Degree Thesis

Master of Science in Engineering, Computer Science and Engineering, 300 credits

Domain Transfer for End-to-end Reinforcement Learning

Computer Science and Enginering, 30 credits

Halmstad University, July 28, 2020
Anton Olsson, Felix Rosberg

HALMSTAD UNIVERSITY

ABSTRACT

In this master thesis project a LiDAR-based, depth image-based and semantic segmentation image-based reinforcement learning agent is investigated and compared for learning in simulation and performing in real-time. The project utilize the Deep Deterministic Policy Gradient architecture for learning continuous actions and was designed to control a RC car. One of the first project to deploy an agent in a real scenario after training in a similar simulation. The project demonstrated that with a proper reward function and by tuning driving parameters such as restricting steering, maximum velocity, minimum velocity and performing input data scaling a LiDAR-based agent could drive indefinitely on a simple but completely unseen track in real-time.

ACKNOWLEDGEMENTS

I, Anton Olsson, would like to thank my family for always supporting me during my studies at Halmstad University. I hope to one day repay you thousandfold.

I, Felix Rosberg, would like to thank my father and my siblings for supporting me during my studies at Halmstad University. In memory of my mother.

We would also as a group like to thank our supervisor Cristofer Englund for providing us with great discussion and feedback and always pushing us in the right direction.
To our teammates, thank you for all the advice, help and good times during this project. You are all awesome. And finally to all our friends during our five years at Halmstad University, you mean the world.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DDPG Deep Deterministic Policy Gradient

C-DDPG Convolutional Deep Deterministic Policy Gradient

AE Autoencoder

GAN Generative Adversial Network

IoU Intersection over Union

ROS Robot Operating System

RC Radio Controlled

RGB Red Blue Green

RGB-D Red Blue Green Depth

TORCS The Open Racing Car Simulator

DQN Deep Q Network

DDAC Deep Deterministic Actor Critic

MSE Mean Squared Error

A3C Asynchronous Advantage Actor-Critic

SV Supervised Learning

B-RL Baseline Reinforcement Learning

EMDQN Episodic Memory Deep Q Network

TD3 Twin Delayed Deep Deterministic Policy Gradient

Part I

INTRODUCTION AND LITERATURE REVIEW

1 INTRODUCTION

This master thesis was conducted at Halmstad University from the end of autumn of 2019 and until summer 2020. The thesis work throughout, consisted of research and engineering in the field of reinforcement learning. The main contribution of the thesis is to further improve the field of reinforcement learning for autonomous driving. Especially addressing the notorious problems with reinforcement learning in real-time and finally test it in real-time using an RC-car. Some problems related to reinforcement learning involves long training times, bridging the gap between simulations and reality and implementing capabilities to learn new or similar tasks. Having a reinforcement learning model acting as both a fast and a safe driver will be crusial for the future of autonomous vehicles, however it requires smart methods to transfer agents developed through simulation to real-life and real-time, as real-time training is expensive. This project aim to control a actual real-time RC car driving in a small scale race track. The output from the reinforcement model is velocity and steering angle. A PID regulator is used in both the simulation and real-time to apply the change in control. The system solution is detailed in Fig. 1.

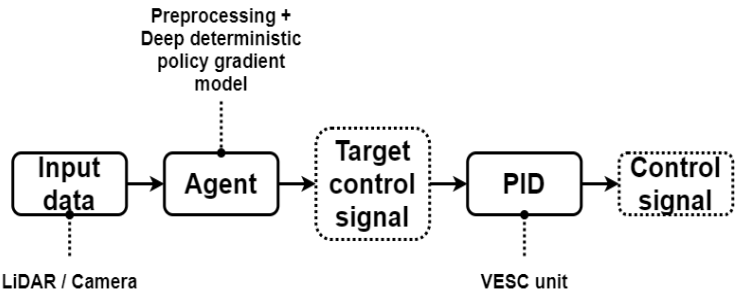


Figure 1: Flowchart detailing the interaction between hardware and software during control of the RC car.

Background

The vehicle engineered within this project will participate in the F1/tenth competition [1]. F1/tenth is an autonomous RC car competition where teams compete against each other using RC cars in both time trials and head-to-head battles. The thesis will use F1/tenth as a test-bed for reinforcement learning agents that are trained in simulation and

1 http://www.f1tenth.org/

transferred to the real domain. F1/tenth was planned to be held in Berlin, Germany July 2020. The competition would have teams from all over the world. During the time-trial, the car would run laps on a race track to receive the fastest time possible and during the head-to-head competition, the car would race against another opponent. These are the two scenarios in which reinforcement learning will have to learn to control velocity and steering angle.
Due to covid-19s impact on the year of 2020 our master thesis will no longer compete in the competition of F1/tenth as travelling and big crowds is out of the picture.
Reinforcement learning is a promising method for learning control in highly dynamic scenarios. However reinforcement learning is dependent on an environment to explore and learn in. Most of the time, this environment is represented in a simulation, as training in real-time is not feasible. For a race car to learn, it requires exploration which usually lead to collision and off-track driving in the early stages of learning. It is self-evident that doing this on a real road is not feasible. Simulation is required, but models learned in simulation suffer from the noise of real-time and other differences. Efforts to try and bridge the gap between simulation and real-time is needed.

The team

This project is a part of a bigger team effort, collaborating with two other master thesis groups. One of these explores AI for path search/-trajectory generation and the other explores interesting research possibilities for perception using the LIDAR and/or the RGB-D camera.
The path search and trajectory generation thesis consists of Jonatan Jönsson and Felix Stenbäck. Since their group also is involved with the trajectory of the car we were able to test our solutions against each other. The other thesis is by Harald Lilja and he is focused on the perception of the car. Both trajectory generation and reinforcement learning group could use his work as input to our models.

1.1 PROBLEM FORMULATION

Simulation

The models will be trained in a simulated environment. Different algorithms will be tested so that we can test and evaluate them against each other. When the best one has been found, the work to transfer the simulation to the real world will continue. Approaches will contain testing agents on lidar, depth- and camera-data. One method that was explored includes segmented images during the training of the reinforcement learning agent instead of using original images [1]. Improvements and novel approaches could be explored to improve training in simulation and adaptation in real-time.

Simulation to real world

How to best transfer the learning from the simulations to the real world? Currently, there are interesting approaches that use image translation. Work has been done to translate virtual images to segmented images to realistic images, they try to bridge the gap between simulation and reality by using the translation of virtual images to realistic images for training [2]. However, we will explore using segmented images during the training of the model instead of the high dimensional images. We will also explore what datatype from segmented images, depth-data or lidar-data in simulation, that is best used for domain transfer for reinforcement learning.

Real world racing

The novelty of our research will be tested and evaluated by driving capabilities in a completely new real-time surrounding after the agent has been trained in a simulation. The aim is to develop a car, based solely on data collected from simulation, that can maneuver within a track in a new domain.
Since the competition in Berlin is cancelled, the plan is to compete against the other team-members in a Time-trial on several different tracks.

Research questions and novelty

- Can an agent learn using segmented (or translated image) and adapt in real-time using same class scene parsing for simulated and real images?
- Can an agent learn on depth data and adapt to the real-time scenario?
- Can an agent learn on LiDAR data and adapt to the real-time scenario?

Budget

The thesis shares a budget with two other projects. The budget will go to the test-bed in the form of sensors and car. The money needed for the tickets to Berlin and accommodation for the first week had also been acquired. This has been provided by Halmstad University. Extra sponsorship has been acquired by the company HMS [2].

2 https://www.hms-networks.com/sv/hem

Risks

It can not be assumed everything will go smooth, especially when the goal is to adapt our agent to real-time. There are a few risks to consider and prepare for.

Table 1: Risk analysis presenting risks, general expected probability and consequences.

Severity	**Likely**	**Occasionally**	**Rare**
Negligible	-	-	Difference in driving surface
Minor	Noisy data	-	-
Serious	-	Loss of data	Long training time
Severe	Poor domain transfer	-	No data

We will have several different data risks to work with. The most likely of them is noisy data. The sensors will provide us with data, hopefully with low signal to noise ratio. However, there will be noise and we can not neglect that and we will have to handle it accordingly. Loss of data will happen occasionally and that will affect our model. To handle this we will have to make sure that the model can work even though some data will be lost. Sometimes there will be no data at all, this can happen if the sensors stop working. The solution to this is trouble-shooting and documentation so that it does not happen again.
Another rare risk that can occur is when we run our car on a different surface than it has trained on. This will probably only happen during competition, to counter-act this we will have to test our car running on different surfaces before the final competition.
The biggest risk and the one that will most likely happen is how to handle the difference between simulation and real-time. This can have great consequences such as making the car act significantly different from the simulation. We hope to solve this by making the shift between the domains as small as possible.
The final risk that we state at the beginning of the thesis is that the training time might be very long. The consequence will be that the car most likely can not drive autonomously in a real-time competition. We hope to handle this by having a model with short training time thanks to the preprocessed inputs.

2 LITERATURE REVIEW

This chapter consists of a literature review that put the work into its academic context and can help with the implementation of the reinforcement learning model. The papers include topics such as end-to-end reinforcement learning, methods to increase sample efficiency, generative adversarial networks, imitation learning, and similar research areas.

2.1 END-TO-END DEEP REINFORCEMENT LEARNING FOR LANE KEEPING ASSIST

El Sallab et. al. [3] proposes different reinforcement learning methods for autonomous vehicles. The main contribution of their research is using raw sensor inputs and then using the proposed reinforcement learning methods to output driving actions. These actions can be either discrete or continuous, this is addressed by introducing algorithms to tackle each case. The different algorithms are Deep Q Networks (DQN) and Deep Deterministic Actor Critic (DDAC).

Deep Q Networks

The problem to solve using Deep Q Networks is formulating the Q-function, this becomes harder as the number of states grows and is impossible when the states are continuous. Solving the Q-function will therefor be done using the approximation of the parameters Q(*s*,*a*,*w*), more precisely finding the best set of the *w*-parameter. Using a deep neural network which minimizes the mean squared error (MSE) of the Q-values is used ie. finding the best setting for the weights:

$$l(w) = E[(r + \gamma argmax_{a'} Q_t(s', a', w') - Q_t(s, a, w))^2] \quad (1)$$

$$J(w) = max_w l(w) \quad (2)$$

In equation 1 s, a and w correspond to sequence, action and weights. The equation uses a discount factor γ and r is the reward. The use of apostrophe simply means that it is the next predicted value. The optimization can then be solved using simple gradient based methods.

Deep Deterministic Actor Critic

To learn continuous action values, the authors of the paper [3] suggest an algorithm that learns two functions. The actor and the critic. The actor provides policy mapping from state to action while the critic evaluates the action taken. The critic is the same function as the Q-function. Two neural networks Q(*s*,*a*,*w*) and $\pi(s, u)$ are used to learn the two functions. This is because the objective is differentiable with respect to the Q-function and the policy. The gradient of the critic is found in the same way as for the Deep Q Network while the gradient of the policy function is found in the following way:

$$\frac{\delta J}{\delta u} = \frac{\delta Q}{\delta a}|_{a=\pi(s,u)} \frac{\delta \pi(s, u)}{\delta u} \tag{3}$$

Equation 3 uses two networks, Q based on sequence, action and weights and π uses states and actions which corresponds to s and u in the formula.

Lane Keeping Assist

The first part of the system proposed in [3] is the data collection phase. This is done using several sensors, them being LIDAR, radar, and camera. A sensor fusion approach is then needed so that the Deep Neural Network can extract relevant features from the data such as velocity or position for the car. There need to be environment features extracted for other objects and their position or orientation. This is needed for the algorithm to correctly steer the car.
The authors started their reinforcement learning with simple Q-learning and an approximation function called tile coding. They stated that even though it showed promising results it has drawbacks such as having too high of a confidence level in one estimator for future Q-values. Therefore the authors continued their work with Deep Q-learning. However, this does not solve the problem of continuous actions.
For continuous actions Deep Deterministic Actor Critic is used, therefore the two algorithms are trained with the same goal in mind.

Results

To test their function they used The Open Racing Car Simulator (TORCS) with a plugin that provides car models, tracks and a physics engine. It also has obvious parts such as velocity, acceleration, and steering angle. Besides all that, it also provides information about the environment in the style of sensor inputs. Running the simulator with both DQN and DDAC on a track that holds both straight paths and curves showed how important it is to be able to do continuous actions. On

the straight part of the map both DQN and DDAC were able to stay on a straight path however on the curves, DDAC outperformed DQN by being able to make smooth turns.
The authors proposed four termination criteria, 1) No Termination Criterion, 2) Out of track, 3) Stuck and 4) Out of track with Stuck. This means that the car can freely explore the map as long as it doesn't fulfill one of these criteria which will result in a negative reward. Each time a termination criterion was fulfilled TORCS had to be terminated.
The authors also concluded that by increasing the amount of termination criterion increased the training time needed for convergence. Consequently, they also showed that DDAC was the better algorithm as it handled curves far better.

Insight

The paper gave insight on how important continuous actions will be for a reinforcement learning model for an autonomous vehicle. As the car we will use also will drive on a race track with curves. It also introduced a termination criterion something that can be used for training of the car as the car is assumed to be on the racetrack while driving and therefor it should not have to learn how to drive in one of the situations that appear while in one of the termination criterion.

2.2 VIRTUAL TO REAL REINFORCEMENT LEARNING FOR AUTONOMOUS DRIVING

Virtual to Real Reinforcement Learning for Autonomous Driving (VRRKAD) is a concept that touches on the ideas of being able to transfer a reinforcement learning model that is trained in simulation to real-time for cars [2]. There are many promising possibilities to learn about driving policies with the help of reinforcement learning. However training autonomous vehicles with reinforcement learning in a real environment is costly because it involves learning from mistakes, e.g colliding. VRRKAD proposes an image translation network to translate the simulated image that the agent is training on into realistic images. Experiments show that reinforcement trained driving policies using this method can adapt to real data.

Realistic Translation Network

One problem with trying to construct a direct image-to-image translation using virtual and real images is that there are no paired images of the two domains. The authors in [2] address this problem by using scene parsing representation. The proposed network to do the scene

representation is SegNet [4]. The authors in [2] used two image translation networks. One takes the virtual frames as input and translates them into segmented images and the second takes segmented frames and translates them into realistic images. The two image translation networks consist of two conditional generative adversarial networks (GANs). The difference between a GAN and a conditional GAN is that the former learns the mapping function from a random noise vector z to an output image $s : G : z \rightarrow s$, while the latter learns the mapping function from a random noise vector z and an image x to an output image $s : G : \{x, z\} \rightarrow s$. Where s often is in a different domain than x, like in the case of translating a segmentation to a real image. The conditional GAN is expressed as in the following equation:

$$\begin{aligned} &\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,s\sim Pdata(x,s)}[\log D(x, s)] \\ &+ \mathbb{E}_{x\sim Pdata(x),z\sim p_z(z)}[\log(1 - D(x, G(x, z))] \end{aligned} \tag{4}$$

Equation 4 G denotes the generator the tries to minimize the objective and D denotes the adversarial discriminator that counters G to maximize the objective and $\mathbb{E}$ denotes the expected value. L1 loss regularization is used to suppress blurring. The L1 loss regularization is expressed as:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,s\sim Pdata(x,s),z\sim p_z(z)}[||s - G(x, z)||_4] \tag{5}$$

This yields an overall representation of the conditional GAN with regularization weight λ:

$$G* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda\mathcal{L}_{L1}(G) \tag{6}$$

Reinforcement Learning for Training an Autonomous Vehicle

Mnih et. al. [5] used a conventional reinforcement learning method called Asynchronous Advantage Actor-Critic (A3C). The implementation details for this algorithm can be found in the paper Asynchronous Methods for Deep Reinforcement Learning [5]. The reward function defined by the authors is particularly interesting as it is designed to encourage the agent to drive fast and avoid a collision. The reward function is defined as:

$$r_t = \begin{cases} (v_t * \cos(\alpha) - \text{dist}^{(t)}_{\text{center}}) * \beta & \text{no collision,} \\ \gamma & \text{collision.} \end{cases} \tag{7}$$

In equation 7 v_t denotes the speed (in m/s) of the agent at time step t, α denotes the angle between the speed of the agent and the tangent

line of the track, and $dist_{center}^{(t)}$) denotes the distance between the center of the agent and the middle of the track. β and γ are constant parameters than are determined before training. In the case of this paper, the authors used $\beta = 0.006$ and $\gamma = -0.0025$.

Experiments

Two sets of experiments were done to evaluate the [5] method. The first set of experiments consists of virtual to real reinforcement learning on real-world driving data. The virtual to real translation network was first trained and then fed with the virtual image to translate those into realistic images. The translated images were then fed into the A3C to train a driving policy. The newly trained A3C was then evaluated on real-world data by looking at the steering angle prediction accuracy. The method was further evaluated by comparing a supervised learning approach to the real-world data and a baseline reinforcement learning model trained directly on virtual frames. The simulation environment used is TORCS [6].

The second set of experiments consists of transfer learning in virtual driving environments. Three agents were trained. First, an A3C agent directly on a TORCS track called Cg-track2. This agent is expected to perform the best on Cg-track2. The second agent was trained on a TORCS tracked called E-track1 and evaluated on the Cg-track2. The authors point out that the visual appearance of Cg-track2 and E-track1 are different. The third agent was trained with domain randomization using a method similar to as described in the paper CAD2RL: Real Single-Image Flight without a Single Real Image [7]. This was done on ten different simulated tracks and then finally evaluated on Cg-track2.

Results

Mnih et. al. [5] presents four kinds of results. Segmentation results, image translation results, reinforcement learning results, and transfer learning results. As described, the virtual scenes and the real scenes look vastly different. The scene parsing segmentation shows clearly similar images for both domains. The translation network yielded realistic-looking images. The virtual images were described as appearing darker than the translated images. The real data set did contain images from a sunny day so this result makes sense.

As mentioned above, reinforcement learning was evaluated on real data. The three methods compared were their proposed method, supervised learning (SV) and baseline reinforcement learning (B-RL). As seen in Table 2, supervised learning performs best. However, out

of the two reinforcement learning methods, the author's method performs significantly better than baseline reinforcement learning.

Table 2: Action prediction accuracy for the three methods [2].

Accuracy	Authors	B-RL	SV
Real data	43.40 %	28.33%	53.60%

Insight

This paper touches on one important challenge for reinforcement learning, namely domain transfer from simulation to real time. As mentioned in many cases, especially for autonomous driving, training in real time in a real environment is not feasible because learning comes from trail and error. And trail and error in real time is costly. One way to bridge this gap is to make hyper realistic simulators, however this is costly as well. For end-to-end reinforcement learning our project will take the methods and ideas from this paper. However some new novel ideas will be introduced. One idea is to instead of translating into realistic images in simulation we directly train the agent on segmented images. One inspiration for this is how human thinks, we are capable of determining scenarios as similar despite huge differences in the environments. We are capable of learning to drive in a city and instantly being able to drive in a forest environment. Another important aspect that will be introduced is testing in an actual real environment. The agent will be trained in simulations and tested in a real environment using an autonomous RC car. As mentioned in the background, this project was planned to participate in the F1/tenth competition. This would have provided a step in the direction of feasible and scalable end-to-end reinforcement learning.

2.3 IMITATING DRIVER BEHAVIOR WITH GENERATIVE ADVERSARIAL NETWORKS

Kuefler et. al. [8] uses imitation learning with the help of Generative Adversarial Networks to best learn how to imitate driver behavior. Imitation learning uses data recorded by some expert, often a human to best learn a policy that best imitates said expert. The policies learned can be expressed with neural networks with less parameters than that of a rule-based method. Inverse reinforcement learning assumes that the expert follows an optimal policy with an unknown reward function, if that reward function can be extracted then reinforcement learning can be used to find a policy that behaves in the same way as the expert. This is however computationally expensive, therefor solutions that improve this such as GAIL is of great impor-

tance [9]. The authors apply GAIL to model highway driving behavior. The main contributions according the authors are their extension of GAIL for recurrent neural networks and their application of the models in a highway simulator. In their model maps raw LIDAR data and handpicked road features to continuous actions.
They present two ways of optimizing the policy needed for controlling the vehicle. The one of interest uses GAIL to learn a surrogate function directly from data to act as the reward function.
The data set used for their experiments consists of several vehicles running on a highway in California. The traffic density goes from uncongested to congested and there is a lot of traffic interaction with actions such as merging and overtaking.

Experiments and results

The authors run their experiments on the *rllab* reinforcement learning framework [10]. The simulations are run for 100 steps and are prematurely exited if the car drives backwards, crashes or drives off-road. All the experiments run with the same set of core features such as vehicle length and speed but also uses features from the LIDAR-like sensor mounted on the vehicle. Finally the authors also created three features which indicates whether or not the car has encountered a undesirable state such as a crash. All features are then fed in to the models. To be able to validate their proposed models they compared them to other baseline models. Then three different ways of validating was used, root-weighted square error, Kullback-Liebler Divergence and Emergent behaviour. Except for a hand-coded controller the GAIL models achieve the lowest collision and off-road driving rate.

Insight

This paper proposes an interesting approach of reinforcement learning for controlling autonomous vehicles. Using a GAIL model to best imitate expert driving. One of the biggest problems in reinforcement learning when it comes to autonomous vehicles is that there is no given way of knowing the reward function for driving. This paper proposes a possible solution by using GAIL to find a surrogate reward function from the data itself.

2.4 EPISODIC MEMORY DEEP Q-NETWORKS

As reinforcement learning has made its success and deep learning as well, it is natural that the two have combined to produce powerful reinforcement learning models. However one big drawback with reinforcement learning, and especially deep reinforcement learning is that it is sample inefficient [1]. It can take up to millions of steps for

a reinforcement learning agent to achieve great results. In Lin et. al. [1] an algorithm inspired by biology introduces episodic memory for deep q-learning. Experiments evaluated on Atari games show that the agent can learn a successful policy in 1/5 of the interactions with the environment.

Episodic Control

The algorithm consist of two parts, one being the deep q network (DQN) part which is described a little further in the review of End-to-End Deep Reinforcement Learning for Lane Keeping Assist [3]. The other is the episodic memory part. The paper has proposed to use episodic control (EC) to increase data efficiency and learn good policies in high dimensions games using model free episodic control (MFEC). Methods that uses EC to approximate action values has also been introduced, that average future returns. Neural episodic control (NEC) is another method that utilizes differentiable neural dictionary to store slow-changing keys and fast-updating values and retrieves useful values by context-based lookup for action selection. MFEC and NEC are referred to as table-based algorithms and are considered to lack generalization. Inspired by how humans make decision using both our reflexes and the memory, the authors proposes that the slow optimization of DQN and lack of generalization of EC can complement each other.

Episodic Memory Deep Q-Networks

For episodic memory deep q-network learning (EMDQN), EC is used to accelerate the learning of DQN. The authors want to address three aspects of DQN. Slow reward propagation, single learning model and sample inefficiency. EMDQN simulates the striatum (reflex) and the hippocampus (memory). EMDQN simulates striatum to provide an inference target denoted as S and simulates hippocampus to provide a memory target denoted as H. A new proposed loss function is introduced as:

$$L = \alpha(Q_\theta - S)^2 + \beta(Q_\theta - H)^2 \tag{8}$$

In the above equation (eq. 8) Q_θ denotes the value function and acts as the decision system for the agent. Where as α and β denotes two appropriate weights.

$$S(s_t, a_t) = r_t + \gamma max_{a'} Q_\theta(s_{t+1}, a') \tag{9}$$

$$H(s_t, a_t) = max_i R_i(s_t, a_t), i \in \{1, 2, ..., E\} \tag{10}$$

Above equations expresses the inference target (eq. 9) and the memory target (eq. 10). E represent amount of episodes and $R_i(s, a)$ represent future return when taking action a in state s in the i-th episode. As mentioned above, the memory is a growing table for each action-state pair, (a, s, r). Where as γ denotes the discount factor, r_t reward at time t. The object function is represented as the equation below (eq. 11). Here $\lambda = \frac{\beta}{\alpha}$ represents the relative weights between H and S. D denotes a mini-batch of experiences.

$$\min_\theta \sum_{s_i, a_i, r_i, s_{i+1} \in D} [(Q_\theta(s_i, a_i) - S(s_i, a_i))^2 + \lambda(Q_\theta(s_i, a_i) - H(s_i, a_i))^2] \tag{11}$$

Experiments

Here the evaluation of [1] will be kept brief. The EMDQN algorithm was evaluated by comparing with other popular reinforcement learning algorithms. Including DQN, MFEC, NEC and double DQN (DDQN). This were done over 40 million frames in 57 different Atari games. DQN and DDQN was evaluated at 200 million frames. The used measurement is human-normalized score [11]. This score provides a summary statistic over several games using equation 12. For the mentioned equation, $score_{human}$ denotes the score of a human, $score_{agent}$ denotes the score of the agent and $score_{random}$ denotes the score of a random agent. As seen in Table 3, EMDQN provides a significant improvement performance.

$$score_{normalized} = \frac{score_{agent} - score_{random}}{score_{human} - score_{random}} \tag{12}$$

Table 3: Mean and median human-normalized scores at 40 Million frames over 57 Atari games [1].

	Mean	Median
DQN(40M)	151.2%	52.7%
MFEC(40M)	142.2%	61.9%
NEC(40M)	144.8%	83.3%
EMDQN(40M)	**528.4%**	**92.8%**
DQN(200M)	227.9%	79.1%
DDQN(200M)	330.3%	114.7%

Insight

Lin et. al. [1] addresses one of the biggest problems in reinforcement learning, data sample inefficiency. The authors demonstrate a state-of-the-art method, EMDQN, to increase learning for the agent. Their algorithm provides a reduced training time to 1/5 of previous training time in complex Atari games. There are some relevant and interesting novel research to explore to build upon their method. As mentioned above the weight between memory and inference target λ is a fixed constant and could be gated to act dynamically and probably improve the performance further. However for our project, the interesting part is using episodic memory to increase learning capabilities of our RC car.

2.5 ADDRESSING FUNCTION APPROXIMATION ERROR IN ACTOR-CRITIC METHODS

This paper addresses the function approximation error that can occur in common Actor-Critic methods [12]. The algorithm proposed in the paper builds on double Q-learning and delays the update of values to reduce the error that occurs. It can be seen as an improved version of the Deep Deterministic Actor-Critic mentioned in Section 2.1. This has a documented increase in performance.

Experiments and results

To test their algorithm Twin Delayed Deep Deterministic policy gradient algorithm (TD3) the authors run the algorithm on several Gym-environments and compared their returned reward values to other state-of-the-art algorithms. In Fig. 2 displays the results. The figure is taken from Addressing Function Approximation Error in Actor-Critic Methods [12].

Insight

The paper addresses the approximation error that can occur in actor-critic models which is something we want to avoid. By introducing a delay in policy updates and the inclusion of the twin, the overall learning speed and performance of the model can be improved greatly for tasks with continuous actions.

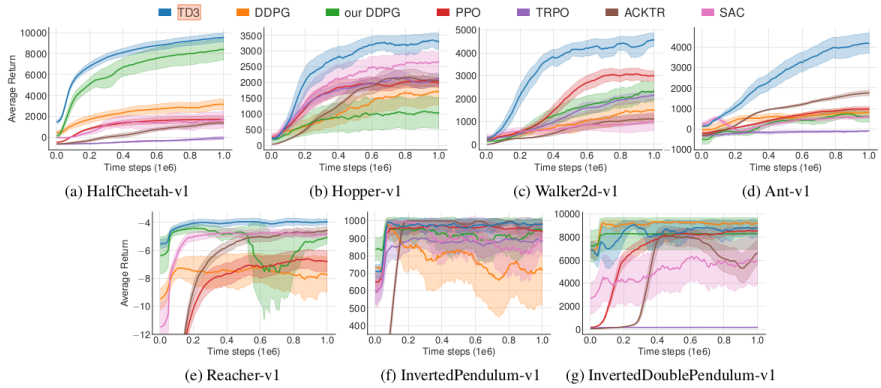


Figure 2: Figure taken from Addressing Function Approximation Error in Actor-Critic Methods [12] and displays the reward values. The reader can see that the proposed TD3-algorithm outperform many of the other state-of-the-art algorithms.

2.6 CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

In Lillicrap et. al. [13] the authors addressed the need for continuous actions by leveraging success in deep Q-learning (DQN) and actor critic methods. The research is motivated by limits in DQN. Though DQN has been successful in handling high dimensionality states and provide human level performance in many tasks, it only can do so for discrete and low dimensional action spaces. As reality has high degrees of freedom, high dimensional action spaces and continuous action spaces, DQN fails to approach these tasks. The suggested method to handle this problem would be the deep deterministic policy gradient (DDPG) which combines the strength of DQN and actor critic methods. Using one actor network, one critic network, one target actor network and one target critic network.

Experiments and results

The DDPG was tested in several different tasks and domains, including a driving scenario using TORCS. The DDPG experiments was setup with a learning rate of 10^{-4} for the actor, a learning rate of 10^{-3} for the critic, a discount factor of $\gamma = 0.99$ and a soft update of the target networks using $\tau = 0.001$. The learning rate decides the step size that the optimizer takes when looking for a global minimum. The discount factor decides how much the agent cares about rewards in the future. A discount factor of $\gamma = 0$ would mean the agent only cared about immediate rewards. Soft update is done by having the target network slowly update with below equation (Eq. 13). For eq. 13, θ represents the weights of the critic network (denoted as Q) and the actor network (denoted as μ). As the name suggest

is this method deterministic. To introduce exploration during training Ornstein-Uhlenbeck noise is added to the actions. A stochastic process that is temporal dependant and acts as a modified random walk.

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{aligned} \tag{13}$$

The DDPG was tested with both fully connected layers and convolutional layers for different tasks. One relevant experiment was done in the TORCS simulator to add control to a racing car. Using a reward function of $v * cos(\theta)$ and -1 for collision. Here v denotes velocity, motivating the agent to drive fast and θ denotes the angle of the car in respect to the roads direction, motivating the agent to keep steering to a minimum.

Insight

The configuration of the DDPG used in [13] will be implemented and used. Differences in the layers may occur. The DDPG will act as a baseline model to compare to a DPPG using our applied theories. This include training on segmented images, filtered image and latent information using a state representation learning autoencoder. This provides a robust baseline to both compare performance during training and behaviour in the real-time domain.

Part II

METHODOLOGY AND EXPERIMENTS

This part consists of Chapter 3, describing our approach to the project and Chapter 4, presents the experiments. The mthodology to answer the research questions is detailed here. The steps taken to construct and preprocess data for a LiDAR-based agent, a depth image-based agent and a semantic segmentation-based agent is included. Chapter 4 contains the experiments we have conducted during the project and we also describe how we evaluated our results from the experiments.

3 METHOD

3.1 TOOLS

This section will briefly describe the software tools that are used, the hardware that the RC car utilizes and the simulator used. The master thesis is heavily focused on the deep learning aspect of the task but the hardware and where to find documentation of the hardware will be discussed as well.

3.1.1 *Software*

All the reinforcement learning algorithms are based on deep learning networks. Therefore, this thesis will utilize libraries such as TensorFlow [14] and Keras [15] for constructing the neural networks required by algorithms such as deep deterministic policy gradient.

Python used with ROS [16] will provide the framework of doing the necessary calculations and sending the control signal for both the simulated car and the real-time car. Communication within ROS and the simulation tools for ROS is done with a subscriber/publisher system. Different topics can be created where parts of the system may either publish data and information to the topic or subscribe on the topic to gather data and information. For example, the module responsible of gathering LiDAR or camera data, publish the data to a topic of a given name. Then for making observations of the environment, one can subscribe to this same topic to asynchronous receive data from the system. In the agent part of Fig. 1 is where the software will be run. Detailed flowchart for each agent (LiDAR-based, depth image-based and semantic segmentation-based) related to its research question can be found in Fig. 14, Fig. 13, and Fig. 12.

3.1.2 *Hardware*

The car in the simulation and real-time is equipped with a LiDAR with 270 degrees field of view and a depth camera that can provide RGB images along with a depth point cloud. LiDAR will be used for early assessment of reward functions and chosen reinforcement learning architecture as well as one method to compare to. The end goal is to compare and discuss the behaviour of the different agents

train on LiDAR data, segmented images and depth data.

For actually controlling the car there is a few other components used. Central computing component used is a Nvidia Tx2 Jetson along with an Orbitty Carrier board. For PID regulator a VESC unit was used to bridge the agent output (Action values in Fig. 4a) and controlling the velocity and steering angle (Fig. 1). Detailed build and installation documentation can found at F1tenths build page[1].

3.1.3 *Simulator*

The simulation framework used for ROS and training the agent is Gazebo [2]. Gazebo provides a physics engine for the car to act in. The way we use Gazebo is by running ROS-launch nodes to spawn a world containing our different models. These models will range from maps, robot and obstacles. These models are either created by us from scratch in the form of map-models for our vehicle to operate and altered vehicle-models created by MIT. Why we alter the vehicle-model is to create a vehicle that closely resembles the one that we use on the real racetracks. We can then during training use these models to alter the starting point of the vehicle, which map it will spawn in or spawn different objects at different locations such as boxes around it will have to avoid.

3.2 STATE REPRESENTATION LEARNING

To provide a driving agent that can learn based on images and possibly current internal state as velocity and steering angle to produce end-to-end control signals, state representation learning is investigated. The idea builds on using low dimension and filtered images to train in simulation to bridge the gap between simulation and real-time driving. To improve information gain and efficiency representation learning will be explored [17]. Recently, a new activation function named swish has been introduced and showed promising results in boosting performance slightly in most tasks compared to relu [18]. For this project we are going to explore its capabilities to retain information when autoencoding states compared to an autoencoder trained traditionally with relu activation function.

Since our theory is that segmented or depth images will provide a way for the agent to both learn faster and recognise objects in real-time easier, data from both domains could be combined to provide improved robustness in real-time scenarios. This approach is however not necessary for LiDAR data as the input vector is a lot smaller

1 https://f1tenth.org/build.html
2 http://gazebosim.org/

and is not as computationaly heavy to map to actions compared to images.

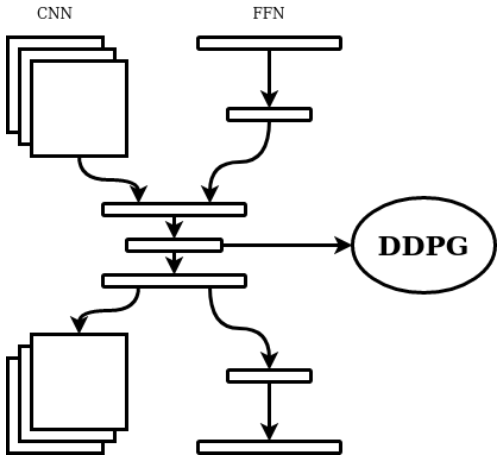


Figure 3: Basic flowchart of state representation autoencoder.

3.3 DEEP DETERMINISTIC POLICY GRADIENT

The project deploys the deep deterministic policy gradient algorithm (DDPG) for reinforcement learning [13]. DDPG can handle the continuous action space that our agent will act in. Being able to output actions in a continuous fashion is an important aspect of autonomous driving as described in 2.1. To improve convergence and sample efficiency during training a memory will be added. Episodic memory was discussed in 2.4, and our system will deploy a replay-buffer to sample state, reward, action and next state from batches in each learning step. The hyper-parameter configuration will follow that of [13], with the exception of network structure. More details is discussed in the experiment chapter.

The DPPG leverage the strengths in deep Q-learning and actor critic methods. Utilizing an actor network μ for mapping state to action values and a critic network Q to estimate the value of those actions as in actor critic method and a target network for actor network μ' and the critic network Q' as in Q-learning. The target network is slowly tracking the learned networks using soft update as in eq. 14. τ denotes a value of how much to update from each network and θ denotes the weights for each network. In our case $\tau = 0.001$. The target networks act as time-delayed versions of their learned counterparts.

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{aligned} \tag{14}$$

The use of target networks greatly increase the stability during the learning phase. This is because when calculating the loss L for the critic network (eq. 16) one uses the state s, next state s' and it depends on the target critic network as the targets is calculated with it (eq. 15). For the equation below, γ denotes the discount factor, a the action taken and N the mini-batch size sampled from the replay buffer. Q' represents the expected future reward, produced by the target critic network. Where as r denotes the immediate reward received.

$$y(r, s') = r + \gamma Q'(s', \mu'(s')) \tag{15}$$

$$L = \frac{1}{N}\sum (Q(s, a) - y(r, s'))^2 \tag{16}$$

The Critic network can now be updated with gradient descent using the loss L. The Actor network is updated by calculating the value of an action with the Critic network as the equation below (eq. 17) using gradient descent. Our model uses the Adam optimizer [19] for all four networks.

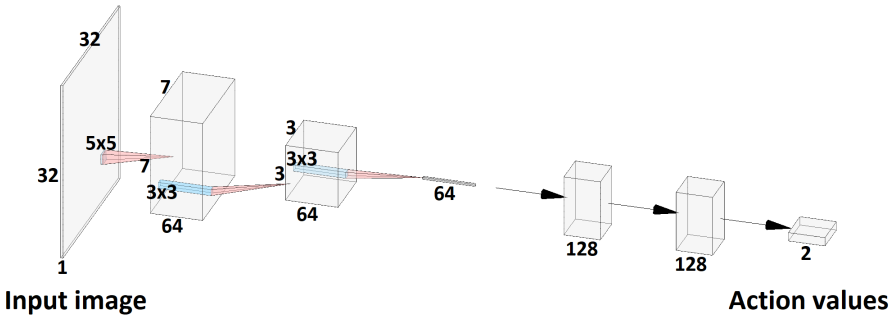
$$\frac{1}{N}\sum Q(s, \mu(s)) \tag{17}$$

3.3.1 *Convolutional Deep Deterministic Policy Gradient*

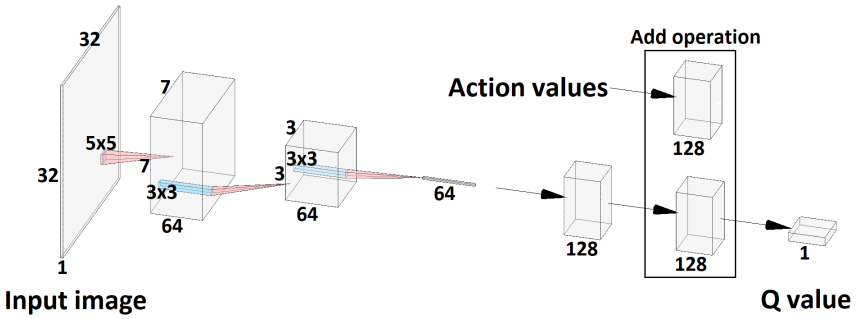
For a single stage network, the DDPG can be constructed with convolutional layers within the Actor and the Critic networks. The virtual to real paper mention at Chapter 2.2 used a single stage deep Q network with three convolutions. A convolutional DDPG (C-DDPG) was constructed and to be deployed using depth image data. Fig. 4a and Fig. 4b illustrates the full architecture of the C-DDPG. The first convolutional layer uses a stride of 4 and the two last convolutional layers uses a stride of 2. After each convolution layer a batch normalization is done.

Ornstein-Uhlenbeck Noise

The mentioned paper above [13] that introduces the DDPG applies the Ornstein-Uhlenbeck process [20] to include exploration during training. This exploration noise is needed because DDPG is a deterministic offline-policy algorithm, meaning it will always choose the same action for the same state. The noise is added to the action values that the agent generate during training to force the agent into new action-spaces. Equation 18 describes how the noise is generated. θ, σ and d_t denotes constants for our purpose and is chosen as 0.2, 0.15



(a) Actor network.



(b) Critic network.

Figure 4: Network architecture of the C-DDPGs actor and critic networks.

and 0.01 respectively, as in [13]. G denotes a Guassian normal distribution in shape of amount of actions. μ denotes the action policy produced when the model chooses an action. Where as t represents the timestep.

$$\aleph_{t+1} = \aleph_t + \theta(\mu - \aleph)d_t + \sigma\sqrt{d_t}G \tag{18}$$

3.4 REPLAY-BUFFER

If you were to train the DDPG only on the sample that the agent sees it would take a long time to reach any meaningful solution. To counter the sample inefficiency of the DDPG a replay buffer is used. This acts like a memory with a fixed size. A sample $S \in \{s, a, s', r\}$, where s is state, a is the action policy, s' is the next state and r is the reward, is observed and measured is placed into the replay buffer. If the maximum size of the replay buffer is reached, the oldest sample is dequeued. Then during every training iteration the agent samples a random mini batch of size N from the replay buffer and pass it through the network.

3.5 REWARD SHAPING

Arguably one of the most important aspect of reinforcement learning is the reward function. The reward function essentially decides how the agent should behave. For racing games a few common reward function used that utilize the velocity v, the angle α of the car in relation to the parallel line from the car and the walls and the distance d from the middle of the road, would be $v * \cos(\alpha)$ from the DDPG paper (2.6), $v * (\cos(\alpha) - d)$ [21]. However these papers has access to really accurate measurements of the distance to the middle of the road and the angle of the cars direction and tangent line along the roads middle, that their simulation provides. Gazebo will not have access to this in our case and the training has to rely on calculating relations with the environment using the equipped LiDAR. Above reward functions will be tested, however new ones will be developed, tested and compared.

3.6 IMAGE SEGMENTATION

As mentioned the thesis aim to investigate the possibilities of bridging the gap between simulation and real-time. One theory is that the same-class segmentation of simulated data and real-time data will provide a way of making simulation data and real-time data indistinguishable, inspired by the paper mentioned in Section 2.2. Instead of training on high dimensional and high resolution images the idea is that agent will be capable of learning to drive unseen tracks in real-time using preprocessed images. This is under the assumption that the two domains have some underlying strong similarity. This includes similar objects, scenes and world interaction. In this case this would be some kind of raised wall, ground, obstacle and the noise outside the circuit for the racing scenario.

To perform the image translation from simulated or real-time images to segmented space, a pix2pix general adversarial network (GAN) is deployed [22]. The pix2pix GAN has shown to predict accurate image translation by utilizing the strength of U-net [23] and a patch GAN discriminator. The generator uses the U-net architecture to generate translated images and the discriminator consist of a patch that instead of classifying the entire image as fake or real, classifies on pixel level. The patch-GAN-discriminator motivates the GAN to produce higher quality images compared to using only a binary classifier. Two GANs are trained for each 'domain transfer', one on the simulated images and one on the real-time images. Our current environments are simple, consisting of three scene classes for each images. These are 'wall', 'ground', 'obstacle' and everything else is regarded as noise. The GANs will be setup to translate the images to one channel for

each class. Fig. 5 how the semantic segmentation-based agent would preprocess image data to train on.

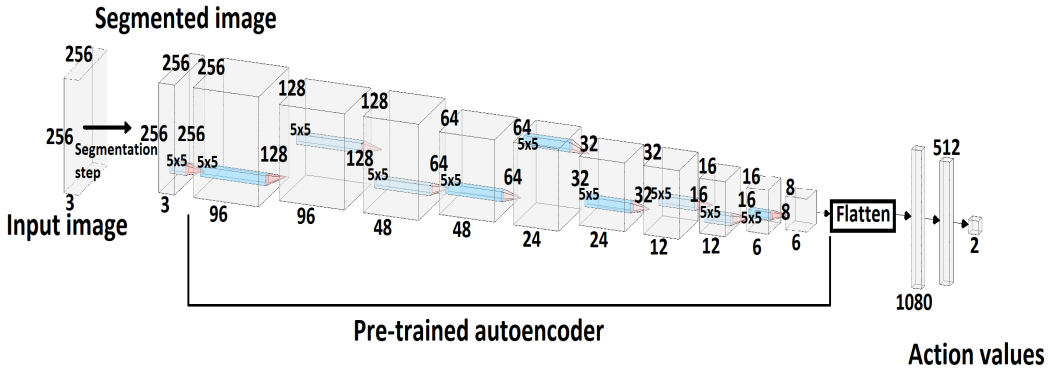


Figure 5: Detailed illustration of data flow for the whole system of the semantic segmentation-based agent. Note that max pooling by a factor of 2 is done in between each convolutional layer inside the encoder part of the autoencoder.

3.7 DATA AUGMENTATION

Along with the training of the segmentation GANs, data augmentation will be done. It is widely known that deep learning algorithms benefits with more data and that data augmentation is a way of introducing more data points from the data you have, thous improving the capabilities of different deep learning tasks [24]. The augmentation methods of interest in this project is flipping the image along the y axis, positive and negative rotation, cropping the image to only contain a smaller area and adding an additive Gaussian noise on the input.

4 EXPERIMENTS

This chapter will focus on the experiments that were run during the master thesis and preparation for the F1/tenth-competition. The experiments are structured in three main parts. Experiments in which the data needed are collected, then there is the part of training our proposed method in a simulation and lastly, there is the part of evaluating the method on a real track. The results of the experiments are presented in Chapter 5.

4.1 DATA COLLECTION

We have collected data from two domains for this project. The main bulk of our data has been collected in the simulated environment of Gazebo. The first part of the thesis solely focused on a map provided by the University of Virginia. This map can be seen in Fig. 6.

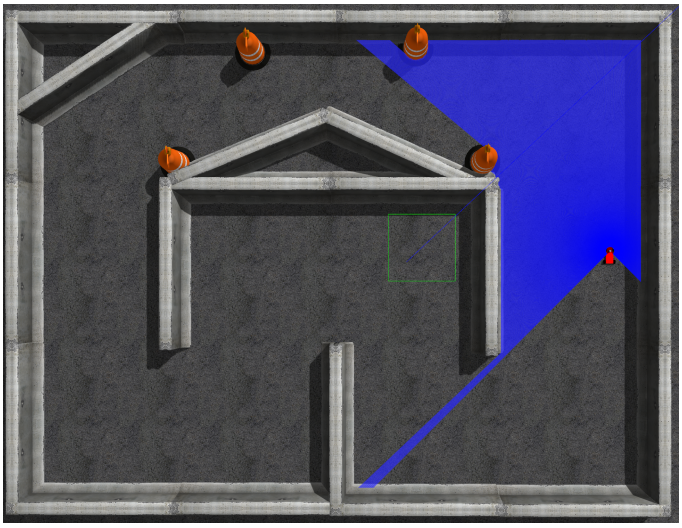


Figure 6: The track used for training of the reinforcement learning model in the simulator Gazebo. The track consists of sharp turns and object in the shape of cones.

This environment provides a good base to collect camera, lidar, and depth data. The environment shows some of the aspects needed for our car to perform in a racing environment, such as obstacles, sharp turns, and walls for the sensors to detect. We did, however, feel that this environment was missing other vital parts, such as long straights for the car to speed up in and other racing-related attributes. This lead us to create or find several different worlds for the car to operate in, many of them longer and with more race-like attributes. One such example can be seen in Fig. 7.

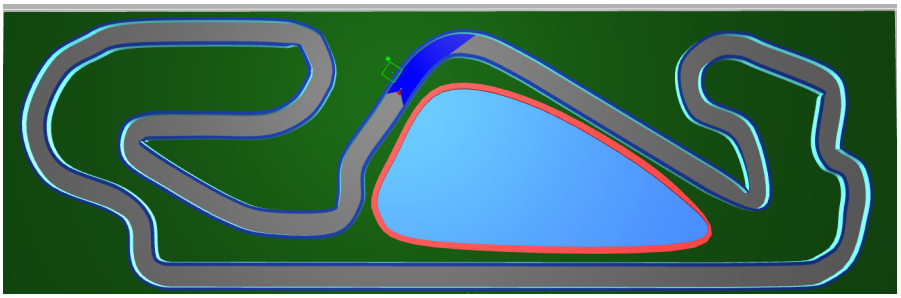


Figure 7: The track used for training of the reinforcement learning model in the simulator Gazebo. The track consists of sharp turns and object in the shape of cones.

The first iteration of Gazebo simulation had the camera close to the ground, this can be seen in Fig. 8.

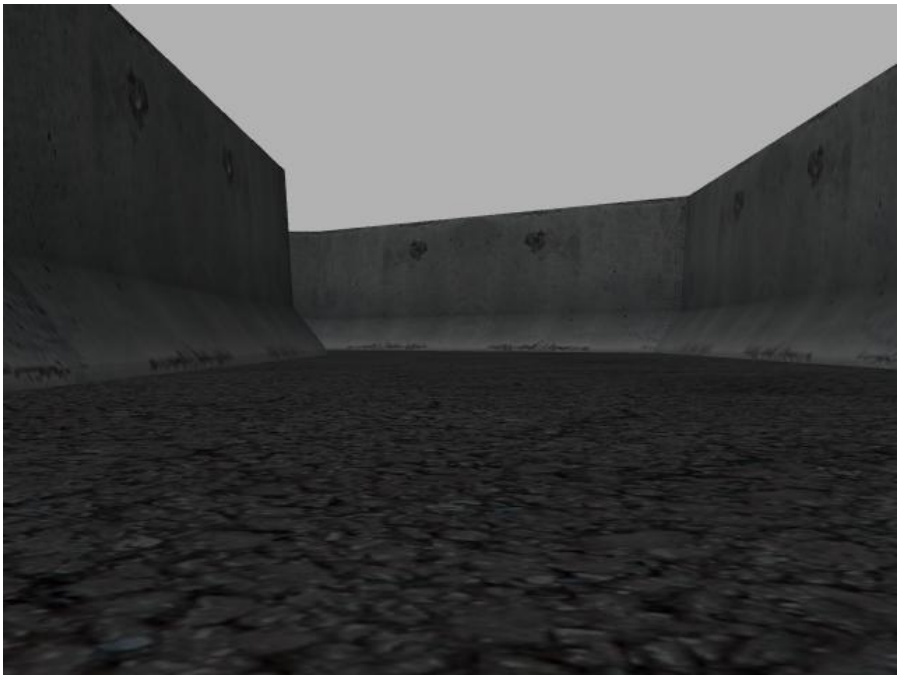


Figure 8: Camera data from the first iteration of Gazebo-experiments

In the second iteration of Gazebo-experiments we altered the model used for the car by raising the camera to a higher position, from 5 cm from the ground to 40 cm. We chose 40 cm as that was the absolute highest elevation possible, to maintain stability of the RC-car while racing. The camera was also tilted 25 °towards the ground to obtain a better sense of depth. The same was done for the real RC-car. We also altered the maps in where we ran camera-based experiments to have lower walls. The result of this can be found in Fig. 9.

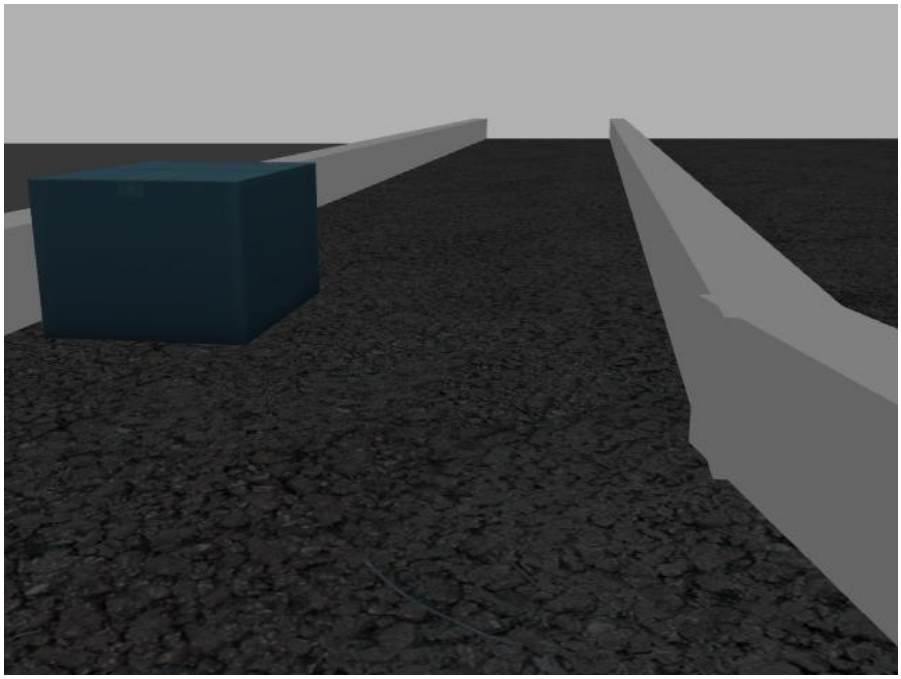


Figure 9: Camera data from the first iteration of Gazebo-experiments

For the experiments using depth-camera or lidar the setup was closer to the one seen in Fig. 8.
We also ran collection phases of real data, for the camera-based approach. We built many tracks in our local gymnasium to closely resemble the racetracks the car would later perform in. To build the tracks we used floorball rims to make walls for the lidar, mats to counter-act drifting and cardboard-boxes to act as objects. An example of this can be found in Fig. 10. For the second iteration of the collection phase we flipped the floorball-rims and applied a thin roll of cardboard. This created a setup closely resembling the one we used in the simulation. We also mounted the camera on a new height of 40 cm. An example of this can be seen in Fig. 11.

Figure 10: One of the test tracks in which the car has been tested. The walls are built using floorball rims. In this experiment the ground was covered with indoor mats to counter-act drifting and to resemble the ground used in the Gazebo-simulation.

Figure 11: One of the test tracks in which the car has been tested. The walls are built using floorball rims. In this experiment the ground was covered with indoor mats to counter-act drifting and to closely resemble the ground found in simulation. This is the second version with lower walls and the camera mounted higher.

4.2 TRAINING EXPERIMENTS

4.2.1 *Labelbox, GANs and Image Augmentation*

After the collection of camera-data, the next thing was to increase the amount of data and to label the different parts of the images. This is described in detail in Section 3.7. The GAN is fully trained before the training of DDPG is started.

4.2.2 *Autoencoders*

The autoencoders, described in Section 3.2, was trained to have the smallest possible dimensions while still containing the vital information found in the image. The training of the autoencoder is completed before the training of DDPG is started.

4.2.3 *Simulation*

Due to the nature of reinforcement learning the training has been done in a simulated environment, in our case in Gazebo. The experiments conducted in Gazebo for training ranged from testing basic control algorithms such as wall-following and follow-the-gap. The next step of training experiments was creating reinforcement learning algorithms for control. This was first on a basic level to achieve a better understanding for us to determine if the algorithms were good enough for our goal. We settled on DDPGs based on lidar and camera. For depth-data we used a C-DDPG.
The DDPG is described in detail in Section 3.3, with the C-DDPG described in Section 3.3.1.
The experiments conducted for the DDPGs were mainly focused on tuning the parameters, determining the correct reward function, and encouraging exploration. These three parts are connected and take up the majority of the training time. To evaluate the performance of the DDPGs the drive duration and the mean reward is used.
The flow of data during training for the camera based DDPG is illustrated in Fig. 12.

Figure 12: A flowchart describing the flow of data during training.

For an explanation for how the GAN processes the input see Section 3.6 and for the autoencoder see Section 3.2. Finally see Fig. 5 for the data flow for the semantic segmentation-DDPG.

The flow of data for the depth-based agent can be seen in Fig. 13. See Fig. 4a and Fig. 4b for the flow of data within the C-DDPG.

Figure 13: A flowchart describing the flow of data during training for the depth-based agent.

The flow of data for the lidar-based agent can be seen in Fig. 14. As the model suggest the flow of data for LiDar-data is straight forward, the input LiDar-data is fed raw to DDPG. The DDPG produces a control signal based on the flow found in Section 3.3.

Figure 14: A flowchart describing the flow of data during training for the lidar-based agent.

The camera-based DDPG was first trained in environments that can be seen in Fig 8 and then later in environments seen in Fig 9. The environment used for the lidar-based DDPG and C-DDPG is found in Fig. 7.

4.3 EVALUATION EXPERIMENTS

To evaluate the training done in Gazebo we transfer the learning from Gazebo and apply it to the real RC-car. All three approaches were tested for driving capabilities on a completely new track we built using floorball rims and mats.

4.4 INTERSECTION OVER UNION

To determine the performance of the GANs and the autoencoders, intersection over union (IoU) was used. It determines by how much a class overlaps with the ground truth. Equation 19 demonstrates how the IoU is calculated. TP denotes the true positives, FP the false positives and FN the false negatives for a pixel-wise prediction.

$$\mathrm{IoU} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{19}$$

Part III

RESULTS AND DISCUSSION

This part consists of Chapter 5 and Chapter 6. This is where the results of the project is presented. Discussion is focused on explaining the results.

5 RESULTS

This chapter will present the results achieved during the projects course. The results from reward shaping was attained during simulation part of the experiments are found in Section 4.2.3. The results from the different agents was attained during experiments explained in 4.2.3 and evaluated in 4.3. The results for image segmentation and state representation autoencoding was attained during experiments as explained in 4.2.1 and 4.2.2.

5.1 REWARD SHAPING

Different types of reward functions were explored. Initially $r = v * (\cos(\alpha) - d)$ was used in early testing, where v denotes velocity of the action, α, denotes the angle between the velocity and the direction of the road parallel to the walls and d denotes distance from middle of the road (path). The function provided decent behaviour as used with LiDAR in the early construction and testing of the DDPG. The one reward function that provided an acceptable behaviour, is the normalized reward and the final version of the function is described in the equation below (eq. 20). This reward function relies on measurements from the LiDAR, where it calculates the distance from the middle of the path, the angle between the RC cars velocity and the parallel line of the RC car and the wall and the angle from the direction of the RC and the longest measurement of a search space in front of the RC car. For this reward function, β denotes the angle between the RC direction and the longest measurement direction of a search space of a cone in front of the RC. α denotes the steering angle action, v the velocity action and d distance from the middle of the path.

$$r = \begin{cases} -1 & \text{if collision,} \\ -|\beta| & \text{if steering away from gap,} \\ v + \cos(\alpha) - 1 - d \end{cases} \tag{20}$$

The above reward function gave the agent a heavy negative penalty if the agent tries to steer away from the direction of the path. If the agent pass the check of going towards the right direction it rewards the agent based of its position in the path, its velocity and how much it is steering. During training the agent would for an ideal action gain a reward close to 1 if it keeps maximum velocity, does minimum steering necessary and stays in the middle. This reward function starts to drastically reduce the reward towards 0 for poor actions and

towards −1 if it starts steering away from the direction of the path. Poor actions in this case would be starting to steer towards the wall. The reward function was tested with an agent trained on segmented data, depth data and LiDAR data as well.

5.2 LIDAR-BASED AGENT

A regular DDPG trained one LiDAR points achieved a policy capable of navigating the tracks using the reward function mentioned above (eq. 20). This agent was trained on a track with a variety of smooth curved turns and several long halls. After every crash, the agent was restarted at a random check point somewhere on the track to introduce a varied situation to learn from. As seen in Fig. 15 and Fig. 16 the agent maintained a stable exploring during training. The behaviour of the drive duration going up and down is the effect of the training cycle with action noise. The agent explores both driving by what it has learned and pseudo-random steps in the environment. The peak at the end represent the agent driving without any action noise interfering with its actions, suggesting the agent has learn to navigate the track somewhat. The agent was tested in a real-time environment, in a small track with similar curves but completely unseen before by the agent. This track contained turns that did not match any from the training simulation. At first, on low speed, the agent managed to drive the track smoothly one lap before cutting the turns to sharp and crashing. After testing in the real-time environment it achieved consistent smooth driving by tuning the driving parameters for the test-bed. This include limiting the steering angle to 80% of possible angle, delaying action by a few milliseconds and scale the LiDAR data down slightly by a factor of 2.5 to trick the agent that he is slightly closer to the walls than it thinks. The agent now drove the new track smooth and fast with continuous actions. With the current simulation setup and reward scaling, the agent learned that always maximizing the velocity is the best course of action along with quick and heavy turns.

5.3 DEPTH IMAGE-BASED AGENT

This agent was trained with a C-DDPG on the same track as the LiDAR-based agent with the same reward function (eq. 20). The CDDPG takes the input depth image of shape (32, 32, 1) (Fig. 19). Full network architecture for both the critic and the actor is illustrated in chaper 3.3.1. The training was done with decaying action noise over time. However the training was continued once, introducing full action noise again (Fig. 17). Which in this case introduces exploring again, thus causing more crashes until the action noise decays again. Just as the LiDAR-based agent, the depth image-based agent

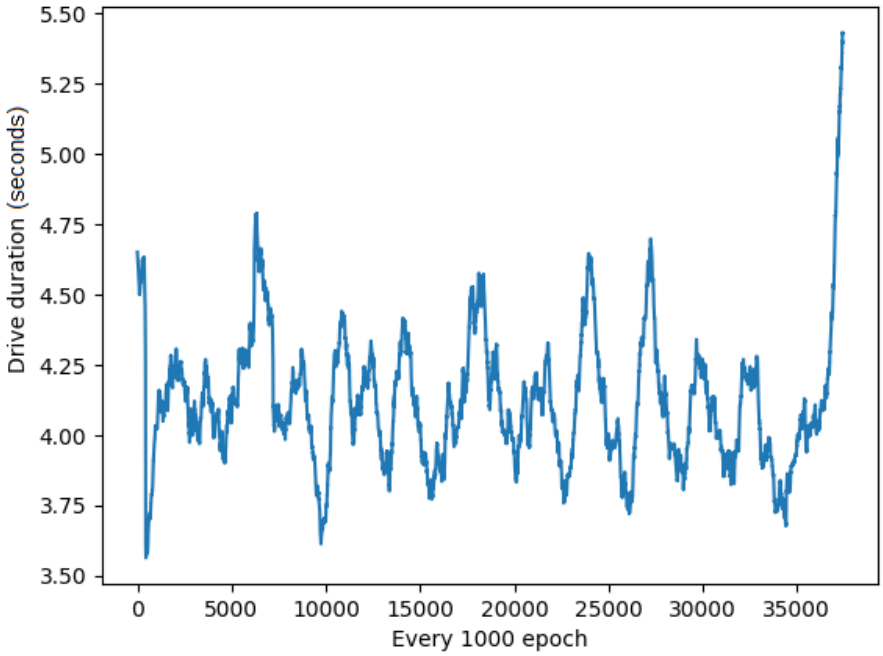


Figure 15: Graph showing the mean drive duration in seconds over a 1000 epoch window before a crash.

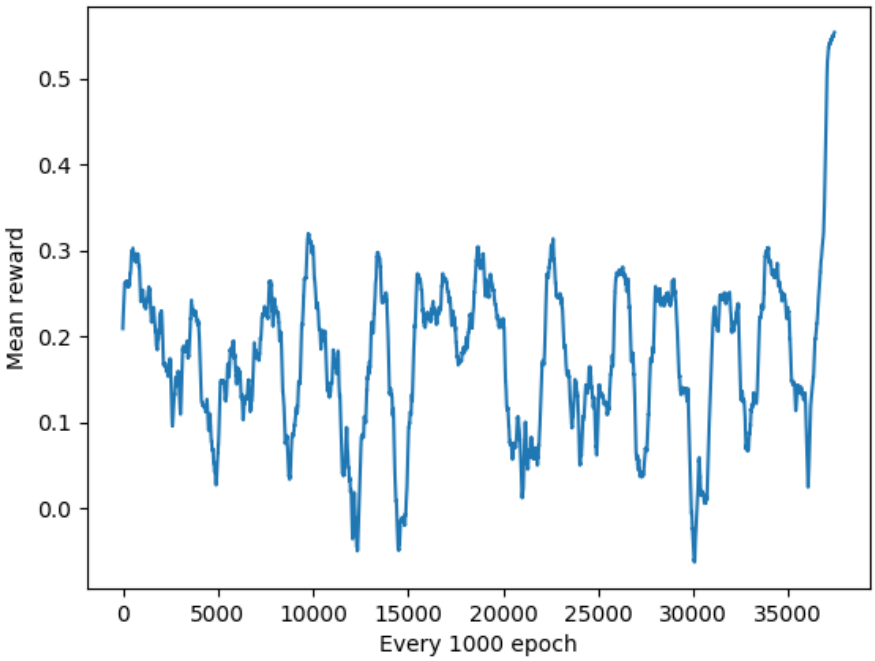


Figure 16: Graph showing the mean reward over 1000 epochs before a crash.

was trained on different checkpoints on the track after each collision. The agent seem to maintain an even stronger reward signal than the LiDAR-based agent just into a few thousand iterations into training as marked by the green area in Fig. 17. After continued training with decayed action noise, the agent started to receive a reduced reward signal over time, suggesting that the agent was getting over trained. The drive duration, as shown in Fig. 18, suggest further that it is over trained. It is worth noting that the two fully connected layers in the C-DDPG only contained 128 neuron each. The agent seem to still match similar reward signals as the LiDAR-based agent in the end. In the real-time scenario the depth image-based agent failed to understand the noisy input data that the real-time sensors provide.

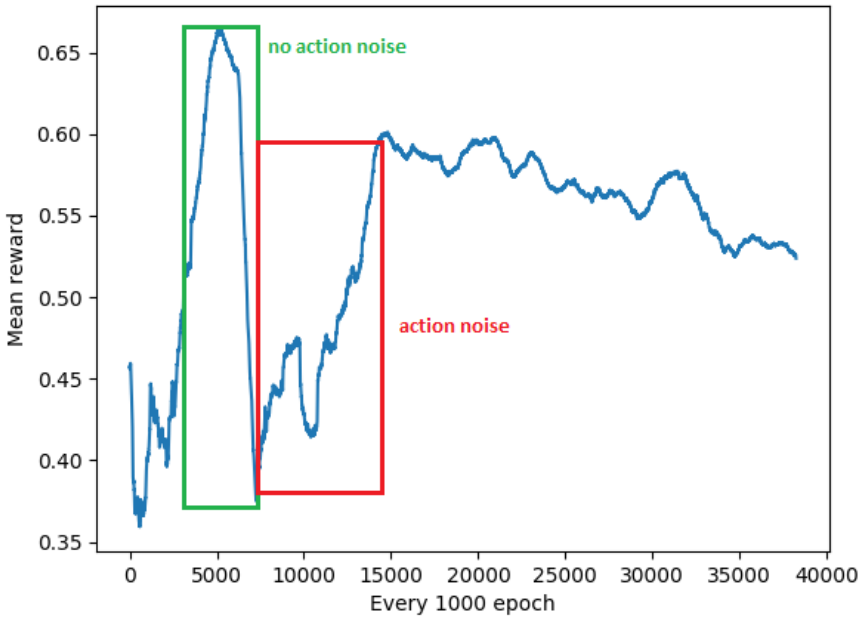


Figure 17: Graph showing the mean reward over a 1000 epoch window before a crash. Green area shows training iterations with none to very little action noise. Red area shows training iterations with action noise added again and decaying over time.

5.4 SEMANTIC SEGMENTATION-BASED AGENT

This section will be divided into four parts. First the results from the preprocessing steps used. That is segmentation, autoencoding the input data, the results from the ground level view agent setup and the birds eye view agent setup. The first one of the agents will focus on the simulation and test-bed results with high walls and the camera mounted low. The second agent will focus on the simulation and test-

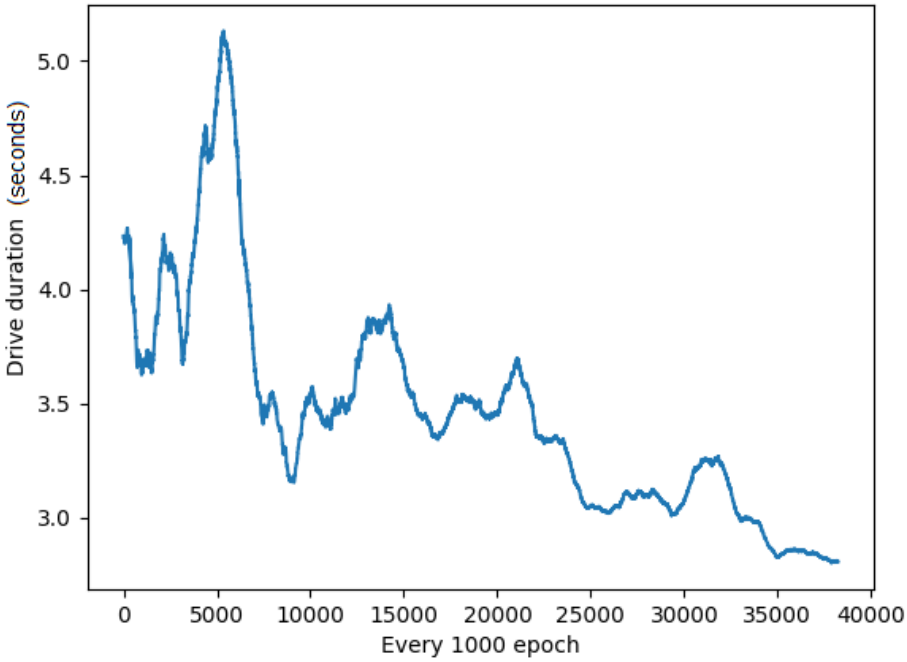


Figure 18: Graph showing the mean drive duration over a 1000 epoch window before a crash.

Figure 19: Illustration of input depth image that the depth image-based agent trained on.

bed results with the camera mounted higher and with lower walls. Both iterations of the car used an autoencoder that compressed the image to the smallest possible dimensions while still keeping the relevant information. More on this can be found in Section 5.4.2.

5.4.1 *Image Segmentation*

Four GANs in total were trained, two for each experiment setup for learning control with segmented images. For the first setup that uses images from the camera at ground level, the two GANs for each domain (simulated and real-time) obtained strong results (Fig. 20a, 20b). The simulation GAN reached an IoU of 0.97 on the test set. The real-time GAN reached an IoU of 0.90 on the test set. Two more GANs were trained for the other setup that uses images from the camera at an elevated position together with lower walls. These GANs obtained strong results as well, achieving an IoU 0.96 for the simulated images and the real-time images (Fig. 21a, 21b). One benefit of having trained GANs with high performance was that it gave us the capability to generate new data. This data was then used to further improve the autoencoder by providing more data to work with.

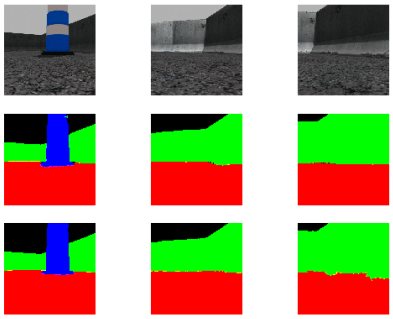
5.4.2 *State Representation Autoencoding*

The autoencoder used for each setup (ground level view and birds eye view) was trained on data from both domains. We tested the information retained compared to how small the images become compressed in the bottleneck of the autoencoder. As seen in Fig. 22 the largest latent vector of $(8, 8, 6)$ retained most information for an autoencoder trained with the swish activation function. This would represent a latent vector of size $(8 * 8 * 6) = 384$ to be used for input for a DDPG. To compare the autoencoder that uses the *swish* activation function, another autoencoder was trained in the same way but with *relu* activation function. As seen in Fig. 23 all vectors except $(8, 8, 1)$ and $(8, 8, 2)$ maintained an IoU of $\tilde{o}.95$.

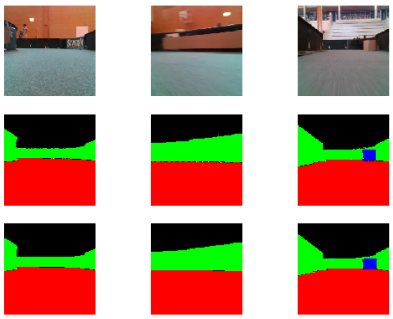
5.4.3 *Ground level view*

The first iteration of the car was trained on images resembling the ones found in Fig.20a and Fig.20b. The car used the same setup in simulation and as in the real racetracks. The walls in simulation was set to the same height as the floorball rims.

At first glance the result found Fig. 24 might seem very good. However the results achieved with this setup had one defining characteristic, it would only learn do drive in circles. This means that the agent did not learn anything from the image input. This lead us to

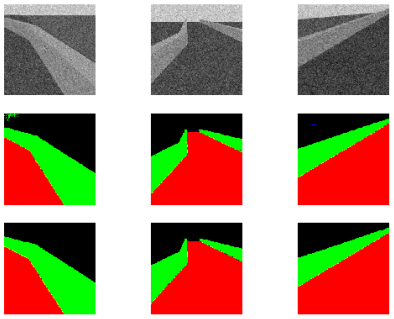


(a) Simulation.

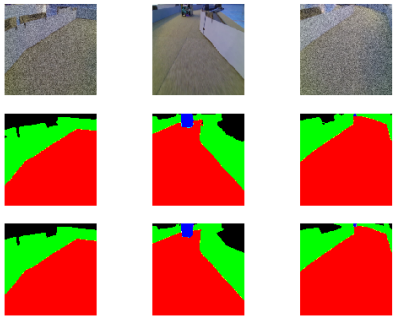


(b) Real-time.

Figure 20: Results from GAN trained on images for a ground view setup. First row is input images, second row the generated images by the GAN and third row the ground truth.



(a) Simulation.



(b) Real-time.

Figure 21: Results from GAN trained on images for a ground view setup. First row is input images, second row the generated images by the GAN and third row the ground truth.

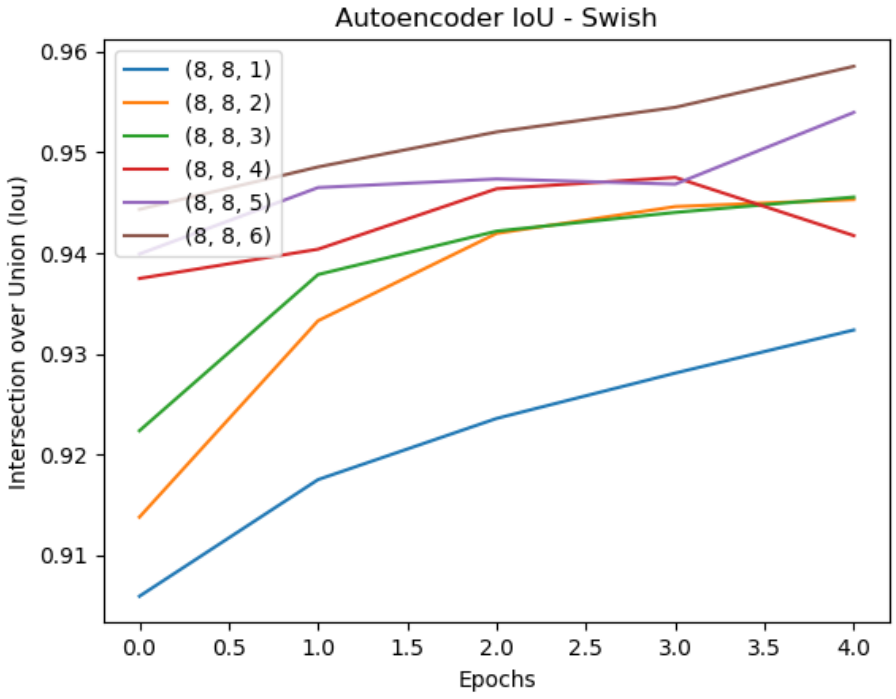


Figure 22: Graphs shows maintained validation IoU over 5 epochs with different latent sizes for the autoencoder using the swish activation function. The latent size of shape (n, m, k) represent the output of the bottleneck convolution layer.

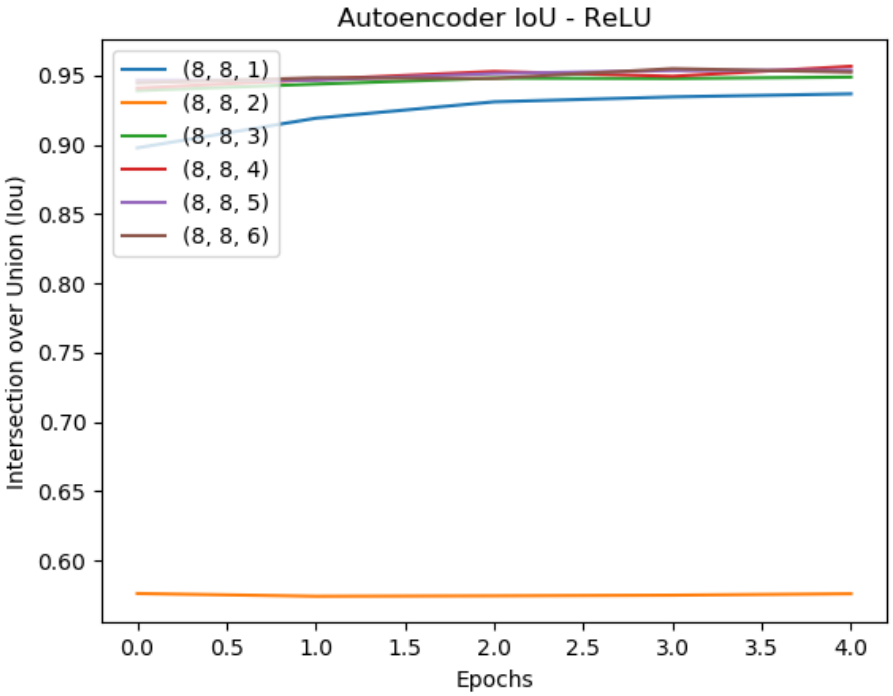


Figure 23: Graphs shows maintained validation IoU over 5 epochs with different latent sizes for the autoencoder using the relu activation function. The latent size of shape (n, m, k) represent the output of the bottleneck convolution layer.

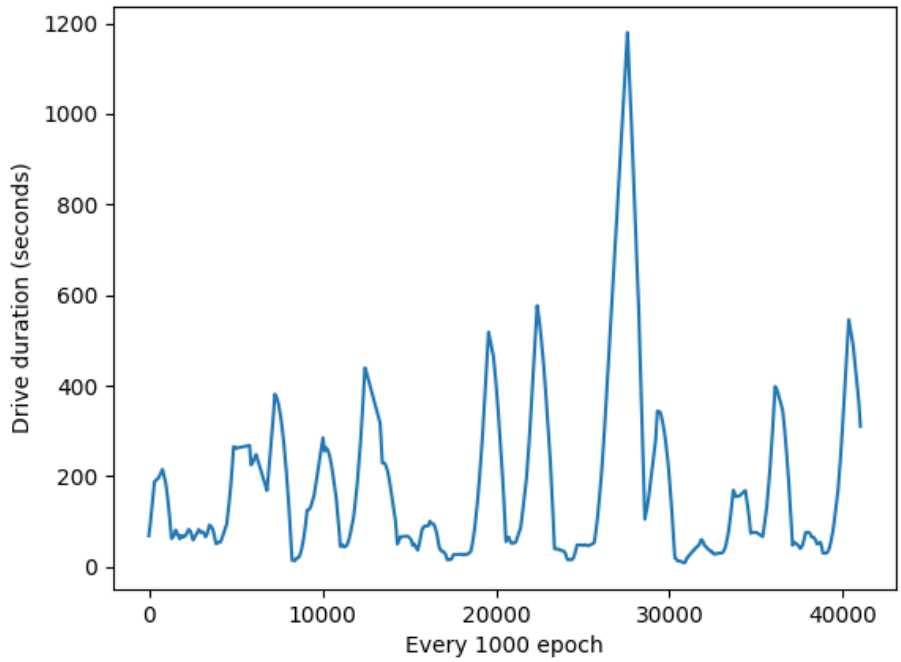


Figure 24: Graph displays the overall drive duration in simulation for the semantic segmentation-based agent with the ground level view setup.

find a new approach for the camera-based agent and the results from that can be found in the next part.

5.4.4 *Birds eye view*

This agent was trained with a DDPG on a track with lower walls and with the camera raised 40 cm and with an angle towards the ground of 25°. It uses the same setup in simulation as on the RC-car. The simulated tracks used a height on the walls of 20 cm and the same was built for the real racetrack. The reward function remains the same (eq. 20).

Using the new setup of the car the agent showed promise in simulation by learning from the images and converting them to valid control signals. It did no longer drive in circles and managed to complete different parts of the maps. This did require longer training times, which can be seen in Fig. 25 compared to Fig. 24. The mean reward found in Fig. 26 stabilized after reaching a value of 0.6. However the learning in simulation did not convert to the real world as the model produced control signals that resulted in crashes. The segmentation results correct with the input data closely resembling the data found in simulation.

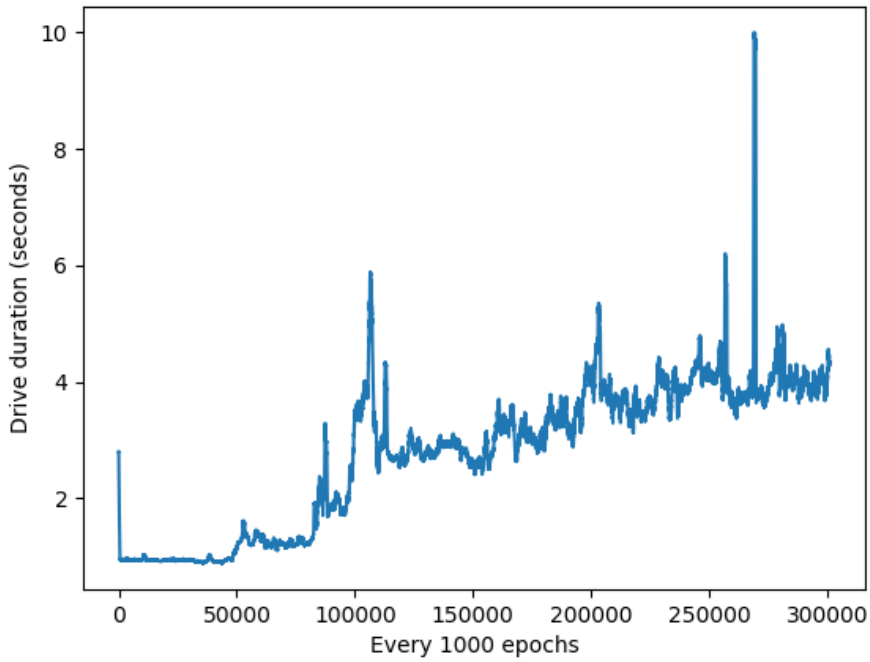


Figure 25: Graphs displays the overall drive duration in simulation for the semantic segmentation-based agent with the birds eye view setup.

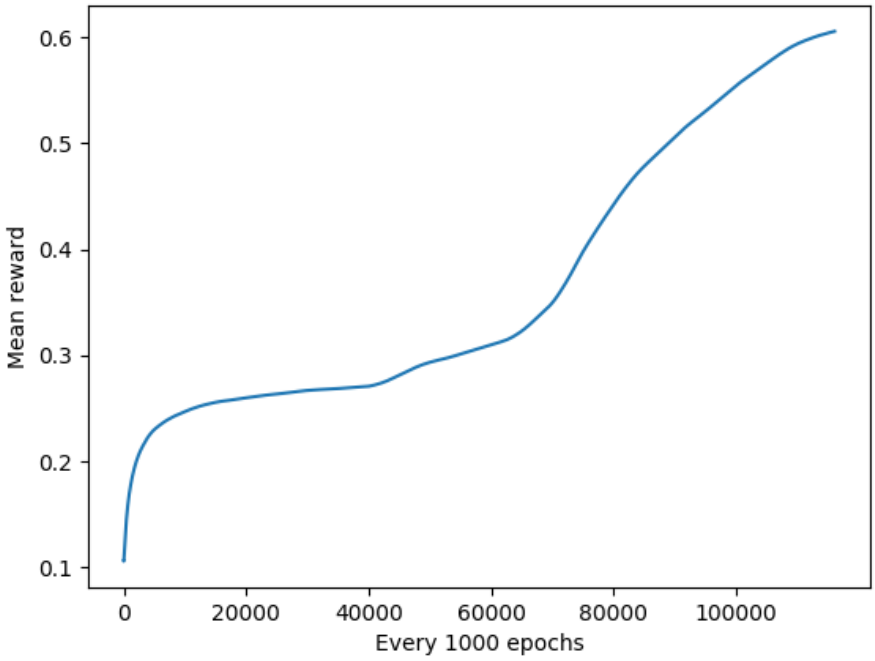


Figure 26: Graphs displays the overall mean reward in simulation for the semantic segmentation-based agent

THE FOLLOWING CHAPTER

Our conclusion and discussion of the result can be found in the next chapter.

6 DISCUSSIONS AND CONCLUSION

This chapter will conclude our master thesis. In Section 6.1 we will present our final thoughts about the project as well as our reasoning to why this project was successful and we will summarize our conclusions from the results in Section 6.2. In Table. 4 we summarize the final behaviour of the agents when deployed in the a real-time scenario, running the agents on an actual hardware test-bed.

6.1 DISCUSSION

Below is a discussion about the three different agents and how they answer the research questions posed for this master thesis. The discussion will also include what was proven and how it could be improved.

LiDAR-based Agent

Research question

- Can an agent learn on LiDAR data and adapt to the real-time scenario?

In this master thesis project we demonstrated that using LiDAR as input data into a DDPG is a robust end-to-end method of transferring the learned policy into a real-time scenario. Thus clearly answering the research question about a reinforcement learning agent trained in simulation with LiDAR data mentioned in Chapter 1. It requires a lot less computation power compared to image and convolution-based methods. The DDPG trained on LiDAR data could handle a completely new simple track and scenario in real-time and combined with a robust novel normalized reward function (eq. 20), that focus on keeping the steering stable and speed fast, it could drive autonomously in real-time. The DDPG can not directly be deployed into a real-time scenario, but driving parameters had to be manually tuned before. We investigated tuning of speed, tuning of restriction in steering angle, delay on action and performing input data scaling and achieved autonomous driving in a new setting without crashing. Without tuning the driving parameters the agent could drive autonomously but eventually made too sharp turns before curved turns.

The policy learned could be improved further by including several time steps of measurements, current and past velocity and steering

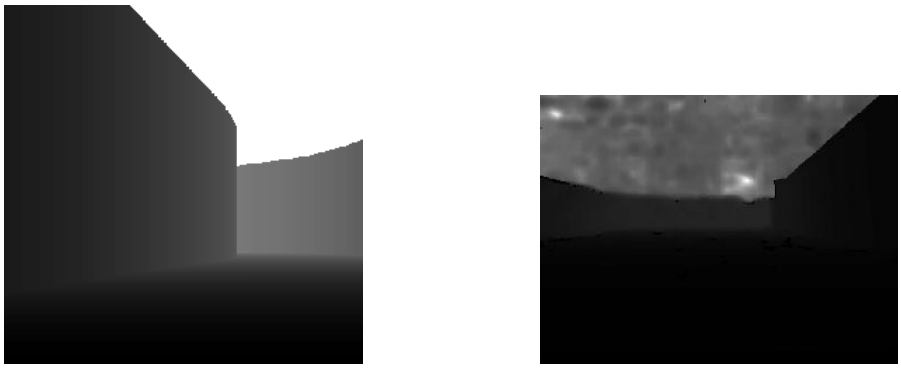
angle actions as input data. Also the agent was not trained with objects in the map, which for a racing scenario could be included to improve object avoidance further. The agent had a set range of possible velocity values that it could reach during training. The agent learned that maximum speed of an action value 1 was best policy in almost every situation. The agent did not really experience the need to perform braking during training. This is one drawback if we would want to push the real-time speed further and navigate along with other cars and obstacles. One suggestion to perhaps introduce a policy of preventing frontal collision is probably to have a logarithmic scaled reward for velocity between $\log(1)$ and $\log(10)$ and train at high speed ranges. Logarithmic scaled reward would introduce diminishing returns for pushing the speed further and the agent could possibly learn to exploit lower speeds to learn a better racing policy. This is currently our strongest candidate for racing in the F1/tenth competition. Another possible positive feature of LiDAR-based DDPG is that due to the lower computation cost, it could map the competition map and train on it in the preparation phase. However as the preparation and tuning phase of the competition is only one day, it is likely a tight window of time of learning optimal policy for that scenario.

Depth Image-based Agent

Research question

- Can an agent learn on depth data and adapt to the real-time scenario?

For an agent trained on depth data we showed results that highly suggests that the agent can learn a navigating policy in the simulation by converting the traditional DDPG into a CDDPG. Using the same reward function, average reward signals matched similar to the LiDAR-based agent in a simulated environment. It is a promising approach to include depth information as it seem clear from an theoretical stand point and the result from the LiDAR-based agent (which uses depth in a different way) that having access to information about driveable space and depth is useful to learn a driving policy. However the real-time depth camera sensor is prone to more noise compared to a LiDAR and the simulated depth camera (Fig. 27a, 27b). The agent could not understand the depth information provided from the depth camera in the real-time scenario. There are still interesting results because if you were to provide methods of mimicking the unique noise in the real-time depth camera for training, there is promise of being able to transfer an end-to-end reinforcement learning agent based on depth data. An interesting approach that could be investigated in the future is to see if variational autoencoders could possible provide a



(a) Simulation. (b) Real-time.

Figure 27: Depth image from real-time and simulation.

way to sample a cleaner or noisier image by passing the input data and tweak the latent space. We feel that this answers the depth question posed in Chapter 1 using our approach.

Semantic Segmentation-based Agent

Research question

- Can an agent learn using segmented (or translated image) and adapt in real-time using same class scene parsing for simulated and real images?

We can not answer the question if an agent can learn continuous actions using semantic segmented images, however our results show some promise as the drive duration and mean reward was increasing over time. That can be one of two things, either the agent is able to learn if so slowly or the more likely option that the agent is over trained. Experiments on new tracks would help with the understanding of over training or not. When comparing the results from the LiDAR- and depth-based approaches of this project it becomes clear that having information about the environments depth is crucial for an agent to learn efficiently, which is why the learning improved when introducing a sense of depth in the input data. To the naked eye the segmented images that the GAN produced was indistinguishable from the ones that the model had trained on in simulation, even though that was the case there was something that we could not see that made the two domains separate. Some interesting future work to consider for navigating with semantic segmented images, is too try introduce semantic segmentation classes that provide enough depth and path direction information. Some ideas is to try and to segment each wall or side of the road as two different classes, 'left wall' and 'right wall. Other approaches could be to try to estimate distances from the image and mark different part of the image. We feel that if we were to introduce some kind of noise similar to the one found

on real data could improve the results. If we were to continue this project of autonomous driving based on semantic segmentation one interesting experiment would be to create a network that produces discrete control signals instead. If that experiment proves successful, then further work to produce continuous signals would ensue, but since the scope of this project included having the car race on a real track, discrete actions were out of the scope already from the start.
We feel that while semantic segmentation was unsuccessful for continuous control of an autonomous vehicle, but it could prove useful when used together with another sensor such as LiDAR.

Table 4: Concluded results for how the car performed with noise or learning.

Agent	**LiDAR**	**Depth Image**	**Semantic Segmentation**
Real-time drive duration	Indefinitely on a simple track	2.2s	1.8s

6.2 CONCLUSION

This thesis propose a new method of constructing an autonomous control system for a RC car, based of reinforcement learning, that can learn in simulation and drive autonomously in real-time using simply steering angle and velocity as output control signals (Fig. 1). This thesis, is to our knowledge, one of the first demonstrations of actually deploying a reinforcement model related to vehicles in a real-time scenario. The most substantial problem facing reinforcement learning in the autonomous vehicle industry is the problem of domain transfers [13]. We have created a reinforcement learning model that performs a remarkable domain transfer between simulation and real-time. Our LiDAR-based model is able to drive completely autonomously in simulation and were then able to transfer that policy to a vehicle in real-time posed with the problem of racing on a simple unseen track from a different domain. The vehicle is then able to drive indefinitely with high velocity without crashing. Final behaviour is concluded in Table 4.

We also created two more models based on the same approach of DDPG where the one trained on depth-data had adequate results in simulation but struggled when transferred to the autonomous vehicle due to noise in the depth-data. The last model was a DDPG trained on semantic segmented images, we had high results from the connected networks in the form of GAN and AE but the car struggled to learn continuous control signals in simulation.

We have with this project demonstrated that by finding a proper reward function, LiDAR-data and tuning driving parameters, one can transfer the policy to the real-time domain and drive autonomously.

BIBLIOGRAPHY

[1] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[2] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving, 2017.

[3] Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks, 2018.

[4] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, Dec 2017.

[6] Volodymyr Mnih, AdriÃ PuigdomÃšnech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[7] Bernhard Wymann, Christos Dimitrakakisy, Andrew Sumnery, and Christophe Guionneauz. Torcs: The open racing car simulator, 2015.

[8] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image, 2016.

[9] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE, 2017.

[10] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[11] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[12] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[14] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[15] François Chollet et al. Keras. https://keras.io, 2015.

[16] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[17] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2012.

[18] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[20] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Phys. Rev.*, 36:823–841, Sep 1930.

[21] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi. End-to-end race driving with deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2070–2075, May 2018.

[22] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[24] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

PO Box 823, SE-301 18 Halmstad
Phone: +35 46 16 71 00
E-mail: registrator@hh.se
www.hh.se