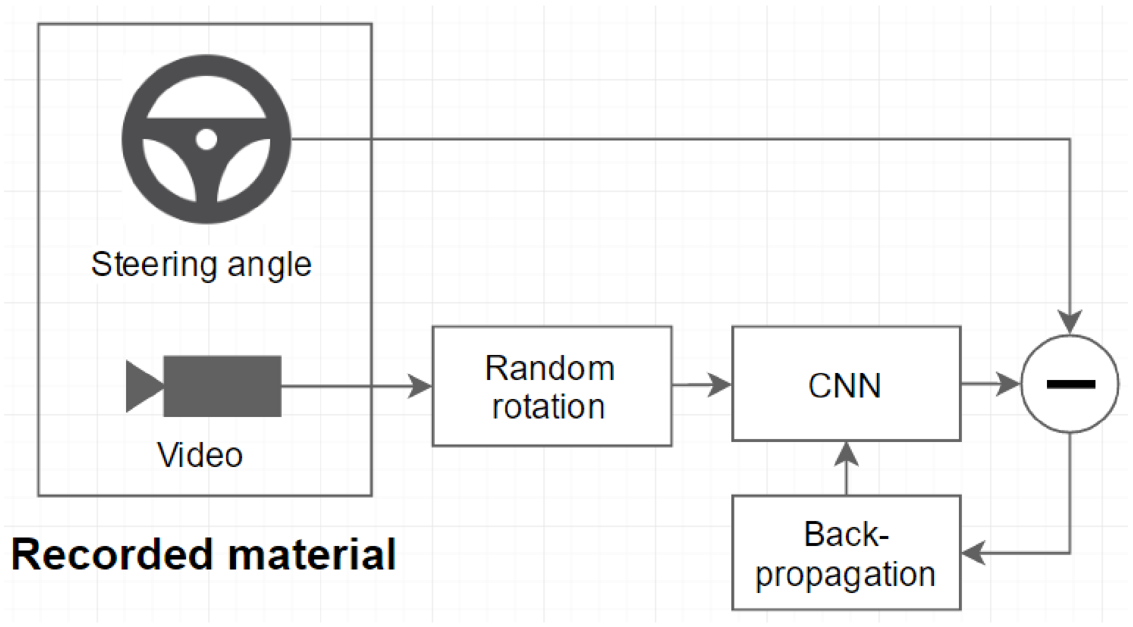
CHALMERS
UNIVERSITY OF TECHNOLOGY



Combining Deep Learning with traditional algorithms in autonomous cars

An introductory exploration of combining some deep learning techniques with computer vision based algorithms for decision making in autonomous vehicles

Bachelor's thesis in Computer Science

Albin Falk
David Granqvist

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Bachelor's thesis 2017

Combining Deep Learning with traditional algorithms in autonomous cars

An introductory exploration of combining some deep learning techniques with computer vision based algorithms for decision making in autonomous vehicles

Albin Falk & David Granqvist

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Combining Deep Learning with traditional algorithms in autonomous cars
An introductory exploration of combining some deep learning techniques with computer vision based algorithms for decision making in autonomous vehicles
Albin Falk
David Granqvist

© Albin Falk & David Granqvist, 2017.

Supervisors:
Erland Holmström, Department of Computer Science and Engineering
Alixander Ansari, Sigma Technology

Examiner: Peter Lundin, Department of Computer Science and Engineering

Bachelor's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology / University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Cover: Representation of the adapted *Nvidia model's* learning process.

Typeset in LaTeX
Gothenburg, Sweden 2017

Combining Deep Learning with traditional algorithms in autonomous cars
An introductory exploration of combining some deep learning techniques with computer vision based algorithms for decision making in autonomous vehicles

Albin Falk & David Granqvist
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

Research of autonomous technologies in modern vehicles are being conducted as never before. For a long time, traditional computer vision based algorithms has been the primary method for analyzing camera footage, used for assisting safety functions, where decision making have been a product of manually constructed behaviours. During the last few years deep learning has demonstrated its extraordinary capabilities for both visual recognition and decision making in end-to-end systems. In this report we propose a solution of introducing redundancy by combining deep learning methods with traditional computer vision based techniques for minimizing unsafe behaviour in autonomous vehicles.

The system consists of a computer vision based lane detection algorithm in combination with a fully connected *Deep Neural Network*, and combines the advantages of both technologies by constructing a control algorithm responsible for consolidating the sub systems calculations of the correct steering angle, used to keep the vehicle within the lane markings of the road.

The solution proposed show that we can increase the performance of our system by applying a combination of the two technologies in a simulator resulting in a safer system than we could achieve with the technologies separately.

Keywords: deep learning, computer vision, lane detection, autonomous vehicles, deep neural networks.

Deep Learning kombinerat med traditionella algoritmer i autonoma bilar
Introduktion och utforskning av hur några tekniker inom deep learning kan kombineras med computer vision-baserade algoritmer för beslutsfattande i autonoma fordon

Albin Falk & David Granqvist
Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet

Sammanfattning

Forskning av autonoma teknologier i moderna fordon bedrivs i en allt ökande grad. Under lång tid har traditionella algoritmer för datorseende varit den primära metoden för att analysera kameramaterial, som används för att bistå säkerhetsfunktioner, där beslutsfattandet har varit en produkt av manuellt konstruerade beteenden. Under de senaste åren har djupt lärande visat sig ha extremt bra förmågor för både visuell igenkänning och beslutsfattande i system av typen *end-to-end*. I denna rapport föreslår vi en lösning för att införa redundans genom att kombinera *deep learning* med traditionellt datorseende för att minimera osäkert beteende i autonoma fordon.

Systemet består av en *computer vision*-algoritm för vägfils-lokalisering i kombination med ett helt anslutet *Deep Neural Network*. Systemet kombinerar fördelarna med båda teknikerna genom att bygga en kontrollalgoritm som är ansvarig för att konsolidera delsystemberäkningarna av rätt styrvinkel som används för att hålla fordonet inom körfältet.

Den föreslagna lösningen visar att vi kan öka systemets prestanda genom att tillämpa en kombination av båda teknikerna i en simulator, vilket resulterar i ett säkrare system än vi kunde uppnå med teknikerna separat.

Nyckelord: deep learning, computer vision, vägfils-lokalisering, autonoma fordon, deep neural networks.

Foreword

This report was written as part of our bachelor's thesis in Computer Science at Chalmers University of Technology, Department of Computer Science and Engineering, as the final part of our bachelor's degree. This project was part of a larger cross-functional project, the *Sigma Project*, where the goal was to build an autonomous miniature car. The Sigma Project was initiated by Sigma Technology and consisted of eight students from multiple disciplines working towards a common goal. The goal of this project was to research and develop safe decision making for a miniature car by using computer vision and deep neural networks. Albin Falk was mainly responsible for developing the simulator and communication, whereas David Granqvist was mainly responsible for developing the deep neural networks.

Albin Falk & David Granqvist, Gothenburg, June 2017

Acknowledgements

Firstly, we would like to thank Alixander Ansari and Sigma Technology for providing us with this super interesting and relevant cross-functional project. Alixander was our supervisor at Sigma and his engagement and support during the project has been very valuable. We would also like to thank all members of the project group that we have been working with including Jesper Rask, Daniel Posch, Vishnu Shyam, Paul Lazar, Anders Sundkvist and Andreas Jakobsson. Furthermore, we would also like to thank our supervisor at Chalmers, Erland Holmström, for his dedication and input to the project.

Finally, we would both like to thank our family and friends for their continuous support throughout the project.

Albin Falk & David Granqvist, Gothenburg, June 2017

Contents

1 Introduction **1**

2 Theory **3**
2.1 Artificial Intelligence 3
2.2 Autonomy & Safety 8

3 Hardware & Software **11**
3.1 Hardware 11
3.2 Software 12

4 Methods **15**

5 Implementation **19**

6 Results and Discussion **27**
6.1 Results 27
6.2 Discussion 27

References **31**

Acronyms **33**

Glossary **35**

A Tools **I**

1 Introduction

Background

Autonomous technologies is a highly attractive research field within computer science and it consists of many areas, where artificial intelligence and machine learning are at the forefront.

During the last few years Deep Learning (DL) has demonstrated its extraordinary computational potential by transcending its abilities into more and more complex areas, where pattern matching, image recognition and complex game play describes a small selection of examples. This newly demonstrated computational potential makes deep learning a highly desirable field of knowledge, competence and research for many companies within many different fields of work.

An area that has started to express great interest for this technology is the automotive industry, where deep learning is being applied as a mean of enabling autonomy in vehicles. The benefit of autonomous cars are many, including less emissions due to traffic flow, higher utilization due to car fleets, and safety due to less human errors. As one of Sweden's largest consultant companies, with consultants working closely with the automotive industry, this technology presents a highly valuable area of interest and competence for Sigma Technology and its partners. Therefore an introduction to research regarding autonomous technologies and safety requirements for vehicles that apply the technology is a highly valuable asset.

Purpose

The main purpose of this project is to investigate how a combination of traditional computer vision techniques and modern Deep Neural Networks can minimize unsafe behaviour in autonomous vehicles.

The intended result will consist of a distributed system with multiple components working in synergy, including a *control server*[1] and multiple *control units*[2] with a supporting communication system. It will also consist of a customized simulator and a *Deep Neural Network (DNN)* modified for this project. Furthermore, an evaluation of the solution and its performance will also be presented.

[1]Central server making the autonomous decisions
[2]Unit responsible for controlling the vehicle and collecting telemetry

Cross-functional project

This thesis project is part of a larger cross-functional project at Sigma Technology, referred to as the *Sigma Project* throughout this report. The Sigma Project group consists of the following four teams with two members in each team: *Computer Vision (CV)*, *Product Development* (PD), *Mechatronics* (MT) and this team, *Decision Making* (DM). The common goal for the Sigma Project was to build an autonomous miniature car.

Delimitations

Since the Sigma project is cross-functional, it's very important to separate the responsibilities. This project is solely focused on software development and our results will therefore not include anything related to hardware, that area will instead be part of the MT project. The results will also not consist of anything related to computer vision, it will instead be part of the CV project. Furthermore, the simulated and physical car will only be functional in the environment used during development where there is good lighting conditions and clearly visible roads without too much noise.

The software of this project will be developed using many existing frameworks and some of the code will be continued work on existing open source projects. This project will not be focused on performance so the result will not be optimized, this applies to the simulator, communication and DNN. The DNN will only be trained towards a level of accuracy sufficient for enabling the vehicle to follow lanes in a slow speed.

2
Theory

The theory consists of machine learning fundamentals, neural network theory, autonomous car principles and safety requirements within the automotive industry. The theory behind this thesis has had a heuristic focus, and has therefore mainly been focused on knowledge about modern frameworks for building deep neural networks.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is a field within computer science which defines itself as the study of "intelligent agents". It is a broad field with many subsets, such as machine learning for example [1].

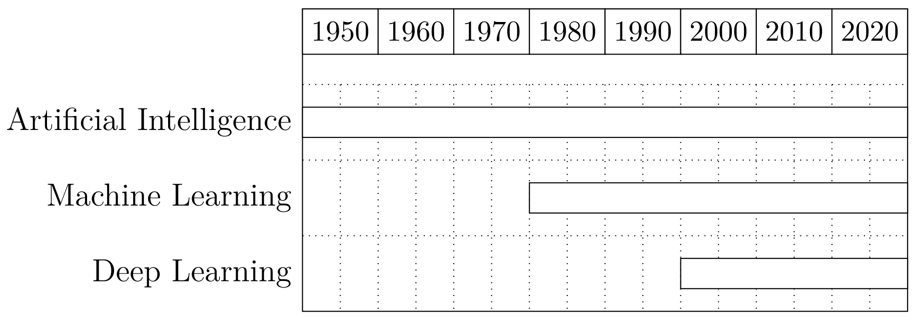


Figure 2.1: Artificial intelligence timeline

Machine Learning

Machine Learning (ML) is a dominant field within artificial intelligence and it's commonly defined as a "field of study that gives computers the ability to learn without being explicitly programmed". The foundation of ML is to construct algorithms that can learn and act upon given data sets [2].

Machine learning can be divided into two primary categories, supervised and unsupervised learning. In addition it also have a less commonly known category called *Reinforcement learning* [3].

Supervised learning The goal of supervised learning is to learn a mapping between inputs to outputs, that yields the desired output for all inputs used in a set of labeled input/output-pairs [3]. A supervisor is used to provide the correct values of the output, used to compare with what the actual output of the mapping is. By having a supervisor present, the mapping can be adjusted towards the correct input to output mapping [4].

Unsupervised learning In unsupervised learning, the supervisor used in supervised learning is not present, and the only thing that is accessible is the input data [4]. The goal of unsupervised learning is to discover structures of interest based on the data, which is done by finding regular patterns in the input data. The learning consists of an unconditional *density estimation*, i.e. a contexture of certain patterns appearing with higher frequency than other patterns from the input data [4], which is more generalizeable and complex than the supervised approach[3].

Reinforcement learning Reinforcement learning can be described as supervised learning, where an intelligent agent makes decision while receiving rewards or penalties based on the outcome of its decisions. The agent finds the decisions that maximizes its reward by running the network repeatedly, and thereby creating the networks *policy* for learning [4]. A single incorrect action is of minior importance in reinforcement learning algorithms, it's instead a sequence of correct actions that is the desired behaviour of the algorithm [4].

End to End Learning In End to End Learning all steps required to obtain the resulting output of the Artificial Neural Network (ANN) is carried out within the network. An ANN that applies end to end learning doesn't have any intermediate steps or sub tasks performed outside of the network [5].

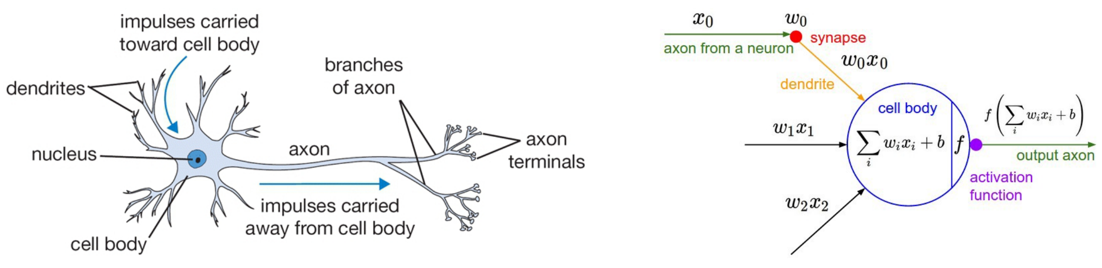
Neural Networks

An Artificial Neural Network(ANN) is a network composed of several simple processing units, referred to as neurons, that is interconnected between each other [6].

Each neuron in the ANN produces a sequence of activations that propagates through the network, activating neurons by weighted connections to itself and other previously activated neurons in the network. The output neurons of the ANN may influence the surrounding environment by triggering actions, where learning of the network is perceived when the weights of the network are adjusted in such a way that the ANN approaches the desired behaviour when these actions are triggered [6].

The construct of a neural network and its structures takes inspiration from neuroscience, where the neuron in an ANN is a simplified model of the biological neurons in the brain [7].

Feed-forward Neural Network A feed-forward neural network is defined as a neural network where the connections only go in one direction; from the current layer to the adjacent forward layer, but not the other way around [9]. It distinguishes itself from *recurrent neural networks* by not having any feedback loops in its architecture.



(a) Biological neuron (b) Artificial neuron

Figure 2.2: Comparison of biological and artificial neurons, From [8]. Reproduced with permission.

Fully-connected Neural Network A fully-connected layer is defined as a network where every unit is connected to every unit of its adjacent forward layer [9].

Loss function A loss function is used in supervised learning for calculating the difference between the output of an ANN and the correct output, specified by the supervisor.

Gradient Optimization The value of the loss function defines a search direction based on the ANN's weights, that the weights are shifted towards. The size of the shift is determined by a parameter known as the learning rate [10].

Backpropagation Backpropagation applies the gradient optimization by propagating backwards through the network, adjusting the weight of the network to compensate for the error calculated by the loss function [11].

Convolutional Neural Networks

A Convolutional Neural Network (CNN) have many similar features of regular neural networks, it consist of processing units that apply a process of learning that uses weights and biases[1] and process data by adjusting its connections. In some cases the network thereafter sends the processed data through a squashing non-linearity that maps the output value to a specified range, more commonly known as a squashing function. It also employs a loss function as a measurement for observing and optimizing the learning rate of the network [10].

What separates a CNN from a regular ANN is the fact that they are created specifically for processing data represented as multi-dimensional arrays. Since a common way of representing image data is by using an array of the dimensions height * width * 3[2]; CNN's are often used for processing this type of data, when using neural networks [12].

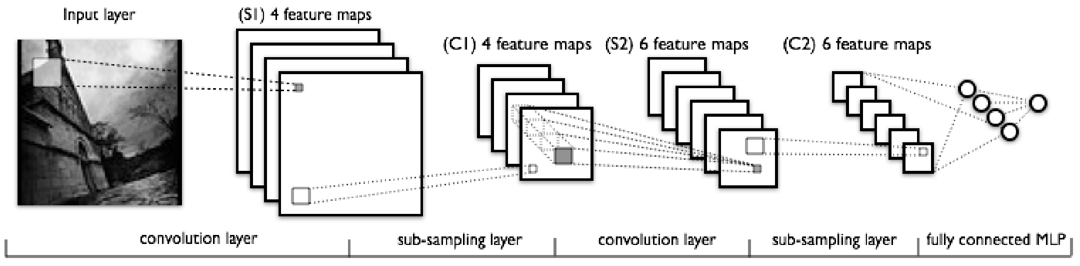
[1]Biases are neurons with constant values, used for shifting the curve of the loss function to the left or the right

[2]RGB channels

The structure of a CNN consist of a series of stages with an input layer, an output layer and one or more convolutional layer, pooling layer, squashing non-linearities and fully connected layers [12].

Deep Learning

Deep learning is based on observations of how the brain processes information, where it is believed that each level in the brain is learning features at increased abstraction levels [3]. The goal of deep learning in a machine learning context is to mimic this behaviour in the form of a computer architecture [3].



(a) Convolutional Neural Network

Figure 2.3: Image representing a Convolutional Neural Network, each layer introduces a higher level of abstraction, From [13]. Reproduced with permission.

Deep Neural Networks Deep Neural Networks (DNN's) are a sub-field of deep learning where one of two main structures is a *multi-layer perceptron* and the the other main structure is a *deep auto-encoder*. A multi-layer perceptron is more formally described as a deep and fully-connected feed-forward neural network using a supervised approach [3] and a deep auto-encoder is more formally described as a feed-forward neural network with the goal of predicting the input to the network. The deep auto-encoder takes a approach that is more closely related to unsupervised learning [3].

The Nvidia Model

The NVIDIA model describes the architecture of a CNN proposed by researchers at *Nvidia* in the paper *End to End Learning for Self-Driving Cars* [14]. The authors of the paper proposes that this architecture demonstrates the advantages of using end to end learning by optimizing all processing steps of the network simultaneously, while generating smaller systems by having self-optimization of the systems components, rather than having criteria selected by humans, which does not always guarantee the best performance of the system [14].

The architecture consists of 9 layers in total, consisting of one normalization layer followed by five convolutional layers and three fully connected layers.

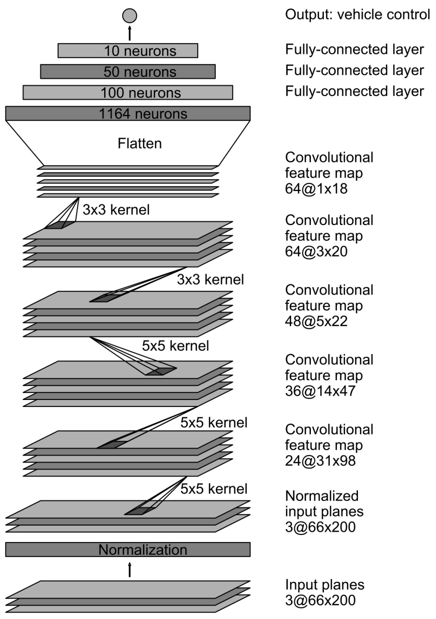
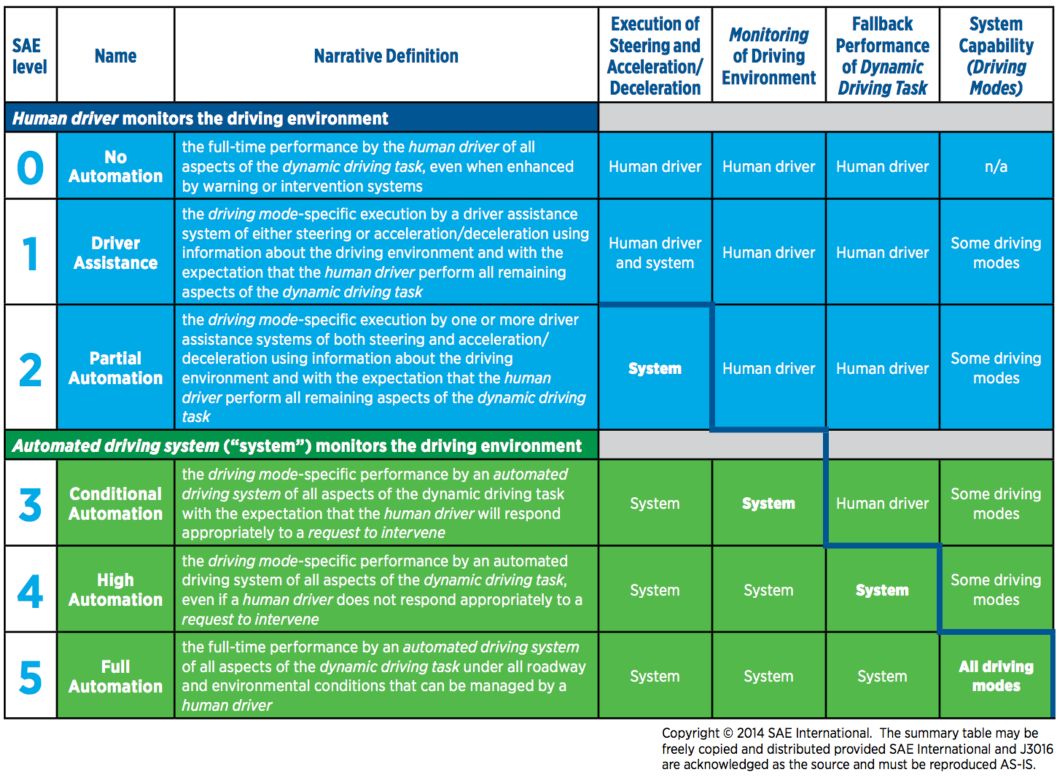


Figure 2.4: Layers in the NVIDIA model, From [15]. Reproduced with permission.

2.2 Autonomy & Safety

A Automated Drive System (ADS) can be structured in many different ways and have significant differences in the level of automation. Therefore, the SAE (The International Society of Automotive Engineers) presented the classification standard J3016, in 2014, that identifies six levels of automation, ranging from no to full automation [16].



SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Copyright © 2014 SAE International. The summary table may be freely copied and distributed provided SAE International and J3016 are acknowledged as the source and must be reproduced AS-IS.

Figure 2.5: The six levels of automation, From [17]. Reproduced with permission.

The J3016 standard defines the different levels of automation based on the automated features of the vehicle, as well as to clarify what role the driver have with regards to the operational parts of controlling the vehicle, for each level of automation. The standard also introduces definitions commonly used when discussing autonomous vehicles, such as *the dynamic driving task*[3], and *driving mode*[4].

Safety Requirements

The international standard *ISO 26262* is a widely accepted safety standard of software in the automotive industry. The standard consists of ten main parts detailing different areas of the development process [18].

As an important part of *ISO 26262* ASIL (Automotive Safety Integrity Level) provides a way of classifying the safety risks of an automotive system in an abstract way

[3]The operational and tactical aspects of the driving task (strategic aspects excluded)
[4]Driving scenario with distinct requirements

by assessing three measurements of the system. These measurements consist of a severity classification, the probability of the event occurring and the controllability of the hazardous event [18].

3 Hardware & Software

3.1 Hardware

Development boards

Two development boards operate as nodes in a distributed system on the physical car. The main development board used for running the control server and Deep Neural Network (DNN) is the *Nvidia Jetson Tegra X2*. It is equipped with an ARM processor and a CUDA GPU. The other development board, used for the control unit on the physical car, is a *Arduino MEGA*.

Table I: Specifications of the two development boards

	Nvidia Jetson Tegra X2	**Arduino MEGA 2560**
CPU	ARM Cortex-A57 @ 2GHz + NVIDIA Denver2 @ 2GHz	16Mhz (ATmega2560)
GPU	256-core Pascal (CUDA)	-
Memory	8 GB 128-bit LPDDR4	256 KB Flash
Storage	32 GB eMMC 5.1	-
OS	Ubuntu 16.04 LTS	-

Cameras

The system is equipped with two cameras; the *Genius F100* with 120 degrees horizontal field-of-view and the *Intel RealSense R200* equipped with a regular RGB-sensor as well as with two infrared sensors for depth perception.

Table II: Camera specifications

Camera	**Intel (3 sensors)**		**Genius Webcam**
	Color sensor	Infrared sensors	Color sensor
Resolution	1920x1080 (Full HD)	640x480 (VGA)	1920x1080 (Full HD)
Aspect Ratio	16:9	4:3	16:9
FOV (DxVxH)	77x43x70	70x46x59	N/AxN/Ax120
Frame Rate	30FPS	30/60FPS	30

Sensors

The system is also equipped with ultrasonic sensors, which measure distance in a single direction using sound waves. This functionality was also replicated in the simulator.

Data link

Ethernet is used as the data link between the development boards.

3.2 Software

This section covers the software used during development.

TensorFlow

TensorFlow is an open source framework created for mathematical operations on multidimensional data arrays (Tensors). Because of the flexible architecture of the framework different hardware configurations can be run using the same API.

TFLearn

TFLearn is a modular high-level API for deep neural networks using TensorFlow, which provides network-layers and helper functions for the most commonly used deep learning models. Furthermore TFLearn enables fast prototyping and experimentation by providing easy revision of implemented models while still providing access to the underlying TensorFlow API.

Unity

Unity is a cross-platform game engine commonly used for developing ordinary video games for platforms such as computers (Windows, Linux and Mac), mobile phones and consoles (Xbox, Playstation etc). It was used to build the simulator of the car and its environment. The two main reasons for choosing Unity was because of previous experience as well as the fact that the project we used as a base for the simulator was built in Unity.

Cuda

CUDA is a programming language which allows everyday programmers to access the power of parallell execution in GPUs. The original purpose of GPUs was to make visual calculations, but primarily thanks to Nvidia's CUDA the GPU platform can be used in areas such as finance and physics. CUDA is crucial for achieving high performance in machine learning and deep learning. This project was developed with CUDA enabled frameworks [19], [10].

Platform.IO

The native Arduino *integrated development platform (IDE)* is often fully sufficient for small projects, but due to its limited functionality Platform.IO was chosen instead[1]. The main reasons for this is that PIO supports cross-platform development platform with a built-in dependency manager.

[1]http://platformio.org/

4
Methods

Software development

The software development was mainly done on laptops, in multiple operating systems and programming environments, without built-in GPUs. To enable GPU access, a remote desktop solution was set up on the main development board, the Jetson.

Training sessions

To enable fast training sessions of the network, a private machine was set up with a training environment and remote access software, allowing remote training while working on other parts of the project.

Evaluation of results

Evaluation of the performance differences between the DNN based approach, the computer vision based approach and the approach of combining the two technologies was carried out by introducing a common performance based metric.

Since the metric of evaluating autonomy, proposed by *Nvidia* in the report *End to End Learning for Self-Driving Cars*[14], were sufficient for this projects demands of evaluating the systems performance, their methodology was applied.

$$autonomy = (1 - (Number\ of\ interventions) * reaction\ time^1/elapsed\ time)$$

Thereafter, a way of measuring the number of interventions of the system was required, where a twofold solution was chosen. The solution consisted of one measurement that was proprietary to the simulator, and another measurement which could be applied to the miniature car as well.

The proprietary measurement was conducted by applying a system using checkpoints on the track combined with collision objects the outer edges of the track in the simulator. When the simulated car collided with a collision object, one intervention was added to the sum of interventions and the simulated car's position on the track was restored to the last checkpoint it passed.

[1]The amount of time required for a human driver to detect a hazardous event, perform action(s) to achieve safety and giving back control to the autonomous system

The other measurement was applied by structuring the control unit after the concept proposed by *Wagner & Koopman*[2] in the report *Challenges in Autonomous Vehicle Testing and Validation*[20], as an actuator/monitor-pair; where the actuator was represented by the control unit of the system, and the monitor was represented by a stand-alone test-module. The monitor was responsible for cross-checking the actuator, by running tests that would ensure that the proposed action was not an unsafe operation, where an unsafe operation was predefined as steering towards an obstacle[3]. By counting the number of times the monitor had to intervene because of an unsafe operation, the amount of interventions was thereafter multiplied by the reaction time of a human[4] and thereafter divided by the total amount of time the system had been active.

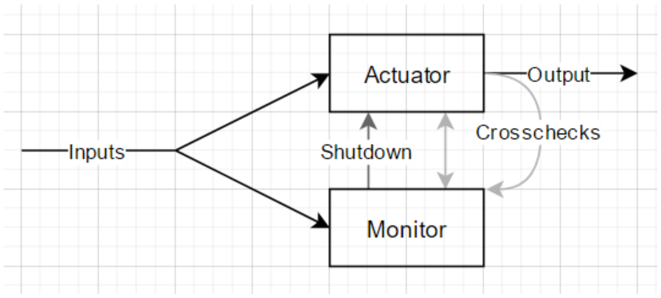


Figure 4.1: Monitor/actuator model, From [21]. Adapted with permission.

Cross-functional implementations

Arduino control unit

The Arduino control unit on the physical car was implemented in collaboration with the MT team, where the steering algorithm and cruise control was developed by the MT team and the communication was developed as part of this project.

At the point in time when the development of the control unit and its communication began, the communication for the simulator control unit was already functional. Due to this, the task was to implement a translated solution.

Inter-process communication

When running the control server in a computer vision mode, where the server uses the computer vision algorithms, the work of tracking lanes is distributed to an application developed by the computer vision team. To achieve this implementation, a TCP socket implementation was made in collaboration with the computer vision team. The control server sends images and retrieves calculated steering angles over IPC.

[2]Different ASIL levels for the actuator/monitor have not been adopted in this project
[3]An obstacle register as detected when triggered by an ultrasonic sensor of a distance < 0.1 m
[4]The reaction time was calculated to be 3 seconds for the miniature car

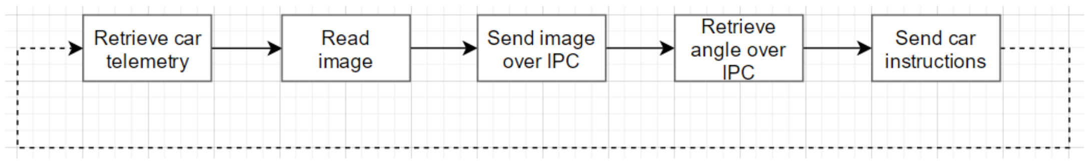


Figure 4.2: Inter-process communication flow.

Dataset

The dataset used in this project consist of images and car telemetry collected from the simulated car, as well as the physical miniature car.

Documentation

During the course of the project there was continuous work on documenting all the components of the project, simplifying future work on this project.

5

Implementation

Udacity

Udacity is an educational website with online courses in many areas of computer science[1]. Much of their learning material acted as a base for this implementation, specifically material from their *Self-driving car-engineer nanodegree program.* As part of this program, they have published an open source project under the MIT license, containing a simulator and a control server.

Control server

Control server refers to the server which is responsible for reading input and generating decisions. The implementation was initially based on the implementation in Udacity's project called *Behavioural cloning*[2], where a DNN is the main factor in the decision making. During the course of this project the whole control server had to be rewritten due to the change of communication method, see *Communication.*

The control server was also extended with functionality to read visual input from multiple sources such as the Intel camera and video files whereas the Udacity version only can read input from the simulator. The control server has two execution modes, either *training* or *running* and is structured like a typcial socket server using TCP. It also has functionality to run with different algorithms and neural nets, in our case both the DNN and the CV team's implementation of lane keeping, see *Inter-process communication* for more on this.

[1]https://www.udacity.com
[2]https://github.com/dyelax/CarND-Behavioral-Cloning

```
c:\>python cs.py -h
usage:
cs.py [-h] [lane_algo] [camera_type] [model] [client_enabled]

Control server

positional arguments:
  lane_algo        Lane keeping algorithm (dnn/cv)
  camera_type      Input video type (cam/vid/sim/sim2)
  model            Path to model h5 file.
  client_enabled   true/false

optional arguments:
  -h, --help       show this help message and exit
```

Figure 5.1: Control server help printed with the -h command for an overview of the run modes.

Control units

Control unit refers to the unit that is responsible for controlling the vehicle's motor and steering, but also collecting telemetry from the car. A control unit is implemented as part of both the simulated and physical car, for a uniform way of controlling both cars from the control server.

Simulator

The physical car was set out to be the main focus of the project, but to not be dependent on working hardware we decided to develop a simulation environment. The main reasons for this is to enable training, validation and fast development iterations. It was decided that the goal was to develop a simulated environment in a way that is interchangeable with reality.

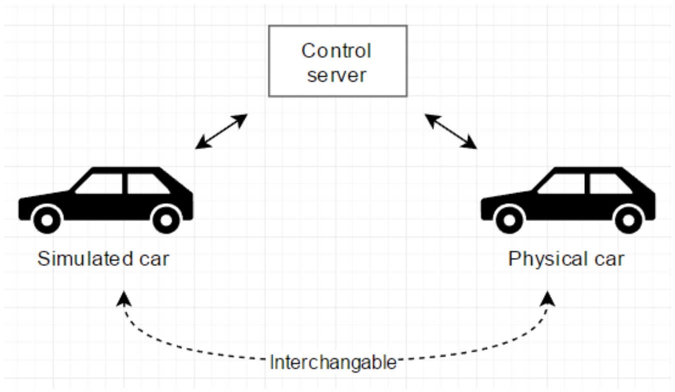


Figure 5.2: Both the simulated car and physical car can be used to connect to the control server, which runs the decision making. Both cars have a unified interface, the control unit.

Communication

The communication is one of the most critical parts of the project due to the fact that it has to work efficiently in a cross-platform setting. This means that the communication has to be functional in Python (control server), Mono (simulated control unit) and C++ (physical control unit), all using Ethernet and TCP communication.

The existing communication provided in the Udacity project was built in the high-level framework Socket.IO (SIO)[3]. SIO was developed mainly for use in web applications and the choice to use it in the Udacity project was probably motivated by its simple integration in high-level applications such as Unity games and Python applications. During the research and experimentation with Socket.IO it was concluded that an alternative communication method had to be used. The conclusion was based on the fact that SIO had a huge overhead and that it took too much control over the data flow with its event based system. During experimentations with SIO on the physical control unit (Arduino) there was also a lot of problems getting a stable connection up and running, probably due to differences in the protocol implementations.

The project group decided that a new communication method had to be developed for the Udacity project, which was used as base for this project. This means that both the control server and control units had to use the new communication method. It was also decided that the most suitable communication method was TCP due to its reliability, flexibility and extendability.

To achieve effective communication with a low overhead, but still enable fast modifications to the data sent over TCP, the data is encoded in the JSON-format and sent using a custom protocol built on top of TCP. The custom protocol was needed because TCP only guarantees that every byte should be transmitted in a correct order, but no data regarding the message size. The custom protocol was structured like the figure below. The first part of the message contains an integer representing the size of the remaining message (except delimiter) which is followed by a delimiter character and then the message content. This way, the receiver can start with reading until the delimiter and then parse those bytes into an integer which is used to read the correct message content.

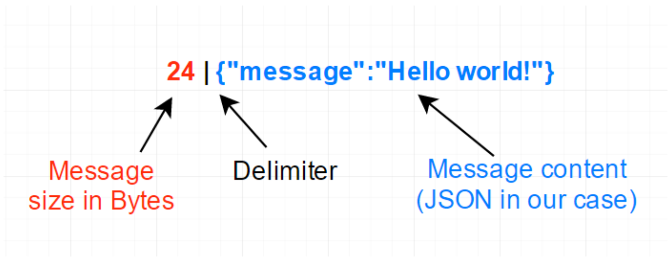

24 | {"message":"Hello world!"}
Message size in Bytes
Delimiter
Message content (JSON in our case)

Figure 5.3: Visualization of the custom protocol built on top of TCP.

[3]https://socket.io/

Neural network

Research and assessment of modern frameworks for creation and management of Deep Neural Networks (DNN) were conducted where relevance, performance and the learning curve of the framework were important factors. This led us to conclude that *TensorFlow* for providing an effortless backend to manage neural networks, in combination with *TFLearn* for providing an easy to use frontend for structures and utilities in *TensorFlow*, was the best frameworks for our purposes. With these frameworks we were able to utilize the GPU on the *Nvidia Jetson TX2* development board as well as use the GPU on the remote computers used to train the DNN. The goal with this was to decrease time used for training and inference which was of great significance since the time of this project was highly limited and since utilization of the GPU on the development board is a key part of enabling real-time performance of the system.

Furthermore, another key part of the decision of the frameworks was the ease of use and as *TFLearn* consist of a high level API for DNN's it enabled rapid development and prototyping which was very important for trying different configurations and having the ability to tweak and optimize significant parameters in a fast and easy way. The DNN that we based the system's DNN on had a similar approach and used *TensorFlow* in combination with *Keras*, however after researching these frameworks we chose TFLearn over *Keras* since TFLearn is a wrapper optimized for using TensorFlow, while Keras is a wrapper optimized for using TensorFlow or *Theano* as backend frameworks and was therefore not as optimized as using TFLearn.

During research of the frameworks another benefit of using TFLearn was discovered, namely that TFLearn has support for TensorFlow's graphical visualization tool *TensorBoard* enabled by default. With this tool, the structure of DNN's, as well as the parameters of the network over time could easily be tracked and adjusted.

Network Architecture

The project's DNN was initially be based on an implementation used for a project called behavioural cloning in the *Udacity Self-driving car-engineer nanodegree program*[4].

The underlying structure of the DNN is a fully connected Convolutional Neural Network (CNN) known as the *Nvidia Model*[14] which is optimized for calculating the steering angle of the car. The network structure adapted well to the requirements of this project, and is optimal since the output from the network only consisted of a steering angle.

[4]https://github.com/naokishibuya/car-behavioral-cloning

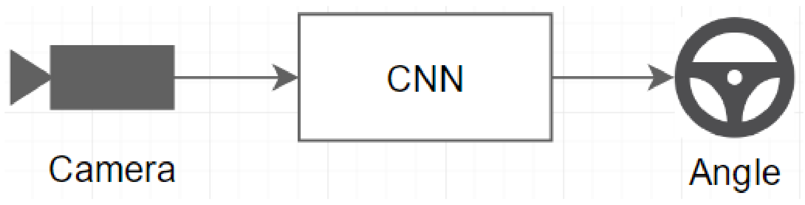


Figure 5.4: Overview of the neural network running, From [22]. Adapted with permission.

Since the research of frameworks for creating and managing DNN's resulted in a choice of using TFLearn and TensorFlow, the original Udacity implementation had to be translated from using Keras to TFLearn instead.

Training

An overview of the training process can be seen in figure 5.5 below. Images from the dataset were preprocessed by a series of augmentations consisting of random rotation, shadows and brightness levels. The augmented images were then fed through the network, resulting in a steering angle as the output. The steering angle of the network were thereafter compared with the correct steering angle of the image from the dataset, the difference between the steering angles were thereafter propagated backwards through the network and was used to adjust the weights of the network to decrease the value of the loss function.

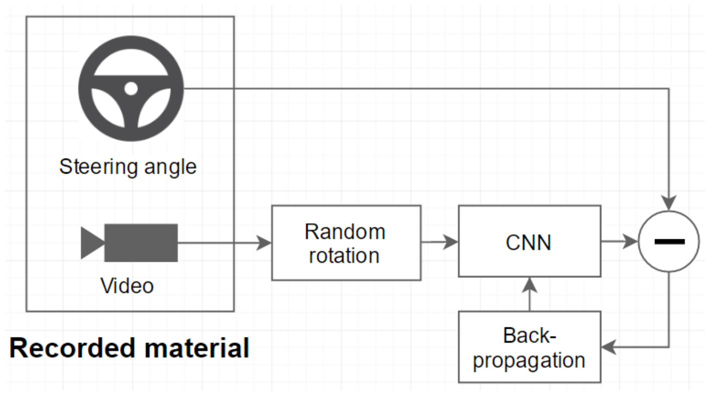


Figure 5.5: Overview of the neural network in training, From [23]. Adapted with permission.

Hyperparameters of the network Training of the DNN was initially performed by using the default values of the hyperparameters[5] taken from the DNN that the systems DNN was based on, as seen in table 5.1 below. This was done to establish a base line for the evaluation of how the training of the network was progressing.

[5]Hyperparameters are distinguished from standard model parameters by the fact that they cannot be directly learned from the regular training process of the network

Table III: Initial hyperparameters of the DNN

Loss function	Mean squared error *(MSE)*
Gradient optimizer algorithm	Adam[6]
Learningrate	0.001

The learning rate was the only hyperparameter that was adjusted during the training process since the results from using the default values for the gradient optimizer algorithm and the loss function were found to be sufficient for this projects purposes. This was decided by researching common adjustments and effects of different hyperparameter values in ANN's. Since the learning rate was found to be one of the most commonly adjusted hyperparameters with a large impact on the results, it was chosen as the primary hyperparameter to adjust for achieving desirable results from the training. The different learning rates showed that the initial value of the learning rate of 0.001 was the best, as can be observed in the graph below.

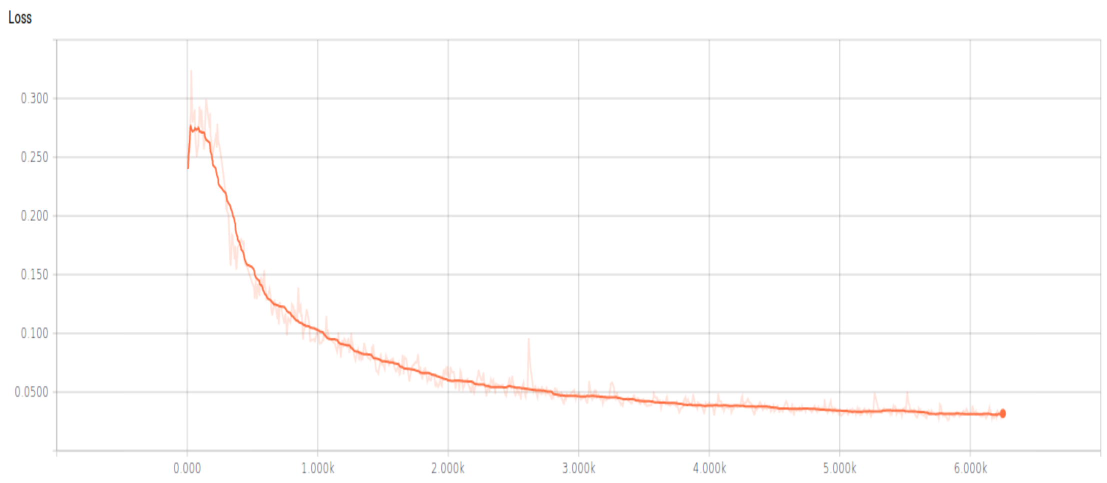


Figure 5.6: Loss of the DNN

Development boards

Two development boards were chosen as nodes in the distributed system on the physical car.

The main development board chosen for running the control server and DNN is the *Nvidia Jetson Tegra X2*. It is equipped with an ARM processor and a CUDA GPU and was chosen mainly because of the GPU, making GPU acceleration of the DNN possible.The GPU acceleration greatly decreases the compute time of neural networks by enabling parallel computation of the different operations that are applied through the network[25]. The other development board, or micro controller, is the *Arduino Mega* which was chosen as the physical control unit, responsible for controlling the vehicle and collecting telemetry.

[6]Adam is thoroughly described by *Kingma & Ba* in the paper *Adam: A method for stochastic optimazation*[24]

Cameras

To experiment with multiple techniques for analyzing the surrounding environment, a stereo camera was chosen because of its depth perception. The initial plan was to use the *ZED Stereo Camera* from Stereolabs, but another camera had to be used due to shipping delays, the *Intel RealSense R200.* During development the group came to the conclusions that the RealSense field-of-view was too narrow for some of the road curvature, and that there was a need for another camera. For that reason another camera was added, the *Genius F100*, which has a high field-of-view of 120 degrees and therefore works in sharp curves.

Sensors To complement the camera ultrasonic sensors were added, which measure distance in a single direction with a pulse-echo. The sensors were chosen for stopping the car in case of emergency, but were also considered for adding redundancy to the system.

Data link

This choice of using Ethernet as data link was made during a discussion with one of the supervisors, Alixander, where he motivated his recommendation based on its universality and debuggability compared to the *CAN bus*, which was another candidate. This choice was also motivated by the fact that the Jetson has built in Ethernet and also that the Arduino can be extended with a cheap *Ethernet shield.*

6
Results and Discussion

6.1 Results

The results of this project include successful implementations of all the components in the distributed system, including the simulator, control server, control units and DNN, all of which were crucial for achieving the purpose of this project.

Our findings from the investigation show that the performance in the simulator increased slightly when applying the combination of the computer vision based lane detection algorithm, the DNN and the system's control unit. The results from the physical miniature car was not evaluated, since the motor controller of the miniature car performed unreliably at the time of evaluation.

In the simulator the following results were measured. The computer vision based algorithm alone consistently accomplished a score of 55% level of autonomy, the DNN based approach alone consistently produced a score of 94% level of autonomy, while the solution of combining both techniques accomplished a 96% level of autonomy[1].

Table IV: Level of autonomy for the different solutions

Level of Autonomy	Technique(s)
55%	CV
94%	DNN
96%	CV + DNN + Control Unit

6.2 Discussion

Learning curve

Machine learning and deep learning are huge research fields with complex math and technicalities, historically there has been a high threshold of understanding before reaching the ability to use neural networks. However, since the introduction of high-level API's such as *TFLearn* the knowledge threshold have been lowered significantly.

The most common neural network structures and algorithms can be applied by using these high-level API's which provides a basis to start from in most projects, yet

[1]The results were obtained using a maximum speed of 20 km/h

basic knowledge regarding the structural components of the neural network is still needed to construct an effective solution.

During this project much of the time regarding theory has been dedicated towards basic understanding of neural networks and it's basic structures, which should be accounted for by anyone that wants to utilize DNN's to their full potential.

Hardware limitations

The use of deep learning has increased exponentially during the last few years and it is thanks to development of both hardware and algorithms and of hardware. Without Graphical Processing Unit (GPU) processing the training might still be possible, but running the DNN in realtime would probably not be possible. This means that the process of developing, testing and running the DNN was very GPU dependent. The development was primarily done on laptops without GPUs and therefore the performance was really bad and much of the training etcetera had to be run on desktop machines in remote locations.

Software dependencies

The goal of the project was to implement neural nets rather than developing everything from scratch. To achieve this we examined existing solutions and tried to use as many frameworks and libraries as we could. Though this has helped us a lot and heavily reduced the development/implementation time, this approach came with another issue. Many of the libraries/frameworks/solutions required a lot of dependencies. Many of the dependencies could be installed easily, but some of them had to be compiled manually, which in some cases took a lot of time or didn't even work. Unfortunately we had to spend a significant amount of time upon managing software dependencies during the course of this project.

Simulator

During initial phase of the project we decided to work with a simulator to not be dependent of hardware, and this was probably the most important decision we took. It took us a lot of time to develop the simulator and its communication, but without it the project would probably not have been finished since the physical car wasn't fully functional until the last week of the project.

Cross-functional team

The Sigma Project, which this thesis is a part of, introduced a way of working that combined the expertise of several different fields which in turn yielded a product with a broader scope than the project would have done with a single engineering discipline. However, it did also introduce a greater complexity in project management since tools and methods differ between the different areas of expertise. Another factor that greatly influenced this complexity was the increase in size of the team, which affects organizational parts of the project and reflect the importance of the project manager.

Improvements

Much of the code that was developed as part of this project is unfortunately not very parallel in execution. The reason for this was because the main focus of the project was optimizing the DNN rather than spending time on perfecting a non-existing solution.

Sustainable development

During the course of this project we have come to several understandings regarding how autonomous cars can impact a sustainable development.

Social development, where autonomous cars probably will have the biggest impact, has been the main focus to study. Safety of humans is the most critical area and our understanding is that these new technologies will decrease the number of traffic accidents by a tremendous amount, if developed and validated in a correct way. Another effect that autonomous cars can have within social development is improved accessibility of transportation for the elderly, handicapped and the youth; given that drivers might not need to be active or even have a drivers license if the technological and juridical challenges behind making a vehicle fully autonomous are met.

Environmental development is another area which will probably be affected by autonomous cars, where the two main areas are less emissions and less amount of vehicles. Less emissions will be the effect of cars driving effectively and having a much better flow in the traffic, so less energy is wasted. Much points towards a decreased demand for owned cars, if fully autonomous cars will be available, mainly because of fleet models where one autonomous car can be shared between a group of people.

Conclusion

This project set out to investigate how a combination of traditional computer vision techniques and modern Deep Neural Networks (DNN) can minimize unsafe behaviour in autonomous vehicles. Much of the development time was spent on setting up the environment and everything needed to conduct the investigation.

Our results indicates that the DNN was the greatest contributor to the results obtained from running the system in the simulator. However, when the different techniques were applied on the hardware of the physical miniature car in a real world environment, the shortcomings of the separate DNN were highlighted, demonstrating the fact that DNNs cannot be relied upon in environments for which they have not been explicitly trained on. In that setting, our initial findings indicate that the separate computer vision based approach outperforms the separate DNN.

Since both systems have their weaknesses, a combination of both computer vision and deep learning is more accurate and safe than running any of the techniques by itself when considering multiple environments. We therefore believe that this combination is the safest way to develop autonomous behaviour, and that future projects will benefit greatly from introducing a system that combines and utilizes the strengths of traditional computer vision based techniques with the computational potential of DNN's.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence : A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2009.

[2] A. Munoz, “Machine Learning and Optimization,” Courant Institute of Mathematical Sciences, 2014. [Online]. Available: https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf

[3] K. P. Murphy, *Machine learning : A probabilistic perspective.* Cambridge, MA, USA: MIT Press, 2012.

[4] E. Alpaydin, *Introduction to machine learning*, 3rd ed. Cambridge, MA, USA: MIT Press, 2014.

[5] Y. Lecun, E. Cosatto, J. Ben, U. Muller, and B. Flepp, “Autonomous Off-Road Vehicle Control Using End-to-End Learning,” Courant Institute/CBLL, Arlington, VA, USA, Tech. Rep. DARPA-IPTO Final technical Report, Jul. 2004. [Online]. Available: http://www.cs.nyu.edu/~yann/research/dave/dave-final-report.pdf

[6] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, doi:10.1016/j.neunet.2014.09.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[7] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation*, ser. Lecture Notes. Redwood City, CA, USA: Addison-Wesley Publishing Company, 1991, vol. 1.

[8] Stanford University, “A cartoon drawing of a biological neuron (left) and its mathematical model (right).” 2017, [Electronic image]. Available: http://cs231n.github.io/neural-networks-1.

[9] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2008.

[10] C. Asplund, “Object classification and localization using machine learning techniques : Designing and training models for use in limited hardware-applications,” M.S thesis, Department of Physics, Chalmers University of Technology, Gothenburg, Sweden, Jun. 2016.

[11] M. Riedmiller, “Advanced Supervised Learning in Multi-layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms,” *Int. Journal of Computer Standards and Interfaces*, vol. 16, no. 3, pp. 265–278, Jul. 1994, doi:10.1016/0920-5489(94)90017-5. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0920548994900175?via%3Dihub

[12] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[13] LISA lab., “The Full Model: LeNet,” 2013, [Electronic image]. Available: http://deeplearning.net/tutorial/_images/mylenet.png.

[14] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to End Learning for Self-Driving Cars,” *CoRR*, vol. abs/1604.07316, Apr. 2016. [Online]. Available: https://arxiv.org/pdf/1604.07316.pdf

[15] Nvidia Corporation, “End to End Learning for Self-Driving Cars,” 2016, [Electronic image]. Available: https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2016/08/cnn-architecture.png.

[16] “Summary of SAE International’s Levels of Driving Automation for On-road Vehicles,” SAE International, 2014. [Online]. Available: https://www.sae.org/misc/pdfs/automated_driving.pdf

[17] SAE International, “Summary Table of the Levels of Driving Automation presented in the J3016 standard,” 2014, [Table]. Available: https://www.sae.org/misc/pdfs/automated_driving.pdf.

[18] *Road vehicles — Functional safety — Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses*, ISO26262-9, 2011.

[19] S. Cook, *CUDA programming : A developer’s guide to parallel computing with GPUs*. San Fransisco, CA, USA: Morgan Kaufmann, 2013.

[20] P. Koopman and M. Wagner, “Challenges in Autonomous Vehicle Testing and Validation,” *SAE International Journal of Transportation Safety*, vol. 4, pp. 15–24, Apr. 2016, doi:10.4271/2016-01-0128. [Online]. Available: http://papers.sae.org/2016-01-0128/

[21] SAE International, “Challenges in Autonomous Vehicle Testing and Validation,” 2016, [Figure]. Available: https://users.ece.cmu.edu/~koopman/pubs/koopman16_sae_autonomous_validation.pdf.

[22] Nvidia Corporation, “End to End Learning for Self-Driving Cars,” 2016, [Electronic image]. Available: https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2016/08/inference.png.

[23] Nvidia Corporation, “End to End Learning for Self-Driving Cars,” 2016, [Electronic image]. Available: https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2016/08/training.png.

[24] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, Dec. 2014. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

[25] B. Nemire, “CUDA Spotlight: GPU-Accelerated Deep Neural Networks,” Nvidia Corporation, 2014. [Online]. Available: https://devblogs.nvidia.com/parallelforall/cuda-spotlight-gpu-accelerated-deep-neural-networks/

Acronyms

ADS Automated Drive System. 8, 32
ANN Artificial Neural Network. 4, 5, 32
API Application Programming Interface. 32
ASIL Automotive Safety Integrity Level. 8, 32

CNN Convolutional Neural Network. 5, 6, 22, 32
CUDA Compute Unified Device Architecture. 32
CV Computer Vision. 2, 32

DL Deep Learning. 1, 32
DNN Deep Neural Network. 1, 15, 19, 22–24, 27–30, 32

GPU Graphical Processing Unit. 28, 32

TCP Transmission Control Protocol. 16, 32

Glossary

Backpropagation Backward pass of the network to adjust weights. 32

Caffe A deep learning framework. 32
Conda Package management system for maintaining dependencies. 32
cross-functional Multidisciplinary collaboration. 32

Ground Truth Term used to refer to direct observation. 32

Tensor Flow A deep learning framework. 32

Unity Game engine for game development. 32

Vagrant Tool for maintaining virtual environments. 32

A
Tools

This appedix is an overview of the tools used during development process.

Git Used for version control.

Trello For providing access to the different sub-teams backlogs.

Slack An accessible communication plattform.

Google Drive Common file storage solution.

Conda A package management system for maintaining dependencies in Python and other languages. Conda was chosen to have a shared development environment for the control server.

Vagrant A tool for maintaining virtual environments. Vagrant was chosen for the purpose of having a shared development environment while developing towards the Jetson board.