Institute of Computer Science
Faculty of Mathematics and Computer Science
Database Department

End-to-End Reinforcement Learning Training of a Convolutional Neural Network to achieve an autonomous driving agent resilient to light changes

Exposé Master's Thesis

submitted by:
Georg Schneeberger

matriculation number:
3707914

Supervisor:
Dr. Thomas Burghardt

© 2023

Abstract

This master's thesis investigates the use of neural networks for self-driving. The goal is to create an autonomous driving agent that is resilient to changes in light conditions. The agent will employ preprocessing steps and a convolutional neural network to achieve this. The agent will be trained using reinforcement learning in a simulated environment with changing light conditions to help the agent generalize.

The reinforcement learning agent's task is to complete a parcour in a simulated environment without collisions. The thesis builds upon previous student's work at the ScaDS.AI [1], the task specifications and evaluation approaches are reused for comparability. The focus of this thesis is to improve the resiliency to changing light conditions.

Georg Schneeberger
3707914

Inhaltsverzeichnis

1. Motivation

The increasing utilization of artificial intelligence in academia and industry have lead to massive efficiency improvements for all kinds of tasks. The development of autonomous vehicles promises to greatly reduce the number of traffic accidents and transportation cost [2]. As a result, researchers and private enterprises from all over the globe are making progress towards fully autonomous driving agents and integrating them in commercial vehicles, many companies started to integrate adaptive cruise control and lane centering assistance [3]. Due to the recent developments in artificial intelligence and the very high complexity of the task of autonomous driving, artificial intelligence often plays a big role in these systems [4].

Predictions for the future of autonomous driving have been very optimistic and although huge progress has been made, the task of fully autonomous driving is still far from being solved [5]. This thesis aims at contributing to the research in this field by applying reinforcement learning to autonomous driving agents in a simulated environment. This work builds upon the work of [1] and will use the same task and evaluation metrics. This thesis focusses on improving the agent's resiliency to changing light conditions by training a convolutional neural network end-to-end using reinforcement learning.

2. Research Goals

The goal of this thesis is to contribute in the domain of autonomous driving by investigating the use of reinforcement learning to train an autonomous driving agent that is resilient to changes in light conditions. The agent is evaluated on simulated parcours that consist of a series of goals indicated by two blocks, a parcour is successfully completed if the agents drives through all goals without collisions. This thesis builds upon previous work at the ScaDS.AI [1] and uses the same parcour and task specifications. The agent from previous work was not able to reliably complete parcours under changing light conditions, motivating the research goals of this thesis.

The self-driving agent is trained using reinforcement learning in a simulated environment, the training process will include changing light conditions and possibly data augmentation to help the agent generalize. Parcours of different difficulties and lighting settings are used to evaluate the agent's reliability and generalisation capabilities. The most important evaluation metric is the success rate. A parcour is considered a success when the autonomous driving agent passes all goals without any collisions.

2.1. Question 1 - Is it possible to train an autonomous driving agent consisting of a convolutional neural network with end-to-end reinforcement learning to reliably solve the parcours of all difficulty levels?

The previous work [1] showed that it is possible to train an agent using reinforcement learning to solve the evaluation parcours, however the trained agents were not successful in reliably traversing the parcours of higher difficulty levels. Furthermore this work will implement the agent in a fundamentally different way, the agents developed in previous work utilized an extensive preprocessing pipeline to extract the relevant information from the camera images whereas the agents in this thesis will use a convolutional neural network to learn and extract the relevant information themselves.

Due to these differences in implementation and as a prerequisite for question 2 and 3, it is first important to investigate if it is possible to train an agent to reliably solve the parcours of all difficulty levels. This raises question 1: Is it possible to train an autonomous driving agent consisting of a

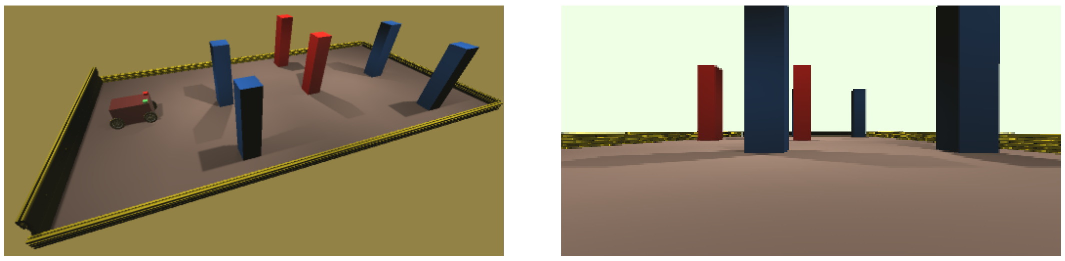


Abbildung 2.1.: Example image of a parcour and the agent's camera

convolutional neural network with end-to-end reinforcement learning to reliably solve the parcours of all difficulty levels?

The question will be answered by training agents that have been developed based on related work and analyzing their performance on the evaluation parcours. The evaluation parcours consist of different difficulty levels, the agent's success rate will be primarily used to answer the question.

2.2. Question 2 - Is it possible to use an end-to-end trained CNN to make the agent robust to changing light conditions?

While question 1 simply investigates if it is possible to train an agent to reliably solve the parcours of all difficulty levels, question 2 investigates if it is possible to train an agent that is robust to changing light conditions in addition to being capable of solving parcours of all difficulty levels. The performance of agents from previous work [1] declined massively under changing light conditions. This raises question 2 - Is it possible to use an end-to-end trained CNN to make the agent robust to changing light conditions?

The question will be answered by training agents that have been specifically designed to be robust to changing light conditions, the agents will be trained using reinforcement learning in a simulated environment with changing light conditions and possibly further data augmentation to help the agent generalize and learn. The agents will be evaluated on the evaluation parcours used in question 1 with changing light conditions. Similarly the success rate will be primarily used to evaluate and compare the agent's performance. The difference in performance for different light conditions will be used to answer the question, if the performance is similar for all light conditions the agent is considered robust to changing light conditions.

2.3. Question 3 - Is it possible to use a neural network that can be transfered to a physical robot?

One goal of the ongoing research at the ScaDS.AI is to build real life robots for demonstration and research purposes [6], the robots are based on the NVIDIA JetBot platform. The robots are equiped with a camera, wheels and a small computer. A future goal is to transfer a trained agent onto these robots, however the limited processing power of these robots might not be sufficient for more complex agents that utilize neural networks. This raises question 3 - Is it possible to use a neural network that can be transfered to a physical robot?

The question will be answered by investigating the processing power required to run the preprocessing steps and neural networks used in the agents that are developed in this thesis. This will be evaluated empirically by creating replays of the agents in simulations and running these replays on the physical robots. If the robots are able to reproduce the behaviour from the replays, the agents can be considered transferable to the robots.

3. Related Work

This thesis will reinforcement learning in the training of an autonomous driving agent that utilizes a convolutional neural network to process visual input. The approach used in this thesis differs greatly from previous work at the ScaDS.AI [1] both in the agent design and training setup. Therefore research relating to reinforcement learning algorithms, convolutional neural networks and self-driving will be reviewed.

3.1. Reinforcement Learning

Reinforcement Learning algorithms have been around for a long time, but only recently have they been able to achieve superhuman performance in games and control tasks [7]. Most reinforcement learning algorithms formalize the problem as consisting of an environment and an agent. The environment consists of a state space and an action space and a reward function that takes state-action pairs as input. Reward functions assign positive rewards to actions that are deemed to be desirable by the algorithm designers, for example scoring a goal in a football match. Reward function can also assign negative rewards to undesirable actions, for example collisions in a driving simulation. The agent processes the environment, takes actions and observes the assigned reward. The observed rewards are then used to update the policy 3.1. RL algorithms train the agent to select an action in a given state and maximize the cumulative reward along the state transitions. The process of selecting an action is called the policy π [8].

Reinforcement learning algorithms are classified into two major groups. RL algorithms that use a model of the environment are called model-based algorithms, algorithms without such models are called model-free algorithms. Algorithms from both groups have been successfully used in a wide range of applications, model-based algorithms are often much more complex but have been shown to be successful at many task that require planning [9]. Model-free approaches are often simpler and more flexible, they have shown great success in various control tasks [7].

Early implementations of RL algorithms used functions with a discrete input space for their policy such as for example tables that store state-action pairs and associated values. These algorithms belong to the family of value-based algorithms and include Q-learning and deep-Q learning. In

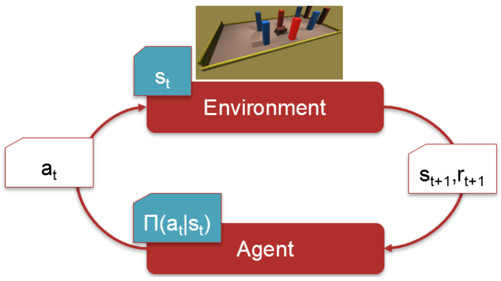


Abbildung 3.1.: RL Training Cycle: The agent selects action a_t based on policy $\pi(a_t|s_t)$ at state s_t and recieves the next state s_{t+1} and rewards r_{t+1} from the environment. Observed rewards are used to update the policy.

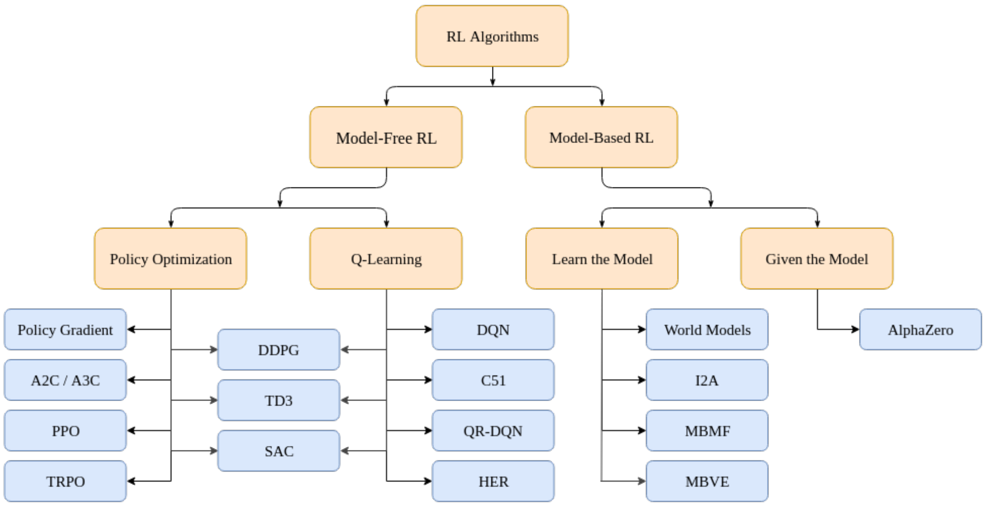


Abbildung 3.2.: Taxonomy of RL algorithms from OpenAI's Spinning Up course [10]

Q-learning, the trained policy function takes a state as input, searches the state in the table and selects the associated action with the highest value. For many problems the use of these tables in not feasible due to the amount of state-action pairs, many extensions to the algorithms have been developed such as for example deep-Q network (DQN). DQN uses a deep (convolutional) neural network to approximate the Q-values [7]. Although they have been used to great success for control tasks they will not be used in this thesis, as they require discrete action spaces and the environment in this thesis consists of a continuous action space.

The other family are policy-based algorithms, instead of learning the values associated to state-action pairs these algorithms learn the policy directly. This allows for the use of continuous action spaces. The Proximal Policy Optimization algorithm was developed to improve the stability of policy-based algorithms [11]. The PPO algorithm restricts the size of policy changes caused by parameter updates, which ensures the policy cannot change drastically and improves stability. PPO is currently one of the most popular algorithms for reinforcement learning, and has already been successfully used in the domain of autonomous driving [1]. The PPO algorithm can be used with convolutional neural networks as well and will therefore be used in this thesis.

Convolutional Neural Network for Reinforcement Learning

Convolutional neural networks are a neural network architecture specifically developed for processing image data, they consist of a number of filters and a fully connected neural network. The filters are applied to the image in a sliding window fashion, the filters detect patterns in the image such as for example edges and corners. Multiple successive applications of such filters enables the network to learn hierarchical information and recognize more complex structures. The fully connected neural network analyses the results of the filters and makes the final prediction [8].

CNNs are often used in Reinforcement Learning since RL problems often require an agent to process visual input. Furthermore CNNs can be trained end-to-end in Reinforcement Learning compared to other feature extraction methods, which means the CNN can learn what features are important for the task at hand. Therefore a convolutional neural network will be used to process the camera images instead of a hand-crafted feature extraction method.

CNNs typically do not take the raw camera/simulation images but rather preprocessed images, e.g. greyscaled images [7]. Preprocessing steps can help in reducing the complexity of the input space. Convolutional neural networks require a lot of data to learn, data augmentation can help increase the size of the training set and to make the agent more robust. Data augmentation generates new samples from already collected ones by applying transformations to the samples.

3.2. Self-Driving

As mentioned before, there has been a lot of progress in the domain of self-driving in recent years. Sophisticated self-driving algorithms often consist of many components to achieve satisfying performance, Tesla's self-driving for example uses separate object detection, occupancy and planning components that are built on top of convolutional neural networks [12]. Self-driving in a real world environment is a very complex task, especially when including other traffic participants. It requires agents that consist of multiple complex components [4] and is beyond the scope of the thesis. Instead I aim to contribute to the domain by expanding on previous research and focus on the training of a convolutional neural network. Approaches from the domain of self-driving will be used to improve the training of the agent, for example reward shaping [4].

This thesis builds directly upon the work of [13], [6] and [1]. [13] built a self-driving agent that was trained to avoid collisions in a simulated arena using an evolutionary approach to neural network training. The agent used a preprocessing pipeline to extract information from visual input. The extracted features were given to a neural network policy. [6] investigated the feasibility of transferring the agent to the real world. The research showcased many difficulties, most notably the object recognition part of the preprocessing pipeline. [1] investigated a different task than the two previous papers, the agent was trained to pass a parcour by driving through a sequence of goals. This task is identical to the one investigated here. [1] successfully used PPO to train the agent which also used a preprocessing pipeline similar to [13]. The instability of the hand-crafted preprocessing pipeline and promissing results by CNNs from other RL researchers in the domain of self-driving [14] motivate the choice of CNNs as the feature extraction method in this thesis.

3.3. Simulation for Reinforcement Learning and Self-Driving

Simulations play a huge role in reinforcement learning and thus the development of self-driving agents. Simulations provide a huge number of benefits over real world experiments. They are much cheaper and faster to run than real world experiments, furthermore they can be run in parallel. In addition the programmers have direct and perfect control over the environment, as such programmers can for example change the simulation speed. This allows for fast experimentation and training of reinforcement learning agents. Simulations also allow for the creation of scenarios that are not possible in the real world. This is especially useful for reinforcement learning agents that are trained to avoid collisions. Simulations also allow for the creation of ground truths such as perfect sensor data and object bounding boxes [15].

Simulated environments often serve as baselines for reinforcement learning algorithms, most famous are the atari games [7]. The Python Gynasium API was developed for easy reuse and comparison of reinforcement learning algorithms for different problems [16], the Gymnasium API defines an interface that can be used to model tasks as reinforcement learning problems. A wide range of reinforcement learning frameworks support the Gymnasium API, for example Google's dopamine [17] and OpenAI's baselines [18]. Advanced simulations like the Unity engine [19], the physics simulator MuJoCo [20] and the driving simulator Carla [15] can be integrated with the Gymnasium API.

There are also dedicated frameworks for reinforcement learning that directly integrate with simulation engines. [1] used the ML-Agents framework [21] to train the self-driving agent in Unity directly.

In this thesis Unity will be used for the simulation, the simulation will be integrated with the Python Gymnasium API and PPO algorithm. This approach is chosen instead of the ML-Agents framework since it allows for more flexibility and control over the simulation and training process.

4. Methods

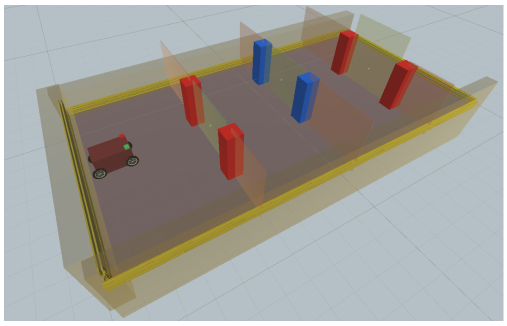
4.1. Task Description

In this section describes the task and the simulation environment, which are the foundation of this thesis and will remain unchanged throughout the course of the project. The task is to develop an agent using reinforcement learning that is able to complete a parcour in a simulated environment without collisions. The agent has to traverse the parcour by passing through a number of goals indicated by pairs of either red or blue blocks without collisions. This problem belongs to the class of single player continuous state and action space problems. The observation space of the agent consists of an image that is taken from its front facing camera. At each timestep the agent uses a neural network to process the image and produce two actions. The two actions are the acceleration values of the left and right wheel, being applied to the wheels until a new action is selected. The task and agent are simulated using the Unity engine 4.1. The engine handles the rendering of the environment, collisions, agent movement and reward functions.

4.2. Reinforcement Learning Algorithm and Frameworks

As outlined in the related works section, many different RL algorithms can be used to solve single player continuous state and action space problems. The PPO algorithm is most commonly used for problems of this class and has already been successfully used in the investigated task [1]. The PPO algorithm will be used as discussed in Related Work.

In this thesis, the PPO algorithm from the stable-baselines3 library [18] will be used. The library is based on the PyTorch framework and provides APIs for training, logging, visualization and evaluating reinforcement learning agents. The training and evaluation algorithms can be modified easily. RL algorithms from stable-baselines3 are applied on Gymnasium environments [16]. As mentioned before the Unity simulation is integrated in a Gymnasium environment. The communication between Unity and the Gymnasium environment is realized using the Peaceful Pie library [22]. The


(a) Example image of the agent at the start of a parcour with 3 goals in Unity

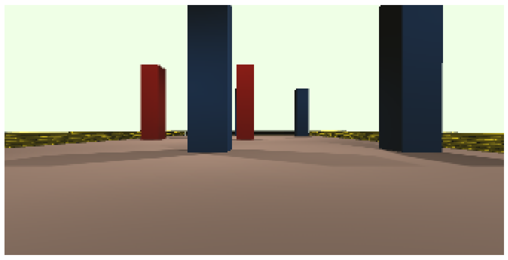

(b) Agent camera view

Abbildung 4.1.: Unity simulation environment and agent camera view

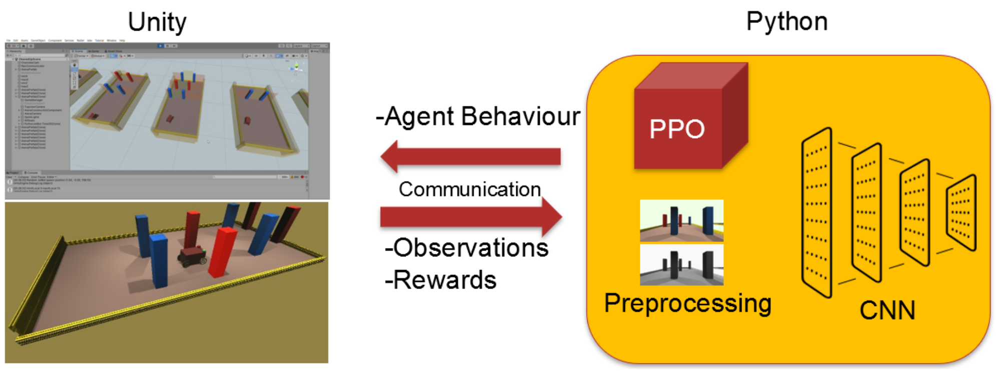


Abbildung 4.2.: Unity and Python communication

PyTorch framework is used to implement the convolutional neural network, see 4.2 for a summary of the Unity-Python interaction.

4.3. Reinforcement Learning Algorithm Details

4.3.1. Reward Function

The design of the reward function is crucial for the success of any reinforcement learning algorithm and agent. The reward function should encourage the desired behaviour. In our case, the agent should drive through all goals without collisions as quickly as possible. The previous work by [1] awarded a reward of 100 for completing the parcours, a reward of 1 for passing a goal and a reward of -1 for missing a goal, colliding with a wall/obstacle and timeouts. This should encourage the agent to navigate the simulation without collisions and to pass through all goals, see EventReward 4.3. Furthermore a reward proportional to the agent's velocity at each step was awarded to encourage speed, resulting quick parcour completions, see VelocityReward 4.3.

The combination of the EventReward and VelocityReward includes everything needed to encourage the desired agent behaviour. However the agents might fail to learn to navigate the parcour, since the EventReward is very sparse. Reward shaping is the practise of providing reinforcement learning agents with frequent and accurate rewards. This helps the agent develop the desired behaviour quicker and more reliably since reward signals are less sparse and less delayed [4]. The VelocityReward can be considered as reward shaping since it provides the agent with a reward at each timestep.

Here the reward function will be further extended with a DistanceReward and OrientationReward, see 4.3. The DistanceReward is proportional to the difference in distance between the agent and the next goal during a timestep, this should encourage the agent to drive towards the next goal. The OrientationReward is proportional to the cosine similarity between the agent's direction and the direction towards the next goal, this should encourage the agent to drive in the direction of the next goal. The partial rewards are combined using a weighted sum. The weights will be determined during the experimentation phase. Setting a weight to zero will disable the corresponding reward shaping component.

$$
\begin{aligned}
R(s_t, a_t) &= c_1 \cdot \mathit{DistanceReward}(s_t, a_t) + c_2 \cdot \mathit{OrientationReward}(s_t, a_t) \\
&\quad + c_3 \cdot \mathit{VelocityReward}(s_t, a_t) + c_4 \cdot \mathit{EventReward}(s_t, a_t) \\
\mathit{DistanceReward}(s_t, a_t) &= \Delta \mathit{distance}(\mathit{Agent}, \mathit{NextGoalPosition}) \cdot \Delta T \\
\mathit{OrientationReward}(s_t, a_t) &= S_C(\mathit{NextGoalPosition} - \mathit{AgentPosition}, \mathit{agentDirection}) \cdot \Delta T \\
\mathit{VelocityReward}(s_t, a_t) &= v \cdot \Delta T \\
\mathit{EventReward}(s_t, a_t) &= \begin{cases} 100, \text{completed the parcour} \\ 1, \text{passed a goal} \\ -1, \text{missed a goal} \\ -1, \text{collision with wall or obstacle} \\ -1, \text{timeout} \\ 0, \text{otherwise} \end{cases}
\end{aligned}
$$

Abbildung 4.3.: Complete reward function R with all its components
S_C: cosine similarity c_i: weights
s_t: state t a_t: action in state t

4.3.2. Frame Stacking

Two configurations of the agent by [1] used a memory to enhance the agent's input. The memory consisted of the input from the last few steps of the agent. This technique of stacking the history has been widely used in RL for continuous [7] and discrete action spaces [23]. This allows the agent to perceive object movement, time and velocities [7]. This frame stacking will also be used to enhance the agent's input, since the next goal may leave the agent's current field of vision.

4.4. Implementation Details for Light Setting Robustness

4.4.1. Convolutional Neural Networks

The works by [6] and [1] showed that the agent's performance greatly depended on the quality of the input preprocessing pipeline. This object detection pipeline had difficulties detecting objects under varying light settings. This thesis uses convolutional neural networks instead of a hand crafted object detection pipeline. CNNs with an adapted training process should make the agent more robust to varying light settings and improve the performance of the agent. Convolutional networks are common practise in the field of reinforcement learning, due to the ability of these networks to adapt and process images. The convolutional neural network could potentially learn to identify relevant information in images more reliably than the previously used image detection pipeline. The research by [6] showed that not all the information provided by the object detection pipeline was considered to be relevant by the neural network. As a starting point for experimentation, the CNN architecture will be the same as [24], which proved successful for simple control tasks.

4.4.2. Feature Reduction and Preprocessing

Several preprocessing approaches can be used to prepare an image, before processing it with convolutional neural networks. Downsampling, greyscaling and rescaling steps will be applied to the images. The goal of downsampling and greyscaling is to reduce the input space size. Both were used in the foundational Atari paper [7]. This results in increased processing speed and can reduce the chance of overfitting of the network. The greyscaled images in 4.4 show that the colour information is not crucial, as the closest blocks remain easily recognizable. Reducing the input space size also helps the PPO algorithm, since more data entries fit in memory, which allows for more efficient training. Pixel values are rescaled to between 0 and 1, which can help the neural network learn quicker [25].

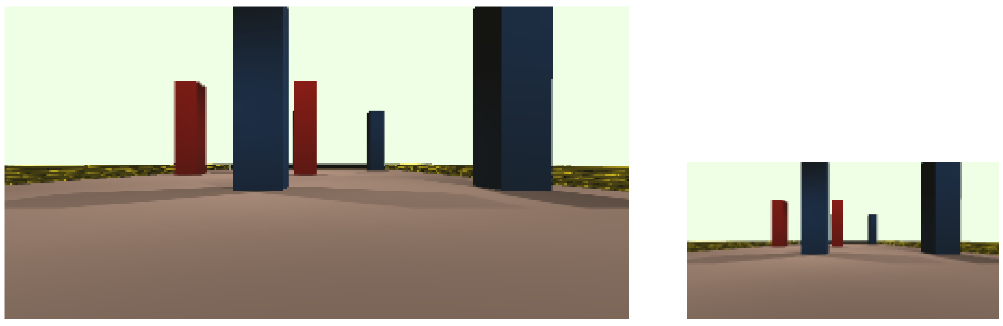
4.4.3. Histogram Equalization

The previous work this thesis builds upon used the HSV colour space to extract the differently coloured objects. Colours in this space consist of three values, referring to hue, saturation and brightness. The hue value was used for extracting the blue and red goal posts. In theory, the utilization of this colour space should make the object detection resilient to changes in brightness, since this information does not affect the hue value. However, this proved to be invalid in practice as shown by [1]. Convolutional neural network typically use the RGB colour space or a greyscale colour space. Image transformations such as brightness and contrast changes can be applied to images before they are processed by the CNN. A histogram equalization of the input images could play a big role in making the agent more resilient to changes in illumination. Image d) in 4.4 shows the effect of histogram equalization on an image, however the image looks worse for identifying the goal posts since the shadows became much darker. This suggests the equalization might not be necessary, nor useful. Therefore equalization and other preprocessing steps will be investigated during the implementation and experimentation phase.

4.5. Training Process

Training Parcours

The achieve good results, the PPO Reinforcement Learning algorithm requires the agent to be placed in a training environment similar to the evaluation environment [8]. In the training process by [1], the goal objects were spawned semi-randomly with a minimum and maximum distance to the next goal. Two different training regimes were used. In Single-Goal-Training the episodes were stopped after completing the first goal or upon collision. In Full-Map-Training the episodes were stopped after completing the whole map or upon collision. The reasoning behind Single-Goal-Training is that the agent will encounter a bigger variety of states during training, since it will start at different positions in the parcour. However, the Full-Map-Training is closer to the evaluation


(a) Original Image (b) Downsampled image

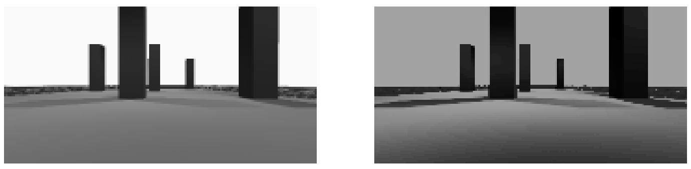

(c) greyscale (d) histogram equalized image

Abbildung 4.4.: 4 Stages of preprocessing images for the CNN

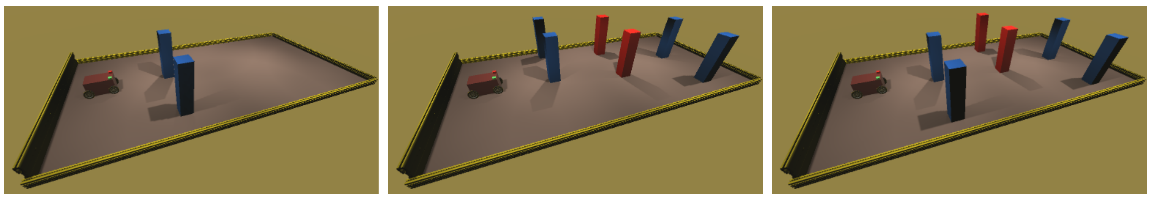

(a) Single-Goal-Training (b) Full-Map-Training (c) Fixed-Training (hard parcour)

Abbildung 4.5.: Training regimes

scenario, since the agent has to complete multiple goals in succession during evaluation. Single-Goal-Training performed worse than Full-Map-Training in the previous work [1] for all evaluation parcours except for the difficult one.

These two approaches will be experimented with, in addition to another approach called Fixed-Training. The most successful approach will then be used for the remainder of the thesis. In Fixed-Training the agent will be trained on the same parcours that are later used for the evaluation. This is standard practise in the domain of Reinforcement Learning [8]. In Fixed-Training, a random evaluation parcour is chosen to train the agent on in each episode.

Training Light Settings

Since the agent utilizes a Convolutional Neural Network to be resilient towards changing light conditions. Furthermore it is necessary to train the agent with varying light conditions, otherwise the adaptability of the CNN would not be fully utilized. This way, the agent will be able to generalize and learn to deal with different light conditions. The light conditions will be randomized for each training parcour similar to the light conditions during evaluation. Training with fixed light settings could also provide interesting insight when comparing the results against training with varying

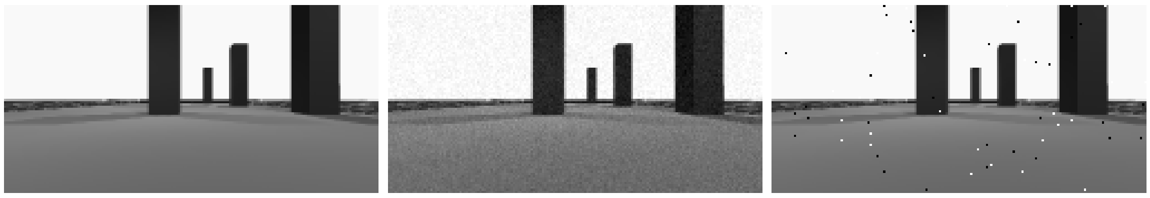

(a) Original image (b) Gaussian Noise $\mu = 0$ $\sigma = 5$ (c) salt-and-pepper noise

Abbildung 4.6.: Data augmentation examples

light settings. This comparison would show if the varying light settings during training help in generalizing to different light settings.

Data Augmentation

Convolutional Neural Networks require huge amounts of data to learn and generalize. Data augmentation is a technique to increase the amount training data by applying transformations to the collected data such as rotation, translation, scaling, flipping and colour changes. Data augmentation is inexpensive, since the new data points are not collected in simulation, as well as providing a more diverse set of training data. [26] employed a diverse set of data augmentation for their imitation learning approach that used a CNN.

During the training process, the collected images will be augmented by applying random transformations to them. The transformations change the image similarly to how different environment conditions (e.g. lighting, camera quality and fog) might change the image. It is not yet decided which transformations will be used, possible candidates are changes in contrast, brightness and tone, as well as filters like Gaussian blur, Gaussian noise, salt-and-pepper noise. Geometric transformations such as translations and rotations are not used since our control commands are not invariant to these transformations.

5. Evaluation and Experimentation

5.1. Evaluation Metrics

During the training and the final experiments, the agents are primarily evaluated based on the success rate, as well as the average completion time and the collision rate. Success rate is defined as the percentage of episodes in which the agent successfully completes the parcour. These metrics, which were previously used by [1], measure the agent behaviour's most important properties.

Additional metrics will be monitored during the training process to identify weak-points and erroneous behaviour of the agent. Monitoring the training process can provide insights into the agent's behaviour and aid in selecting appropriate hyperparameters. TensorBoard will be used to visualize the monitored metrics 5.1. These metrics may include be the average cumulative reward, the average number of passed goals, the average distance travelled, the average amount of collisions, the average game duration and the average speed of the agent.

5.2. Experiments

The proposed experiments build on the scenarios from [1]. These experiments included three parcours with different difficulty levels 5.2 and conducted 3 different experiments with each trained agent. The first experiment used the same settings as the simulation environment. The second experiment was conducted under different lighting settings. The third one changed the motor power of the agent's two front wheels. All experiments primarily used the success rate to evaluate the agent's performance, the success rate is the proportion of successfully completed parcours.

The first two experiments will be used in this thesis as well, using the same experiments allows for an easy comparison to the previous research. The experiment with varying motor power will be omitted, since it is not related to the research goals of this thesis.

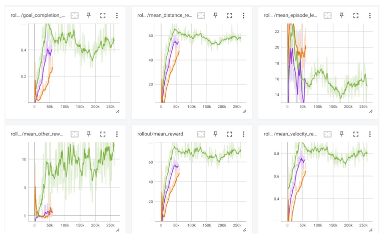


Abbildung 5.1.: Evaluation Metrics in TensorBoard

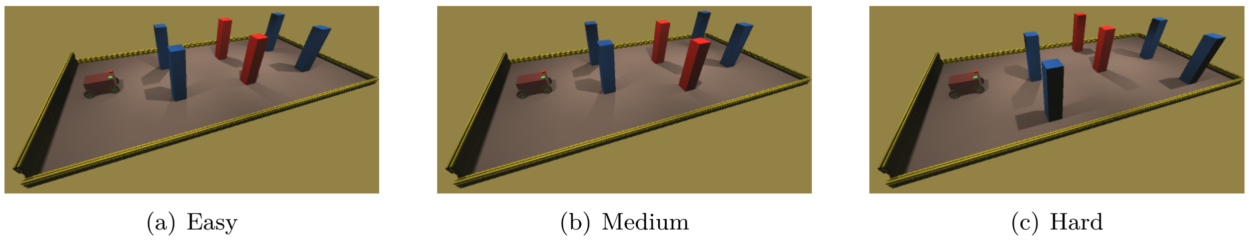

(a) Easy (b) Medium (c) Hard

Abbildung 5.2.: Evaluation Tracks of different difficulties

5.2.1. Research Question 1 - Is it possible to train an autonomous driving agent consisting of a convolutional neural network with end-to-end reinforcement learning to reliably solve the parcours of all difficulty levels?

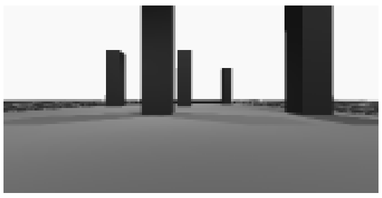
The experiments under minimal changes will be used to judge if the agent is able to reliably solve all parcours. 5.2 shows three parcours of different difficulty levels, there are further variations of these parcours that change the positioning and colour of the obstacles. The parcours are evaluated using the success rate, if the agent is able to reliably solve all parcours the agent and training implementation can be considered successful.

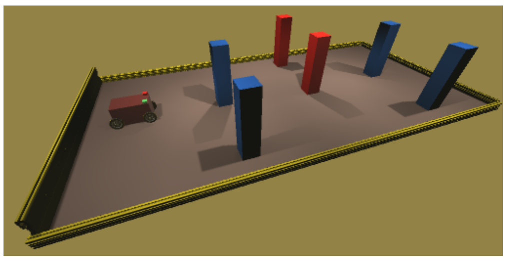
5.2.2. Research Question 2 - Is it possible to use an end-to-end trained CNN to make the agent robust to changing light conditions?

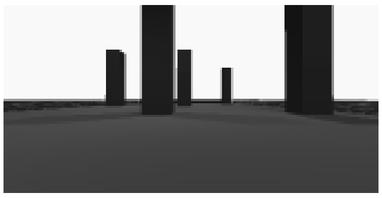
The evaluation parcours will be used with varying light settings to evaluate the agent's robustness towards changing light conditions. The agent's performance will be measured using the success rate, if the agent performs similarly across all light settings, the agent can be considered robust to changing light conditions.

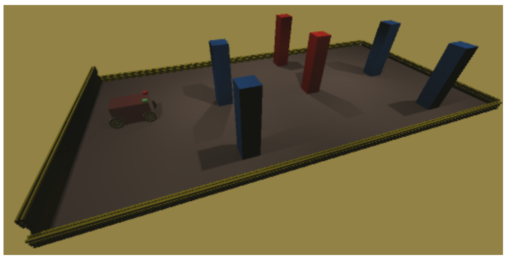
5.2.3. Research Question 3 - Is it possible to use a neural network that can be transfered to a physical robot?

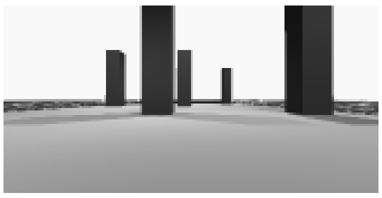
To answer question 3, the developed agent will be evaluated on the JetBot's processing unit. There will be no physical experiments with the JetBot due to time constraints. Instead a replay of an evaluation parcour is generated in Python. The replay is then processed on the JetBot to measure its processing capabilities. The replay will consist of input-output pairs and metadata, such as processing times. If the JetBot is able to reproduce the behaviour from the replay at sufficient speed, the agent can be considered transferable to the JetBot.


(a) Standard Lighting Agent POV


(b) Standard Lighting Arena


(c) Reduced Lighting Agent POV


(d) Reduced Lighting Arena


(e) Increased Lighting Agent POV

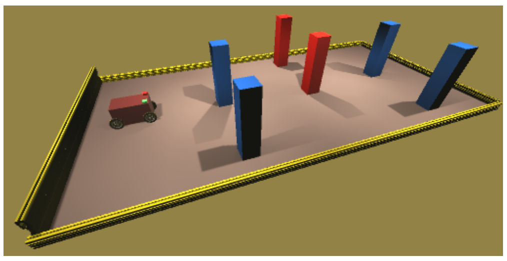

(f) Increased Lighting Arena

Abbildung 5.3.: Agent Camera POV and Arena Screenshots of the different light settings

6. Schedule

- 1 month - Literature research: Search for further approaches to increasing light setting robustness and CNN training.
- 2.5 months - Implementation of the training algorithm. Including preliminary testing/evaluation of configs and hyperparameters.
- 1 month - Training and Evaluation
- 1.5 months - Writing the thesis

Literatur

[1] Maximilian Schaller. „Train an Agent to Drive a Vehicle in a Simulated Environment Using Reinforcement Learning“. Magisterarb. Universität Leipzig, 2023.

[2] Johannes Deichmann u. a. *Autonomous driving's future: Convenient and connected.* 2023. URL: `https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected` (besucht am 09. 12. 2023).

[3] Mike Monticello. *Ford's BlueCruise Remains CR's Top-Rated Active Driving Assistance System.* 2023. URL: `https://www.consumerreports.org/cars/car-safety/active-driving-assistance-systems-review-a2103632203/` (besucht am 24. 10. 2023).

[4] B Ravi Kiran u. a. *Deep Reinforcement Learning for Autonomous Driving: A Survey.* 2021. arXiv: `2002.00444 [cs.LG]`.

[5] Collimator. *The State of Autonomous Vehicles: Seeking Mainstream Adoption.* 2023. URL: `https://www.collimator.ai/post/the-state-of-autonomous-vehicles-in-2023` (besucht am 16. 02. 2023).

[6] Merlin Flach. „Methods to Cross the simulation-to-reality gap“. Bachelor's Thesis. Universität Leipzig, 2023.

[7] Volodymyr Mnih u. a. *Playing Atari with Deep Reinforcement Learning.* 2013. arXiv: `1312.5602 [cs.LG]`.

[8] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

[9] Julian Schrittwieser u. a. „Mastering Atari, Go, chess and shogi by planning with a learned model“. In: *Nature* 588.7839 (Dez. 2020), S. 604–609. DOI: `10.1038/s41586-020-03051-4`. URL: `https://doi.org/10.1038%2Fs41586-020-03051-4`.

[10] OpenAI. *OpenAI Spinning Up Part 2: Kinds of RL Algorithms.* 2018. URL: `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html` (besucht am 09. 12. 2023).

[11] John Schulman u. a. *Proximal Policy Optimization Algorithms.* 2017. arXiv: `1707.06347 [cs.LG]`.

[12] Think Autonomous. *Tesla's HydraNet - How Tesla's Autopilot Works.* 2023. URL: `https://www.thinkautonomous.ai/blog/how-tesla-autopilot-works/` (besucht am 15. 09. 2023).

[13] Jonas König. „Model training of a simulated self-driving vehicle using an evolution-based neural network approach“. Bachelor's Thesis. Universität Leipzig, 2022.

[14] Nilesh Barla. *Self-Driving Cars With Convolutional Neural Networks (CNN).* 2023. URL: `https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn` (besucht am 09. 12. 2023).

[15] Alexey Dosovitskiy u. a. „CARLA: An Open Urban Driving Simulator“. In: *Proceedings of the 1st Annual Conference on Robot Learning.* 2017, S. 1–16.

[16] Mark Towers u. a. *Gymnasium.* März 2023. DOI: `10.5281/zenodo.8127026`. URL: `https://zenodo.org/record/8127025` (besucht am 08. 07. 2023).

[17] Pablo Samuel Castro u. a. „Dopamine: A Research Framework for Deep Reinforcement Learning“. In: (2018). URL: `http://arxiv.org/abs/1812.06110`.

[18] Antonin Raffin u. a. „Stable-Baselines3: Reliable Reinforcement Learning Implementations“. In: *Journal of Machine Learning Research* 22.268 (2021), S. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[19] Unity Technologies. *Unity Engine*. 2023. URL: `https://unity.com`.

[20] Emanuel Todorov, Tom Erez und Yuval Tassa. „MuJoCo: A physics engine for model-based control“. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, S. 5026–5033. DOI: `10.1109/IROS.2012.6386109`.

[21] Andrew Cohen u. a. „On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning“. In: *RL in Games Workshop AAAI 2022* (2022). URL: `http://aaai-rlg.mlanctot.info/papers/AAAI22-RLG_paper_32.pdf`.

[22] Hugh Perkins. *Peaceful Pie, Connect Python with Unity for reinforcement learning!* 2023. URL: `https://github.com/hughperkins/peaceful-pie` (besucht am 23. 10. 2023).

[23] David Silver u. a. „Mastering the game of Go with deep neural networks and tree search“. In: *Nature* 529 (Jan. 2016), S. 484–489. DOI: `10.1038/nature16961`.

[24] Volodymyr Mnih u. a. „Human-level control through deep reinforcement learning“. In: *Nature* 518 (2015), S. 529–533. URL: `https://api.semanticscholar.org/CorpusID:205242740`.

[25] Jason Brownlee. *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. 2020. URL: `https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/` (besucht am 09. 12. 2023).

[26] Felipe Codevilla u. a. *End-to-end Driving via Conditional Imitation Learning*. 2018. arXiv: `1710.02410 [cs.RO]`.