EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, *SVERIGE 2019*

Performance Evaluation of Imitation Learning Algorithms with Human Experts

ERIK BÅVENSTRAND

JAKOB BERGGREN

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Abstract

The purpose of this thesis was to compare the performance of three different imitation learning algorithms with human experts, with limited expert time. The central question was, *"How should one implement imitation learning in a simulated car racing environment, using human experts, to achieve the best performance when access to the experts is limited?"*. We limited the work to only consider the three algorithms Behavior Cloning, DAGGER, and HG-DAGGER and limited the implementation to the car racing simulator TORCS. The agents consisted of the same type of feedforward neural network that utilized sensor data provided by TORCS. Through comparison in the performance of the different algorithms on a different amount of expert time, we can conclude that HG-DAGGER performed the best. In this case, performance is regarded as a distance covered given set time. Its performance also seemed to scale well with more expert time, which the others did not. This result confirmed previously published results when comparing these algorithms.

Keywords

Imitation Learning, DAGGER, HG-DAGGER, Behavior Cloning, Machine Learning, TORCS

Abstract

Målet med detta examensarbete var att jämföra prestandan av tre olika algoritmer inom området imitationinlärning med mänskliga experter, där experttiden är begränsad. Arbetets frågeställning var, ”Hur ska man implementera imitationsinlärning i en bilsimulator, för att få bäst prestanda, med mänskliga experter där experttiden är begränsad?”. Vi begränsade arbetet till att endast omfatta de tre algoritmerna, Behavior Cloning, DAGGER och HG-DAGGER, och begränsade implementationsmiljön till bilsimulatorn TORCS. Alla agenterna bestod av samma sorts *feedforward* neuralt nätverk som använde sig av sensordata från TROCS. Genom jämförelse i prestanda på olika mängder experttid kan vi dra slutsatsen att HG-DAGGER gav bäst resultat. I detta fall motsvarar prestanda körsträcka, givet en viss tid. Dess prestanda verkar även utvecklas väl med ytterligare experttid, vilket de övriga inte gjorde. Detta resultat bekräftar tidigare publicerade resultat om jämförelse av de tre olika algoritmerna.

Nyckelord

Imitationsinlärning, DAGGER, HG-DAGGER, Behavior Cloning, Maskininlärning, TORCS

Acknowledgements

There are a few people we would like to thank for their support of this thesis. We would like to sincerely thank our supervisor Mika Cohen, both for his time and effort put into this thesis. also, we would like to thank our examiner Anders Västberg for his guidance in this thesis. A special thank you to Bernhard Wymann for his immense work in developing TORCS.

Authors

Erik Båvenstrand erikbav@kth.se
Jakob Berggren jaberggr@kth.se
Information and Communication Technology
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden

Examiner

Anders Västberg
KTH Royal Institute of Technology

Supervisor

Mika Cohen
KTH Royal Institute of Technology

Contents

Glossary

AI	Artificial Intelligence.
ANN	Artificial Neural Network.
BHC	Behavior Cloning.
DAGGER	Dataset Aggregation.
FNN	Feedforward Neural Network.
HG-DAGGER	Human-Gated Dataset Aggregation.
IL	Imitation Learning.
ML	Machine Learning.
MSE	Mean Squared Error.
RL	Reinforcement Learning.
RMSE	Root Mean Squared Error.
TORCS	The Open Racing Car Simulator.

1 Introduction

Racing simulation presents a multitude of different challenges for machine learning (ML). The most prominent one being the imitation of human intuition and in turn, human intelligence. The use of racing simulations can be compared with the way board games have been used to research artificial intelligence (AI), with the obvious example of deepMind and their alphaZero in chess [21]. In comparison to board games, the use of a fast-paced racing simulation is a definite step towards solving more complex problems in the ML stratosphere, while keeping the decision time to a minimum. The complexity often comes in the form of non-determinism, large dimensional space-state, and a requirement of fast decision making.

1.1 Background

One way to learn is through imitation of actions by someone else. What might not be as well known is that the principle of imitation learning (IL) is also applicable to computers, and more specifically artificial neural networks (ANN). It is not surprising when taken into consideration the close relation between ANN and biological neural networks [20]. One convenient property of IL is that it can be performed on recorded data which reduces the need for live training.

The IL technique to be explored is DAGGER and was first introduced in a paper from 2011 [18]. It can be briefly explained as an iterative algorithm that makes use of an expert policy to record a dataset. The dataset is then used for training a new policy made to mimic the expert policy. For the next iteration, the newly recorded dataset is aggregated to the previous set of data before training the new policy.

The Open Racing Car Simulator (TORCS) has proven to be a good choice when researching AI in driving environments [25]. TORCS will be the simulator of choice in this thesis.

1.2 Problem

Racing simulation is a problem with considerable complexity and has a close connection to the paradigm shift of self-driving cars in the car industry. The grand problem can, therefore, be formulated as follows:

How should self-driving cars be implemented to achieve the best possible performance?

However, due to the time constraints of this thesis, this problem is far too broad and open for us to answer and will, therefore, be narrowed down to something more tangible. IL could be a solution to the complex problem presented above. Therefore, we identify the problem of this report to be comparative, where we will compare the performance of different IL algorithms. More specifically, the viability of behaviour cloning (BHC) [1], dataset aggregation (DAGGER), and human-gated dataset aggregation (HG-DAGGER) [9] will be evaluated, when using a human as expert for IL in car racing simulations.

How should one implement IL in a simulated car racing environment, using human experts, to achieve the best performance when access to the experts is limited?

1.3 Purpose

The purpose of this thesis is to investigate three different IL algorithms performance in a simulated car racing environment with limited access to the expert. This is done to contribute to further development in the field of IL.

1.4 Goal

The goal is to provide scientific value by comparing the three different IL algorithms, DAGGER, HG-DAGGER, and BHC. The comparison will be done in a simulated racing environment, to verify previously achieved results in a different environment. In essence, we want to create research material for future work within IL.

1.5 Benefits, Ethics and Sustainability

Hopefully, this study will be useful outside of the simulator and affect the development of autonomous driving in the real world. Given that there are organizations such as Drive Sweden, who are backed by the Swedish government, working on various research projects focusing on autonomous vehicles, there is an interest of the topic at a governmental level in Sweden [22]. The argument can be made that Sweden may directly benefit from this study. Secondary effects such as lowered costs and emissions of transport through the development of autonomous cars may also be relevant to consider. Especially since Greenblatt et al. forecasts the emissions of greenhouse gases and energy use to be lowered by as much as 80% through the use of autonomous vehicles [5]. This gives reason to assume that the work conducted in this thesis may have an indirect positive effect on sustainability and help achieve environmental goals set out by, for example, the EU.

1.6 Methodology

The research methodology employed in this thesis is *Experimental research*, which is quantitative. It is the study of cause and effect and is conducted by manipulating one variable while keeping the other variables constant while observing for differences in the result [6]. This method is commonly used when analyzing performance in a system and is, therefore, a good fit for this thesis since the different IL algorithms are to be compared in performance when given different amounts of expert time.

1.7 Stakeholders

The main stakeholder of this project is Mika Cohen, a researcher at the *Swedish Defense Research Agency* (FOI), and our supervisor for this project. The project is conducted upon his appointment.

1.8 Delimitations

Concerning the goal of this thesis, work will be limited to only regard BHC, DAGGER, and HG-DAGGER as the algorithms of interest. Furthermore, the environment in which the different AI's will be deployed will be the car racing simulator TORCS. Firstly, because it widely used for ML research in race driving, some examples of this are related works presented in section 1.1, and secondly for being open-source [25]. Further, and more specific delimitations of the thesis will be presented in section 3.

1.9 Outline

Section 2 will in depth present relevant theoretical background information about ANN, IL and the algorithms, BHC, DAGGER and HG-DAGGER. It will also describe the environment (TORCS) of the implementation. Section 3 presents a thorough explanation of the methodology used and the reasoning behind it. Section 4 will provide the reader with a practical description of the implementation of the method described in the previous chapter. Section 5 presents the results achieved by the thesis work. Section 6 presents the conclusion and provides a discussion of the conducted thesis project. Future work is also included in that section.

2 Background

In this section, a detailed description of the background of the thesis is presented together with related work. If a further understanding of the background is desired, we kindly refer you to the original works, as cited.

2.1 Artificial Neural Network

An ANN consists of artificial neurons in layers, and acts as a framework for different ML algorithms [8]. Every ANN needs to have an input layer of nodes where the data is received from the environment. However, the size of this layer is highly dependent on the application and can vary immensely. The same principle also applies to the last layer of nodes in the ANN. The last layer of an ANN is the output layer, which outputs the processed data to the environment. The appearance of it is highly dependent on the environment, just as the input layer. Between the first and last layers, there can also exist so-called hidden layers of neurons that are not in contact with, nor visible from the outside environment. There exist infinitely many configurations of the layers and how they are connected, but for this thesis, only feedforward neural network (FNN) are of interest. In a FNN, each layer receives inputs from the previous layer and computes an output that is propagated to the next layer. A visual representation of a FNN can be seen in Figure 2.1.

2.1.1 Artificial Neuron

The neuron is the fundamental building block for all ANNs and is in reality just the mathematical equation 1 [3, p. 42]. For each input to the neuron, there exists a corresponding weight that in some sense describes the importance of that input. A bias value is added to the sum that is used to make sure that even if all the input values to the neuron are 0, it is still going to activate. Before the neuron propagates the value forward, an activation function is applied to the value.

$$Y = \Sigma_{i=1}^{n}(w_i * x_i) + b \tag{1}$$

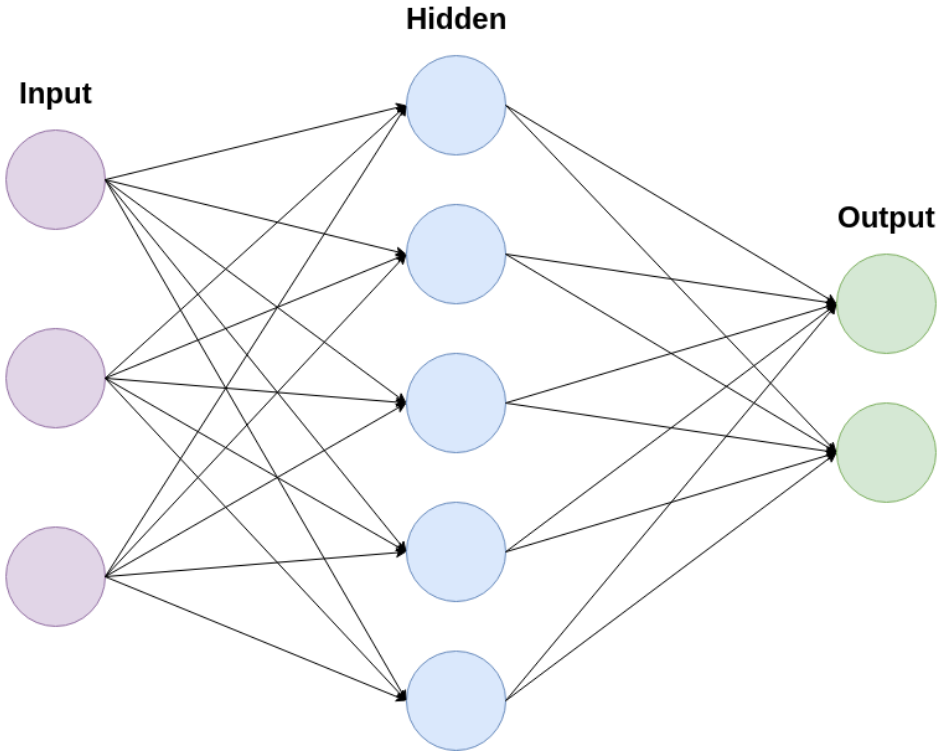


Figure 2.1: An FNN.

2.1.2 Activation Function

The purpose of an activation function is to introduce non-linearity to an ANN [27, pp.129-137]. Some examples of such functions are Sigmoid [27, pp.118] and rectified linear unit (ReLU) [27, pp.116]. The Sigmoid activation function is a *squashing* function and returns values in the range 0 to 1. It is defined in equation 2. The Sigmoid function is commonly used for binary classification problems that require the range [0, 1]. Unfortunately, it is prone to saturation since large input values result in values close to either 0 or 1. This slows down the learning process since the gradient of the Sigmoid function is very small at large values. In simple terms, the network can get *stuck* when training because the gradients of the calculated values are too small.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

ReLU returns values in the range $[0,+\infty]$ and is described in equation 3. The reasoning behind ReLU is that it provides a very simple nonlinear transformation.

It will retain positive values while discarding negative values. ReLU does not have the same problem with vanishing gradients as with Sigmoid. Unfortunately, it has the property of causing neurons to *die* if the weight turns negative. The neuron will most likely never be *resurrected* and will not provide any use to the network anymore [27, pp.139-140].

$$\text{ReLU}(x) = max(0, x) \tag{3}$$

2.1.3 Loss Function

A loss function is required to determine an ANNs performance by computing the deviation of its predictions. These functions are often immediate implementations of standard statistical functions, such as the mean squared error (MSE) [27, p.18] shown in equation 4. The MSE is a typical function to use for error computation. It measures the average of squared differences between the network's prediction and the label of the data. It has an interesting computational characteristic. It is independent of direction due to the nature of squaring a difference, which may be problematic if the direction of the error is of importance. This may lead to an emphasis of larger errors and a neglection of smaller ones.

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(y_i - \hat{y}_i)^2 \tag{4}$$

2.1.4 Optimization Function

The optimization function, in its most basic form, consists of maximizing or minimizing the output in an attempt to find the optimal solution. In the case of this thesis the function to be optimized is the loss function MSE presented in section 2.1.3. There exists a multitude of optimization functions, however the one of interest for this thesis is ADAM [11], displayed in equation 5. This function uses both gradients and second moments of gradients. The parameters given are represented as $w^{(t)}$ and the loss function $L^{(t)}$. t is the current iteration of training, and the loss function is MSE in our case. The small constant ϵ is used to prevent division by zero, and β is used as a forgetting factor to give a decaying effect on

previous gradients.

$$
\begin{aligned}
m_w^{(t+1)} &= \beta_1 m_w^{(t+1)} + (1-\beta_1)(\nabla_w L^{(t)}) \\
v_w^{(t+1)} &= \beta_2 m_w^{(t+1)} + (1-\beta_2)(\nabla_w L^{(t)})^2 \\
\hat{m}_w &= \frac{m_w^{(t+1)}}{1-(\beta_1)^{t+1}} \\
\hat{v}_w &= \frac{v_w^{(t+1)}}{1-(\beta_2)^{t+1}} \\
w^{(t+1)} &= w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w + \epsilon}}
\end{aligned}
\tag{5}
$$

2.1.5 Training an ANN

The previous subsections have covered the individual parts of training an ANN, and now the full composition will be explained. The training is conducted through a method called backpropagation. It works by computing the current error with a loss function and calculating the changes to the ANN through an optimization function [27, pp.176-177]. When configuring the training setup, there are a few important variables to set. They will all affect the training and the outcome of the trained ANN regarding stability and accuracy. The variables are batch size, the number of epochs, and learning rate [27, pp.95-96].

Batch size is an important variable of the training and is vital for computational feasibility when using large datasets [27, pp.431-432]. A large amount of data poses a computational problem, due to the vast amount of calculation needed for each epoch of training. Therefore, it is natural to process the data in batches. The optimal size for the batches is highly subjective and should be viewed as a trade-off between accuracy and speed.

The number of epochs used when training is, as with batch sizes, highly subjective. For each epoch, the whole dataset is trained on, which allows for more training to be conducted without needing additional data [27, p.107]. While one would optimally have enough data only to run one epoch, it is not feasible in practice. A complete dataset is not practical nor necessary to achieve a somewhat optimal result and can, therefore, be substituted by several epochs.

Lastly, the learning rate of the ANN has to be selected, which is a scalar to the tweaking of the weights and biases [27, pp.422-423]. It is a trade-off of the same character as with the batch size. A higher learning rate can increase the training speed drastically but presents the risk of sub-optimally tweaking the ANN, by making too large changes.

2.1.6 Selecting a Topology for an ANN

When designing an ANN many different components need to be tweaked to fit the problem domain. There is seldom a known optimal configuration for the specific application at hand, and it will often require some tweaking through trial and error, to achieve acceptable performance. This is a direct cause of the complex environment in which ANN's are used. However, there are a few guidelines to follow in the design. Firstly, the number of necessary input and output neurons should be decided. The number of input parameters can be represented as the spatial dimensions of the ANN, while the number of output parameters is the number of solution surfaces, per Rafiq et al [17, p. 1545]. If there are too many neurons in the input layer, the risk of confusing the ANN runs higher while also increasing the computational power required. Too few and the problem resolution becomes too small for precise calculations. When designing input and output layers, there are a few suggested ways of doing it. For example, using statistical methods or dimensional analysis [17, p. 1545].

The more abstract part of the architectural design of an ANN are the hidden layers. The design of the hidden layers is problem specific and dependent on the amount, and quality of the training data. Too many neurons in the hidden layers and it might encourage over-fitting or modeling of the data with unnecessary complexity [23]. One should strive for the design to be simple enough for generalization, yet sufficiently complex for the correct modeling of the problem.

Rafiq et al. finds that some researchers use an upper bound for neurons in the hidden layer of twice the number of input neurons [17, pp. 1546-1547]. However, this comes with no guarantee of generalization of the network. To properly design a hidden layer, a parametric study should be carried out by changing the design

gradually and observing the difference in performance until convergence is found or a predefined threshold achieved. The unit of measure is the root mean squared error (RMSE).

2.1.7 Keras

Keras [10] is a high-level API for neural networks built to be easy to use and allowing for fast prototyping. It runs on top of other popular ML frameworks, such as TensorFlow, CNTK, or Theano.

2.2 Imitation Learning

IL, or learning by demonstration as it is also known, is one of many ML techniques. IL is based on the notion that there exists a learning agent and an expert that operates in a teacher-student manner [3, Section 18]. The expert is viewed as an oracle and the actions taken by it are regarded as optimal by the agent. Each state is an observation of the environment and a corresponding action taken by the expert. The goal of the agent is to learn the expert policy by minimizing the total loss of all actions taken by the agent.

2.2.1 Agent

The learning agent is commonly implemented using an ANN. The only criteria are that the agent can learn and to generalize the expert policy. Since the agent is commonly an ANN, the architecture of it has to be implemented in a way that is appropriate for the learning environment.

2.2.2 Expert

The expert in a IL setting can be any type of algorithm or entity that can make decisions [3, Section 18.4]. Preferably, it should either be a computationally expensive algorithm or something with inherit intuition, like a human. In basic IL, like BHC, the dataset used for training the agent can be pre-computed. Thus,

allowing the dataset to be generated by exhaustive search of the state-space or similar expensive computations. This is not the case for other forms of IL, such as DAGGER, that requires the expert policy to make decisions in parallel with the agent to aggregate the dataset. An advantage of using an algorithmic expert is that the policy itself is sure about the action taken, given a certain observation of the environment. A human might react differently depending on many factors, such as the focus on the task or prior knowledge. This requires the agent to account for variance in the expert actions given the same observation.

2.3 Behavior Cloning

BHC is one of the most basic forms of imitation learning. It requires only a demonstrator or records of demonstration data, which are actions taken by the expert and the given state in which the action was taken [3]. The algorithm consists of two significant steps. Firstly, the dataset D is sampled from the expert policy π^*, and secondly, the policy $\hat{\pi}$ is trained on the dataset D.

A benefit of BHC is the simple characteristics of its implementation, as well as its efficiency. However, its simple design does come with disadvantages. Such a disadvantage is the lack of ability to plan and handle long term planning [26]. These properties make BHC useful when 1-step deviations are not critical and/or expert trajectories ”cover” the state space. The technique is more appropriately used for classification problems, rather than sequential decision-making problems, where the current state is dependent on previous decisions.

Algorithm 1 BHC

1: Initialize $D \leftarrow \emptyset$
2: Sample trajectories using π^*.
3: Get dataset $D = \{(s,\pi^*(s)\}$ of visited states by π^*
4: and actions given by expert.
5: Train policy $\hat{\pi}$ on D.
6: **return** $\hat{\pi}$ on validation.

2.4 DAGGER

In a paper published in 2011, written by Ross et al [18], the first version of the DAGGER algorithm was presented. It is described in pseudo-code as seen in algorithm 2. However, a thorough explanation is necessary.

DAGGER is an iterative algorithm in the area of IL, that aims to teach an agent the policy of the expert. D is the collected dataset which, during each iteration, is aggregated with the newly collected dataset D_i. The agent policy $\hat{\pi}_1$ is initialized to a random policy at the start. π^* is the expert policy and β_i is the probability of intervention by the expert. It is initialized to 1 and slowly decays towards 0 during N iterations, where the agent is completely in charge of the decisions. At the start of each iteration, a policy π_i is created. π_i is a combination of expert and agent decisions. A sampling of new states by the policy π_i is done and for each step, the observation and the expert decision are stored in the dataset D_i. The dataset D_i is then aggregated to D and the new agent policy $\hat{\pi}_{i+1}$ is trained on the newly updated dataset D.

The issue that DAGGER attempts to solve is that plain IL often leads to a compounding error in sequential decision-making problems. This is because the previous actions affect the current state of the agent, thus affecting the actions of the future. To counteract this, Ross et al. introduced a stochastic mixed policy π_i that gradually allows the agent to control more and more of the decisions.

The work conducted by Ross et al. regarding DAGGER is highly relevant for this thesis since their work in formulating the algorithm sets the foundation for us to analyze the technique in specific problem domains.

2.5 HG-DAGGER

The original DAGGER algorithm was implemented with a stochastic variable β_i which sets the amount of expert intervention for that epoch; however, this might not be optimal when using human experts. The weighted coin toss with the probability β_i is assumed to be uniformly distributed across all steps. Because of this, the expert will experience a form of input lag since the decisions are only

Algorithm 2 DAGGER

Initialize $D \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\prod$.
for i = 1 **to** N **do**
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
    Sample T-step trajectories using π_i.
    Get dataset $D_i = \{(s,\pi^*(s)\}$ of visited states by π_i
and actions given by expert.
    Aggregate datasets: $D \leftarrow D \cup D_i$.
    Train policy $\hat{\pi}_{i+1}$ on D.
return best $\hat{\pi}_i$ on validation.

partly made by the expert. As the probability β_i approaches 0, it will severely impair the expert's decisions negatively. Kelly et al. propose HG-DAGGER [9], a derivative of DAGGER which is specifically developed to be used with a human as an expert. This method of DAGGER was developed under the assumption that better labels can be acquired if the human expert can demonstrate uninterrupted for longer periods [9, 18]. The pseudo-code for HG-DAGGER is provided in algorithm 3.

HG-DAGGER relies on the fact that an ensemble of independent ANN will slightly differ in their answers to the same question. The l_2 norm of the answer's covariance matrix is then calculated to find a doubt value. The doubt is a value of how much the ANNs differ in their answers to the same question. If the doubt value is high it means that the ensemble is uncertain in their decision. This could mean that the demonstrations provided do not cover the given state with high doubt. Therefore, when the doubt crosses a certain threshold, the controls are given to the expert until the expert decides to hand them back. This ensures that the agent only is given new training data when corrections need to be made to better imitate the expert policy.

The dataset D is initialized with the set D_{BC} that contains demonstrations performed by the expert. I is a list of doubt values. The algorithm is executed for K epochs. M rollouts are executed, which each contains several timesteps T. A timestep is a single observation of the environment with the corresponding action. If the expert is requesting to take control of the car, the current doubt value of the ensemble is recorded into I_j before recording the demonstrations to D_j. The

demonstrations D_j are appended to D and the doubt values I_j are appended to I. A doubt threshold is calculated by finding the mean of the latest 25% of all doubt values in I. If the doubt value crosses the threshold, the control is automatically given to the expert since the agent is indecisive. When the rollout is complete, all the ensemble networks are trained on the same aggregated dataset D.

Algorithm 3 HG-DAGGER

procedure HG-DAGGER(π_H, π_{N_1}, D_{BC})
    $D \leftarrow D_{BC}$
    $I \leftarrow []$
    for epoch i = 1 **to** K **do**
        for rollout j = 1 **to** M **do**
            for timestep $t \epsilon T$ of rollout j **do**
                if expert has control **then**
                    record expert labels into D_j
                if expert is taking control **then**
                    record doubt into I_j
            $D \leftarrow D \cup D_j$
            append I_j to I
        train $\pi_{N_{i+1}}$ on D
    $T \leftarrow F(I)$
    return $\pi_{N_{K+1}}, T$

2.6 TORCS Environment

TORCS is an open-source racing simulation game that is commonly used for AI research in smart control systems for cars. It simulates the driver through a low-level API which gives only partial access to the simulation state [25, p. 1].

The simulation engine uses discrete time, paired with simple Euler integration of differential equations. It is developed to be simple while still handling all the basic properties of vehicular dynamics. Some of these properties are as follows:

- Mass and rotational inertia of car components.
- Mechanical components such as differentials, suspensions, and links.
- Friction profile of tires, both static and dynamic, dependent on the profile of the ground.

- Aerodynamic model, which includes both slip-streaming and ground effects.

While the simulation engine is running at a time discretization of 0.002s of simulated time, the drivers only have an opportunity to interact with the simulation every 0.02s. This means that the simulation is perceived by the drivers as it would be running at 50 frames per second, while the simulation is running 500 frames a second [25, pp. 1-2].

Total documentation of the actuators and sensors available in the TORCS environment can be found in appendix A. The sensors available are found in the Table A.1, and the actuators in Table A.2. This setup of sensors and actuators are per the official rules for the world championship in AI racing for TORCS [13].

2.7 Related Work

In this section, we will present a few related works and give a brief explanation of why that is.

2.7.1 DAGGER

The DAGGER algorithm was introduced in a paper from 2011 [18] with the intent to solve issues regarding IL in sequential decision-making problems, where the future actions depend on past actions. The DAGGER algorithm attempts to solve these issues and was tested on a driving game, similar to TORCS, with good results. The DAGGER algorithm is one of the three algorithms compared in this thesis.

2.7.2 HG-DAGGER

In a paper published 2019 [9], the HG-DAGGER algorithm was described as a combination of ENSEMBLE-DAGGER and SAFE-DAGGER, to be used for IL with human experts. This is very relevant to this thesis as the expert is human. The results achieved in this paper compares the algorithm to both DAGGER and BHC, which are the three algorithms this thesis compares.

2.7.3 Temporal Exploration

A paper released by Heuel et al. [7] examines the domain of reinforcement learning (RL) in the driving simulator TORCS. The approach of this thesis and their work is similar to ours in the way that inputs to the agents are from sensors. However, due to vast differences in implementation and underlying techniques the results of this paper will not be comparable to the results presented in this thesis.

2.7.4 A Human-Like TORCS Controller

Muñoz et al. presented a human-like TORCS controller in a paper from 2010 [16], where they describe the implementation of a controller and the ANN that operates the car in great detail. Controllers that mimic humans are not the norm and therefore this paper is relevant for the implementation of the controllers in this thesis. The results of this paper are not comparable to the results of this thesis due to the approach of creating a human-like TORCS controller. The human-like TORCS controller presented in their paper consisted of several hard-coded policies that aid the ANN in its decisions. This would create an unfair advantage compared to the agents in this thesis.

2.7.5 Vision-Based Neural Networks for TORCS

The paper Evolving Large-Scale Neural Networks for Vision-Based TORCS, by Koutník et al. [12] experiments with large scale neural networks to achieve good performance with a vision-based agent in TORCS. The choice of vision-based input sensors provides an interesting contrast to the sensors provided by TORCS. The results of this paper cannot be compared to the results of this thesis since the input sensors are completely different, and the agents of the paper are trained and validated on the same track.

3 Method

In this section, the preliminaries for the experiment will be described as well as the method used to answer the central question of the thesis.

3.1 Implementation

The project is entirely written in Python 3.6 [19] and is using SnakeOil [4] to interface with the TORCS server. SnakeOil is a client interfacing with the TORCS API through sockets. The TORCS server is a modified version of release 1.3.7, according to the simulated car racing manual [13] and is available for download on GitHub [15]. The two following python libraries are also used. TensorFlow 1.12.0 [14], which contains Keras and is used to construct and run the ANN. Keras [10] is a high-level API for neural networks built to be easy to use and allowing for fast prototyping. It runs on top of other popular ML frameworks, such as TensorFlow, CNTK, or Theano. Pygame 1.9.4 [2] is a free and open-source game library used for receiving inputs from a steering wheel and pedals. The steering wheel and pedals used are Logitech G29 Driving Force. All three algorithms used the car car1-trb1 [24]. The ANN architecture is the same for all three algorithms and they all have access to the same sensors and actuators.

The implementation of the three different imitation learning algorithms BHC, DAGGER, and HG-DAGGER have been done following the original published papers in which these algorithms were first introduced. Pseudo code of the different algorithms can be found in sections 2.3 - 2.5.

3.2 Testing

To analyze the performance of the different algorithms, parametric analysis was used. All parameters were kept identical across the three algorithms. The changing parameter is expert time and it is defined as the time where the expert is either demonstrating to or actively observing an agent. Many of the parameters that are not explicitly defined in the different implementations were selected from

related works that achieved good results. The main performance metric evaluated is *distance covered* for a given *time*.

When experimenting, T tracks are used to train the agent while V tracks are used for validation and measuring the performance of the different algorithms. The experiment is divided into X number of instances. The first instance will contain the least amount of expert time while the last instance will contain the most. Each instance is defined as follows: All agents will get an equal amount of pre-training demonstration on all T training maps. This will then be followed by individual training where all agents will be given the same amount of expert time. This will allow for equal treatment for all techniques regarding expert time, even though BHC is not actively using an expert. Each validation track V is run multiple times to gather data that is not skewed by the non-determinism of TORCS.

To ensure feasibility when conducting the testing, we had to introduce some limiting factors to the experiment. Specifically, we limited the experiment as follows:

- Training tracks T and validation tracks V are all on tarmac.
- The agent was alone on the track during training and validation.
- Automatic transmission was used.

3.3 Research Paradigm

Quantitative research was done in the form of an experimental study. Specifically, a parametric study [6] was conducted to answer the research question of this thesis.

How should one implement IL in a simulated car racing environment, using human experts, to achieve the best performance when access to the experts is limited?

An inductive approach was taken to answer this question, which means that a general answer for the question was induced for the question based on the limited data collected. There are two reasons for choosing an inductive approach. Firstly,

the authors were not experienced enough in the field of IL to propose a relevant hypothesis on the topic. Secondly, the feasibility of conducting this thesis with limited time also needed to be considered, therefore an inductive approach to the problem was considered most suitable.

4 Implementation

In this section, the implementation, conducted following the method from the previous section, will be described in detail.

4.1 Agent

The agents that were trained with the three different techniques all share the same basic ANN architecture. The input layer was decided to have 10 neurons. Specifically, the sensors used by the neural network are the following: (i) The speedX sensor that reads the current speed in the direction of the car. (ii) Nine of the track sensors that measure the distance from the car to the edge of the track along the directions {-90°, -60°, -30°, -10°, 0°, 10°, 30°, 60°, 90°}. The distance measuring sensors are displayed in Figure 4.1 The hidden layers consist of 256 and 128 fully connected ReLU neurons. The hidden layer architecture is inspired by a previous work that successfully managed to drive a car in TORCS with good results [7]. The output layer consists of two neurons with Sigmoid activation. The two outputs control the steering and the throttle of the car. No further investigation of the architecture was conducted due to the scope of this project.

4.2 Tracks

All tracks were selected from the available tarmac tracks in TORCS 1.3.7. In total, 15 tracks were selected. 10 of them were used for training while 5 of them were used for validation. A diverse set of training data is required to achieve good results in the validation set. Because of this, one might want to manually make sure that the validation set and training set is kept diverse. However, to not induce any unnecessary human interference to the experiment, we chose to select the sets of maps at random. The 10 training tracks can be seen in Figure 4.2 and the 5 validation tracks can be seen in Figure 4.3.

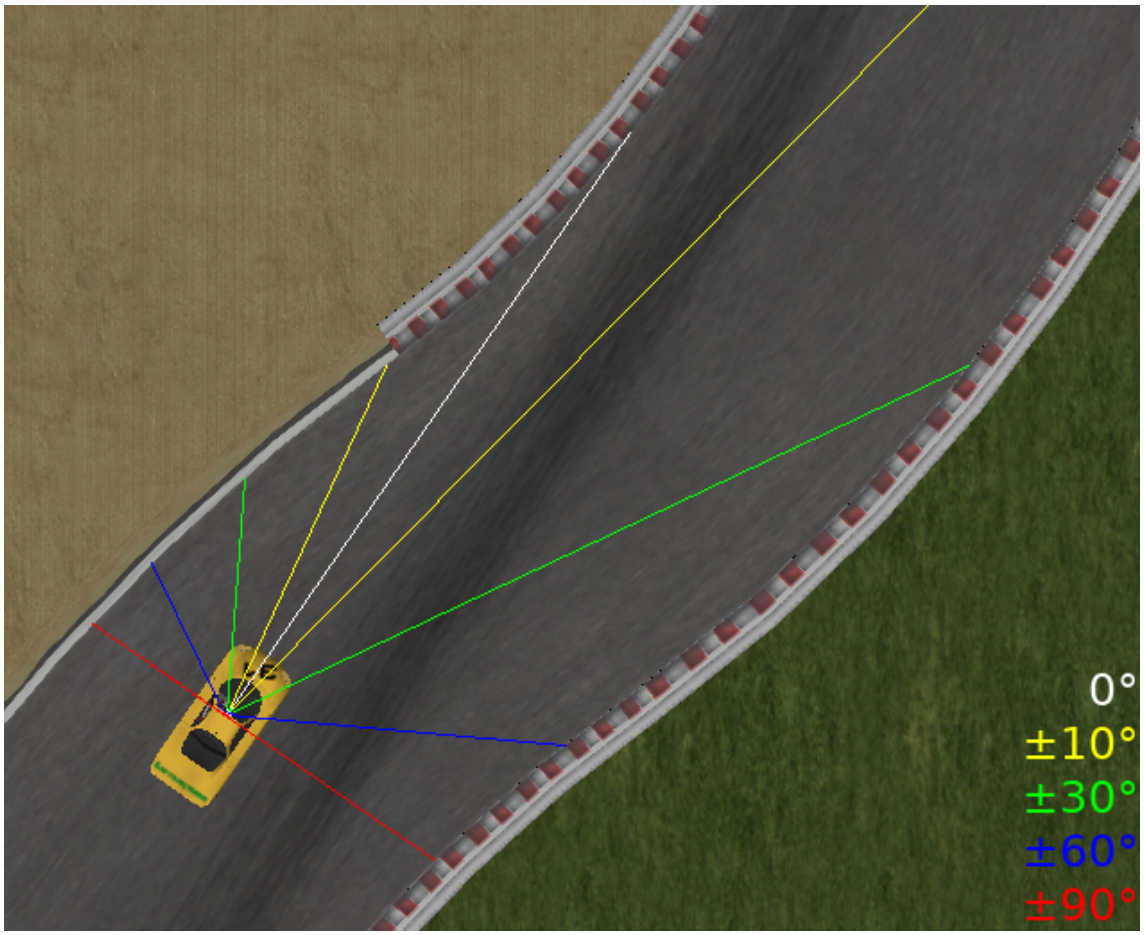
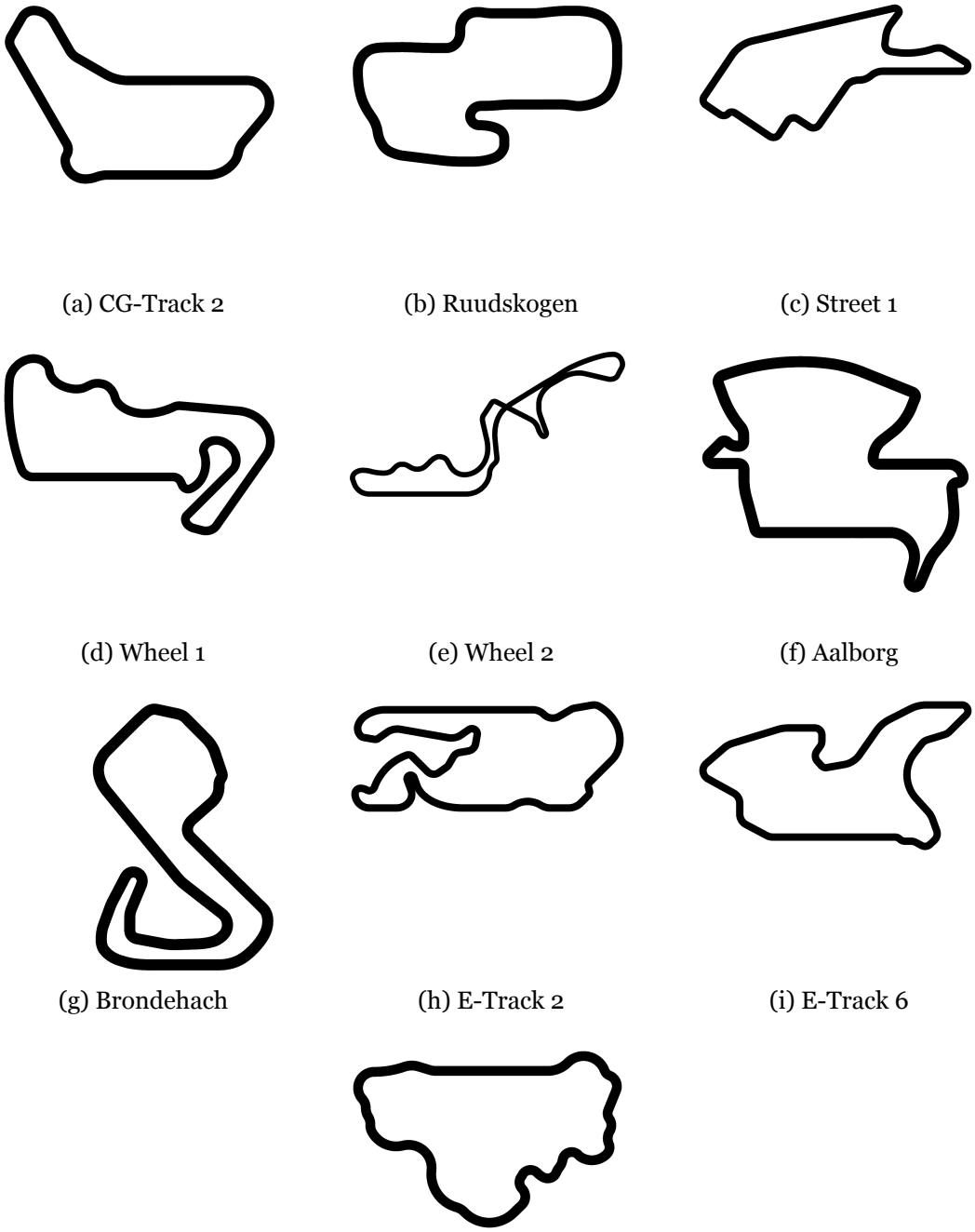


Figure 4.1: The distance sensors used for input to the ANN.



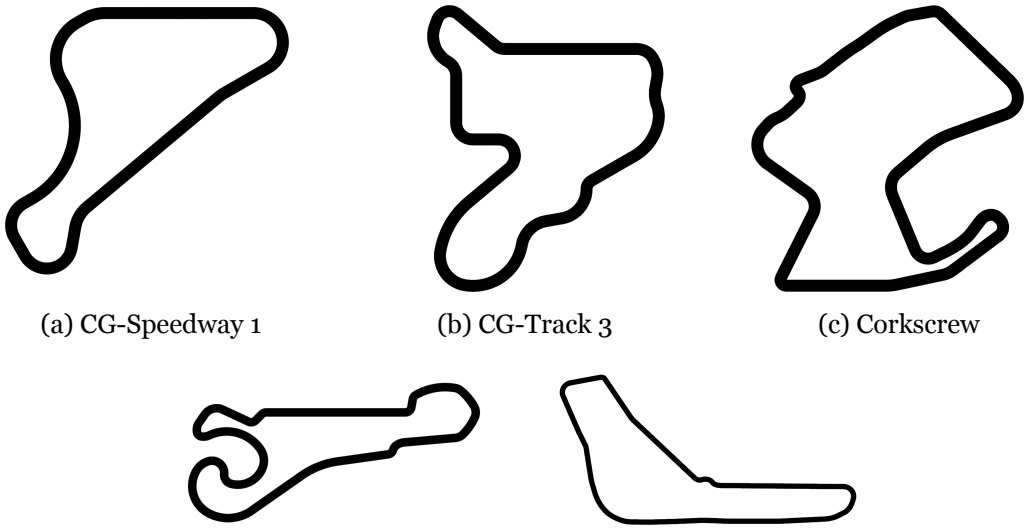
(a) CG-Track 2 (b) Ruudskogen (c) Street 1

(d) Wheel 1 (e) Wheel 2 (f) Aalborg

(g) Brondehach (h) E-Track 2 (i) E-Track 6

(j) E-Road

Figure 4.2: Training tracks



(a) CG-Speedway 1 (b) CG-Track 3 (c) Corkscrew

(d) E-Track 3 (e) Forza

Figure 4.3: Validation tracks

4.3 Automatic Transmission

An automatic transmission was implemented using a deterministic gear change policy with inspiration from an earlier work that implemented a human-like controller for the 2010 Simulated Car Racing Championship [16]. The policy can be found in Table 4.1, and it shows the speed or rpm of the car needed to shift gear. For instance, if the car is currently in third gear and its rpm and speed are 7000 and 140 respectively, it would gear up. To prevent oscillating gear changes, a reset timer prevents the gear to be changed within 1 second of the previous gear change. This, unfortunately, puts a limit to the gear shifting and its aggressiveness. However, since all experiments were run using the same transmission policy, the prerequisites are still equal for all.

4.4 Pre-Training

A dataset containing sensor observations and actuator actions by the expert was collected and used for pre-training the agents. The dataset that contained 50

Table 4.1: The gear changing policy used for all agents and experts.

Current Gear	*RPM*		Speed *km/h*	
	Gear Up	Gear Down	Gear Up	Gear Down
Reverse	0	-	0	-
Neutral	0	-	0	-
1	8500	-	30	-
2	8900	4000	80	30
3	9000	6000	130	80
4	9000	7000	180	130
5	9000	7500	230	180
6	-	8000	-	230

seconds of demonstrations for each of the 10 training tracks in Figure 4.2 was collected. In total, 500 seconds worth of demonstrations was used to pre-train the policies of all agents.

4.5 Expert Time

To measure the effectiveness of expert time, the three algorithms were trained on three different amounts of expert time on top of the pre-training. The three levels of expert time per track were: 50 seconds, 100 seconds and 200 seconds. In total, that amounts to 500 seconds, 1000 seconds and 2000 seconds worth of expert time. The amount of expert time per track was doubled each increment with the intent to better display a relationship between performance and expert time. The tracks were run and trained in the same order as in Figure 4.3 are presented.

4.6 Behavior Cloning

Since the BHC algorithm only requires labeled data, the expert time was spent by demonstrating each track an additional time without interference from the agent. The agent was then trained on the accumulated dataset after each track was demonstrated.

4.7 DAGGER

The DAGGER algorithm is based on the probability β_i that is slowly decreasing to give the agent more control of the trajectory taken. β_0 was initialized to 0.85 with the decreasing factor of 0.85 after each iteration to achieve a similar learning process as the one evaluated in the paper that presented HG-DAGGER [9]. The decreasing factor is multiplied with β_i at the end of each iteration to find β_{i+1}. For each track, the observations and actions by the expert were collected and aggregated into the training dataset, and subsequently used for training the agent.

4.8 HG-DAGGER

The HG-DAGGER agent is an ensemble of ANNs, and therefore the size of the ensemble had to be decided. To make sure that the results of such an agent would be positive, it was decided that the size of the ensemble would be 10, as presented in the HG-DAGGER paper [9]. According to the paper, it is also important to initiate the ANNs in the ensemble to different weights, and therefore it was done so in our implementation as well.

4.9 Validation

The experiment and validation of the thesis were conducted by running the agents on all validation tracks five times each, as seen in Figure 4.3. The decision to run each track multiple times was taken to ensure correctness because the calculations made in the TORCS physics engine are prone to approximation errors. The agents were left to run for 200 seconds on each track, and the distance measured was a point along the race line of the track, that the agent was positioned at when time ran out.

5 Result

In this chapter, results from the work conducted will be presented and graphs will be explained. In section 5.1, the tables containing the result from the experiments will be explained. In section 5.2, the graphs will be explained.

5.1 Tables

All results are available in tables B.1 to B.12, in appendix B.

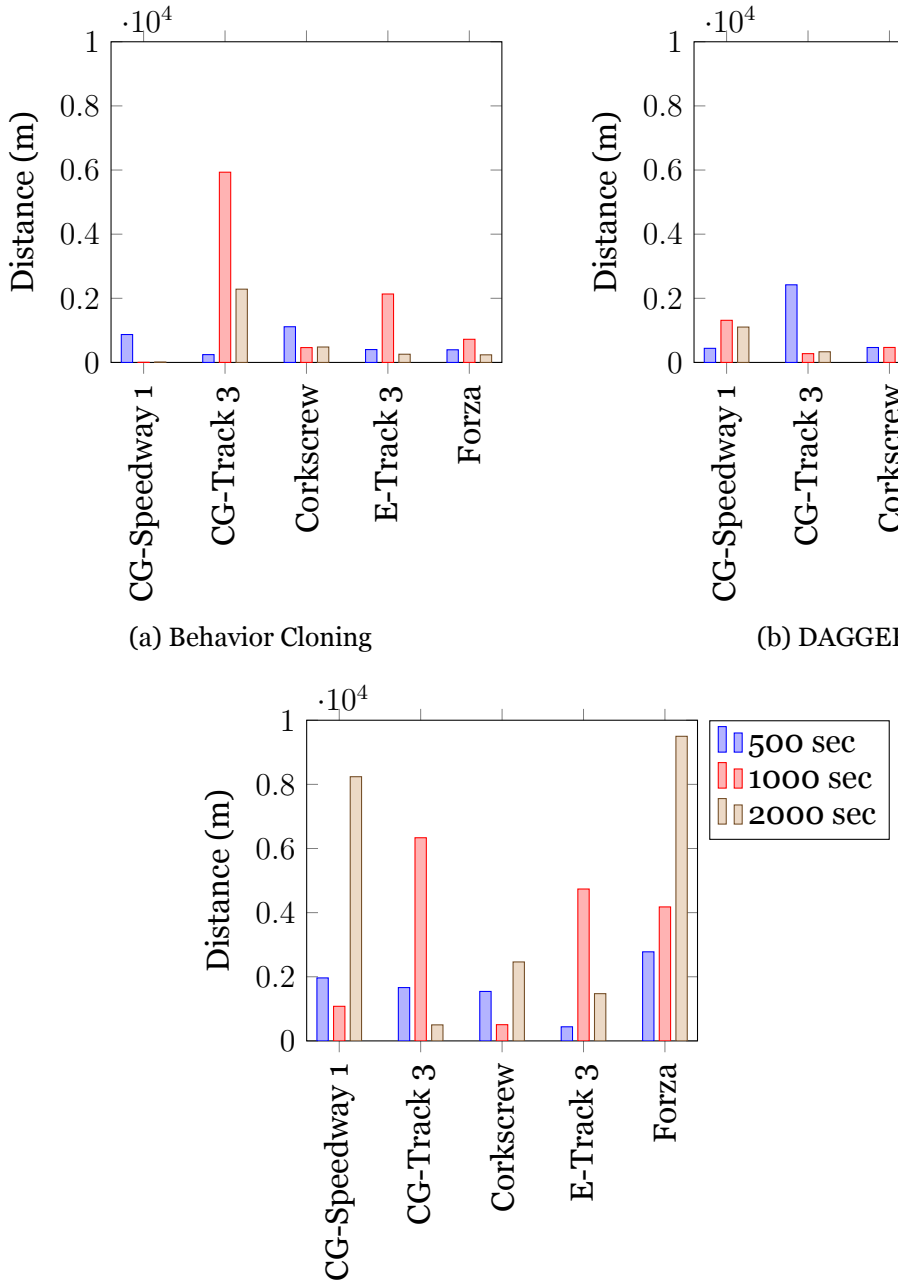
Tables B.1 to B.3 display the results of the BHC agents with three different levels of expert time. Table B.1 corresponds to 500 seconds of expert time while table B.3 corresponds to 2000 seconds of expert time. Each track was run five times each and the min, max, median and mean distances was calculated for each one. The results of DAGGER can be found in tables B.4 to B.6. The results of HG-DAGGER can be found in tables B.7 to B.9.

In table B.10 the distance traveled by the expert during 200 seconds is displayed. These distances are used for comparing the performance of agents against the expert that trained them.

Table B.11 displays the sum of the median distances for each of the algorithms for all three levels of expert time. The reason for choosing the median over mean is to prevent distortion of the distances by extrema. The median is an achieved result and therefore better coincides with the expected result. Table B.12 displays the relative distance of the algorithms compared to the sum of median expert distances.

5.2 Graphs

The graphs are displayed in Figures 5.1 to 5.3. The y-axis is in distance covered and the x-axis is the five different validation tracks in Figure 4.3, unless otherwise stated. The distances for all graphs are directly taken from Tables B.1 to B.12. The graphs are all using the median distances from the tables in appendix B.

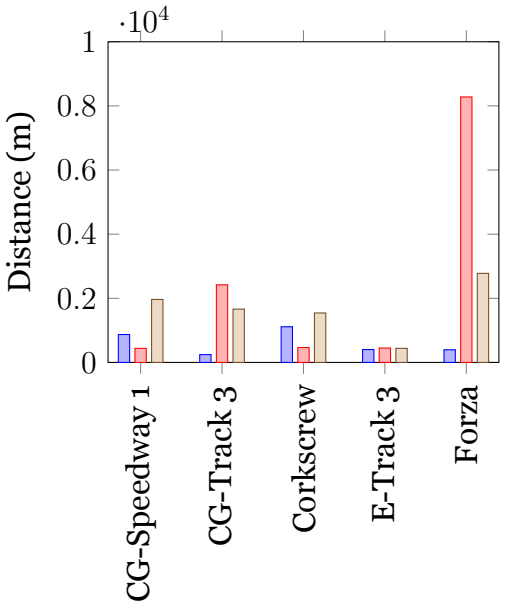


(a) Behavior Cloning

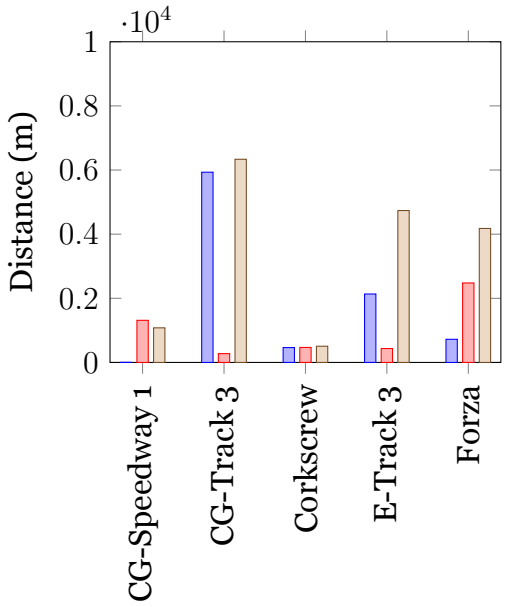
(b) DAGGER

(c) HG-DAGGER

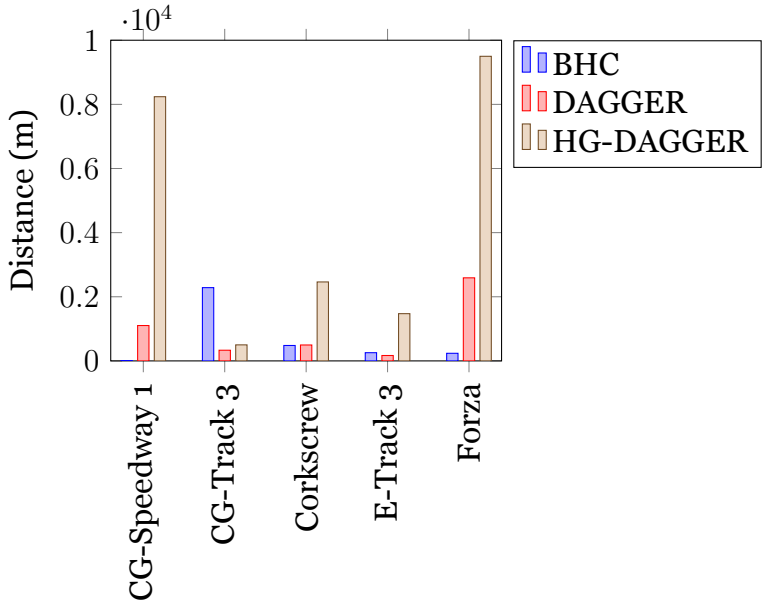
Figure 5.1: Graphs displaying the distance covered of an algorithm with three different levels of expert time, during 2000 seconds on the validation tracks in Figure 4.3.



(a) 0.5K sec expert time

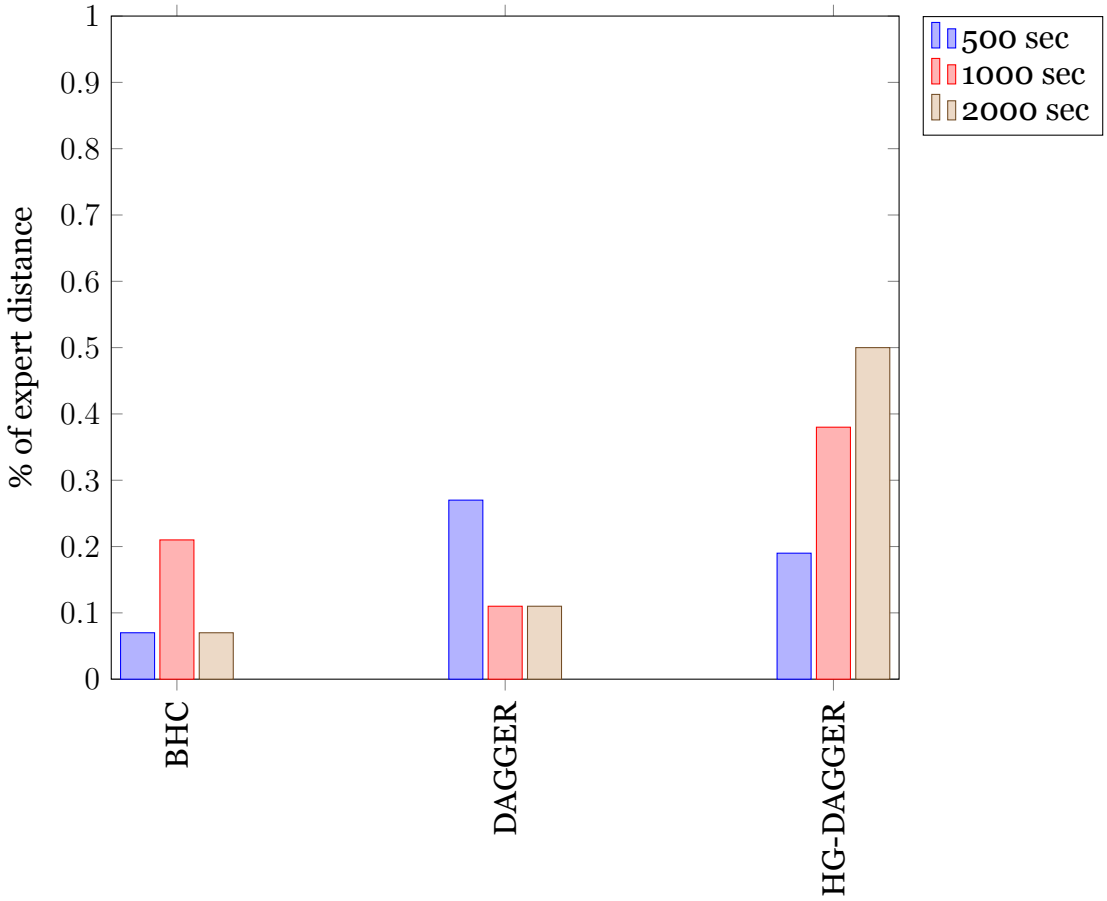


(b) 1K sec expert time



(c) 2K sec expert time

Figure 5.2: Graphs displaying the distance covered of all three algorithms, given the same amount of expert time, during 2000 seconds on the validation tracks in Figure 4.3.



(a) Performance

Figure 5.3: Graphs displaying the relative distance covered of an algorithm with three different levels of expert time, during 2000 seconds on the validation tracks in Figure 4.3.

5.2.1 Expert Time Comparisons

In Figure 5.1, three different graphs are displayed. Each of the three graphs contains distances from a specific algorithm and compares the different levels of expert time. Figure 5.1a displays the performance of the BHC agents with three different levels of expert time. The data is the median distances taken from Tables B.1 to B.3. Figure 5.1b displays the performance of the DAGGER agent with data taken from the median distances of B.4 to B.9. Figure 5.1c displays the performance of the HG-DAGGER agent with data from the median distances of Tables B.7 to B.9.

5.2.2 Algorithm Comparisons

The graphs in Figure 5.2 displays the performance of the three algorithms compared to each other, with the same expert time. In Figure 5.2a, the three algorithms with 500 seconds of expert time are compared to each other. The data is taken from the median distances of Tables B.1, B.4 and B.7. Figure 5.2b compares the three agents with 1000 seconds of expert time. The data is taken from the median distances of Tables B.2, B.5 and B.8. Figure 5.2c compares the three different agents with 2000 seconds of expert time. The data is taken from the median distances of Tables B.3, B.6 and B.9.

5.2.3 Performance Comparison

Figure 5.3 displays the relative distance of the three algorithms compared with the expert. It is the sum of the median distance covered on all validation tracks as a percentage of the sum of median distance covered by the expert. The data is taken from the median distances of Table B.12. The purpose of the graph is to give some insight into the absolute performance of the different algorithms.

5.3 Observations

In Figures 5.2b and 5.2c, the distance covered by the 1000 seconds and 2000 seconds HG-DAGGER agents are substantially better than the two other algorithms displayed. Out of 5 tracks, HG-DAGGER traveled the furthest on 4 of them, for both levels of expert time. One thing to note is that the performance of HG-DAGGER did not increase on all tracks with an increased amount of expert time, specifically on CG-Track 3 and E-Track 3, as can be seen in Figure 5.1c.

BHC seems to achieve its best performance with either 500 seconds or 1000 seconds of expert time, as can be seen in Figure 5.1a. A notable spike in distance is on CG-Track 3 with 1000 seconds of expert time. This spike is also present in Figure 5.1c with 1000 seconds of expert time, on the same track. Additionally, the track Forza seems to have a disproportionate distance covered for both DAGGER and HG-DAGGER with all levels of expert time, as can be seen in Figures 5.1c and 5.1b.

It can be observed in Figure 5.3 that HG-DAGGER is the best performing algorithm of the three evaluated. There is an increase in the relative distance, for each increase of expert time.

6 Conclusions

In this section, we will attempt to answer and discuss the central question of the thesis, based on the achieved results.

How should one implement IL in a simulated car racing environment, using human experts, to achieve the best performance when access to the experts is limited?

The central question of the thesis will be answered in section 6.1 while the discussion will be in section 6.2.

6.1 Conclusion

It is seen in Figure 5.3 that the agents only cover half the relative distance of the expert in the best case. The best performance was achieved by HG-DAGGER with 2000 seconds of expert time while the worst performance was achieved by BHC with 500 and 2000 seconds of expert time respectively. The bad performance of BHC was expected since it is well established that BHC preforms notoriously bad in sequential decision-making problems [18] [9].

An observation of the learning capabilities of the three different algorithms can be made with Figure 5.3. The only algorithm with a clear increase in performance with an increased amount of expert time is HG-DAGGER. The BHC and DAGGER agents both have trouble with increasing their performance according to the increase in expert time. The reasons for this will be discussed in section 6.2.

With the delimitations of this thesis in mind, the HG-DAGGER algorithm is by far the best option of the three algorithms presented, in the environment of this thesis. This result is of course entirely dependent on the delimitations and subject to change if any of the factors are changed. A comparison of related work is done in section 6.2.4 where a similar result to the one achieved in this thesis, has been achieved in a different environment. The result achieved in this thesis supports the results presented in the paper that introduced HG-DAGGER [9].

6.2 Discussion

The results achieved on the validation tracks certainly show that the agents are not on par with the human expert. The best result achieved is almost exactly half of the experts, which was achieved using HG-DAGGER in 2000 seconds of training. There exist many potential factors that contributed to this result and it will be discussed in section 6.2.1.

6.2.1 Potential Factors of Error

A different result might have been achieved if a different ANN architecture was used. However, due to the delimitations of this project, no attempt to find answers to this question was done.

If different levels of expert time were used, there might have been a different comparative result. An example of this can be directly observed if one focuses on the 500-second point, where DAGGER can be seen outperforming the other algorithms. It is not improbable that if a lower amount of training was used, such that the 500 seconds was the highest level of expert time, DAGGER might have been the algorithm with the best overall performance.

The approach to training the agents also needs to be discussed. A certain amount of pre-training was used, which was equal for all agents. However, presumably that the decisions made when designing the learning configuration make a difference. Perhaps a different amount of training or even a different type of training overall would have positively affected the result. But the delimitations of the thesis apply for this aspect of the project as well. It would not have been feasible for this thesis to investigate every variable with the potential to affect the result. Instead, the result can be seen as a general guideline for the relative performance of the different algorithms, with the same prerequisites.

Lastly, one has to take into consideration that the time constraint of the thesis. The most prominent symptom of the tight time constraint is the small sample size used in the validation of the agents. With only 5 runs per track, the data is not enough to make a statement or conclusion about the absolute performance

of the algorithms. This causes the data to possibly misrepresent the algorithm's general performance. However, since the result achieved in this thesis is very similar to previous work with the same comparison, it strengthens the validity of our conclusion that HG-DAGGER is the best performing IL algorithm, of the three.

6.2.2 Human Vision Versus Sensor Input

The result and conclusions of this thesis are naturally dependent on the performance of the expert. However, there is a substantial difference in how the agent and the expert perceive the environment. The expert is completely vision based and therefore can see beyond the next turn. The agent on the other hand only has access to 9 distance sensors and one speed sensor. This is limiting in the way that the agent is only able to observe the current turn and cannot prepare for the next turn. It can also cause the agent to become confused when training, since the expert's actions will not only depend on the current turn but also the next. This, in turn, causes conflicting training data. One example of a controller for TORCS using a vision-based input is from a paper written by Koutník et al. [12].

6.2.3 Validation Track Difficulty

An interesting observation can be made in Figure 5.2a, where a significant spike in performance can be observed on the track Forza with the DAGGER algorithm. It seems rather counter-intuitive. Why would it perform incredibly well on this specific track, with the least amount of training available, and then never perform at the same level when more training is introduced? A certain answer to this question cannot be given due to the nature of ML, however, an attempt to provide a few probable reasons for this anomaly will be made.

Over-training is a very common reason for performance decrease with an increased amount of training. However, over-training mostly occurs when the dataset is too small or too similar. This is not the case for the training approach taken for this thesis since more data points are added as the expert time level

increases. Therefore, it should not be over-training per se, yet it could be something related to the nature of over-training. When analyzing this anomaly, the characteristics of the track into consideration, which can be found in Figure 4.3, picture e. Forza is a very simple track with long straights and few curves, in comparison to the training tracks found in Figure 4.2. One possible explanation is that the track might not be well represented in the training data, and a solution to the track is not achieved through a generalization of the training maps. The small amount of training on the non-related maps to Forza would, therefore, allow for a more general solution for Forza, in a sense.

Another point to consider is the characteristics of the DAGGER algorithm. One observation made when training the ANN with DAGGER was the difficulty of making small corrections, like changing positioning on a straight, etc. If the expert was unfortunate and not given enough control because of the stochastic implementation, exaggerated turning data would create contradict the previous data and training. The paper that introduced HG-DAGGER brought up the issue of pilot-induced oscillations, when the pilot overcorrects the system in an attempt to stabilize an aircraft. This causes the aircraft control system to react in a more violent counteraction, leading to a greater response from the pilot in turn [9]. In reality, the expert only wanted to tweak the positioning of the car, but it could easily cause the agent to become confused when faced with a similar situation in the future. This is only one example of the problem with contradictory data, that occurs with the DAGGER implementation.

6.2.4 Validation of result

In the paper concerning the HG-DAGGER algorithm by Kelly et al., they perform a similar comparison of the same IL algorithms. Their implementation is an actual self-driving car and not in a simulator, but they still achieved very similar results to the ones presented in this thesis. It suggests that the results achieved in this thesis are not specific to our implementation, but a more general conclusion. Some points of similarity observed between the results are:

- HG-DAGGER scales the best with increased expert time.

- DAGGER and BHCs performance worsen at higher amounts of training.

6.3 Future Work

For future work, one could attempt to recreate our experiment but with different data, and perhaps widen the scope to more points of comparison. Preferably both higher and lower values than used in this thesis, to create a more general understanding of the performance of the algorithms. It would certainly be interesting to see how the performance of HG-DAGGER would evolve when given more training data.

The choice of topology in the ANN's is also something that needs a thorough investigation to find the optimal setup for the different algorithms. Perhaps they perform differently, relatively speaking, with different networks which therefore makes it more suitable to compare the algorithms with different ANN's rather than the same, as is done in our case.

References

[1] Bratko, Ivan, Urbančič, Tanja, and Sammut, Claude. “Behavioural cloning: phenomena, results and problems”. In: *IFAC Proceedings Volumes* 28.21 (1995), pp. 143–149.

[2] Community, Pygame. *Pygame*. 2018. url: https://www.pygame.org/wiki/about (visited on 04/24/2019).

[3] Daumé III, Hal. *A Course in Machine Learning*. Self-Published, 2013.

[4] Edwards, Chris X. *SnakeOil*. 2017. url: http://xed.ch/p/snakeoil/ (visited on 04/22/2019).

[5] Greenblatt, Jeffery B. and Shaheen, Susan. “Automated Vehicles, On-Demand Mobility, and Environmental Impacts”. In: *Current Sustainable/Renewable Energy Reports* 2.3 (Sept. 2015), pp. 74–81. issn: 2196-3010. doi: 10.1007/s40518-015-0038-5. url: https://doi.org/10.1007/s40518-015-0038-5.

[6] Håkansson, Anne. “Portal of Research Methods and Methodologies for Research Projects and Degree Projects”. In: *Proceedings of the International Conference on Frontiers in Education : Computer Science and Computer Engineering FECS’13*. QC 20131210. CSREA Press U.S.A, 2013, pp. 67–73. isbn: 1-60132-243-7. url: http://www.world-academy-of-science.org/worldcomp13/ws.

[7] Heuvel, Jeroen van den, Wiering, Marco A., and Kosters, Walter A. *Temporal exploration for reinforcement learning in continuous action spaces*. 2016.

[8] Hopfield, John J. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[9] Kelly, Michael et al. “HG-DAgger: Interactive Imitation Learning with Human Experts”. In: *CoRR* abs/1810.02890 (2018). arXiv: 1810.02890. url: http://arxiv.org/abs/1810.02890.

[10] Keras. *Keras*. 2019. url: https://keras.io/ (visited on 04/10/2019).

[11] Kingma, Diederik P and Ba, Jimmy. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[12] Koutnik, Jan et al. “Evolving large-scale neural networks for vision-based torcs”. In: (2013).

[13] Loiacono, Daniele, Cardamone, Luigi, and Lanzi, Pier Luca. *Simulated Car Racing Championship Competition Software Manual*. 2013.

[14] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. url: https://www.tensorflow.org/.

[15] Mirus, Florian. *Torcs-1.3.7-SCR-Patch*. 2018. url: https://github.com/fmirus/torcs-1.3.7 (visited on 04/23/2019).

[16] Muñoz, Jorge, Gutierrez, German, and Sanchis, Araceli. “A human-like TORCS controller for the Simulated Car Racing Championship”. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE. 2010, pp. 473–480.

[17] Rafiq, M.Y, Bugmann, G, and Easterbrook, D.J. “Neural network design for engineering applications”. In: *Computers & Structures* 79.17 (2001), pp. 1541–1552. issn: 0045-7949. doi: https://doi.org/10.1016/S0045-7949(01)00039-6. url: http://www.sciencedirect.com/science/article/pii/S0045794901000396.

[18] Ross, Stéphane, Gordon, Geoffrey, and Bagnell, Drew. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[19] Rossum, G. van. *Python tutorial*. Tech. rep. CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI), Apr. 1995.

[20] Schaal, Stefan. “Is imitation learning the route to humanoid robots?” In: *Trends in Cognitive Sciences* 3.6 (1999), pp. 233–242. issn: 1364-6613. doi: https://doi.org/10.1016/S1364-6613(99)01327-3. url: http://www.sciencedirect.com/science/article/pii/S1364661399013273.

[21] Silver, David et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. issn: 0036-8075. doi: 10.1126/science.aar6404. eprint: http://science.sciencemag.org/content/362/6419/1140.full.pdf. url: http://science.sciencemag.org/content/362/6419/1140.

[22] Sweden, Drive. *Drive Sweden*. 2019. url: https://www.drivesweden.net/ (visited on 04/03/2019).

[23] Swingler, Kevin. *Applying neural networks: a practical guide*. Morgan Kaufmann, 1996.

[24] Wymann, Bernhard. *car1-trb1*. 2017. url: http://www.berniw.org/trb/cars/car_view.php?viewcarid=5 (visited on 04/25/2019).

[25] Wymann, Bernhard et al. “Torcs, the open racing car simulator”. In: *Software available at http://torcs. sourceforge. net* 4.6 (2000).

[26] Yue, Yisong and Le, Hoang M. *Imitation Learning*. International Conference on Machine Learning Presentation. 2018.

[27] Zhang, Aston et al. *Dive into Deep Learning*. http://www.d2l.ai. 2019.

Appendices

Appendix - Contents

A TORCS sensors and actuators

Table A.1: Available sensors in the TORCS environment [13, pp. 13-14]

Name	Range (unit)	Description
angle	$[-\pi, +\pi]$ (rad)	Angle between the car direction and the direction of the track axis.
curLapTime	$[0, +\infty)$ (s)	Time elapsed during current lap
damage	$[0, +\infty)$ (point)	Current damage of the car (the higher is the value the higher is the damage).
distFromStart	$[0, +\infty)$ (m)	Distance from the start line along the track line
distRaced	$[0, +\infty)$ (m)	Distance covered by the car from the beginning of the race

focus	$[0, 200]$ (m)	Vector of 5 range finder sensors: each sensor returns the distance between the track edge and the car within the range of 200 meters. When noisy option is enabled sensors are affected by i.i.d. normal-noises with a standard deviation equal to the 1% of sensors range. The sensors sample, with a resolution of one degree, a five-degree space along a specific direction provided by the client (the direction is defined with the *focus* command and must be in the range [-90,+90] degrees w.r.t the car axis). Focus sensors are not always available: they can be used only once per second of simulated time. When the car is outside of the track (i.e. pos is less than -1 or greater than 1), the focus direction is outside the allowed range([-90,+90] degrees) or the sensors has been already used once in the last second, the returned values are not reliable (typically -1 is returned).
fuel	$[0, 200]$ (m)	Current fuel level
gear	$-1, 0, 1...6$	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 6.
lastLapTime	$[0, +\infty]$ (s)	Time to complete the last lap

opponents	$[0, 200]$ (m)	Vector of 36 opponent sensors: each sensor covers a span of 10 degrees within a range of 200 meters and return the distance of the closest opponent in the covered area. When noisy option is enabled, sensors are affected by i.i.d. normal noises with a standard deviation equal to the 2% of sensors range. The 36 sensors cover all the space around the car, spanning clockwise from -180 degrees up to +180 degrees with respect to the car axis.
racePos	$1, 2, ..., N$	Position in the race with respect to the other cars.
rpm	$[0, +\infty)$ (s)	Number of rotations per minute of the car engine.
speedX	$(-\infty, +\infty)$ (km/h)	Speed of the car along the longitudinal axis of the car.
speedY	$(-\infty, +\infty)$ (km/h)	Speed of the car along the transverse axis of the car.
speedZ	$(-\infty, +\infty)$ (km/h)	Speed of the car along the Z axis of the car.

track	$[0, 200]$	Vector of 19 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 meters. When noisy option is enabled sensors are affected by i.i.d. normal noises with a standard deviation equal to the 10% of sensors range. By default, the sensors sample the space in front of the car every 10 degrees, spanning clockwise with -90 degrees up to +90 degrees with respect to the car axis. However, the configuration of the range finder sensors (i.e., the angle w.r.t. to the car axis) can be set by the client once during initialization, i.e., before the beginning of each race. When the car is outside of the track (i.e., pos is less than -1 or greater than 1), the returned value is not reliable (typically -1 is returned)
trackPos	$(-\infty, +\infty)$	Distance between the car and the track axis. The value is normalized w.r.t the track width: it is 0 when car is on the axis, -1 when the car is on the right edge of the track and +1 when it is on the left edge of the car. Values greater than 1 or smaller than -1 mean that the car is outside of the track.
wheelSpinVel	$[0, +\infty]$ (rad/s)	Vector of 4 sensors representing the rotation speed of wheels.
z	$(-\infty, +\infty)$ (m)	Distance of the car mass center from the surface of the track along the Z axis.

Table A.2: Available actuators in the TORCS environment [13, p. 14]

Name	Range (unit)	Description
accel	$[0, 1]$	Virtual gas pedal (0 means no gas, 1 full gas).
brake	$[0, 1]$	Virtual brake pedal (0 means no brake, 1 full brake).
clutch	$[0, 1$	Virtual clutch pedal (0 means no clutch, 1 full clutch).
gear	$-1, 0, 1, ..., 6$	Gear value.
steering	$[-1, 1]$	Steering value: -1 and +1 means respectively full right and left, that corresponds to an angle of 0.366519 rad.
focus	$[-90, 90]$	Focus direction in degrees.
meta	$0, 1$	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

B Experiment Data

Table B.1: Distance that BHC agent with 500 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

BHC 0.5K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	870.37	244.93	1377.68	401.41	89.70
RUN 2	1537.64	243.79	1367.67	401.38	394.90
RUN 3	925.93	265.88	1077.56	401.32	394.59
RUN 4	263.63	243.58	914.49	401.06	91.24
RUN 5	266.09	243.73	1112.99	401.23	394.90
MIN	263.63	243.58	914.49	401.06	89.70
MAX	1537.64	265.88	1377.68	401.41	394.90
MEDIAN	870.37	243.79	1112.99	401.32	394.59
MEAN	772.73	248.38	1170.08	401.28	273.07

Table B.2: Distance that BHC agent with 1000 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

BHC 1K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	1.46	5932.29	463.51	2255.67	722.98
RUN 2	1.26	5939.45	1352.02	2133.93	717.38
RUN 3	1.26	4306.36	463.08	2108.92	721.79
RUN 4	1.26	5944.48	463.12	2023.74	721.47
RUN 5	1.26	5798.56	457.65	2246.99	724.42
MIN	1.26	4306.36	457.65	2023.74	717.38
MAX	1.46	5944.48	1352.02	2255.67	724.42
MEDIAN	1.26	5932.29	463.12	2133.93	721.79
MEAN	1.30	5584.23	639.88	2153.85	721.61

Table B.3: Distance that BHC agent with 2000 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

BHC 2K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	10.39	2724.75	498.26	256.72	237.27
RUN 2	9.44	2284.83	480.56	255.90	235.24
RUN 3	10.83	2281.10	480.07	255.90	237.27
RUN 4	10.21	2759.88	480.89	254.94	238.16
RUN 5	11.01	2281.39	498.54	255.90	237.27
MIN	9.44	2281.10	480.07	254.94	235.24
MAX	11.01	2759.88	498.54	256.72	238.16
MEDIAN	10.39	2284.83	480.89	255.90	237.27
MEAN	10.38	2466.39	487.66	255.87	237.04

Table B.4: Distance that DAGGER agent with 500 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

DAGGER 0.5K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	448.32	2420.77	465.39	449.39	8333.44
RUN 2	439.76	1300.09	465.47	448.73	8298.11
RUN 3	447.71	1306.12	465.44	447.94	8279.47
RUN 4	434.16	2432.55	465.66	447.38	2859.13
RUN 5	438.55	2458.27	465.23	448.75	3885.19
MIN	434.16	1300.09	465.23	447.38	2859.13
MAX	448.32	2458.27	465.66	449.39	8333.44
MEDIAN	439.76	2420.77	465.44	448.73	8279.47
MEAN	441.70	1983.56	465.44	448.44	6331.07

Table B.5: Distance that DAGGER agent with 1000 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

DAGGER 1K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	1309.60	419.05	466.83	433.38	2507.56
RUN 2	1315.74	19.05	471.04	433.37	2477.08
RUN 3	1314.42	274.12	467.41	433.37	2478.08
RUN 4	1305.04	19.04	475.23	433.45	5086.34
RUN 5	1313.78	274.17	466.84	433.38	2476.04
MIN	1305.04	19.04	466.83	433.37	2476.04
MAX	1315.74	419.05	475.23	433.45	5086.34
MEDIAN	1313.78	274.12	467.41	433.38	2478.08
MEAN	1311.72	201.09	469.47	433.39	3005.02

Table B.6: Distance that DAGGER agent with 2000 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

DAGGER 2K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	1103.08	332.95	496.62	165.44	2593.79
RUN 2	1108.30	332.97	496.56	168.05	2550.35
RUN 3	1107.86	335.96	496.59	168.34	2772.05
RUN 4	1100.73	332.85	496.56	468.20	2590.57
RUN 5	1102.10	332.90	496.43	466.38	2545.18
MIN	1100.73	332.85	496.43	165.44	2545.18
MAX	1108.30	335.96	496.62	468.20	2772.05
MEDIAN	1103.08	332.95	496.56	168.34	2590.57
MEAN	1104.41	333.53	496.55	287.28	2610.39

Table B.7: Distance that HG-DAGGER agent with 500 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

HG-DAGGER 0.5K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	1965.37	1666.70	1542.64	438.85	5102.42
RUN 2	1962.88	1663.34	1542.61	440.01	2776.73
RUN 3	1968.66	1662.82	1542.52	440.00	2774.71
RUN 4	1964.44	1660.52	1542.66	440.17	2777.88
RUN 5	1960.94	1661.03	1540.15	440.35	3138.74
MIN	1960.94	1660.52	1540.15	438.85	2774.71
MAX	1968.66	1666.70	1542.66	440.35	5102.42
MEDIAN	1964.44	1662.82	1542.61	440.01	2777.88
MEAN	1964.46	1662.88	1542.12	439.88	3314.10

Table B.8: Distance that HG-DAGGER agent with 1000 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

HG-DAGGER 1K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	1078.98	6335.57	506.60	4736.72	8108.11
RUN 2	1056.78	6329.21	506.50	4684.04	4178.30
RUN 3	1056.03	6336.20	505.82	7324.76	8183.41
RUN 4	5217.70	1855.23	505.79	472.33	4171.90
RUN 5	3112.62	6340.88	504.59	7398.01	2567.23
MIN	1056.03	1855.23	504.59	472.33	2567.23
MAX	5217.70	6340.88	506.60	7398.01	8183.41
MEDIAN	1078.98	6335.57	505.82	4736.72	4178.30
MEAN	2304.42	5439.42	505.86	4923.17	5441.79

Table B.9: Distance that HG-DAGGER agent with 2000 sec of expert time covered on validation tracks in Figure 4.3 during 2000 sec.

HG-DAGGER 2K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	8239.90	502.63	2463.70	1466.43	9350.27
RUN 2	8246.25	164.20	2462.18	1452.30	9509.90
RUN 3	8233.34	159.35	2462.32	1479.57	9525.46
RUN 4	8238.92	505.23	2460.15	1473.35	3866.06
RUN 5	8223.52	500.09	2462.32	1473.27	9499.01
MIN	8223.52	159.35	2460.15	1452.30	3866.06
MAX	8246.25	505.23	2463.70	1479.57	9525.46
MEDIAN	8238.92	500.09	2462.32	1473.27	9499.01
MEAN	8236.39	366.30	2462.13	1468.98	8350.14

Table B.10: Distance that the expert covered on validation tracks in Figure 4.3 during 2000 sec.

EXPERT 2K sec	CG-Speedway 1	CG-Track 3	Corkscrew	E-Track 3	Forza
RUN 1	9221.68	8066.86	7723.20	8107.72	11151.36
RUN 2	9342.13	7659.33	7654.91	7958.30	11297.87
RUN 3	8945.60	8143.50	7549.50	7991.59	10957.70
RUN 4	9111.62	8021.90	7891.47	8276.24	11432.71
RUN 5	9150.90	8267.30	7566.78	8158.78	11249.11
MIN	8945.60	7659.33	7549.50	7958.30	10957.70
MAX	9342.13	8267.30	7891.47	8276.24	11432.71
MEDIAN	9150.90	8066.86	7654.91	8107.72	11249.11
MEAN	9154.39	8031.78	7677.17	8098.53	11217.75

Table B.11: Sum of median distance for all three algorithms with three different levels of expert time.

Sum of median values	BHC	DAGGER	HG-DAGGER
0.5K	3023.06	12054.17	8387.76
1K	9252.39	4966.77	16835.39
2K	3269.28	4691.50	22173.61

Table B.12: The median relative distance covered of all three algorithms in regard to the expert on the validation tracks in Figure 4.3.

Relative Distance	BHC	DAGGER	HG-DAGGER
0.5K	0.07	0.27	0.19
1K	0.21	0.11	0.38
2K	0.07	0.11	0.50

TRITA-EECS-EX-2019:255