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 and the practicability of using reinforcement learning for the training process. The reinforcement learning agent's task is to complete a parcour in a simulated environment without collisions. The thesis will build upon previous student's work at the Scads.AI by changing implementation and training details. The goal is to improve the agent's performance and evaluate the changes. The work will be compared to the results of the previous thesis [1].

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. 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 [2]. 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 [3].

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 [4]. This thesis aims to contribute to the research in this field by applying reinforcement learning to autonomous driving agents in a simulated environment. Different implementations will be tested and evaluated to investigate their contributions to the performance of the agent. This work builds upon the work of [1] and will use the same task and evaluation metrics.

2. Research Goals

The goal of this thesis is to contribute in the domain of autonomous driving by investigating the efficiency and contributions of different implementation details using reinforcement learning in a simulated environment. The thesis will build on the work of [1] and review the utilised algorithms and implementation details. Implementation changes will be proposed based on research in the field of reinforcement learning and autonomous driving. The contributions of these implementation details will be evaluated on their own and in combination with each other.

The self driving agent is trained and evaluated on a simulated parcour. Different parcours, lighting settings and motor-power 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.

Train to successfully complete all parcours

The trained agents from [1] were not able to reliably complete all the evaluated parcours, especially the parcours of higher difficulty levels. I will investigate the question: Is it possible to train an agent to reliably solve the parcours of all difficulty levels?

The agents [1] with memory mechanisms performed better for the more difficult parcours, although they were outperformed for the simple parcours. This behaviour is unexpected since memory mechanisms such as frame-stacking have shown to be very useful in reinforcement learning [5]. The agents trained in this thesis will also use a memory mechanism and will be compared to agents without memory mechanism to investigate the contributions of the memory mechanism.

Train end-to-end to handle varying light conditions using CNNs

The preceding work on the parcour employed autonomous driving agents which used a dedicated preprocessing pipeline. The pipeline was programmed to extract the coordinates of the goals/obstacles from the agent's camera. The agents were not able to complete the parcour under changing light conditions. The agents in this thesis will use Convolutional Neural Networks (CNN) to extract the relevant information directly from the camera images. I will investigate the question: Is it possible to use an end-to-end trained CNN to make the agent robust to changing light conditions?

The complexity of Convolutional Neural Networks might prevent them from being used in embedded environments. A possible future work is to transfer the trained autonomous driving agents to real life NVIDIA JetBots at the Scads.AI. Therefore the required size and complexity of the CNNs will be investigated. Is it possible to use a CNN which is small enough to be used in the JetBot?

3. Related Work

Since this thesis will employ reinforcement learning in the training of the autonomous driving agent it is important to review the current state of the art in reinforcement learning and autonomous driving. Some techniques of the researched works might not be applicable in this thesis due to the smaller scope and availible resources of this thesis, it is still important to review them to understand the current state of the art and to be able to utilise aspects of them. Simulation environments for reinforcement learning and self driving will also be reviewed, since they play a bit role in the state of the art of both fields and will be employed in this thesis. Finally, some papers about the use of reinforcement learning for 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. Most reinforcement learning algorithm formalize the problem using a state space and an action space and a reward function associated with state transitions. The RL algorithms are built to select an action in a given state and try to maximize the cumulative reward along the state transitions. This process of selecting an action is called the policy.

The classical version of the Q-learning RL algorithm keeps a table of state-action pairs and their associated quality values which are learned/computed during training. These Q-values are used by the algorithms' policy. The amount of state-action pairs can be very large and not feasible to compute in many cases, especially for environments with continuous state and action spaces.

To solve this problem neural networks have been used to approximate the Q-values, they take a representation of a state from the state space as input. This approach is called Deep Q-Learning, convolutional neural networks are often used [6]. The initial success of Deep Q-Learning has led to many improvements and variations of the algorithm, since then RL algorithms have proved to be very useful in a wide range of applications.

A mayor weakness of the initial Deep Q-learning algorithm was its stability during training, the parameter updates of the Q-learning agent's neural network can result in a collapse in performance. Proximal Policy Optimization (PPO) and other algorithms have been developed to improve the training process stability. The PPO [7] 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].

Another major improvement in the domain of reinforcement learning was the combination of neural networks with traditional planning and search algorithms, the most famous example for this is AlphaGo [8]. The search algorithms (typically Monte Carlo Tree Search) evaluate the possible actions at a given state. The neural networks are used for the evaluation of states and actions in the search algorithms. AlphaGo was developed for a 2 player deterministic game with a discrete

state and action space. Since then the combination of neural networks and search algorithms has also been used for all kinds of problems, for example single player continuous state and action spaces [9].

Convolutional Neural Network for Reinforcement Learning

As mentioned already convolutional neural network are often used in 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 pattern in the image such as for example edges and corners. The fully connected neural network analyses the results of the filters and makes the final prediction. CNNs are ideal for the use in Reinforcement Learning since Reinforcement Learning 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. CNNs typically do not take the raw camera/simulation images but rather preprocessed images, e.g. greyscaled images. Preprocessing steps are discussed in the Methods section 4.3.2. Furthermore data augmentation can be done during training, this is another kind of preprocessing. Data augmentation generates new samples from already collected ones by applying transformations to the samples, this can be used to increase the size of the training set and to make the agent more robust.

3.2. Self Driving

As mentioned before there has been a lot of progress in the domain of self driving in recent years, often driven by private enterprises that might not publish their findings in academic journals. Sophisticated self driving algorithms often consist of many components to achieve satisfying performance, Tesla FSD for example uses separate object detection, occupancy and planning components [3]. This thesis builds directly upon the work of [10], [11] and [1]. [10] 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 visual inputs that were preprocessed before feeding it to the neural network. [11] investigated the feasibility of transferring the agent to the real world, this research showcased many difficulties, most notably the object recognition part of the agent. [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 was fed preprocessed data similar to [10].

Another interesting approach to building self driving agents is using imitation learning [12], imitation learning agents learn to mimic the behaviour exhibited in the training set. The training set is usually generated by humans driving the car, this approach is rumored to be used by Tesla to train their self driving agents [3].

3.3. Simulation for Reinforcement Learning (and Self Driving)

Simulations play a huge role in reinforcement learning and the development of self driving agents. Simulations provide a 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. This allows for much faster 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, which are often not available in the real world such as perfect sensor data and object bounding boxes. Ground truths are used to evaluate the performance of the agent and can be used to train the agent. In addition the physics of the simulation can be modified, for example it is possible to increase the simulation speed.

Simulated environments, especially games have served as baselines for reinforcement learning algorithms. The most famous baseline are the atari games [6]. The need for easy use of reinforcement learning algorithms has led to the development and adoption of the Gynasium API [13], the Gymnasium API defines an interface that can be used to model tasks as reinforcement learning problems. This interface is called a Gym environment, an allows for easy testing of reinforcement learning algorithms on a wide range of tasks. Furthermore the Gymnasium API allows for easy comparison of different algorithms on the same task. A wide range of reinforcement learning frameworks support the Gymnasium API, for example Google's dopamine [14] and OpenAI's baselines [15].

Wrappers for Gymnasium environments are often used to facilitate the use of more advanced simulations for environments, such as for example the game engines Unity and Unreal or the physics simulator MuJoCo [16]. The complexity and interest in self driving has also led to the development of dedicated simulators, such as the Carla [17] and the AirSim [18] simulator. The Carla simulator provides researchers with useful features such as weather control and ground truths for object detection and segmentation.

There are also dedicated frameworks for reinforcement learning that directly integrate with game engines, such as the ML-Agents framework [19] for Unity.[1] used this framework to train the self driving agent.

3.4. Reinforcement Learning for Self Driving

[20] review the use of reinforcement learning for autonomous driving, they also describe many improvements for reinforcement learning algorithms that can improve the training stability and performance, such as for example reward shaping. In addition to published research papers there have been a lot of experiments, tutorials and demonstrations of self driving agents on YouTube and GitHub. The University of Tübingen published their full lecture series on Self-Driving Cars [21], the series also includes a section on reinforcement learning.

4. Methods

4.1. Task Description

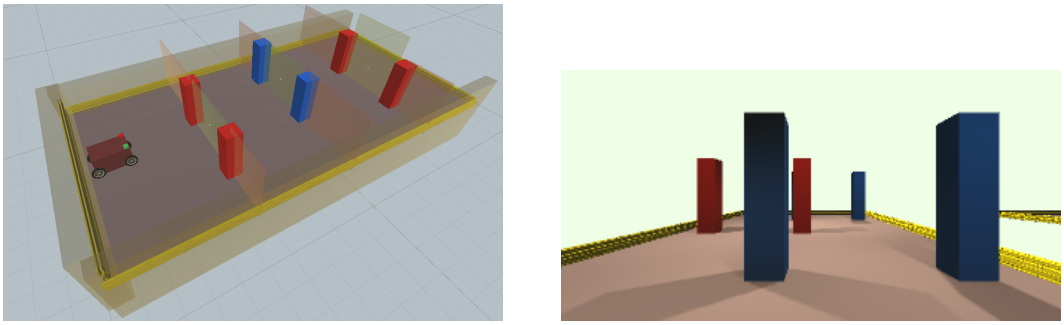
In this section I will be describing the task and the simulation environment as these two aspects are the foundation of this thesis and will not change over the course of the project. The task is to develop an agent using reinforcement learning that is able to complete a parcour in a simulated environment 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. At each timestep the agent uses a neural network to process an image from the environment 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 game engine, the game 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 algorithms is most commonly used for problems of this class and has already been successfully used in the investigated task [1]. The MuZero [9] could also be used, however this algorithm does require a lot more compute resources during training and inference. In this thesis the PPO algorithm will be used.

There are multiple approaches to training an agent in the Unity simulation environment as highlighted in the related works section, it is not yet decided which approach will be chosen, therefore I will quickly describe the possible scenarios and highlight their advantages and disavdantages.



(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

Unity and ML-Agents

Reinforcement learning agents can be trained directly in the Unity simulation environment using the ML-Agents framework [19]. This approach has the advantage of being very simple to implement and use, since the agent and simulation are in the same framework. The biggest disadvantage of this approach is that it might be difficult to implement some of the implementation details that might be used in this thesis such as the proposed changes for dealing with delayed rewards. The PPO algorithm by [19] was used in [1] to successfully train the agent on the investigated task.

Unity and separate reinforcement learning frameworks

There are many reinforcement learning frameworks publicly available that can be used to train agents in simulated environments, such as the OpenAI's baselines [15] and Google's dopamine [14]. These frameworks often support the Gymnasium API [13], wrapping the Unity simulation environment in a Gymnasium environment would allow for easy use of these frameworks. This approach has the advantage of being able to use the many features of these frameworks and makes it easy to change the training algorithms which might be necessary for the investigated implementation details. The disadvantage of this approach is that it might be difficult to wrap the Unity simulation environment in a Gymnasium environment, there already exist frameworks to help with this [22].

4.3. Investigated implementation details

The goal of this thesis is to answer two questions with a subitem each.

1. Is it possible to train an agent to reliably solve the parcours of all difficulty levels?
    - Are memory mechanisms necessary in achieving this?
2. Is it possible to use an end-to-end trained CNN to make the agent robust to changing light conditions?
    - Is it possible to use a CNN which is small enough to be used in the JetBot?

4.3.1. Improvements for training the agent

4.3.1.1. Reward shaping

TODO make the Reward function a seperate subsection? one subsubsection per xxxReward?

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 with reward shaping [20]. This practice was already employed by [1] by providing the agent with a reward proportional to its speed, this encourages the agent to drive faster and thus hopefully complete the parcour quicker. This thesis will investigate the use of a

$$R(s_t, a_t) = c_1 \cdot DistanceReward(s_t, a_t) + c_2 \cdot OrientationReward(s_t, a_t) + c_3 \cdot VelocityReward(s_t, a_t) + c_4 \cdot EventRe$$

$$OrientationReward(s_t, a_t) = S_C(NextGoalPosition - AgentPosition, agentDirection) \cdot \Delta T$$

$$DistanceReward(s_t, a_t) = \Delta distance(Agent, NextGoalPosition) \cdot \Delta T$$

$$VelocityReward(s_t, a_t) = v \cdot \Delta T$$

$$EventReward(s_t, a_t) = \begin{cases} 100, \text{completed the parcour} \\ 1, \text{passed a goal} \\ -1, \text{missed a goal} \\ -1, \text{collision with wall or obstacle} \\ -1, \text{timeout} \end{cases}$$

Abbildung 4.2.: Reward Function, S_C is the cosine similarity

$$SmoothedReward(s_t, a_t) = \sum_{i=0}^{n} \gamma^i \cdot Reward(s_{t+i}, a_{t+i})$$

Abbildung 4.3.: Reward Function that uses the reward of current timestep t and the next $n-1$ timesteps, γ is the discount factor (between 0 and 1)

reward proportional to the difference in distance to the next goal between timesteps, this should encourage the agent to drive towards the next goal and to drive faster in this direction.

TODO change paragraph to include orientation reward

4.3.1.2. Dealing with delayed rewards

Delayed rewards can be a big problem in reinforcement learning and make the training process difficult. Delayed rewards are rewards that are not obtained immediately after the responsible action is taken. In our environment an action a_n (e.g. turning right) may lead to a collision at state s_{n+x} which results in a negative reward for action a_{n+x}. The RL algorithm might fail learn to avoid the action a_n.

There are a multiple approaches to dealing with delayed rewards. These approaches use the reward at the current timestep and the near future. These approaches result in a more accurate and dense reward signal and can improve the training stability and performance of the agent. N-step bootstrapping uses the rewards at the current step and the next n steps [23]. A similar approach does not use the reward from the next n steps but rather the cumulative reward encountered in the next n seconds [24]. Due to the continuous nature of the environment this approach might be more suitable than N-step bootstrapping.

$$BootstrappedReward(s_t, a_t) = R(s_t, a_t) + \gamma R(s_{t+1}, a_{t+1}) + \gamma^2 R(s_{t+2}, a_{t+2}) + ... + \gamma^n R(s_{t+n}, a_{t+n})$$

Abbildung 4.4.: N-step bootstrapping reward function [23]

4.3.1.3. Time perception

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 [6] and discrete action spaces [8]. This allows the agent to perceive object movement, time and velocities [6]. In our case the agent needs a history since the next goal could leave the agent's current field of vision.

4.3.2. Improvements for Light setting robustness

4.3.2.1. Convolutional Neural Networks

The works by [11] 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 will investigate using convolutional neural networks instead of an object detection pipeline, 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 [11] 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 [5].

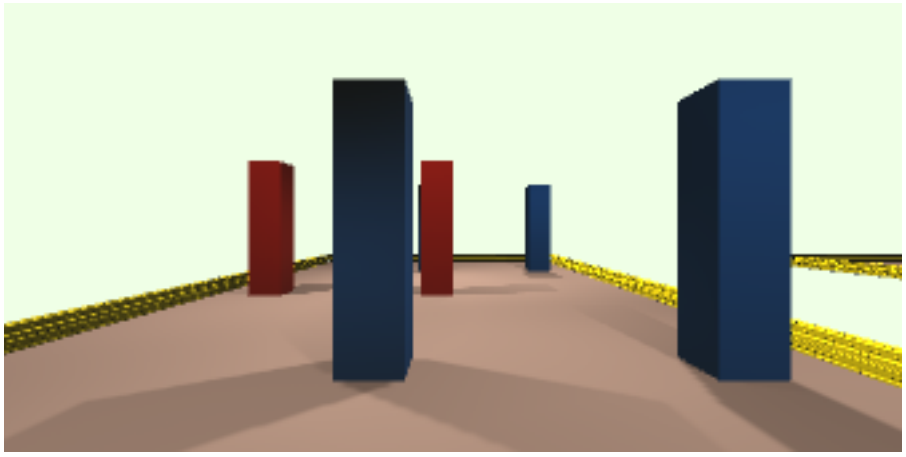
4.3.2.2. Feature reduction / preprocessing

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 [6] paper. These techniques will be used due to the similar complexity of our task and many atari games.

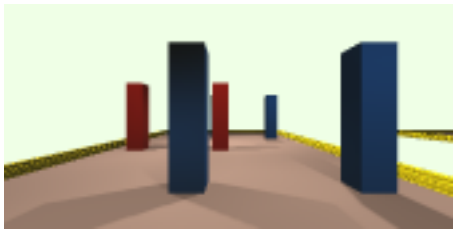
Downsampling reduces size of input space

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

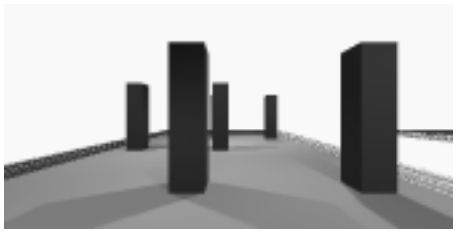
Rescaling pixel values to between 0 and 1 can help the neural network learn quicker



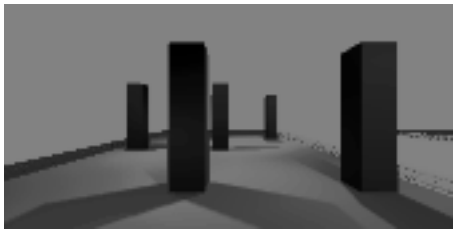
(a) Original Image



(b) Downsampled image



(c) greyscale



(d) histogram equalized image

Abbildung 4.5.: 4 Stages of preprocessing images for the CNN

4.3.2.3. 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 4.5 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.

4.4. 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.

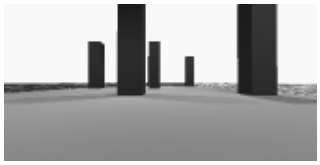
A third possible regime would be Full-Map-Training with randomized starting positions, this combines both approaches. The Single-Goal-Training is not strictly worse or better than Full-Map-Training. Therefore is not clear what training regime to chose for this thesis, therefore SGT, FMT and FMT with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

Training Light Settings

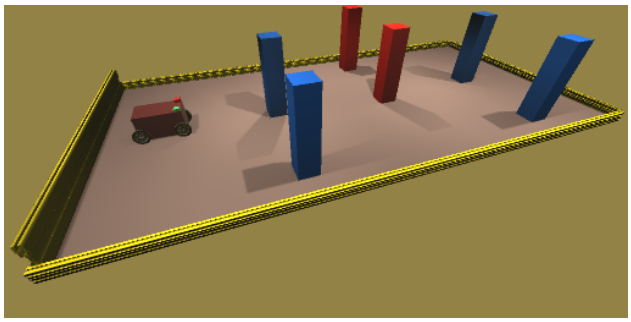
Since the agent utilizes a Convolutional Neural Network to be resilient towards changing light conditions it is also necessary to train the agent with varying light conditions, otherwise the adaptability of the CNN would not be fully utilized. This way the agent will be able to learn to generalize to 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

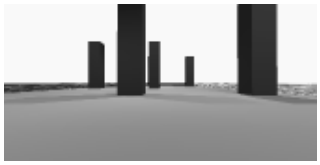
Convolutional Neural Networks require a lot of data to learn and generalize. Data augmentation is a technique to increase the amount of data available during training by applying transformations to the collected data. Collected data can be used to produce many more training examples by applying transformations 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



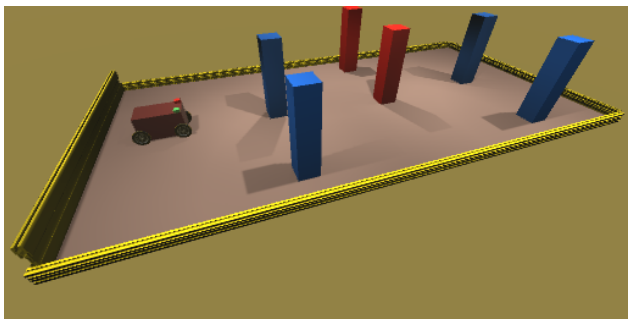
(a) Standard Lighting Agent POV



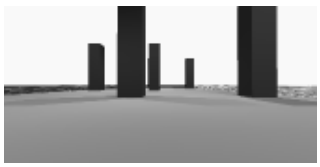
(b) Standard Lighting Arena



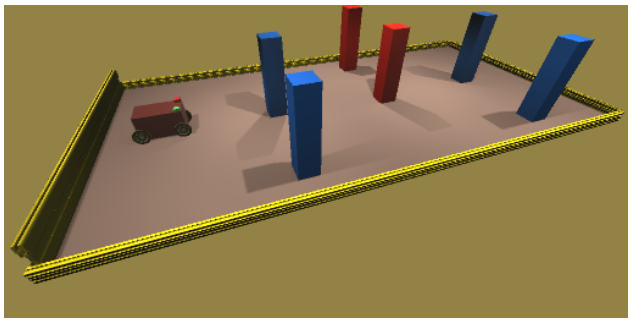
(c) Reduced Lighting Agent POV



(d) Reduced Lighting Arena

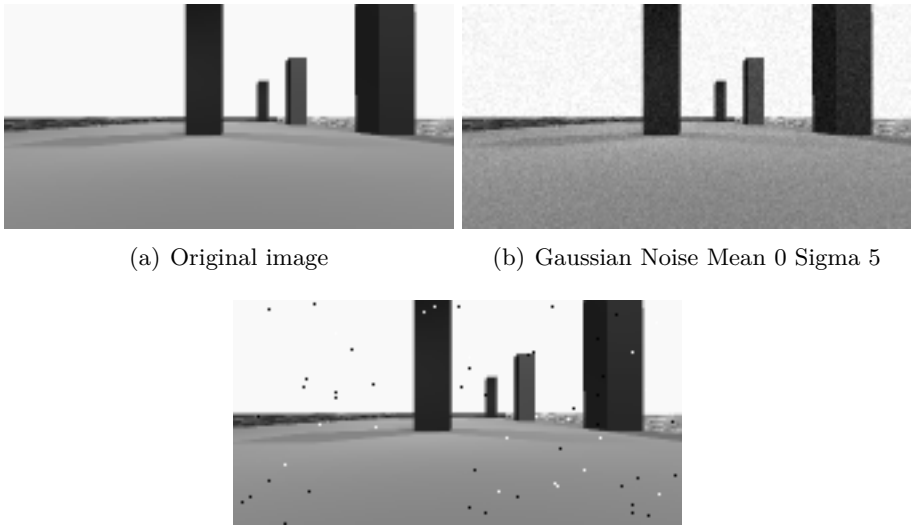


(e) Increased Lighting Agent POV



(f) Increased Lighting Arena

Abbildung 4.6.: Lighting Agent POV and Arena TODO real images wait for information by Maximilian about how lighting was done



(a) Original image (b) Gaussian Noise Mean 0 Sigma 5

(c) salt-and-pepper noise

Abbildung 4.7.: Data augmentation examples

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. 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.

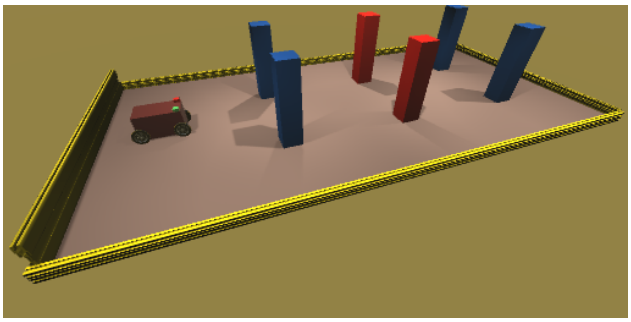
The experiment with varying lighting settings will be used to evaluate the agent's robustness towards changing light conditions and hence answer the second research question. Comparing the results of the agents with differently sized CNNs will provide insights into the required size and complexity of 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.

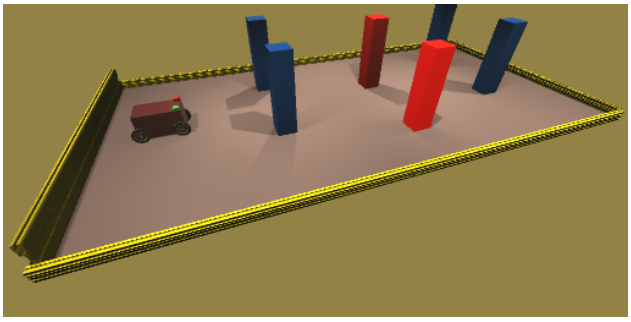
5.2. Evaluation

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.

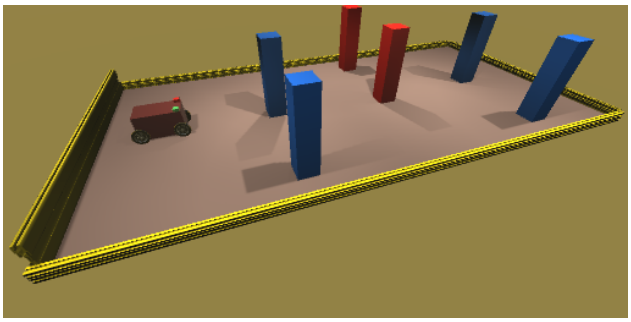
There will be additional metrics monitored during the training process to identify weak-points of the agent and erroneous behaviour. Monitoring the training process can provide insights into the agent's behaviour and help with choosing appropriate hyperparameters. These metrics could be the average cumulative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

[3] Think Autonomous. *Breakdown: How Tesla will transition from Modular to End-To-End Deep Learning*. 2023. URL: https://www.thinkautonomous.ai/blog/tesla-end-to-end-deep-learning/ (besucht am 15. 09. 2023).

[4] 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).

[5] 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.

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

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

[8] 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.

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

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

[12] Joonwoo Ahn, Minsoo Kim und Jaeheung Park. „Autonomous driving using imitation learning with look ahead point for semi structured environments“. In: *Scientific Reports* 12 (Dez. 2022), S. 21285. DOI: 10.1038/s41598-022-23546-6.

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

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

[15] 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.

[16] 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.

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

[18] Shital Shah u. a. „AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles“. In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[19] 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.

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

[21] Prof. Andreas Geiger. *Self-Driving Cars*. 2022. URL: https://www.youtube.com/watch?v=GYnlqiSqZiU&list=PL05umP7R6ij321zzKXK6XCQXAaaYjQbzr&index=13 (besucht am 23. 10. 2023).

[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] 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.

[24] Linesight. *Trackmania AI Learns To Drift and Beat Pros ? | Hockolicious*. Youtube. 2023. URL: https://youtube.com/clip/Ugkxfx5GF0CSNmsVP_s-JZN4RQMePaatG6Vn?si=wptMMB8ytxCX3op8.

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