UNIVERSITÄT
LEIPZIG

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

Expose Master's Thesis

submitted by:
Georg Schneeberger

matriculation number:
3707914

Supervisor:
Dr. Thomas Burghardt

Abstract

In my master's thesis I will investigate using 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. This thesis focusses on improving 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 to contribute 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

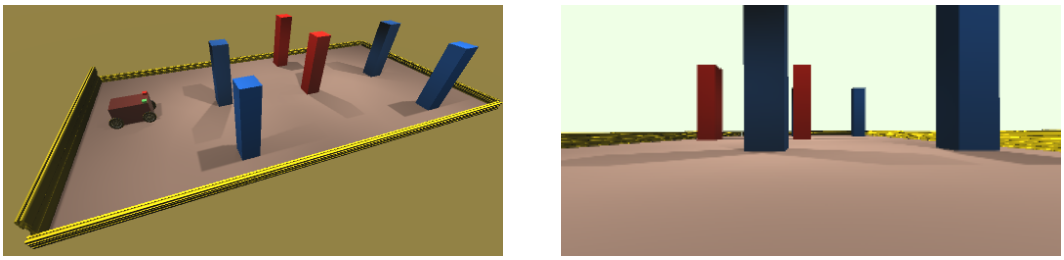


Abbildung 2.1.: Example image of a parcour and the agent's camera

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 algorithm 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 mayor 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

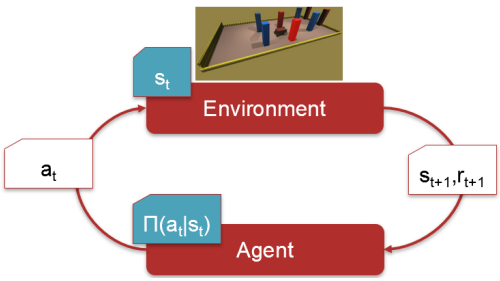


Abbildung 3.1.: RL Training Cycle: The agent selects action a_t based on policy $\pi(a_t|s_t)$ at state s_t and recieves the next state s_{t+1} and rewards r_{t+1} from the environment. Observed rewards are used to update the policy.

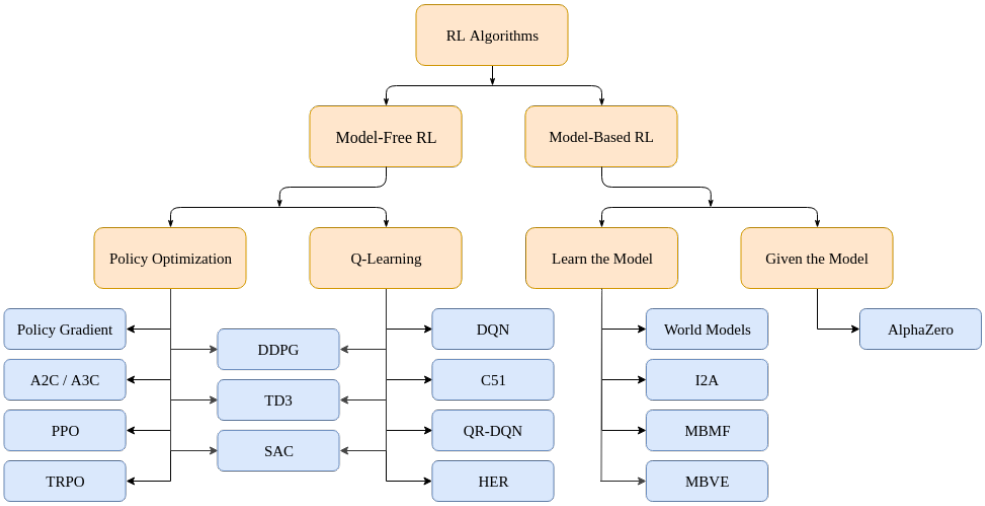


Abbildung 3.2.: Taxonomy of RL algorithms from OpenAI's Spinning Up course [10]

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) that 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, this 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, this 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 including other traffic participants is a very complex task. It requires agents that consist of multiple complex components [4], this 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, this information was given to a neural network policy. [6] investigated the feasibility of transferring the agent to the real world, this 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, such as the ML-Agents framework [21]. [1] used this framework to train the self driving agent in Unity.

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

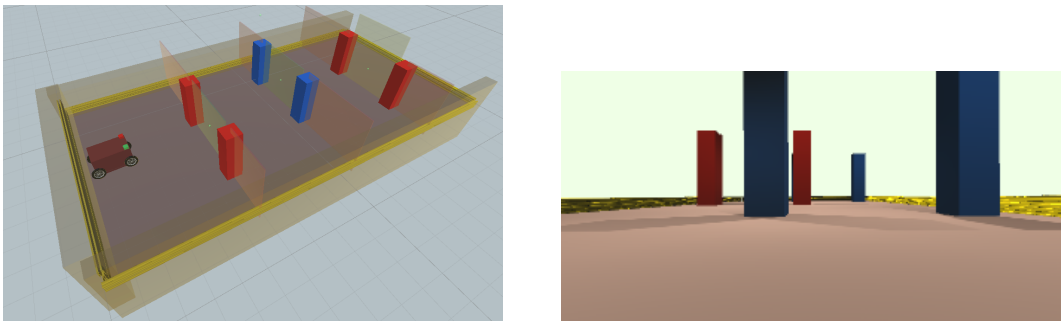
4.1. Task Description

In this section I will be describing the task and the simulation environment as these two aspects are the foundation of this thesis and will not change over 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, these acceleration values are 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]. In this thesis 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 and evaluating reinforcement learning agents. It provides logging and visualization, it can also be extended for example to modify the training and evaluation algorithm. RL algorithms from stable-baselines3 are applied on Gymnasium environments [16], as mentioned before the Unity simmulation is integrated in a Gymnasium environment. The communication between Unity and the Gymnasium environment is realized using



(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

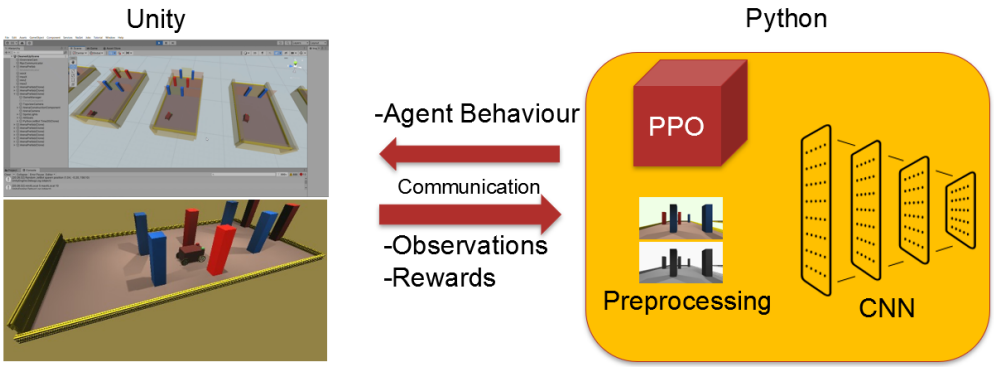


Abbildung 4.2.: Unity and Python communication

the Pieceful Pie library [22]. The PyTorch framework is used to implement the convolutional neural network, see 4.2 for 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 quick 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 not make any 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 and thus quick parcour completions, see VelocityReward 4.3.

The combination of the Event- 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.

In this thesis 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.

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

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 of 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 could 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 an object detection pipeline. Combined with an adapted training process, this should make the agent more robust to varying light settings and improve the performance of the agent. Using convolutional networks to process images is a common practise in the field of reinforcement learning, due to the ability of these networks to adapt. 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, this could be mitigated by training the CNN end-to-end. As a starting point for experimentation the CNN architecture will be the same as [24] which proved succesful for simple control tasks.

4.4.2. Feature reduction / preprocessing

There are several preprocessing approaches that can be used to prepare an image for processing by a convolutional network. The goal of these approaches is to reduce the feature space to increase processing speed, they can also help encourage the network to generalize. The following steps were taken in the foundational [7] paper. These techniques will be used due to the similar complexity of our task and many atari games.

Downsampling reduces size of input space

converting to greyscale reduces size of input space and potentially removes irrelevant information

Rescaling pixel values to between 0 and 1 can help the neural network learn quicker

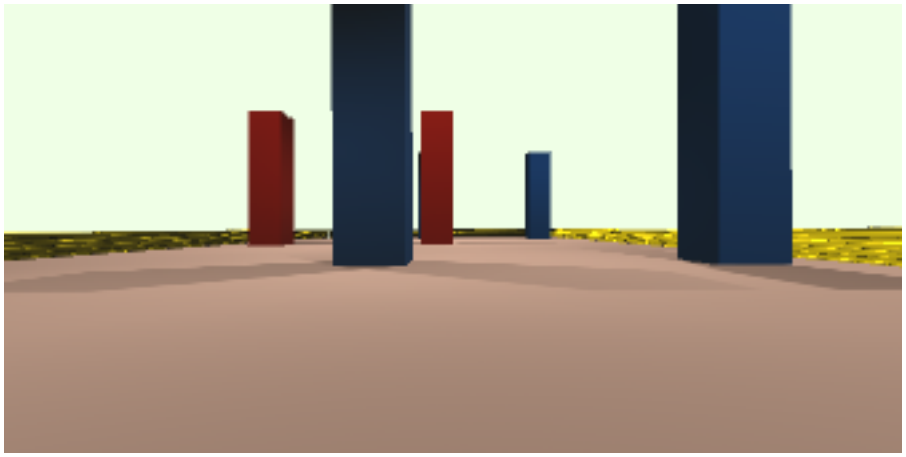
4.4.3. Histogram Equalization

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, one for the hue, saturation and brightness. The hue value was used for extracting the objects. 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 not be true in practice as shown by [1]. Convolutional neural network typically use the RGB colour space or a greyscale colour space. A histogram equalization of the input images could play a big role in making the agent more resilient to changes in illumination, with which the agent by [1] struggled with. Image d) in 4.4 shows the effect of histogram equalization on an image, the image looks worse for identifying objects. This suggests the equalization might not be necessary/useful, it is to be investigated during the implementation/experimentation phase.

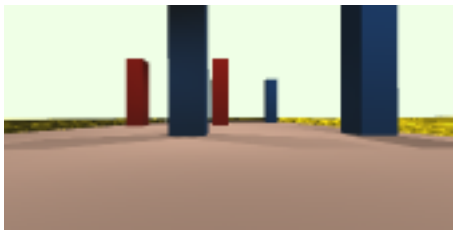
4.5. Training Process

Training Parcours

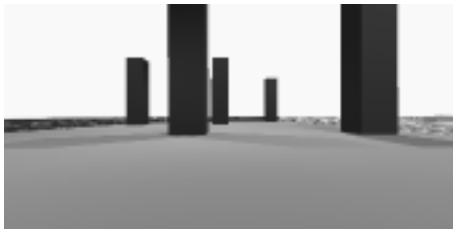
The PPO Reinforcement Learning algorithm requires the agent to be placed in a simulation environment similar to the evaluation environment to achieve good results. The agent will be placed in an arena with goal objects during training. In previous work [1] two different training regimes were used, Single-Goal-Training and Full-Map-Training. In Single-Goal-Training the training was stopped after completing the first goal or upon collision. In Full-Map-Training the training was 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 map. However the Full-Map-Training scenario is closer to the evaluation 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.



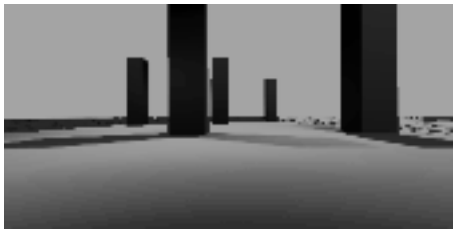
(a) Original Image



(b) Downsampled image



(c) greyscale



(d) histogram equalized image

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

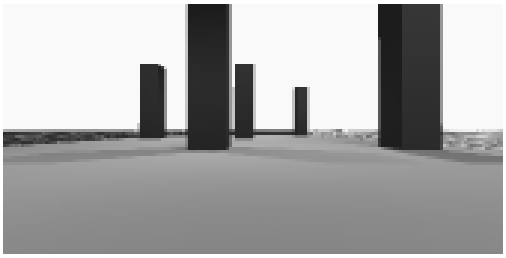
A third possible regime would be Full-Map-Training with randomized starting positions, this combines both approaches. The Single-Goal-Training is not strictly worse or better than Full-Map-Training. Therefore is not clear what training regime to chose for this thesis, therefore SGT, FMT and FMT with randomized starting positions will be compared during the experimentation phase.

Training Light Settings

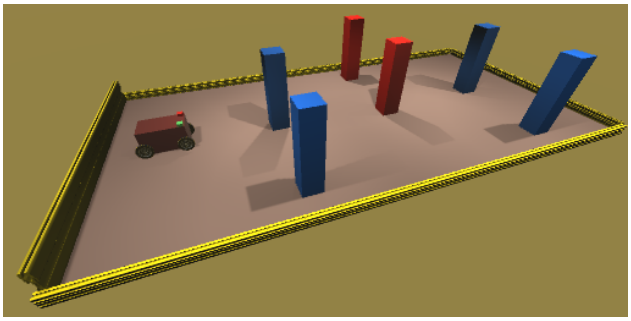
Since the agent utilizes a Convolutional Neural Network to be resilient towards changing light conditions it is also necessary to train the agent with varying light conditions, otherwise the adaptability of the CNN would not be fully utilized. This way the agent will be able to learn to generalize to different light conditions. The light conditions will be randomized for each training parcour. Training with fixed light settings could also provide interesting insights when comparing the results to the results of training with varying light settings. If there is enough time the agent will be trained with fixed light settings as well. The comparison would show if the varying light settings during training helps for the generalization to different light settings.

Data Augmentation

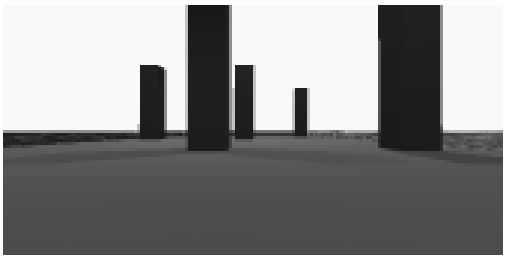
Convolutional Neural Networks require a lot of data to learn and generalize. Data augmentation is a technique to increase the amount of data available during training by applying transformations to the collected data. Collected data can be used to produce many more training examples by applying transformations such as rotation, translation, scaling, flipping and colour changes. In addition to providing a more diverse set of training data, this saves a lot of time since the new data points are not collected in simulation. [25] 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.



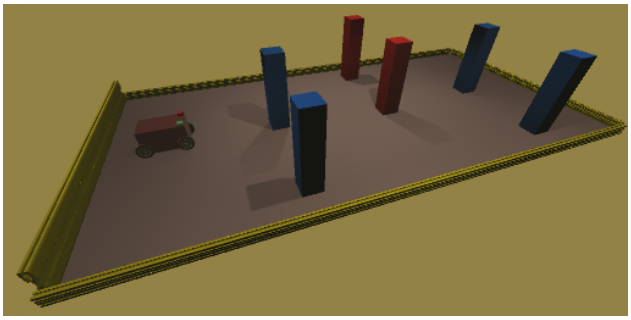
(a) Standard Lighting Agent POV



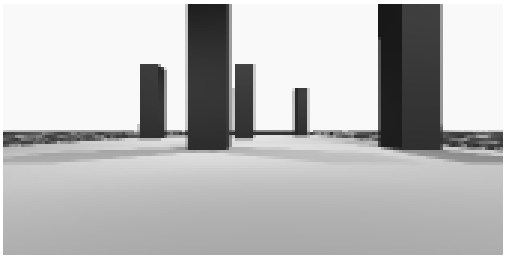
(b) Standard Lighting Arena



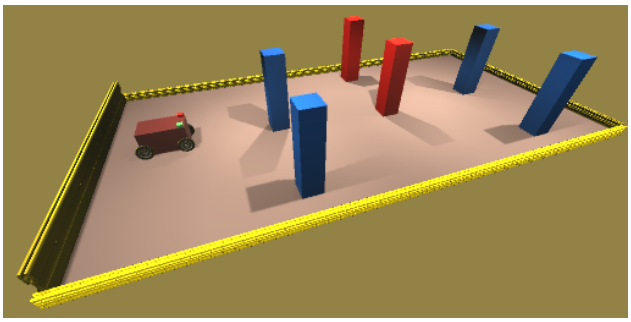
(c) Reduced Lighting Agent POV



(d) Reduced Lighting Arena

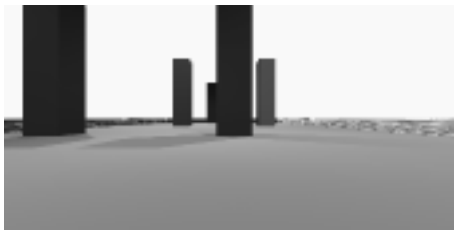


(e) Increased Lighting Agent POV

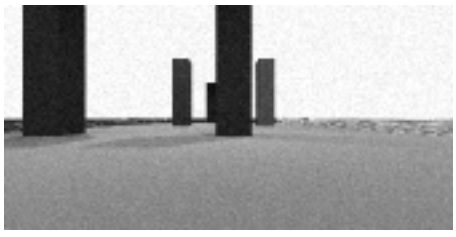


(f) Increased Lighting Arena

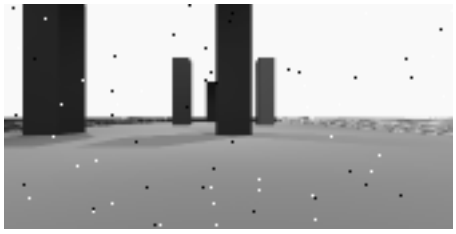
Abbildung 4.5.: Agent Camera POV and Arena Screenshots of the different light settings



(a) Original image



(b) Gaussian Noise Mean 0 Sigma 5



(c) salt-and-pepper noise

Abbildung 4.6.: Data augmentation examples

5. Experiments and Evaluation

5.1. Experiments

The proposed experiments will build on the scenarios from [1]. The experiments included three parcours with different difficulty levels 5.1 and conducted 3 different experiments with each trained agent. The first experiment was under optimal conditions with minimal changes from 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.

The experiment with minimal changes and changed lighting settings will be used to evaluate the agent developed in this thesis. 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.

The experiments under minimal changes will be used to judge if the agent is able to reliably solve all parcours and hence answer the first research question. Agents with and without memory capabilities will be compared to investigate the contributions of the memory mechanism.

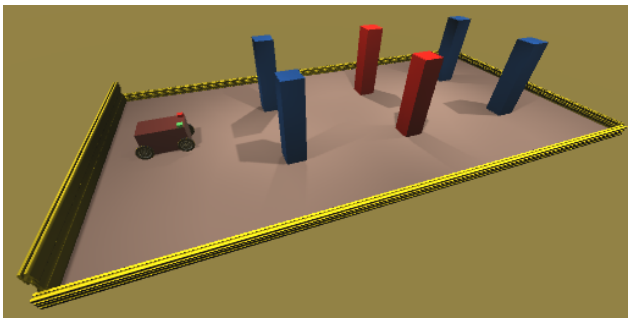
The experiment with varying lighting settings will be used to evaluate the agent's robustness towards changing light conditions and hence answer the second research question. Comparing the results of the agents with differently sized CNNs will provide insights into the required size and complexity of the CNNs. This will be used to judge if the agents can be transferred to real-life NVIDIA JetBots at the Scads.AI research facility.

The results of both experiments will be compared to the previous work's results to see if the CNN approach outperformed the engineered preprocessing pipeline.

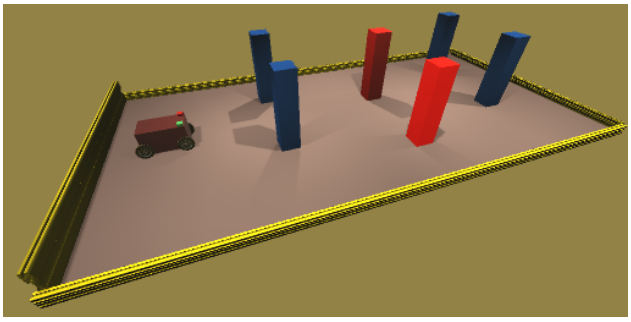
5.2. Evaluation

During the training and the final experiments, the agents are evaluated using the success rate, the average time needed to complete the map and the collision rate. The success rate is the percentage of episodes in which the agent successfully completed the map. These metrics were already used by [1] and measure the most important properties of the agent's behaviour.

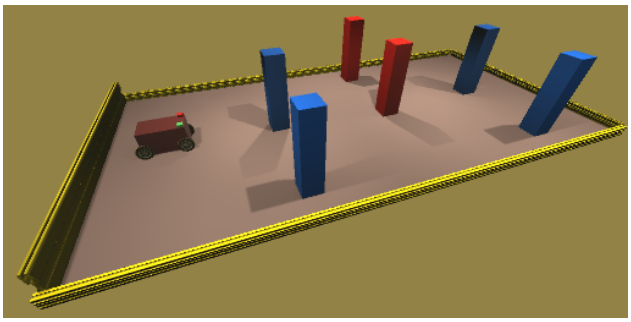
There will be additional metrics monitored during the training process to identify weak-points of the agent and erroneous behaviour. Monitoring the training process can provide insights into the agent's behaviour and help with choosing appropriate hyperparameters. These metrics could be the average cumulative 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.



(a) Easy



(b) Medium



(c) Hard

Abbildung 5.1.: Evaluation Tracks of different difficulties

6. Schedule

- 1.5 months - Literature research: Finding a suitable RL-training algorithm and training framework. Researching implementation details for the task of autonomous driving.
- 2 months - Implementation of the training algorithm. Including preliminary testing/evaluation.
- 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] Felipe Codevilla u. a. *End-to-end Driving via Conditional Imitation Learning*. 2018. arXiv: 1710.02410 [cs.RO].