learning environment are the costs of training. Amazon offers easy, albeit functionally limited ways of training RL models in their cloud services. This sort of contradicts the intended use, as the DeepRacer is supposed to offer a simple and affordable entry into the ways of machine learning. In order to circumvent this cost barrier we – like others before us – began setting up a training environment on one of the more powerful computers in the robotics lab. After searching the internet we found a very cost efficient method, to train the model on our own computer. Given that in our robotics-lab we gain access to a "super-computer", which would train our models easily and fast we decided to train it by ourselves. Amazon doesn't provide a easy to use interface to download and upload models because they don't want to support you using your own servers and computers to train. Following this idea it seems that a lot of people worked their way around and made their own GUI's and interfaces. The setup for local training is available on GitHub

6.1.1 Prerequisites

[1]. In order to function properly the computer had to meet the following requirements:

- Docker + Docker Compose
- vncviewer
- Nvidia CUDA/cuDDN
- Minio
- Python

6.1.2 Docker

Docker

Docker is free to use software which isolates applications using container virtualisation. It simplifies application deployment because the container containing all the necessary packages, and package dependencies, can be easily transferred and mapped into a so-called Docker image. The container ensures that the resources used on the computer are separated and managed. This includes: code, runtime-modules, system tools, system-libraries etc.[2]

[1] **https://github.com/aws-deepracer-community/deepracer**.
[2] *Overview of Docker Compose*.

Docker-Compose

Compose is a tool used, to handle multiple Docker images. Using a YAML-File you can configure all the services a application needs. Then with only one command, you can start all your files and services from the configuration.

YAML

YAML, a super-set of JSON, is a human-readable language used for serialisation of data. Its most common use is in data files, in which data is stored and transmitted to an application. It targets the same communication applications as the Extensible Markup Language(XML). [3]

6.1.3 VNC-Viewer

VNC

In computing, virtual network computing (VNC) is a platform independent, graphical desktop sharing system that uses the remote frame buffer protocol (RFB) to remotely control another computational system. You differentiate between a VNC-Client and VNC-Server, multiple Clients can connect to the same server.

RFB

The Remote Frame-buffer Protocol(RFB) is a TCP Network-protocol using the Port 5900 + Desktop Number. Its main Task is to transfer data such as screen-contents and user-inputs.[4]

VNC-Viewer

VNC-Viewer is the Program, which allows the Client to connect to a server and remotely control it. On the other site, the server will run a VNC-Server application.[5]

6.1.4 Nvidia CUDA/CUDDN

CUDA

CUDA (previously known as Compute Unified Device Architecture) is a programming technology developed by Nvidia that allows part of the program to be processed by a graphics processing unit (GPU). Providing additional computing power in the form of GPU is necessary,in a highly parallelisable program sequence (high data parallelism), because the GPU is usually much faster than the CPU. CUDA is mainly used for scientific and technical computing. CUDA is usually programmed in C, although multiple wrapper for other languages like Ruby, Python or Java exist.[6]

cuDDN

Nvidia cuDDN(CUDA Deep Neural Network) is a library that's used for GPU accelerated deep neural network learning. It provides highly optimised implementations for routines,

[3] *YAML-File*.
[4] Richardson and Levine, *RFB-Protocol*.
[5] *Overview of VNC*.
[6] *Nvidia-CUDA*.

like pooling, forward and backward convolution, normalisation and activation layers.[7]

6.1.5 Minio

Minio is a popular High Performance open source-based object storage server.It's optimised to be used with Amazon S3 cloud storage service. You can use it to build a performance-based infrastructure for machine learning, application data workloads and analytics. If you want more control over the object storage server, you can also configure applications that are configured to communicate with Amazon S3 to make Minio a alternative to Amazon.

6.1.6 Python

Why use Python?

All the Amazon services provide an API that is used to communicate between them. Python3 is used in all scripts to build an establish a connection and handle the user defined functions. The Artificial Intelligence itself is also coded in Python. Therefore its the easiest and quickest way to write Code an extend the existing Codebase in Python. The rising number of Machine Learning Library's for python is one of the key points why Amazon decided to code in this language.

Python

Python is a low level Programming-Language used in many different use-cases. Its easy to learn and has a very good documentation. You have to preinstall it, because the local training solutions are all coded in Python. Its easy to read and understand if you have a big file and it also keeps the line numbers short.[8]

6.2 Amazon tools

To train and create the model according the rules, you have to use the programs provided by Amazon. The major tools we used in our project were:

- Sagemaker
- Robomaker
- S3

6.2.1 Amazon Sagemaker

Sagemaker is a service amazon provides, to create, train and handle machine learning models. It handles the normally complicated learning process to make it easier, training multiple large models and don't have to worry about all the tools and work processes that need to be done to learn efficient. Sagemaker contains a large tool-set to handle all different components of machine learning. It also include a debugger and pro-filer, which helps identifying training-errors and performance-issues.
Sagemaker Studio is a web-based visual user interface that provides you full access, control and insight of all the steps that need to be done to create train and analyse a model. You can upload own models and analyse results, make experience or provide it to get into production.

[7] *Nvidia-cuDDN*.
[8] *Python Documentation*.

Sagemaker Data Wrangler reduces the time required to prepare Data for Machine Learning. It provides easy access to Data-preparation-workflows including data-visualisation, data-adjustment and data-exploration.
Sagemaker Ground on the other side, handles all the quick and easy to use data that is needed to train a model properly. Because a model is only as good as the data that is given to it, with Sagemaker ground you are able to manage all the training data.
Sagemaker Edge Manager enables you to optimise, protect, monitor, and maintain machine learning models on edge device clusters to ensure that the models deployed on edge devices work properly.[9]

6.2.2 Amazon Robomaker

Robomaker is a Cloud solution provided by amazon to create, simulate and test robot-based applications in large amounts. It provides access to a fully scale-able simulation infrastructure for all you robots and your CI/CD-Integration. Your are also given access to an IDE to develop new Robot applications and ROS(Robot Operating System) extensions to make your robot collaborate with the operating System.Robomaker also provides extensions to a lot Amazon Cloud Services like Kinesis(used to Stream Videos), Polly(Speachrecognition) and Cloud Watch(Monitoring and logging) for users that use ROS. It includes a lot of Example Applications that allow an easy and quick Access. Using the AWS-Services you are able to develop complete solutions that include powerful machine learning, voice recognition and speech processing capabilities and integrate them in Robot-applications.[10]

6.2.3 S3

Amazon Simple Storage Service is a data Storage solution on whom you are able to store any kind of objects. It provides a high level of security, scale ability and performance. You can use s3 for a large variety of different use cases such as websites, mobile applications, IoT and Big Data Analytics. You are able flexibly scale your storage resources to meet fluctuating demands - without upfront investment or resource acquisition. With S3 you save money without sacrificing performance by storing data in storage classes that support different levels of data access at appropriate fees.Your data is saved via encryption functions to provide the access management of unauthorised access. Access Point facilitate the data-management access with specific permissions for your applications using a shared data set. Using specific Data-Requests you are able to make Big-Data-Analyses over all your stored S3 Objects.[11]

6.3 Other Tools

These two Tools are getting installed automatically if you run the init.sh Script as explained below. The most important two are Gazebo and rviz.

6.3.1 Gazebo

Gazebo is an 3-Dimensional open source Robotics and Physics Simulator. It integrates the ODE Physics engine, OpenGL rendering ans supports also sensor and actuator control.

[9] *Sagemaker*.
[10] *Robomaker*.
[11] *s3*.

Gazebo uses multiple high-performance realistic physics engines like Bullet. It provides environment rendering like lighting, shadows and textures.[12]

6.3.2 rviz

ROS(Robot Operating System) Visualisation (rviz) is graphical user interface which outputs images and videos. In this use-case it acts as the Visual Component on which you can see the training progress and the Deepracer. Gazebo simulates the Progress and the Physics and then sends the processed output to rviz which shows the video-stream. This is very useful, because you can see a livestream from the virtual Deepracer and its training progress. Based on this you can adjust the hyperparameter to get better performance.[13]

6.4 Implementing the local training methods

6.4.1 Deepracer-Wiki

Although most of the Information about the Deepracer and The Amazon Deepracer League is in the Online Management Console of Amazon, some users made an own independent Deepracer-Wiki, in which you find a lot of information for local training, parameter adjustment and vehicle hacking. It also links to a community-blog, a community-website and a general community-GitHub-page. [14]

Local-Training Solutions

The Wiki includes a total list of 6 different documented GitHub projects from users, who tried to locally install all the amazon services and set up a whole local training environment. In our installation process we tried two different Repository's, because we couldn't get the first one to work on our PC. More Information about the process will be in the next section.

6.4.2 First Approach

Our first approach was to clone a GitHub repository, who implemented the whole algorithm and a graphical user interface. The author of this project relies heavily on another repository from the GitHub user crr0004. Because his instructions were unclear and the project itself was lacking features and had poor usability, he decided to expand the project and make it easier to use. Before getting started with installing the tools needed to run it on your machine you have to check if your computer meets the prerequisites: At first you have to ensure that your operating system is Ubuntu 18.04 or higher, because some of the packages needed are only available on Linux. The next main point is that your system has to run on a Nvidia GPU and have CUDA/CUDNN installed and configured. If your machine doesn't provide a Nvidia GPU you are only able to train on your CPU which is much slower and takes a lot of time. You also should have installed Docker as well as the Nvidia-Docker runtime. This is needed for the training environment to gain access to the GPU and create new timelines. Lastly vncviewer has to be installed, because its the program that controls all interaction between the graphical user interface and the controllers from amazon. If your system meets all the requirements above you have to clone to run the init.sh script. This may take up to 10 minutes and all it does is, it clone crr0004's[15]

[12] *Gazebo*.
[13] WilliamWoodall, *rviz*.
[14] Tptak, *Deepracer Community Documentation*.
[15] Crr0004 and AlexSchultz, *Local Training False Ímplementation*.

repository, runs the required scripts and creates the necessary folder structure including all the needed files, to manually adjust your code, train your model, and upload it to amazon. After the installation and initialisation process is completed, we were able to start a training session but after about 10 minutes the program crashed with a python buffer overflow. Searching the internet we did not find any evidence, that we made errors during the installation process and we also were not able to fix it.

6.4.3 Final Approach

Our second try was to download Matt Camps[16] solution, which also relies heavily on crr0004's GitHub repository, but it has more detailed instructions and doesn't use as much self programmed scripts.The only Prerequisites you have to have installed on your computer are the Nvidia CUDA drivers and docker compose. That made it easy, to install and debug if error occurred. The next step was to clone the Repository and run the init.sh. Although following the tutorial we were not able to start a training session. After debugging and logging we found out, that the project heavily relies on Amazon Sagemaker, but the corresponding docker image was never pulled and installed. After downloading it, the scripts ran flawlessly and no following errors occurred.

6.5 Training and setup process

6.5.1 Installation Guide

After our successful installation i am going to explain the complete installation process. Before the beginning you have to have all the prerequisites installed.[16]

1. At first set the docker-nvidia2 as the default runtime in your daemon.json file that's located in /etc/docker/ . You will find the argument *default-runtime* which you set to *nvidia*
2. The next step is to install Amazon Sagemaker. Use the Command *sudo apt-get install Sagemaker* to download it from the Ubuntu packet-manager. This is necessary, because the following code will depend on the provided API from the Package.
3. Completing the installation clone the GitHub Repositoryhttps://github.com/mattcamp/deepracer-local/ and run the *start-training.sh* Script.

6.5.2 Folder structure

In the Main-Folder, called deepracer-local are all files needed, to train the model. The most important scripts are:

- start-training.sh
- stop-training.sh
- delete-last-run.sh
- upload-current.sh
- mk-model.sh
- local-copy.sh

[16] mattcamps et al., *Local Training Implementation*.

start-training.sh

Starts a new Training-Session. The model data directory must be empty, because the current progress is saved there. If you want to use a pretrained model, you have to set "pretrained": "true" in hyperparams.json. When starting for the first time, it will download 10GB of docker-images, therefore it will take a while to start the training process.

stop-training.sh

Stops the Training. If running, it will take at least 20s to stop, because it will first stop Sagemaker, then it sleeps till Robomaker creates a model.tar.gz in which the model data from the current model is stored and then it will stop completely. In this Format it is ready to be uploaded on a physical deepracer.

delete-last-run.sh

Clears the current bucket to prepair the model data directory, to start a new training session.

upload-current.sh

Loads current model files into a Model directory, that the user can specify before.

mk-model.sh

Creates a complete model from the training model, that can be uploaded to a physical car.

local-copy.sh

Copies the current trained model data to a save directory. This command has to be executed before the delete-last-run.sh script, otherwise all model data will be lost.

other important files that change regularly are:

- hyperparams.json
- config.env
- custom-files directory
- reward.py

hyperparams.json

Includes the Hyperparameters that can be changed, such as batch-size, loss-rate and number of epochs.

config.env

Stores the Training Parameters and the track name. The track name has to be the same as in the training-params.yaml file.

reward.py

Stores the reward function, that can be changed by the user. This is most important file and is changed the most.

6.5.3 starting a training session

If you want to start a new session you have to decided between creating a new model within the session or pretraining an old one and improving the performance.
If you create a new model you have to run delete-last-run, than start-training. This will output the current training parameters and number of epoches.
If you decide to train an older model you have to run local-copy before and change the "pretrained" parameter in the hyperprams.json to true and than follow the instructions as explained above.

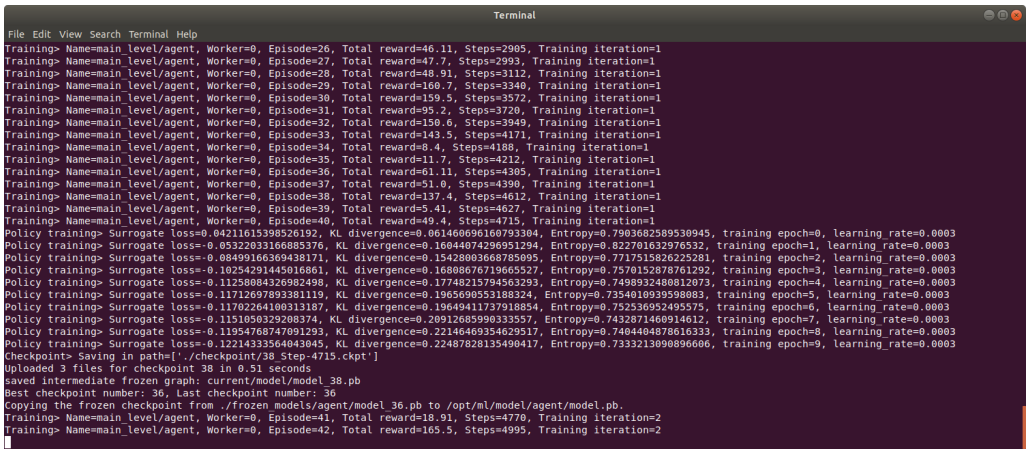


Figure 6.1: Output from start-training[17]

If you are complete with the session, use stop-training. This will take approximately 20 - 25s. When the stopping process is completed you should copy your model via local-copy into a save directory. Completing the whole progress you run mk-model an get the finished model, to upload it to your deepracer.

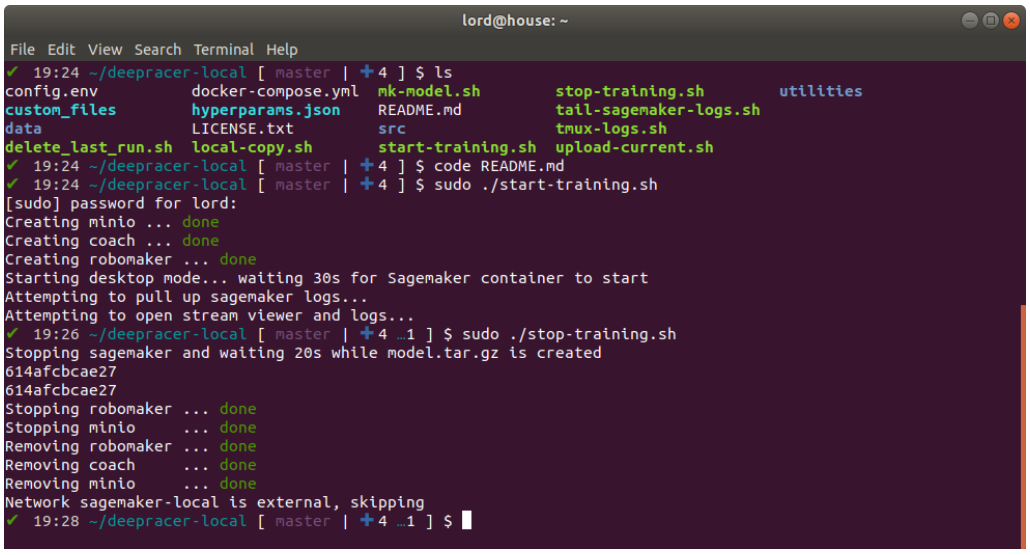


Figure 6.2: Output from stop-training[18]

6.5.4 Uploading a local trained Model

After finishing a training process you have to upload your model into the Cloud. This is necessary if you want to submit your Model in a Racing League or if you want to evaluate the Training information. Amazon doesnt provide an easy interface to upload data in your model garage.I am going to explain the solution we achived to upload a model in the s3 bucket an use it in your garage.

1. At first you have to create a model an train it for at least 5 minutes. This is necessary, because after the training, amazon creates an entry in the s3 bucket, in which all the model information is stored.
2. The next step is to create the model.tar.gz from your locally trained model. This will be achieved by using mk-model.sh command.
3. Afterwards you have to copy
   - model-directory
   - ip-directory
   - metadata.json

   and swap them with the corresponding directories in the s3 bucket.
4. At the End you have to navigate back to your model garage and should now be able to evaluate this model. Although the name doesnt change, the metadata and functions should have been swapped.

After an update of the AWS-Console from the 2021-02-22 there is another easier solution:

1. You have to use the command mk-model.sh on your pretrained model.
2. The next step is to navigate into the S3 bucket, create a new Bucket and import the whole folder including:
   - model-directory
   - ip-directory
   - metadata.json
   - reward-function.py
3. Lastly navigate to your model-garage and import the created S3 bucket into the Garage. The Cloud will automatically detect the new model and you can start an evaluation.

6.6 Errors occurring during the Installation process

At our first try we installed all the needed tools and services, but made the mistake to forget to specify the Nvidia CUDA drivers. Although the installation was successful, we were not able to train the model correctly, because the drivers didn't handle the communication between the individual services well. The GUI had a lot of bugs and was mostly unresponsive. We decided to uninstall all the drivers, files and services and get back to 0. We read the instructions one more time slowly and then realised our human error. After repeating the whole installation process but including the wright driver version everything seemed to work fine. 10 minutes into the training process we started receiving python buffer overflow errors, which is very unusual. Debugging the error and reading all log information, we were not able to detect the problem causing the buffer overflow. We gathered information on the internet and came to a conclusion, that the Amazon tools couldn't allocate the

memory of the graphics cards properly. Trying out different work-around and solutions we were not able to fix the error or reproduce it in another scenario. After two weeks an a lot of frustrating hours we decided to delete the whole installation and try another different approach.
Meanwhile we tried setting up the whole process on our laptops using Ubuntu in an virtual machine. But after discovering the CUDA drivers wouldn't work out of a Virtual Machine because they were not allowed to allocate memory on the physical graphics card, likewise because the hypervisor wouldn't allow that, we tossed out that idea quickly.
Another common error we discovered is that all Repository's rely on the Amazon Tools like Sagemaker and Robomaker, but they dont include the installation in the instructions. You have to read the Ubuntu Wiki and search for the according packages to download.

6.7 Difference between Cloud Training and Local Training

As expected the Local training wasn't performing as good as training in the cloud. In the next section will compare the Hardware and Performance of the PC and the Cloud

6.7.1 Performance-Difference

To exactly measure the difference between the Cloud and the research Computer, we set up a Test-Model. The model specifications are:

Table 6.1: Sample Model Specifications

Hyperparameter	**Value**
Gradient descent batch size:	64
Entropy:	0.01
Discount factor:	0.999
Loss type:	Huber
Learning rate:	0.0003
Number of experience episodes between each policy-updating iteration:	20
Number of epochs:	10

Using the standard reward function, the model was trained on the Oval Track. We choose this test-environment because its very easy to learn, and only given 30 minutes to train the model, we couldn't train it on a more difficult Track, expecting to master it. Moreover the Race-Mode was set to time-trial, because this model isn't trained to race in a league against other Deepracers.
To compare the two models, we started an evaluation in the AWS-Cloud. This process simulates the car on the track and drives 3 laps. Then it determines the probability that the deepracer will successfully complete a lap without crossing the line.

Cloud-Training

After finishing the Evaluation in the Cloud the Results were:

Evaluation results

Trial	Time (MM:SS.mmm)	Trial results (% track completed)	Status
1	00:28.886	100%	Lap complete
2	00:13.854	42%	Off track
3	00:30.031	100%	Lap complete

Figure 6.3: Evaluation Results from Cloud Training

Local-Training

Evaluating the local-Training, the results were as following:

Evaluation results

Trial	Time (MM:SS.mmm)	Trial results (% track completed)	Status
1	00:08.134	44%	Off track
2	00:08.064	43%	Off track
3	00:14.474	81%	Off track

Figure 6.4: Evaluation Results from Local Training

6.7.2 Price-Difference

The Table below shows the Difference between storing and evaluating Data online and offline.

Table 6.2: Pricing for model training with AWS services.

service	pricing AWS	pricing PC
training and evaluation	3.50 US$ per hour	0.29 US$ per Kilowatt per hour
model storage	0.023 US$ per GB and month	free of charge

19 20

6.7.3 Result

After training in the Cloud and training locally, we came to the conclusion, that the choices depends on the use-case. Comparing the training Data, the Cloud is 40% faster, otherwise comparing the Cost-Explorer, the Cloud is 1200% more expensive. The AWS-Management Console has a very detailed documentation and guides that help you creating your first model. Its also very easy and fast to train and evaluate the model. On the other side local training is very cost-efficient although, there is poor to none documentation and if errors occur or, you need to debug you have to have a lot of knowledge of the structure of the Amazon tools and services that are running in the background.

[19] Cited date 2020/08/20
[20] *CostPower*.

Chapter 7

Deepracer League

7.1 Online League

7.2 Tournament

Index

Autoren Index

Literatur Index

Bibliography

AWS. *AWS DeepRacer: Developer Guide*. URL: https://docs.aws.amazon.com/deepracer/latest/developerguide/awsracerdg.pdf.

CostPower. URL: https://www.e%5C-control.at/konsumenten/strom/strompreis/was%5C-kostet%5C-eine%5C-kwh#:~:text=Unterschiede%5C%20nach%5C%20Wohnort%5C-,Eine%5C%20Kilowattstunde%5C%20Strom%5C%20kostet%5C%20einen%5C%20durchschnittlichen%5C%20Haushalt%5C%20mit%5C%203.500%5C%20kWh,in%5C%20Graz%5C%2018%5C%2C47%5C%20Cent. (visited on 02/22/2021).

Crr0004 and AlexSchultz. *Local Training False Implementation*. URL: https://github.com/aws%5C-deepracer%5C-community/deepracer%5C-for%5C-cloud/blob/master/README.md (visited on 12/20/2020).

Gazebo. URL: http://gazebosim.org/ (visited on 02/18/2021).

mattcamps et al. *Local Training Implementation*. URL: https://github.com/mattcamp/deepracer%5C-local/blob/master/README.md (visited on 02/18/2021).

Norris, William. *Modern steam road wagons*. University of California Libraries, 1906. URL: https://archive.org/details/modernsteamroadw00norrrich/page/64/mode/2up.

Nvidia-CUDA. URL: https://developer.nvidia.com/cuda%5C-zone (visited on 02/18/2021).

Nvidia-cuDDN. URL: https://developer.nvidia.com/cudnn (visited on 02/18/2021).

Overview of Docker Compose. Docker Inc. URL: https://docs.docker.com/ (visited on 02/18/2021).

Overview of VNC. URL: https://www.realvnc.com/de/ (visited on 02/18/2021).

Python Documentation. URL: https://www.python.org/doc (visited on 02/18/2021).

Richardson and Levine. *RFB-Protocol*. URL: https://tools.ietf.org/html/rfc6143 (visited on 02/18/2021).

Robomaker. URL: https://aws.amazon.com/de/robomaker/ (visited on 02/22/2021).

s3. URL: https://aws.amazon.com/de/s3/ (visited on 02/22/2021).

Sagemaker. URL: https://aws.amazon.com/de/sagemaker/ (visited on 02/22/2021).

Tptak. *Deepracer Community Documentation*. URL: https://wiki.deepracing.io/Main_Page (visited on 02/18/2021).

WilliamWoodall. *rviz*. URL: http://wiki.ros.org/rviz (visited on 02/18/2021).

YAML-File. URL: https://yaml.org/ (visited on 02/18/2021).