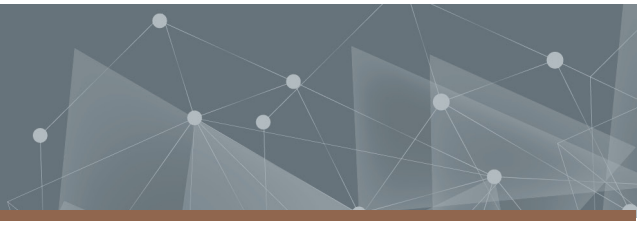
CHALMERS
UNIVERSITY OF TECHNOLOGY



System for Hands-on Steering Wheel Detection Using Machine Learning

Master's thesis in Complex Adaptive Systems

Emil Johansson
Ravi Linder

DEPARTMENT OF MECHANICS AND MARITIME SCIENCE

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

System for Hands-on Steering Wheel Detection Using Machine Learning

Emil Johansson
Ravi Linder

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Science
Chalmers University of Technology
Gothenburg, Sweden 2021

System for Hands-on Steering Wheel Detection Using Machine Learning
Emil Johansson
Ravi Linder

Supervisor: Car-Johan Häll, Volvo Cars
Supervisor: Spyridon Skouras, Volvo Cars
Examiner: Matthijs Klomp, Department of Mechanics and Maritime Science

Master's Thesis 2021:50
Department of Mechanics and Maritime Science
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

System for Hands-on Steering Wheel Detection Using Machine Learning
Emil Johansson
Ravi Linder
Department of Mechanics and Maritime Science
Chalmers University of Technology

Abstract

More cars are getting semi-automatic driving capabilities and on the way towards full automation, the driver is still responsible for the car. An important aspect of controlling the car is holding the steering wheel which is supposed to be done at all times, even during semi-automatic driving. To check that the driver is controlling the vehicle and holds the steering wheel, a system can be created using inputs from the steering gear sensors or a camera. Two different solutions using these inputs are implemented and tested to see if machine learning can be used to differentiate between hands on and hands off situations. The systems are also evaluated running online on a Raspberry Pi single board computer where the efficiency of the systems is considered, as well as performance. The evaluation of the systems shows that the sensor version can be updated frequently at 5 Hz, while being robust when the driver is not intentionally tricking the system by hanging a weight on the steering wheel. The camera model is limited in update frequency by the library used for running the neural network in C++. The neural network had to be downsized in order to make the system update frequently enough to detect change of state. This meant that the system wasn't able to detect some situations as well as the system which was evaluated on offline data. Changing libraries for the neural network would potentially solve this problem and make the online version perform more similarly to the offline system.

Keywords: Hands-off detection, Machine learning, State-space model, Kalman filter, Object classification

Acknowledgements

First we would like to give a big thank you to our supervisors Carl-Johan Häll and Spyridon Skouras for their support and guidance throughout the thesis. We would like to thank Tuschar Chugh for the guidance modelling the car. Lastly, we would like to thank Volvo Cars Corporation for letting us do our thesis there and use all the necessary equips, cars and data to complete the thesis.

Emil Johansson and Ravi Linder, Gothenburg, 06-2021

Contents

List of Figures

List of Tables

Nomenclature

The used physical constants for the state-space model are the following:

δ_{pin} Pinion angle

δ_s Steering wheel angle

b_{belt} Dampning coefficient for calculating M_{mot}

b_{pin} Pinion damper constant

b_{tb} Torsion bar damper constant

c_{belt} Stiffness coefficient for calculating M_{mot}

c_{tb} Torsion bar spring constant

F_s Sampling frequency, $F_s = 1/T_s$

F_{rack} Rack force

i_{mot} Ratio between the motor and pinion

i_{rp} Ratio between the rack and pinion

J_s Steering wheel interita

J_{pin} Pinion interita

M_s Steering wheel torque

$M_{mot,eff}$ Motor torque

M_{rack} Rack force converted to torque through i_{rp}

$M_{s,fric}$ Steering wheel torque friction

$M_{tb,fric}$ Torsion bar torque friction

T_s Time between each sample

1
Introduction

In today's society, automation has been an ongoing process for many years and more and more parts are likely heading in the same direction. Currently lawnmowers, delivery robots and entire warehouses are operated autonomously. The main thread in automation involves monotonous tasks which we as humans find unfulfilling to complete which often leads us to become distracted and increase the risk of making errors. This is not so worrying when mowing a lawn perhaps, but there are other situations where the consequences are higher. One such situation is driving a car.

Moving towards a world of autonomous mobility where predictions indicate self-driving vehicles in the near future, the first step is to let an algorithm drive on highways while the driver still ready to take control if something unexpected were to happen. This is called semi-autonomous driving or driver assist. This type of autonomous system, which is able to control the car for limited periods of time, is classified as level two of the five levels of autonomous cars (Abraham et al., 2017). There are already level two autonomous cars on the roads today and there are several car manufacturers offering this kind of driver assist in their cars. Volvo's Pilot Assist, Tesla's Autopilot and Audi's Traffic jam assist are examples of such systems. To make sure the driver is always ready to assume control over the vehicle in an unexpected situation, it is important to always hold the hands on the steering wheel. Besides making the driver be able to react faster, it also makes it more difficult to focus on other distractions such as looking at one's phone or similar activities which puts the focus away from driving. There are legal requirements on car manufactures for them to implement these systems which states that the driver must at all times keep at least one hand on the steering wheel and always be ready to take control. At this level of autonomy, the driver is responsible for the car even though a driver assist is temporally driving the car. If a driver does not hold their hand on the steering wheel, the regulations are to set an optical warning after at most 15 seconds and to shut down the semi-automatic drive within a minute (United Nations, 2017-11-30). The two most common methods to check for hands off driving are either by a capacitive sensor in the steering wheel or by looking at the torque. Lately, cameras have also been used to check if the driver is paying attention.

1.1 Purpose

The purpose of this thesis is to research and develop a system in combination with machine learning which can detect if the driver is holding the steering wheel dur-

ing semi-automatic driving. The system is based on a camera mounted above the steering wheel and the steering gear sensors available in the car. The developed system should be migrated to work on a single board computer to emulate the computational power in a car for real-time detection.

1.2 Delimitations

The detection system will only consider all hands that are holding the steering wheel, it will not matter if the hands belong to the driver or another passenger. The camera based detection system will not be generalized and gloves and other accessories affecting the view of the hands will not be considered. Solutions to overcome this problem will be discussed. Lastly, the system is designed to work during semi-automatic driving and no general cases outside of semi-automatic driving will be considered.

2
Theory

In this chapter, necessary background theory in relation to the project is presented.

2.1 Neural Network

Neural network is a broad term including many different types of networks. Depending on what task you want the network to perform, you need to find a solution fitting to that specific task. The most common network architecture is the fully connected network (FCN). This network is built with layers of neurons where you can have several layers stacked to produce a deeper network (Goodfellow, Bengio, and Courville, 2016). These network consists of different amount of layers where the first layer is called an input layer and the last layer is called the output layer. The layers in the middle are called hidden layers and can in principle consist of any number of layers. if a network has at least one hidden layer, it is referred to as a deep neural network (Goodfellow, Bengio, and Courville, 2016). An example of a deep fully connected network can be seen in Figure 2.1.

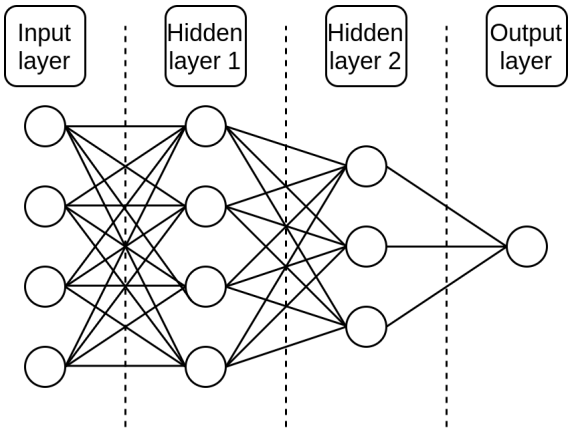


Figure 2.1: Example of a fully connected network with two hidden layers.

2.1.1 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are development of the fully connected network. Instead of connections between neurons being only in a forwards fashion, an RNN has connections which create internal loops and cycles between neurons which enables the exhibit of dynamic temporal behavior (Sherstinsky, 2020). This means signals running through the network can travel both forwards and back which also

means that information in the network is able to travel in different directions leading to RNNs having a short term memory (Hochreiter and Schmidhuber, 1997). The network is able to process new information while taking previous information into consideration. This feature has made RNNs common in language translations, speech recognition and text generation which has been made popular by OpenAI's Generative Pre-trained Transformer (GPT) (Floridi and Chiriatti, 2020).

2.1.1.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) is an architecture belonging to the RNNs. LSTMs were introduced by Hochreiter and Schmidhuber in 1997 as a way of taking care of long-term dependencies (Hochreiter and Schmidhuber, 1997). The difference in the LSTM module is the improved memory capabilities implemented using a memory cell (Pulver and Lyu, 2016). The memory cells can be seen in Figure 2.2 for both the vanilla RNN and the LSTM.

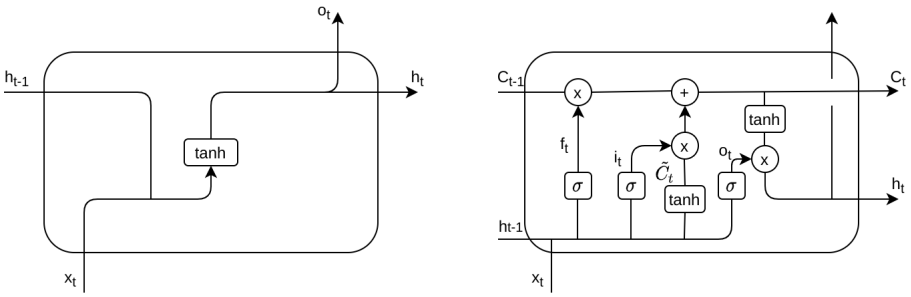


Figure 2.2: A visualization of the layout of a memory cell for the vanilla RNN (left) and the LSTM (right).

This memory cell consist of several layers and the flow of data through this cell is determined by gates consisting of sigmoid layers, which outputs a value between one and zero (Pulver and Lyu, 2016). This value will determine how much information should flow through this cell. If the value is zero, no information will be let through the gate and if the value is one, everything will pass. An LSTM-module has three gates of this kind which control the cell state. These gates are learnable parameters which means that the LSTM module during training, learns how long it should consider past information to be relevant (Pulver and Lyu, 2016). It is possible to represent the passing of time by stacking one of the cell layouts in Figure 2.2 horizontally, where each cell represents one timestep. The output to the right of the cell is used as the inputs to the left of the next cell module.

2.1.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) is most often associated with image based learning, but lately has gained more popularity for analysing time series data (Cai, Pipattanasomporn, and Rahman, 2019). CNNs gained popularity in 2012 with the introduction of AlexNet which was one of the first deep CNN (Alom et al., 2018). AlexNet entered the ImageNet challenge which was a competition where

different neural networks competed against each other in image classification. The performance of the network was much better than the competition, leading to CNNs becoming the standard of image based learning (Alom et al., 2018).

2.1.2.1 2D convolutional network

The most commonly used CNN is a network with a 2D input. This input is usually an image of some kind, and if this image is in colour, the input is divided into three channels for e.g RGB (Albawi, Mohammed, and Al-Zawi, 2017). What happens in a convolutional layer is illustrated in Figure 2.3.

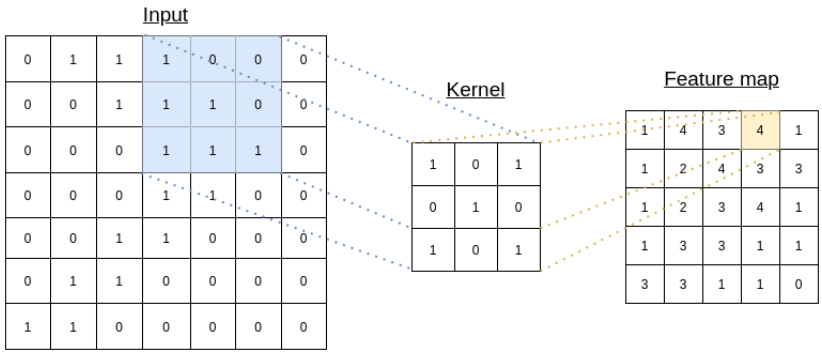


Figure 2.3: Example of a 3x3 kernel operating on a 7x7 input with stride 1. The result is a feature map of size 5x5.

Looking at the figure above, there is a kernel which is a fixed size square matrix with size smaller than the input image. The kernel holds the weights in a convolutional layer, and these weights are operating on the image as the kernel moves over the input (Albawi, Mohammed, and Al-Zawi, 2017). The weights of the kernel are multiplied with the corresponding values of the input image, and the sum of these operations are then the resulting value in the corresponding place in the feature map. When this is done, the kernel moves to the next part of the image with a step size called stride. When the kernel has moved over the entire input, all the resulting values are saved in a feature map, which is then the output of the CNN layer.

If the input is an image the kernel would have three dimensions, one for each of the colour channels. A collection of kernels is sometimes referred to as a filter and it's common to use several filters in a CNN layer since each filter learns one feature in the image (Albawi, Mohammed, and Al-Zawi, 2017).

2.1.2.2 1D convolutional network

One variant of the CNN is operating the network on 1D inputs, or time series. These kind of operations have previously been handled by RNNs and LSTMs, but resent research has shown that CNNs in 1D can perform similarly, or even better compared to the RNN models Cai, Pipattanasomporn, and Rahman, 2019. Even if the networks are performing similarly, there are several advantages when using a CNN. The greatest advantage is the ability to parallelize the training of a CNN on a

GPU which greatly improves the training time, although there are some algorithms to partly parallelize training RNN, it still isn't as efficient (Hwang and Sung, 2017). LSTMs are dependant on the previous output to calculate the next, leading to slower processing. The CNN also has fewer parameters in general compared to a LSTM which also help improve time spent training the network. CNNs also don't suffer from vanishing (or exploding) gradients, especially in combination with the rectified linear unit (ReLU) activation function, making training and tuning the network less challenging (Ide and Kurita, 2017).

2.1.3 Activation Functions

The activation functions are deciding whether the neuron in the network are activated (Sagar Sharma and Simone Sharma, 2017). This is based on the input, which is the sum of the weights and biases which are related to that specific neuron. The activation function is almost always set to be the same in a whole layer.

There are several different activation functions which are used for different purposes. It is common to have different activation functions in different layers in a neural network, and especially in the output layer which will provide the result of the network. The main purpose of the activation functions is to provide non-linearities in the network and help the network learn more complex tasks (Sagar Sharma and Simone Sharma, 2017). Without activation functions, the networks would be a linear regression model.

As mentioned before, there are several different activation functions. Some of the commonly used function can be seen in table 2.1. The rectified linear unit (ReLU) is currently one of the most popular activation function for use in most types of networks, and this function is seen as a revolution in machine learning. The ReLU function returns the input if the input is larger than 0, otherwise it returns 0. This function is linear for half of its domain and non-linear in the other half which still will ensure that the network has complex learning abilities. The function is also popular since it doesn't get saturated like several other functions do. Such as the tanh function which will return 1 for large positive inputs and -1 for large negative inputs. This is a problem when training deep neural network and therfore, the ReLU function is a commonly used activation for hidden layers. Another advantage over many other activation functions is that the ReLU function doesn't suffer from vanishing gradients. Again, comparing with the tanh function when close to saturation will suffer from small gradients.

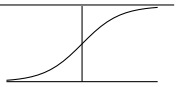
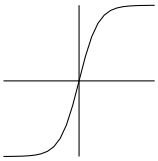
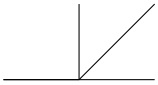
Name	Function	Derivative	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^z + e^{-z}}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	

Table 2.1: Non-linear activation functions.

When back-propagating through the network to adjust for the error, the gradients are multiplied with the error, and if the gradients are close to 0, the learning of the network is very slow. However, since the ReLU function is 0 for negative values, the gradient would be 0 as well which means that no learning will occur after this level is reached. This could be fixed using a ReLU function with a small positive gradient for negative vales, such as the leaky ReLU function. The drawback for using a function like this is that it is more computationally expensive during training, and is most often not used for that reason. The advantage gained is often deemed to small to pay the price of more demanding calculations.

2.1.3.1 Output Layer

The choice of the activation function for the output layer is an important parameter. The output from a neural network will be entirely dependant on the activation functions of the output layer which means that it will be decided by the task in question. For classification tasks, it is helpful if the output is a probability. This is why the most common activation functions for classifications are the sigmoid function and the softmax function. The sigmoid function is used for binary classification and will output a value between zero and one while the softmax function is used for classifications with more than two classes. The softmax function will output a probability for all classes in the dataset and the sum of these values is one. The sigmoid function is a special case of the softmax function for two classes.

2.1.4 Loss Functions

Loss functions, or cost functions, are an important part of training a neural network. The loss function gives a value of the prediction of the network and often, the goal is to minimize this function. Since training a neural network is an optimization problem, where the weights in the network are being optimized, it is necessary to have a validation of the quality of the networks ability to make predictions. The loss function is different from prediction accuracy, as in this case, it isn't how many of the guesses that are correct that is interesting, but how close the guesses are to

the correct answer. A network could have high accuracy and at the same time have a high loss. This usually indicates that the network is uncertain in its predictions.

There are several different loss functions depending on the purpose of the network. If the intention is to make a regression neural network to make predictions, the most commonly used loss function is the mean square error (MSE). The loss in this case is calculated by taking the mean of squared differences between the prediction of the network and the correct value, as can be seen in Equation 2.1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

Neural networks are often used to classify objects. For this scenario, it is common to use the categorical cross-entropy (CCE). This loss function is used when there are more than two classes to identify. The formula for this loss function can be observed in Equation 2.2, where c is the number of classes to be predicted.

$$CCE = -\sum_{i=1}^{c} y_i log(\hat{y}_i) \tag{2.2}$$

If it is a binary classification, binary cross-entropy (BCE) is used which is a special case of the categorical cross-entropy for only two classes.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i) \tag{2.3}$$

For the multiple classification network, it is important to use a Softmax layer as an output layer since this will produce a probability off the prediction for each class. For the binary classification, a Sigmoid function is common for the final layer in the network since the output from this is a probability between zero and one.

2.1.5 Datasets - Training, Validation and Testing

When training networks, best practice is to use three different datasets: training, validation and a test dataset. The training dataset is used for the training of the network. When training a network, it is almost always possible to achieve a high accuracy on the training dataset. The problem is that at some point, the network isn't learning general features but instead, focusing on the features of the training dataset. This means that when this network is introduced to new data, it won't perform as well since the network hasn't learned general features. This is known as overfitting and is the reason why a validation dataset is important.

After each training epoch, the networks performance is validated using the validation dataset. This dataset is used to know when to stop training. Looking at the validations loss, it is when this starts increasing that the network has stopped learning general features and starts overfitting.

When the network has finished its training, the test dataset is used as a final check that the networks performance is as expected. The test dataset has not been involved

in the training process at all which is a good representation of the networks ability to perform at the intended task. When creating the test dataset used for the network evaluation, no timer is used and different lengths of hands off duration is registered to emulate a real world situation as well as possible.

2.2 Car Model

To model the input signals from the car a 2 degree of freedom state-space model is used. The model is the same reference model as in the paper (Chugh et al., 2020). The figure 2.4 shows the model taken from figure 1b in (Chugh et al., 2020) and uses two second order differential equations. The first one is a spring-damper model for the steering wheel and the second one a spring-damper model for the pinion,

$$\begin{aligned} J_{\mathrm{s}}\ddot{\delta}_{\mathrm{s}}(t) &= -M_{\mathrm{tb}}(t) - M_{\mathrm{s,fric}}(t) + M_{\mathrm{s}}(t), \\ J_{\mathrm{pin}}\ddot{\delta}_{\mathrm{pin}}(t) &= -b_{\mathrm{pin}}\dot{\delta}_{\mathrm{pin}}(t) - M_{\mathrm{rack}}(t) - M_{\mathrm{tb,fric}}(t) + M_{\mathrm{tb}}(t) + M_{\mathrm{mot,eff}}(t). \end{aligned} \tag{2.4}$$

From the equations, the motor torque $M_{mot,eff}$ and torsion bar torque M_{tb} can be further simplified to,

$$\begin{aligned} M_{tb} &= b_{tb} \cdot (\dot{\delta}_s - \dot{\delta}_{pin}) + + c_{tb} \cdot (\delta_s - \delta_{pin}); \\ M_{mot,eff} &= -b_{belt} \cdot \dot{\delta}_{pin} \cdot i_{mot}^2 - c_{belt} \cdot \delta_{pin} \cdot i_{mot}^2 \end{aligned} \tag{2.5}$$

from the equations a state-space model can be set up which is used in a Kalman filter.

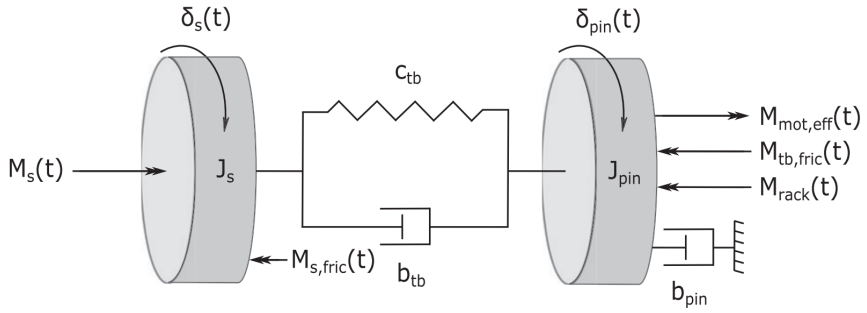


Figure 2.4: The used 2 degree of freedom spring-damper model used for the state-space model. Figure taken from figure 1b in (Chugh et al., 2020).

2.3 Kalman Filter

The Kalman filter estimates the state in a system of linear difference equations. Using the system from 2.2,

$$x_{k+1} = A_k x_k + B u_k + w_k, \tag{2.6}$$

and the output's from the system being,

$$y_k = C_k x_k + v_k \tag{2.7}$$

where v is gaussian with mean 0 and variance Q, and w is another gaussian with mean 0 and variance R (Bishop, Welch, et al., 2001). A priori state estimate $\hat{x}_k^-$, posterior state estimate $\hat{x}_k$ and the true state x_k from equation 2.6 are used to define the error equations where the e_k^- is the priori and e_k is the posterior,

$$\begin{aligned} e_k^- &\equiv x_k - \hat{x}_k^- \text{ and} \\ e_k &\equiv x_k - \hat{x}_k \end{aligned} \tag{2.8}$$

Afterwards covariance matrices can be constructed,

$$\begin{aligned} P_k^- &= E\left[e_k^- e_k^{-T}\right] \text{ and} \\ P_k &= E\left[e_k e_k^T\right] \end{aligned} \tag{2.9}$$

The kalman filter is then built up by a predict stage which estimates the current state and a correct stage which adjusts the prediction with the real measurement (z_k) (Bishop, Welch, et al., 2001). In the Kalman filter if the R variance goes towards zero, the filter trusts the real measurments' more and if the priori covariance (P_k^-) goes towards zero, the estimation is trusted more (Bishop, Welch, et al., 2001).

2.3.1 Predict step

In the prediction step, the priori state and priori covariance is estimated. The next state is estimated with equation,

$$\hat{x}_{k+1}^- = A_k \hat{x}_k + B u_k \tag{2.10}$$

and the covariance with,

$$P_{k+1}^- = A_k P_k A_k^T + Q_k \tag{2.11}$$

the covariance and state update uses the previous posterior state and covariance. (Bishop, Welch, et al., 2001).

2.3.2 Correct step

The correct step uses the priori estimation from the predict step to calculate a Kalman gain (K), and using that to compute a posterior state estimation, aswell as a posterior covariance. The Kalman gain is calculated to minimize the error covariance and derived from (Jacobs, 1993),

$$\begin{aligned} K_k &= P_k^- C_k^T \left(C_k P_k^- C_k^T + R_k\right)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K\left(z_k - C_k \hat{x}_k^-\right) \\ P_k &= \left(I - K_k C_k\right) P_k^- \end{aligned} \tag{2.12}$$

The updated posterior state and covariance are then used in the next predict step and the Kalman filter runs recursively between the steps.

2.4 Fast Fourier Transform

The fourier transform relates the time-domain signal to the frequency domain, and in a computer the signal is sampled with distinct time-steps. A continous signal, sampled can be seen as sequence of x(n) with N total samples and can be transformed to the frequency domain X(k) using the discrete fourier transform (DFT),

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-jk\left(\frac{2\pi}{N}\right)n}, k = 0, 1, 2, 3, \ldots, N-1 \tag{2.13}$$

however the DFT is computationally expensive with a complexity of $O(N^2)$ (Zhang and Jiang, 2021). using a fast fourier transform (FFT) instead, the coputational time can be decreased to $O(N\log(N))$, the FFT needs a signal length of 2^k and if that is not satisfied zeros can be added to satisfy the constraint (Zhang and Jiang, 2021). Nussbaumer writes in more detail about the FFT algorithm in (Nussbaumer, 1981) and works by splitting the discrete sequence into two N/2 sequences one with the even samples (x_{2n}) , the other with the odd samples (x_{2n+1}) which turns the DFT into,

$$\begin{aligned} &X(k) = \textstyle\sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-jk\left(\frac{4\pi}{N}\right)n} \sum_{n=0}^{N/2-1} x_{2n+1} \cdot e^{-jk\left(\frac{4\pi}{N}\right)n} \\ &X(k+N/2) = \textstyle\sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-jk\left(\frac{4\pi}{N}\right)n} - e^{-jk\left(\frac{2\pi}{N}\right)} \sum_{n=0}^{N/2-1} x_{2n+1} \cdot e^{-jk\left(\frac{4\pi}{N}\right)n} \\ &k = 0, \ldots, N/2-1. \end{aligned} \tag{2.14}$$

the same principle can be used recursivly and splitting the equation 2.14 into two new halfs of length N/4 leading to the computational time of $O(N\log(N))$ for a FFT (Nussbaumer, 1981).

3
Related Work

Because of the severe consequences of car crashes previous work has been done within the field of making sure the driver is paying attention and in control of the vehicle while driving. Within hands on detection, literature focuses on the torque sensor in the car because a driver cannot hold the steering wheel without generating torque. For attention detection algorithms cameras have been used to monitor where the hands are in relation to the steering wheel. This chapter provides a summary on how sensor based and camera based approaches have been used. Secondary the chapter introduces neural network prediction models within autonomous driving to see how the two areas can be combined for this thesis.

3.0.1 Hands On/Off Detection Based on EPS Sensors

The paper (Moreillon, 2017) proposes that hands on/off detection in combination with lane-keeping assistance are two aspects used in semi-automatic driving today and the goal is to make sure the driver is always in control of the vehicle. For further use when the car evolves and is more automatic, hands on/off detection can be used for transition between automatic and manual driving, making sure the driver is ready to take over control by placing the hands on the steering wheel. A robust system avoids false positives; the scenario when the system assumes hands are on the steering wheel while in reality the driver is not holding the steering wheel (Moreillon, 2017). Furthermore, this can lead to situations when neither the semi-automatic driving nor the driver is controlling the car, a hazardous situation. By using sensors available in the car the paper proposes a state-space model with 1 degree of freedom based on one spring-damper model to estimate the torque from the driver on the steering wheel (Moreillon, 2017). As the torque is the input in the model, an extended state-space modification is made and a state observer is created to estimate the driver torque. Since there is a large difference between hands on and hands off states the paper puts a hysteresis level to determine if the driver is holding the steering wheel. To avoid the false positive scenario the model only switches to hands off when the torque is below the threshold for a certain time (Moreillon, 2017).

3.0.2 Learning-Based Approach for Online Lane Change Intention Prediction

There has been usage of machine learning algorithms for saftey predictions in autonomous driving. One approch is to use a support vector machine (SVM) to classify and predict upcoming lane changes (Kumar et al., 2013). A SVM maps a non-linear

problem to a higher hyperspace to create a linear plane which separates the classes for prediction. The lane change prediction is done by looking at features important to lane changes such as the steering wheel angle and the car's lateral position in the lane (Kumar et al., 2013). Because a lane change is not instantaneous a sliding window is used to capture the whole lane change. Ground truth is captured to make sure the network has good data to classify on. To further improve the algorithm a baysian filter (BF) is added after the SVM which increases computational time slightly but improves accuracy (Kumar et al., 2013). The SVM with the BF in this paper is able to predict a lane change on average 1.6 seconds before it happens but has several false alarms and an accuracy of 71.5% (Kumar et al., 2013). This thesis differs in classifying ground truth data instead of predicting what is going to happen in the near future. The classification of this thesis is also if the driver is holding the steering wheel versus the paper where the position of the car is predicted.

3.0.3 Distributed Sensor for Steering Wheel Grip Force-Measurement in Driver Fatigue Detection

One way to detect if the driver is holding the steering wheel is through touch sensitive sensors on the steering wheel (Baronti et al., 2009). The sensors then outputs if a driver is holding the steering wheel based on the force put onto the steering wheel. In the paper they use 15 sensors equally distributed around the steering wheel and each is equipped with a micro controller to also know where the driver is putting it's hands (Baronti et al., 2009). The goal of the paper is not only to detect if the driver is holding the steering wheel but also to understand if the driver is fatigued and therefore the capactitve touch sensors are complimented with car sensors signals such as steering angle and vehicle speed (Baronti et al., 2009). The difference from this thesis is that the system will not be estimating fatigue of the driver, instead only classifying whether the driver has their hands on the steering wheel while not using touch sensors.

3.0.4 Development of a new capacitive matrix for a steering wheel'spressure distribution measurement

The paper (Garinei and Marsili, 2014) shows different types of sensors used on the steering wheel to detect if the driver is holding the steering wheel or not. One drawback with the capacitive sensors is that when bending them around the steering wheel, there are measurement inaccuracies because of non linearity's in the sensor. However, optimization techniques have been developed to counteract this. The paper shows how drivers react in different scenarios such as fast acceleration or steering but does not use the sensors for detecting if the driver is holding the steering wheel (Garinei and Marsili, 2014).

3.0.5 On Performance Evaluation of Driver HandDetection Algorithms:Challenges, Dataset, and Metrics

The paper (Das, Ohn-Bar, and Trivedi, 2015) shows how different camera based approaches can recognize the hands of the driver or passenger and also the steering wheel.The paper introduces several camera angles to show which ones are good respectively worse for detection (Das, Ohn-Bar, and Trivedi, 2015). All recorded data is annotated with where the hands are and bounding boxes are created for this. If the bounding boxes of the hands overlap the steering wheel, the driver is holding the steering wheel. The camera takes as input all RGB colors and can miss-classify red hues as a hand leading to false outputs. False outputs are also found when the bounding boxes are poorly fitted something that is in the papers future work (Das, Ohn-Bar, and Trivedi, 2015).

3.0.6 Hands on the wheel: a Dataset for Driver Hand Detection and Tracking

The paper (Borghi et al., 2018) creates an annotated dataset for detection of hands on the steering wheel and an algorithm to detect where the hands are located. The steering wheel detection works by fitting an ellipse with five points to the steering wheel manually in the beginning. The image's are then transformed from a ellipse to a circular from through a homography matrix. Lastly, the image is unrolled to linear space where overlapping hands with steering wheel can be mapped (Borghi et al., 2018). However, the system is running on a real computer with a high performing CPU to be a real time detection algorithm.

4
Methods

This method is divided into two sections, the sensor based approach and the camera based approach. The sensor method is presented first followed by the method for the camera. In the end a description of the online system is presented.

4.1 Sensor based method

For this solution, the already existing sensors from the car are used to determine if the state is either hands on or hands off. The cars all have some standard equipment when it comes to sensor but most will not be of use in this project. Since the interesting parts are the steering system, these are the sensors that will be investigated to come up with a solution. The signals from these sensors can be accessed from the Controller Area Network (CAN bus). Two different neural network architectures are created to compare the results, both are using the same dataset and inputs.

4.1.1 Dataset

The data collection based on sensors are collected by driving a car with driving assist and logging the sensor data through the software CANoe, which collects data from the vehicles CAN-bus. The CANoe software samples the data at around 10 Hz depending on signal. However, a faster sampling frequency can be achieved on the CAN-bus itself and therefore the CANoe data is interpolated to 1 kHz using the previous value. Since it is important to accurately label the dataset, the ground truth is also logged by the passenger pressing a key to switch between hands on and hands off states. The states are represented as a zero for hands on and a one for hands off. To make sure the ground truth label is accurate, 0.25 seconds are removed from the dataset before and after the key is pressed.

When creating a training dataset, it is important that the data is balanced, meaning that there should be similar amount of situations where hands are on the steering wheel and when the hands are off. Otherwise the network will have a bias towards one or the other which will affect the training off the neural network. This is solved by having a timer set to fifteen seconds between switching states when collecting data. The dataset is around 6 hours of logged highway data, with most of it from a Volvo S60. For the testing dataset, the goal was instead to resemble a realistic situation and is made up of both long and short hands off situations with varying duration of hands on driving in between. To make sure the algorithm is generalized

the test is instead from a Volvo XC90 and is in total just over 1.5 hours driving. The test dataset includes a total of 5800 predictions spaced by 1 second.

4.1.2 Estimating the Driver Torque

Since the goal is to be able to differentiate between when the driver's hands are on the steering wheel and when they are off, it would be possible to look at the torque applied to the steering wheel from the driver. Since the steering wheel will still be moving from the autopilot system, other sensors will not be affected from this change of state. The problem is that there is no torque sensor in the steering wheel. However, there are other existing sensors which can be used to estimate the steering torque by using a state-space model, including a torque sensor in the torsion bar and steering angle sensors in the steering wheel and pinon below.

4.1.2.1 State-Space Model

in this thesis, the friction is ignored for an easier model and the goal is to model the driver torque (M_s), therefore M_s is converted to a state instead of an input. Doing so produces an extended state-space model which cannot model high frequency changes as there is no information about the extended state's derivative. The extended state space model is created using the following states $\vec{x}$ and input u,

$$\begin{aligned} \vec{x} &= \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \end{bmatrix} = \begin{bmatrix} \delta_s(t) \\ \frac{\mathrm{d}\delta_s}{\mathrm{d}t} \\ \delta_{pin}(t) \\ \frac{\mathrm{d}\delta_{pin}}{\mathrm{d}t} \\ M_s \end{bmatrix} \\ u &= M_{rack} = F_{rack}/i_{rp} \end{aligned} \tag{4.1}$$

The states are modeled with the following constants in the state-space model,

$$\begin{gathered} a21 = -c_{tb}/J_s \\ a22 = -(b_s + b_{tb})/J_s \\ a23 = c_{tb}/J_s \\ a24 = b_{tb}/J_s \\ a24 = 1/J_s \end{gathered}$$

$$\begin{gathered} a41 = -c_{tb}/J_{pin} \\ a42 = b_{tb}/J_{pin} \\ a43 = -(c_{tb} + c_{belt} \cdot i_{mot}^2) \\ a44 = -(b_{tb} + b_{belt} \cdot i_{mot}^2 + b_{pin})/J_{pin} \end{gathered} \tag{4.2}$$

$$\begin{gathered} c51 = c_{tb} \\ c52 = b_{tb} \\ c53 = -c_{tb} \\ c54 = -b_{tb} \end{gathered}$$

from the constants, the following matrices are set up in continuous time,

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ a21 & a22 & a23 & a24 & a25 \\ 0 & 0 & 0 & 1 & 0 \\ a41 & a42 & a43 & a44 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1/Jpin \\ 0 \end{bmatrix} \tag{4.3}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ c51 & c52 & c53 & c54 & 0 \end{bmatrix}$$

the continuous state-space model is,

$$\begin{aligned} \frac{\mathrm{d}\vec{x}}{\mathrm{d}t} &= A \cdot \vec{x} + B \cdot u \\ \vec{y} &= C \cdot \vec{x} \end{aligned} \tag{4.4}$$

the output vector y contains and estimate of the states x1,x2,x3,x4 and M_{tb} which can be compared to the car's sensor values in order to estimate M_s using a Kalman filter. The model further has to be discrete to run online in the car and using the known sample time T_s it is converted from the continuous state-space model using the following,

$$\begin{bmatrix} F & G \\ H & 0 \end{bmatrix} = \exp\left(\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} \cdot T_s\right) \tag{4.5}$$

where F is the discrete version of A, G is corresponding to B and H is equal to C. The discrete state-space model used is the following,

$$\begin{aligned} \vec{x}_{n+1} &= F \cdot \vec{x}_n + G \cdot u \\ \vec{y}_n &= H \cdot \vec{x}_n = C \cdot \vec{x}_n \end{aligned} \tag{4.6}$$

the equation 4.6 is used as the reference in a Kalman filter for estimating M_s, where the output vector $\vec{y}$ is compared to sensor values in the car.

4.1.2.2 Kalman filter

The Kalman filter uses the state-space model's y output and the sensor values corresponding to the output to determine estimates of the states in the model. It is done through a predict and update sequence as in section 2.3. The Kalman filter uses the

state-space model. Because the model is extended and no information about the M_s derivative is known, a low R variance of 0.1 on the Gaussian distribution is chosen as it gives higher trust towards the sensor measurements. The sensor measurements are filtered through a Butterworth lowpass filter with a cutoff frequency of 25 Hz to remove high frequency noise. Together with the Fourier transformed signal, the driver torque (M_s) amplitude is used as input signal to the neural network, spaced equally every twentieth value to get a total of 100 values for 2 seconds of data using the 1 kHz sampling frequency.

4.1.2.3 Fourier transform

From the Kalman filter the estimated driver torque is converted to the frequency domain using an FFT. Since only lower frequencies are sough after. The input sequence of 2000 values is padded with zeros to 2048 values to be a power of two. The output is truncated to the first 100 values to make it less computational intensive for the neural network and because the relevant frequency information is below 5 Hz (Moreillon, 2017).

4.1.3 Neural Network

To use the collected data in training, the Keras architecture for Python is chosen because of the easy implementation. In Keras, convolutional and LSTM layers can be placed and built to a model, which is then trained by calling a fit method to the model. Keras has several activation functions, regularizes and callbacks to save model weights and history. The input to the neural network are the FFT vector of 100 values and the torque amplitude of 100 values in two separate channels.

4.1.3.1 Convolutional Neural Network model

For the Convolutional neural network (CNN) model, the network structure is shown in Figure 4.1. Starting with a filter size of 64 in the first layer followed by max pooling. There are two more convolutional layers with 128 and 256 respectively filters. The network then goes through 2 fully connected layers with 512 and 1024 neurons. All convolutional layers have a kernel size of 3 and are zero padded to fit the input. The fully connected layers have 20% dropout to combat overfitting.

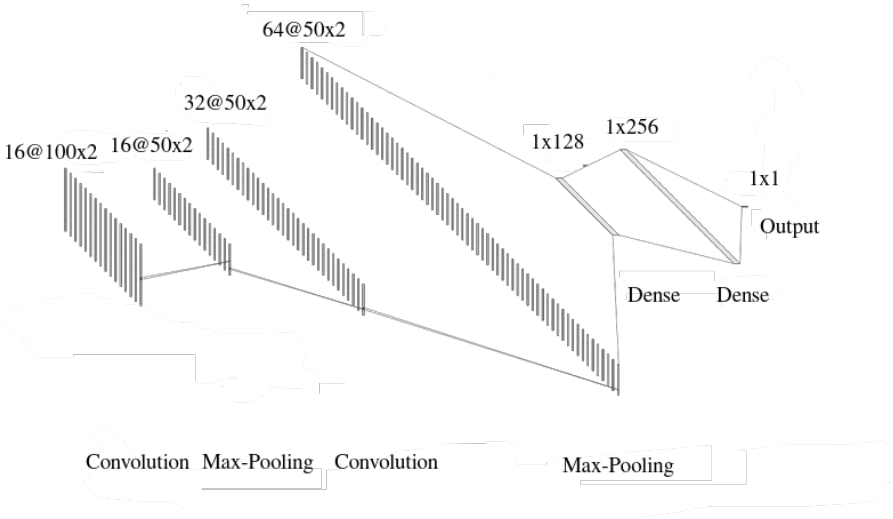


Figure 4.1: 1D convolutional network used for hands off classification. Input to the network is the estimated driver torque and the corresponding FFT of 2 second data. Five layers in total: First a convolutional layer with 16 filters, followed by a max-pooling layer and two more convolutional layers with 32 respectively 64 filters. Ended with two fully connected layers with 128 and 256 neurons.

4.1.3.2 LSTM Model

The LSTM architecture is implemented using Keras, using two LSTM layers with 64 neurons each followed by a fully connected layer with 128 neurons. The last step is a sigmoid activation which gives an output between 0 and 1 and the full architecture can be seen in figure 4.2. The LSTM architecture is smaller compared to the CNN because the LSTM network cannot be parallelized, the current output depends on a memory of previous outputs.

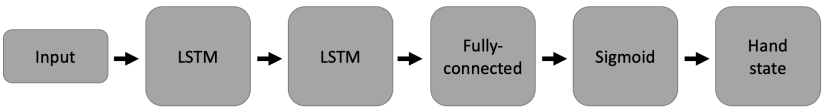


Figure 4.2: The used LSTM architecture. Using two LSTM layers of 64 neurons each, followed by a fully-connected layer of 128 neurons. The class is predicted through the sigmoid activation function in the last layer, giving an output between 0 and 1.

4.1.3.3 Confidence Level Algorithm

To make sure outliers do not sway the results a leaky bucket algorithm is used to keep a confidence level based on current output and the previous 2 outputs, giving a total of 3 samples in the memory. The algorithm is designed so that the current

output has a higher weight and each of the previous values are weighted with an exponential decay, normalized to be between 0 and 1. Afterwards 7 confidence levels are set as follows,

- Confidence below 0.2 = Network is certain of hands on scenario.
- Confidence between 0.2 and 0.4 = Network is quite certain of hands on scenario.
- Confidence between 0.4 and 0.5 = Network is uncertain but leaning towards hands on.
- Confidence between 0.5 and 0.6 = Network is very uncertain and no prediction can be made.
- Confidence between 0.6 and 0.7 = Network is uncertain but leaning towards hands off.
- Confidence between 0.7 and 0.9 = Network is quite certain of hands off scenario.
- Confidence above 0.9 = Network is certain of hands off scenario.

The levels are set after testing different levels and promotes the algorithm to switch quicker towards hands on compared to hands off. The quicker switch to hands on is a wanted attribute for the driving experience. The confidence algorithm sets the output hands on if the confidence is below 0.5 and hands off if the confidence is above 0.6. It keeps the previous state if the confidence is in the middle.

4.2 Camera based method

For the camera approach, a MakerHawk Raspberry Pi camera was mounted in the car and the dataset was created by collecting pictures while driving. Figure 4.3 is showing two images taken from the training dataset. Two methods are made which will be compared in terms of performance. The methods differ in all aspects from dataset to the execution and is therefore presented in two different sections. The first method will be using object classification instead. This method works by feeding images to a network which will then output one of two states. Either hands on or hands off. The second method is object detection which will try to detect hands in images which will then be compared to the position of the steering wheel to determine if the hands are on the steering wheel.

Figure 4.3: Sample images from the mounted camera. The camera is located in the ceiling of the car behind the sun visor.

4.2.1 Object Classification

In object classification, a neural network takes an image as an input and outputs a prediction of which class the network believes that the image belongs to. The output is a probability that represents the certainty of the prediction from the neural network.

When doing an object detection, the first step is to construct the dataset. Then, a network model is created which will train on the dataset and later, be used for making prediction using the image input from the camera in the online system in the car.

4.2.1.1 Dataset

The dataset consists of images separated into the classes the network is supposed to learn, hands on and hands off. There are no existing datasets which could be used for this problem and therefore a dataset is created. A larger dataset means a better result but there are other options as well for expanding a image dataset without adding more images. This will be discussed in the next section.

It is important to add as much variation as possible to the dataset in order to force the network to learn general features. If there is a large variation, the network will focus on the important parts, such as the steering wheel and the arms and hand instead of less important features such as the driver's clothes. Other important aspects that are not included in the dataset includes gloves, different skin tones and other steering wheel colors. If a situation that has not been trained on shows up it could make the network less accurate and the more varied the dataset is, the better. There are many ways one can hold a steering wheel so while making the dataset, varied hand position are important.

The images used in the dataset where sampled from the camera once every third

second and then manually labeled either on or off. The final dataset consists of over 8000 images which is divided into training, validation and test with a split ratio off 0.6|0.3|0.1. For the entire dataset, 52% of the images were hands on and 48% where hands off, ensuring that the dataset is balanced. The file structure for the dataset can be seen in figure 4.4. In the dataset, different sets of clothing were used for training, validation and testing to make sure the images are not too similar and see that the network learns the general features.

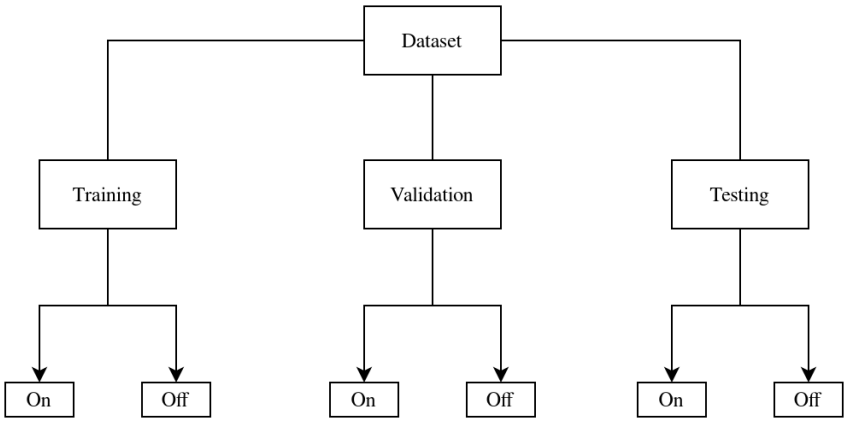


Figure 4.4: Structure of the dataset used for training, validation and testing.

4.2.1.2 Data Pre-Processing

Before feeding an image to a network, the image needs to be processed to maximize the networks ability to make valid predictions. The first step is to crop out unnecessary parts of the image which doesn't add valuable information and only add noise. In this case, the passenger side of the car is removed while keeping the driver and the steering wheel, which are the interesting parts.

The number of parameters in a neural network is dependant on the size of the input image. If a large image is to be fed through a network, it will demand more computational resources compared to a smaller image. Depending on what features are to be recognized, a larger resolution does not always mean better result. Since the features of a hand on a steering wheel is not dependant on a high resolution, the images are resized before entering the network. A commonly used image size by several established networks for images are 224x224, and this resolution will be used in this project. This image size has been established as a valid size both in terms of keeping most features in the image while also being small enough to use reasonable resources.

When training networks on images, it is common to use different transforms which makes the network learn more general features and reduces overfitting. This is especially helpful if the goal is to recognise something which can be in a varied setting. For this project, the setting is mostly constant and there are few thing changing between different situations. There will always be a driver seat and a steering wheel, and the orientation is constant. Because of this, not all transforms will be of help in this situation. For instance, it is unnecessary to apply a random horizontal flip on

images since the camera will never be upside down. There are however some transforms that will be used to apply different brightness, hue, saturation and parameters which will emulate potential real world situations better and help generalize the network.

The last step before feeding the image to the network is to normalize the inputs. The network achieves better results and faster converging since the gradients are more stable if the input has a smaller range. Also, features are equalized so one feature is not dominating all others. A common normalization technique is to subtract the mean and divide by the standard deviation of the images in the training dataset. This is called a Z-score normalization. The same normalization is then applied to the validation and test datasets, and also any future inputs to the network. If all the available images in all datasets where used for normalization, some information from theses dataset would leak into the training which is important to avoid. Also, when using the system live, there are no datasets which means these values have to be set before hand. The result of the normalization is that the pixel values will be centered around zero with a standard deviation of one.

4.2.1.3 Neural Network Architecture

The neural network is built using Keras Tensorflow and consists of primarily two-dimensional convolutional layers as can be seen in figure 4.5, where the design of the neural network is presented. The last three layers are fully connected layers with the output layer consisting of one neuron which will output the binary classification. In order to achieve a network with a reasonable amount of parameters, max-pooling layers are added to reduce the spatial size of the input. There are also some batch normalization layers which can help the network decrease the number of epochs needed to learn. These layers normalize the inputs in the batch to zero mean and unit variance. The batch normalization layers are not shown in the figure below but exists after each convolutional layer.

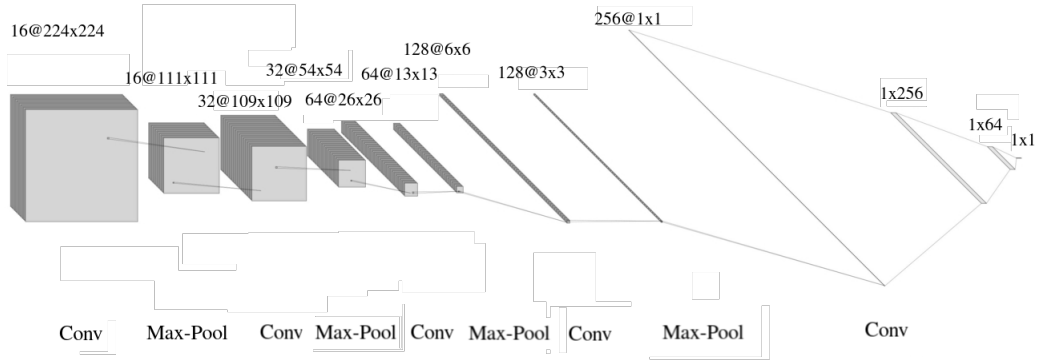


Figure 4.5: Neural network structure used for object classification. Batch normalization layers are not included in the image but exists after the convolutional layers.

4.2.2 Object Detection

The object detection model creates bounding boxes around the target with labels showing which pixels in a picture that includes a certain object. The dataset created includes hands from the dataset created for object classification and adds the ergohands dataset (Bambach et al., 2015), which contains annotated hands from people playing games to detect the general shape and feature of a hand. All data is annotated with square bounding boxes through the software VGG Image Annotator (Dutta and Zisserman, 2019). During training the dataset is extended by randomly altering the images, including flip, rotation and scaling. Because finding the position of hands is not enough to provide an output if the hands are on or off the steering wheel, another dataset and network is created to find the logo on the steering wheel. The logo network is only run on initialization and afterwards the position of the steering wheel assumed to stay constant.

4.2.2.1 Neural Network Architecture

For object detection a concept called transfer learning is commonly used because the object detection models are often large, and takes lots of data to train from the ground up. Instead a model already trained for general objects, often on the 80 general classes from the COCO dataset is used as a starting point. Transfer learning then modifies the neuron weights to fit the classes provided by the user in a smaller dataset and tweaks the network to fit the new situation. The advantage is that a smaller dataset can be used and less training is necessary to get the network up and running, with the disadvantage that a pre-trained network is needed. PyTorch provides a model-zoo with these pre-trained networks and the ResNet-50 model (He et al., 2015) was used as the starting point for both detecting the logo and hands.

4.2.2.2 Detecting the hands off state

The initial model finds the logo of the steering wheel, creating a bounding box with a height and width. From the mid-point of the logo bounding box an ellipse is created. As the camera is mounted manually each time, the ellipse is stretched out and moved manually to match the steering wheel for each video sequence, one example is seen in figure 4.6. The ellipse is made smaller then the real steering wheel considering the bounding box for the hand when holding the steering wheel overlaps quite well with the steering wheel.

The second model takes over, trained to find the position for the hands and predicts bounding square boxes around them. Because the model finds several boxes that could potentially be a hand, a built in PyTorch method is used to only keep the box with highest score if two boxes have an overlapping area of 50% or more. If the final bounding box overlaps with the ellipse of the steering wheel, it is considered a hands on situation. The overlap is calculated as in the paper (Groves, 1963) to see if the closest point to the middle of the ellipse is within the ellipse area. If no bounding boxes overlaps with the ellipse, it is a hands off situation.

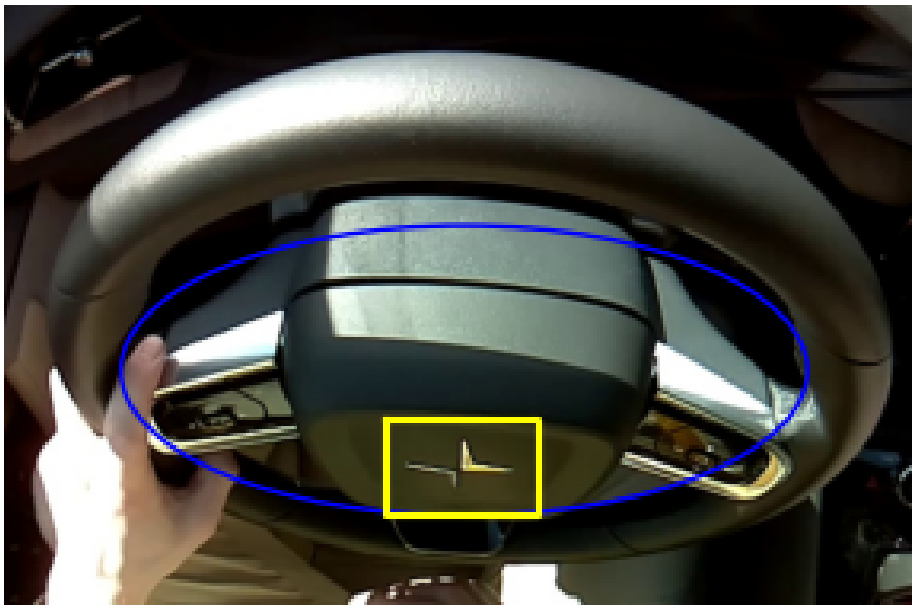


Figure 4.6: Algorithm to model the steering wheel, starting from the yellow bounding box of the logo and extending the blue ellipse to be slightly smaller than the steering wheel.

4.3 Implementing the model for online usage in a car

The models are implemented on a Raspberry Pi model 3b for online usage in the car. The Raspberry Pi emulates the processing power available in a car, and can be powered through a usb port. Because of the lower computational power, both the sensor solution with a CNN and the camera object classification solution is ported to C++. The library Frugally-deep is used to run both trained neural network models in C++. A matrix multiplication package emulating Matlab commands in C++, Eigen is used for the state-space model and Kalman filter. OpenCV is used for image processing and feeding the image in the right size and crop to the object classification model. The camera model and sensor model are running in parallel as two separate solutions.

4.3.1 Sensor solution

The signals are extracted from the CAN-bus using a PiCan board and the included python script for finding the correct signals, the signals are then sent to the C++ script through local UDP. The sampling frequency on the CAN-bus is 100 Hz, and is interpolated to 1 kHz to match previous work. The update frequency is set to 5 times per second to catch the switch between hands off and hands on quicker. The data is kept in a rolling vector that always contains the last 2 seconds of data. The result is sent through UDP from the Raspberry Pi for visualization.

4.3.2 Camera solution

The image consists of three channels and these three channels are each transformed in the same way as the training dataset using the z-score transformation. The z-score transformation variables are saved from the training set and are applied to each of the images so they are as close to the images from the dataset as possible. The image is then fed through the network which is built using the same package as in the sensor solution, but some packages are removed as they are not included in frugally-deep. The result is sent through UDP from the Raspberry Pi for visualization.

In the car, the camera is mounted behind the sunscreen which we can see in figure 4.7 below. The camera is connected through a 15 pin flex cable which is setup in the roof of the car, which is then connected to the Raspberry Pi.

Figure 4.7: The camera can be seen in the left figure. It is mounted above the steering wheel and behind the sun visor. In the right figure, the flex cable from the camera is connected to the Raspberry Pi.

5
Results

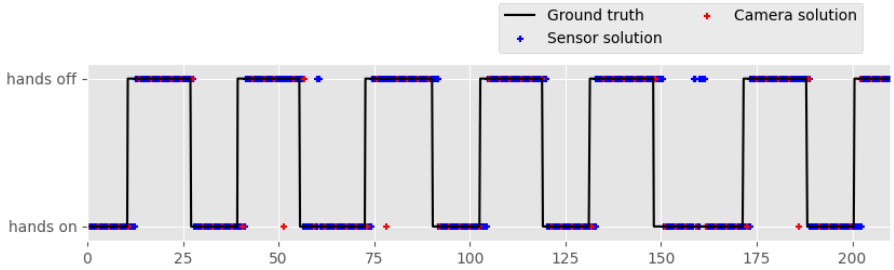
The results are presented as how the algorithm works online with both solutions running in parallel followed by the different network performances in both the sensor and camera solution.

5.1 Final product

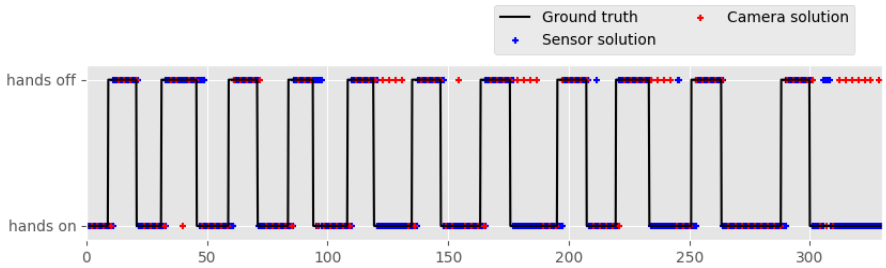
The final prototype runs the C++ script in real-time on the raspberry pi and detects hands off situations from the car's sensors and the camera. The sensor solution indicates hands off after 3 $\pm$1 seconds while reacting on in 1 second when going towards the hands on state. The sensor solution gives updates on average at 4.96 Hz, where each each calculation takes on average 0.032 seconds to complete and the update frequency could be increased. The computational time disregards the overhead of saving new sensor signal values which are done in another thread to not lose any information during the calculation of hands on or off. It keeps the hands off state for the full duration, but sometimes jumps to hands off during hands on driving for a short duration. The approach is sensitive to breaking for a car in front and sometimes jumps to a single hands off sample during these situations before returning to hands on.

The camera approach works as a separate solution with an average update frequency of 0.37 Hz indicating the hands on/off switch after 2 $\pm$ 1 sample. The camera is very good at finding hands off situations and therefore the decision boundary is set to 0.8, where a score higher than 0.8 is hands off and below is hands on. Because of the high decision boundary it shows some wrongful classifications for hands on during hands off driving.

A qualitative figure can be seen in figure 5.1 where the final selection is showed in figure 5.1a and 5.1b for both the camera and sensor version. The sensor version is updated frequently showing a short delay compared to the ground truth. The camera is updated slower but still predicts most situations correctly in figure 5.1a where the driver is holding the steering wheel with two hands at 10 to 2. The camera shows some problems in figure 5.1b, where the wrong state is predicted for longer time, here the driver is instead holding the steering wheel with only one hand, at the bottom of the steering wheel and 3 fingers at 3 o'clock.
In the figure 5.2 the neural network prediction is showed for both versions when the system is fooled when a weight of 350g is hung on the steering wheel. The sequence



(a) A good sequence. The sensor algorithm in blue shows a delay of a around 2 seconds compared to the ground truth moving towards hands off and 1 second moving back to hands on. There are some errors where the prediction is hands off during hands on driving. The camera version in red is updated slower and has a couple of errors showing hands on driving during hands off



(b) A sequence where it does not work as intended. The sensor algorithm in blue is quite robust with few wrong classifications and a delay of a around 2 seconds compared to the ground truth moving towards hands off. The red camera solution shows some errors for quite long periods of time, when the driver is only using one hand.

Figure 5.1: Two situations captured live in a Volvo V90 on the Raspberry Pi. The first one shows a situation where the algorithm performs well and the second one shows a situation where the algorithm does not perform well. The second situation is due to the driver only holding the steering wheel with one hand

shows sensor solution being uncertain, fluctuating between hands on and hands off predictions. Compared to the following hands on scenarios where the hand is the weight, a difference in the predictions can be noticed. The camera on the other hand solves all hands off situations. A similar pattern can be found in figure 5.1b when the driver is holding the steering wheel with one hand, the camera predicts hands off while the sensor shows hands on. In general the camera predicts high for hands on, and close to 1 for hands off leading to the high decision boundary.

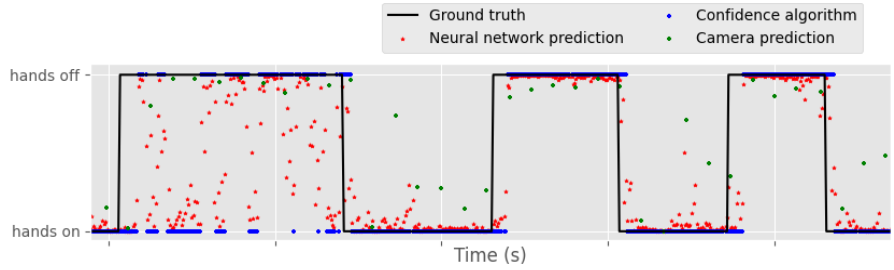


Figure 5.2: A situation where a weight of 350g is hung from the steering wheel in the first sequence followed by 2 normal driving sequences. The camera shows hands off in the first sequence while the sensor version fails. One thing to notice is that the camera predicts high numbers for hands on and almost 1 for hands off, leading to a high decision boundary for the camera approach.

5.2 Evaluating the Sensor Based Method

The first step towards a prediction is to use the signals from the car in a Kalman filter to estimate the driver torque and feed it through an FFT. In figure 5.3 a comparison of the estimated driver torque is shown versus a measured torque from a robot, driving at 60 km/h. The estimated driver torque does not handle high frequency shifts in the beginning but models the lower frequency in the end well.

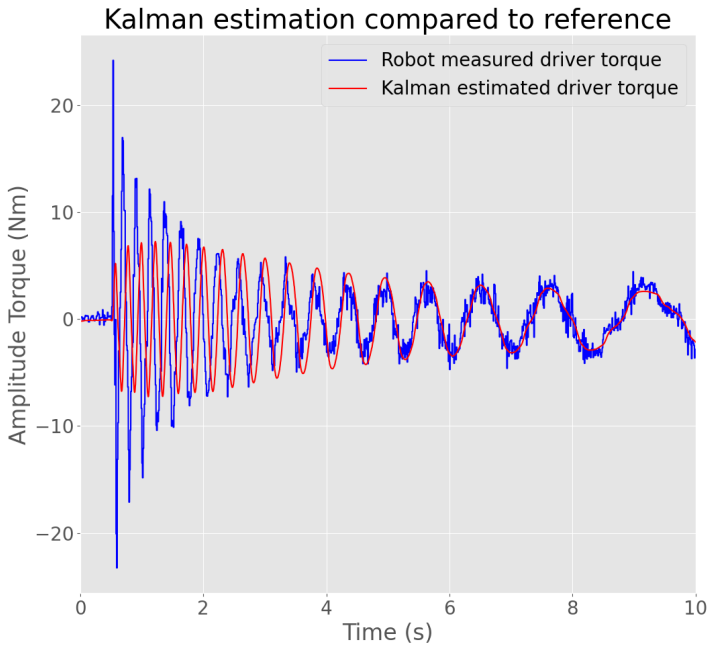
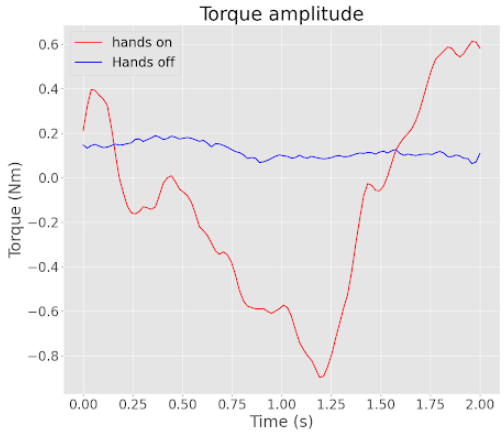


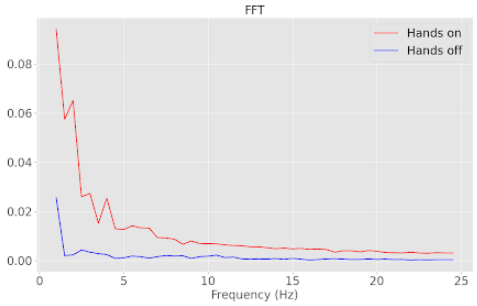
Figure 5.3: The Kalman filter estimation of driver torque (red) compared to a robot measured driver torque(blue). Varying torque applied over the duration.

To further evaluate if the Kalman filter is applicable to determine hands off situations during semi-automatic driving a 2 second sequence for hands on and one for hands

off is shown in figure 5.4 with both the amplitude and FFT. From the figure 5.4a it is a clear distinction between the torque for hands on and hands off, with the hands off situation being almost constant torque while the hands on situation is a signal with a period time. The difference between the period of both signals can be found in the frequency domain, using a FFT as in figure 5.4b. In the figure the hands on sequence shows the periodic behaviour with frequency spikes at around 2 and 4 Hz while the hands off situation has no frequency information.



(a) Estimated torque amplitude for hands off in blue and hands on in red. For hands on driving a curve with frequency information can be approximated, while the hands off scenario looks almost constant over time.



(b) Hands off scenario in blue, and hands on scenario in red. The plot shows frequency spikes at around 2 and 4 Hz for hands on and no frequency information for hands off

Figure 5.4: Estimated torque and the corresponding frequency information for one hands on sequence and one hands off sequence using two seconds of data.

5.2.1 Convolutional network architecture

The first model created is the convolutional one explained in section 4.1.3.1 using both the amplitude and FFT in two separate channels to provide a prediction of the hands on or off state. The model is trained and the training epochs are seen in figure 5.5. The network learns general features of the situations where the hands are on and off, as the loss of the validation data decreases steadily. The accuracy of the validation set increases with the training set and over time the network is learning. The validation loss is lower then the training loss because of the dropout layer, which limits usage of neurons in the fully-connected layers during training but uses all of them during validation. In general there is only a small gap between validation and training so the network is working as expected and is generalized towards the amplitude and FFT features of each class. The built in evaluation tool in Keras shows and accuracy of 95,2% accuracy and a loss of 0.196 on the test dataset. The test loss is higher compared to both the training and validation, because the test

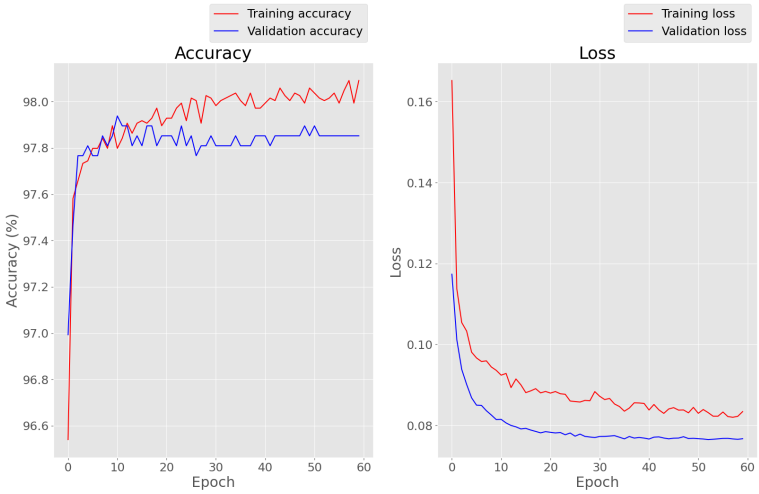


Figure 5.5: Learning curve of the model during training for 60 epochs. Validation loss is dropping together with the training loss while the accuracy's are following each other.

dataset includes more varying situations and short hands off situations. Most errors can be traced to the switching flank between hands on and off states, where the predictions are quite far off leading to a high loss for the initial samples after such a switch. The model performs well after the initial wrong classification and keeps the hands off classification throughout the full duration, switching quickly back to hands off.

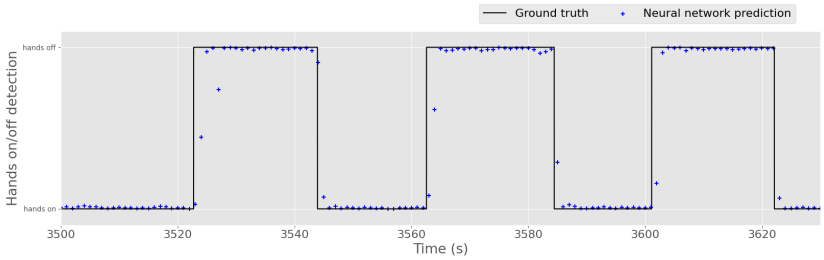


Figure 5.6: A short sequence from the test set highlighting the initial wrong classification when the flank moves from hands on to hands off and the opposite, which decreases accuracy of the model in the Keras evaluation tool.

Adding the confidence algorithm explained in section 4.1.3.3 to the neural network prediction to achieve a final selection of state, either hands on or off which is robust to single outliers. From a test of 10 minutes in figure 5.7 only two samples are outliers where both of them being a classification of hands off during hands on driving. The wrong classification of hands off during hands on driving is seen as less critical compared to the opposite and is an acceptable trade off to having an algorithm that is robust during hands off driving.

For a sequence where the driver is either holding the hands on or off, an error

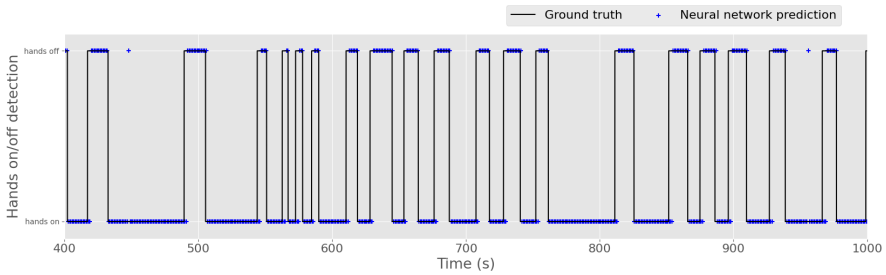


Figure 5.7: Confidence algorithm with selection of state using the CNN architecture. Most samples are classified correctly, with two outliers predicting hands off during hands on driving. A delay of a couple of samples are seen when the driver stops holding the steering wheel

is if one sample during the sequence is classified wrong. In the total test there are 251 sequences with one more hands on situation because the test starts and ends with hands on driving. In the table 5.1 all sequences are summarized, the accuracy is 98.8 % Over the sequences there are in total 3 outliers classified wrongly, all classifying hands off when it should be hands on. In the 3 outliers, there is 1 single sample wrong before returning to the correct label. It does not show any problems dropping hands off classifications during hands off driving.

Table 5.1: Confusion table using the CNN architecture. Three sequences includes a missed sample, where all missed samples are a prediction of hands off when the true label is hands on.

Predicted class / True class	Hands on	Hands off
Hands on	123	3
Hands off	0	125

5.2.2 LSTM network architecture

Another network based on the LSTM architecture was also created for comparison, using the same inputs and data as the CNN. The training and validation loss is in figure 5.8, the validation accuracy and loss is higher than in the CNN model. There is also a larger generalization gap, where the LSTM validation cannot quite reach the training loss even after several epochs and training for longer results in overfitting 5.8. Compared to the CNN, the LSTM model moves quickly from a hands on to off classification without the couple of samples prior to the switch, often taking 1 sample until the prediction is switched to hands off. The LSTM works as good as the CNN switching from hands off to hands on, reacting directly on the next sample. The figure 5.9 shows predictions for a short sequence in the test dataset highlighting the quick reaction time in a couple of samples between hands on and off, while the

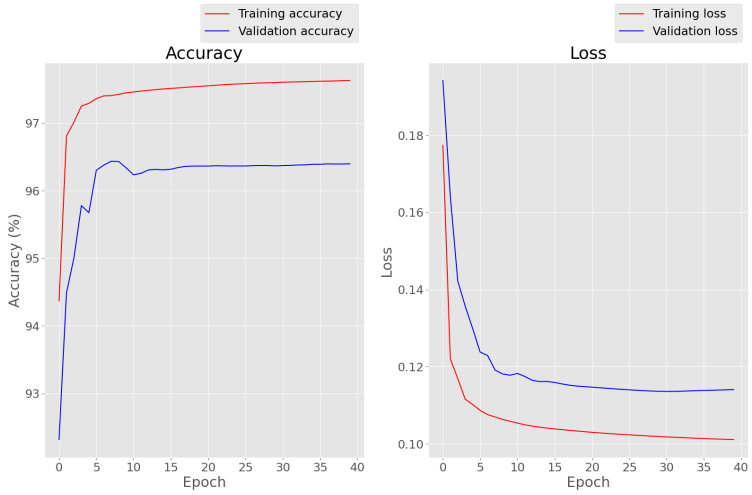


Figure 5.8: LSTM: Learning curve of the model during training for 60 epochs. Validation loss is dropping together with the training loss. The validation loss does not quite reach the training loss

predictions are very close to 0 during hands on driving and very close to 1 during hands off driving.

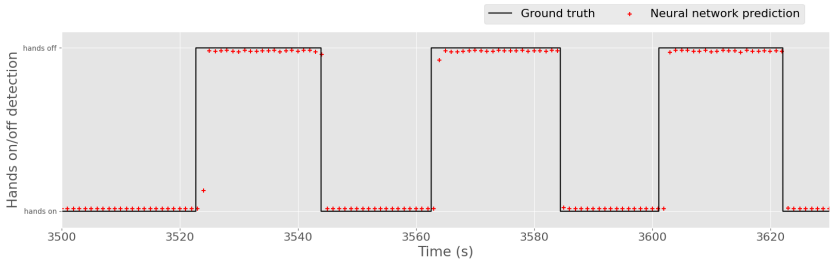


Figure 5.9: LSTM: A short sequence from the test set highlighting the initial wrong classification when the flank moves from hands on to hands off and the opposite, which decreases accuracy of the model in Keras evaluation tool.

Over the same sequence of 10 minutes driving using the confidence algorithm the LSTM performance is the same as the CNN with two outliers seen in figure 5.10. In total the LSTM performance can be seen in table 5.2. A wrong classification is if 1 or more samples are classified wrong during the hands off or hands on sequence, giving an accuracy of 97.2% over the testing data. The LSTM shows one critical error where the output is hands on during hands off driving. A total of 8 samples are predicted wrong out of the 5800 total as there are 2 samples in a row predicting hands on driving during the critical error.

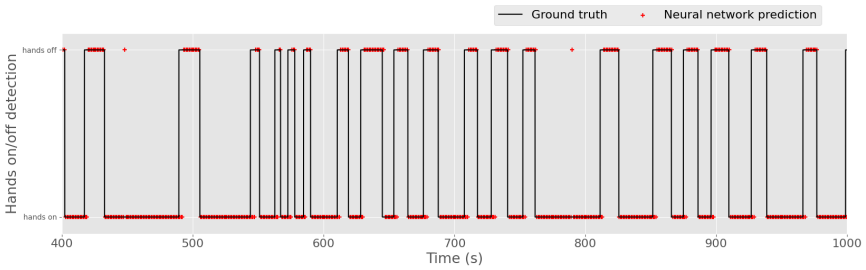


Figure 5.10: A sequence over 10 minutes driving using the confidence algorithm and the underlying LSTM network. The output shows two wrong classifications over the duration.

Table 5.2: Confusion table using the LSTM architecture. The LSTM architecture shows one error predicting hands on during hands off driving, and 6 errors predicting hands off during hands on driving.

Predicted class / True class	Hands on	Hands off
Hands on	120	6
Hands off	1	124

5.3 Evaluating the Camera Based Method

This section shows the results of the two separate camera based solutions. First, the results from the object classification is presented and then the object detection method.

5.3.1 Object Classification

The neural network was trained on the dataset for 300 epochs and the result of the training can be seen in figure 5.11. Here, both accuracy and loss is represented for both training dataset and validation dataset for each epoch. The accuracy is quickly rising for the early epochs before converging slowly towards a steady state. The behaviour of the loss is opposite, where it quickly goes down before converging to a steady state as well. When the loss is not getting lower for some number of epochs, as in figure 5.11, training is complete.

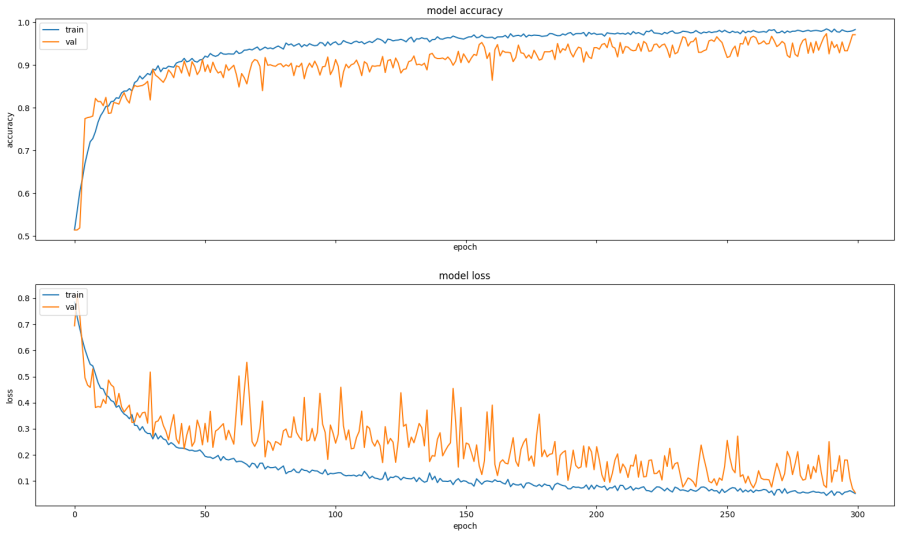


Figure 5.11: Accuracy and loss for the training and validation datasets for the training of the neural network for 300 epochs. The above figure presents the accuracy of the model and the bottom figure presents the loss.

After every epoch, the network is saved only if the loss is lower compared to the last saved network. This ensures that after 300 epoch, the best model is saved and available to . The best model during this training occurred at epoch 258. This network model with the lowest loss had a value off 0.034 for training and 0.046 for the validation loss. The accuracy of the network was 99.7% for training and 99.3% for validation. The test set had an accuracy of 97.9% and a loss of 0.079.

5.3.1.1 Classification of Images

On the entire test dataset, which consists of 528 images, there were a total of 11 classifications that the network predicted wrong. There were 5 misclassifications from the off-class and 6 wrong from the on-class which can be seen in table 5.3.

Table 5.3: Confusion table for the test dataset using the object classification method. The system misclassifies 6 hands on images and 5 hands off images.

Predicted class / True class	Hands on	Hands off
Hands on	267	6
Hands off	5	273

In figure 5.12 below, some sample images are shown which the network had trouble predicting.

(a)
True label: On
Network prediction: Off

(b)
True label: On
Network prediction: Off

(c)
True label: Off
Network prediction: 0n

(d)
True label: Off
Network prediction: On

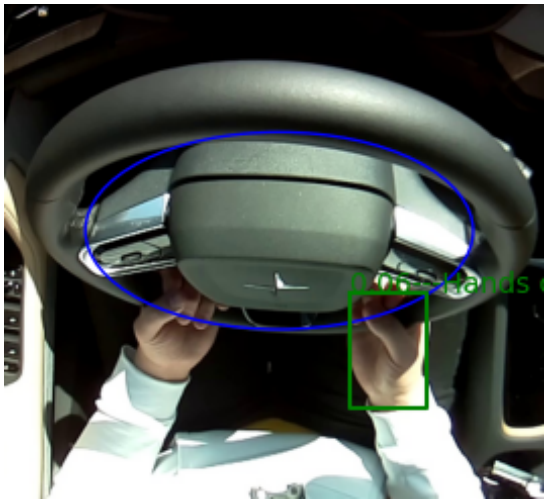
Figure 5.12: Some examples of images that were misclassified by the neural network that represents the general problems for the system to classify. Images (a) and (b) are often misinterpreted as hands off and image (b), when the hand is covered by the steering wheel can be misinterpereted as hands on. Image (d) is between states and therefore difficult to classify.

5.3.2 Object Detection

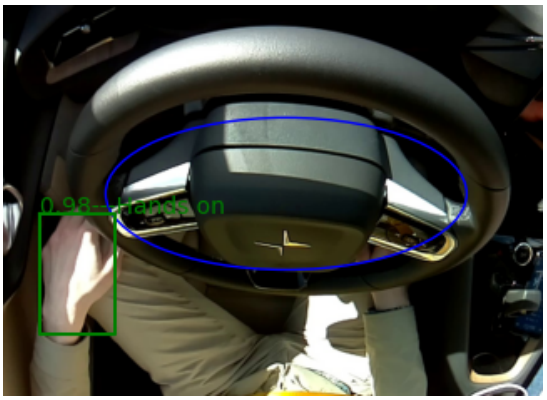
The object detection method uses a pre-trained ResNet-50 model from PyTorch and is trained for 30 epochs to fine tune it into detecting hands instead of the pre-trained classes. The network achieves a precision of 89% when accepting bounding box predictions that are overlapping with ground truth of 50% or more, but only a precision 20% when the overlap has to be above 75%. In general the network has problems detecting hands below the steering wheel, and holding the bottom of the steering wheel seen in figure 5.13a where only one hand is found, with a low score. When tested on a subset of 166 pictures from the object classification test, consisting of only one video because the ellipse is matched to the steering wheel manually. The test results can be seen in table 5.4. There is one misclassification for hands on during hands off and 6 misclassifications for hands off during hands on. A total of 95.7% accuracy. The downside is that the ellipse is matched to the steering wheel for each run and not generalized.

Table 5.4: Confusion table for the test dataset.

Predicted class / True class	Hands on	Hands off
Hands on	93	6
Hands off	1	67



(a) hands on scenario where the network only finds one hand, and the prediction for that hand is very low. A common problem is to detect hands low on the steering wheel



(b) hands off scenario where the bounding box for the hand is large enough to overlap with the steering wheel resulting in a wrong classification

Figure 5.13: The most common errors for wrong classifications. In 5.13a the hands are on the steering wheel but the detection model has a hard time finding the hands. In 5.13b the network finds the hand with a high prediction, but the bounding box overlaps with the steering wheel slightly.

6
Discussion

This section includes the discussion of the two solutions. First is the sensor based solution where the different neural network are compared. Following is an evaluation of the performance between the two camera based method. The online versions of the sensor method and the camera method are compared as two completely separate solutions as well as a potential solution involving both sensors and camera.

6.1 Sensor Solution

The sensor based solution was successfully implemented as an online version capable of running directly in the cars. The system is able to detect the state of the drivers hand for most situation and the performance on the Raspberry pi ensures that the update rate is sufficient to comply with the current laws regarding semi-autonomous cars. The sensor solution is a robust solution when the driver does not intentionally fool the system.

However, it is possible to mislead the system into thinking the hands are on the steering wheel. Since the system uses the force applied to the torsion bar to decide between states, any force should in theory work if applied to the steering wheel. It is possible to hang something from the steering wheel of appropriate mass to mislead the system into thinking that the driver has control of the vehicle. As seen in figure 5.13 the predictions from the network are not necessarily hands on all the time and instead fluctuates, so there could be information in the frequency domain to differentiate between a static object and a hand applying the same amount of force. Another solution is to use a camera which will not be disoriented by similar methods and ensures that hands are on the steering and harder to mislead.

6.1.1 Approximating the Driver Torque

When approximating the driver torque, some data gets lost in the process. The model is done using an extended state space model of two degrees which has some limitations. One such limitation is that the estimation works well for lower frequencies but is struggling when it comes to higher frequencies. Since the system is mainly to be used at highway driving, there should not be any situations that need the higher frequency from the steering. However, some frequencies may be lost which are not related to the steering itself but from vibrations which are present in the entire steering system generated from the car driving at high speeds on the road

and the friction which is disregarded. These frequencies might help the network to recognize hands of or hands on situations. Especially since the solution involves using an FFT which is dependant on the frequency. It is possible that using a more advanced state space model to get more information about the estimated driver torque, could help the network perform better at separating between the two states and even help with differentiating static objects and hands.

6.1.2 Network Architecture

This project featured two different network architectures which were the two most commonly used for analysing time series; LSTM and CNN. Looking at the result presented earlier, both networks performs well in most situations. The LSTM had one situation where it dropped the hands off predcition and a couple more situations where it predicted hands off during hands on, but gave a quicker switch towards hands on and hands off. In the thesis emphasis was put on recognizing situations of hands off driving and not to make the quickest reaction, as 15 seconds are allowed according to regulations United Nations, 2017-11-30. In general both the LSTM and CNN detects hands off situations in most cases, but the CNN has a slight edge on accuracy with the trade-off being slightly slower at detecting the switch. As both solutions uses the same dataset and inputs, it is possible to make the switch to a LSTM solution in the online version, if a quicker reaction time is necessary.

6.2 Camera Solution

For the camera solution, it is also implemented to the Raspberry pi running online but with a slower update frequency. The average update frequency of 0.37 Hz, yields a new state every 2.7 seconds and in theory it indicates hands off driving within the 15 seconds time limit set by the regulations. However, there is an inconvenience for the driver waiting 2.5 to 3 seconds on average for the system to react.
The camera solution is more robust against fooling the system by hanging something on the steering wheel, but more sensitive to the training data. If a situation that is not trained on shows up, the camera solution gives a less accurate result compared to similar situations in the training data.

6.2.1 Limitations of the Dataset

Perhaps the larges limitation when it comes to the object classification solution is the lack of variations in the dataset. Due to the limited time and resources during the project, the dataset had to be quite small compared to what is normally used in projects. The dataset is built up from over 8000 images using two people with fifteen different outfits.
The solution is a proof of concept and if the system were to be implemented in a car, the dataset has to be extended by a large margin. There are a lot of other body types, clothes, accessories and even weather conditions can confuse the algorithm in the current implementation. If one were to use gloves or a wheel cover, the system

would not be able to differentiate between states nearly as well as without them. Extending the dataset is the most important and time consuming part of the camera based solutions and especially for object classification.
For object detection, the state of the dataset is better. This is due to the many already existing datasets which includes labeled hands for using in detection networks. This solution is not dependant on the environment as much as object classification since the focus is on finding hands in images. This means that object detection would be simpler to use when it comes to the dataset, and not as many situations has to be accounted for.

6.2.2 Object Detection or Object Classification

For the camera based solution, two different approaches were tested; object detection and object classification. Both solutions provided a working system but each one has some limitations and challenges.

Object detection is a more complex solution since the network is only detecting hands in the images, leading to more subsystems for it to work as a hands on detection system. For instance, a second neural network is used in order to find the steering wheel based on the logotype in the center of the wheel. This check is done at the startup of the system and the ellipse matching the steering wheel is not generalized. With a more similar position for the camera in each car, the same parameters for the ellipse could possibly be used but there might also be problems with field of view differences when moving the steering wheel position for different drivers. Another issue is that a square bounding box does not always match the hand very well, especially when the fingers are stretched out and away from each other, making a large hand and an even larger bounding box. There is possibilities to use similar networks with keypoints or polygon bounding boxes to fit the hand even better. The two biggest downsides for object detection at the moment is to make a general steering wheel bounding box and the computational time to have it running in real-time.

Object classification is only dependant on the neural network to function. The neural network takes an input in the form of an image and outputs a classification directly. The problem with such a solution is that the system itself acts as a black box with an input and an output. Unlike object detection, which is detecting hands which are then set in proportion to the location of the steering wheel, object classification are looking at features which are unknown outside of the neural network. This uncertainty could potentially make the network in object classification misclassify images due to unforeseen content. The dataset is therefore extremely important and the biggest downside for object classification.

In terms of performance, object classification is more computationally efficient since the neural network is smaller and therefore has less parameters, meaning less calculations. However, even the more computationally efficient object classification is too slow to run as a solution on it's own with an update frequency of 0.37 Hz on

average. There is possibility to move away from the current C++ package Frugally-deep to a more efficient neural network package. Object classification has a higher accuracy compared to object detection, and seems to be the better method to use, but further research has to be made with an extended dataset to see how robust it is in all situations.

6.3 Online Implementation

The sensor solution and the object classification method where successfully ported C++ and were able to run on the Raspberry Pi simultaneously. The update frequency for the systems were limited by the computational power of the Raspberry Pi to update the sensor solution at around 5 Hz with an average calculation time on the Raspberry of 0.032 seconds making it possible to double the frequency if necessary.

The camera at averaged an update frequency of 0.37 Hz, limited partly by the usage of the C++ library frugally-deep. This library has limitations since it is written as a hobby project and contains some bugs. For instance, when using a dilation rate in the neural networks for object classification slowed down the network by a factor of twenty. There were other features that were causing the networks to slow down by an unreasonable amount which had to be removed and worked around. This could be solved by changing to a more C++ friendly machine learning package. A native C++ machine learning library could also contain the possibility of using more than one core in the CPU, further accelerating the process.

6.3.1 Performance online

Both solutions perform slightly worse online compared to the testing offline. The sensor solution is made for a system with 1 Hz update frequency offline and the confidence algorithm keeps the current sample and previous 2 samples, giving a 3 second memory. The online implementation uses the same confidence algorithm with 3 samples but has an update frequency close to 5 Hz, meaning that after 0.6 seconds it can switch state leading to a short set of outliers promotes a switch wrongfully. The problem is solved by keeping a longer memory in the confidence algorithm which corresponds to the update frequency used. To further reduce the false positives an even longer memory can be used since the solution should warn within 15 seconds. The camera solution is sensitive to situations that are not trained on, and depends on the dataset as seen in 6.2.1. The neural network used in the live camera method does not match the training network because of limitations in the C++ package.

6.3.2 Sensor Based or Camera Based Solution

Both methods have their respective strengths and weaknesses. The sensor solution is the most simple option due to being only a software implementation using only existing sensors fitted in all cars. In order to implement the system, it would only

need computational resources from the core system. The sensor system also only depends on the steering gear in the car, and does not depend on any outside modifications the user does. However, although the system is fairly general when it comes to different models of cars, the system has trouble when changing to another platform because of the difference in the state-space model constants and inputs. This means that in order to fit the sensor based system for other platforms, it would be necessary to train the neural network on the new platform, which includes a new dataset. With the current implementation on the Raspberry Pi, the computational time for the sensor system is 0.032 per update making the system a fully functional real-time system.

The camera based solution on the other hand would work well on all models, regardless of platform. Although, there exists other variations in the cars which will make the system perform less well. Another problem is that all variations that does not work well depends on outside variations that the car manufacturer did not foresee beforehand, such as the owner using a steering wheel cover. The camera version is also more computational heavy with updates on average every 2.7 seconds using the same machine learning package as the sensor version.

Looking at object classification, this solution is susceptible to variations in the interior of the car. Specifically the steering wheel which comes in a variety of colors. The training of the object classification network only includes black steering wheels which consequently means that other colors does not work for the detection system. There are several ways of solving this issue. One way is to extend the dataset with all different colored wheels to make sure the detection algorithm learns that not all steering wheels are black. For this solution, it would be important to balance the dataset to not get a bias in the detection system. Another potential solution would be to make different networks for different steering wheels. Although this would be more time consuming, the result could potentially be a more robust system.

6.4 Future Work

For both the camera solution and the sensor solution, there are improvements to be done to further increase the performance the systems. For the camera solution, the most important improvement would be to extend the dataset by a large margin. The dataset would have to include a large variation in people and clothing and it would need to have more variation in ways of holding a steering wheel. This would lead to a more general system which are aware of the the boundaries between states. The detection systems could be more efficient by using a faster library for the neural networks in C++. A limitation with the current library is the ability to only use one CPU-core which slows down execution which is especially seen in the camera solution as it is more computationally extensive. The library also includes some bugs which slows down execution. A change to PyTorch might solve the issue since their C++ library supports all modules for the neural network currently used.

For the sensor solution, the state-space model for the Kalman filter can be extended

to a 3 degree of freedom model and the friction coefficients can be used to model the driver torque even better. A slight difference is seen between a hand holding the steering wheel and a static object, which could be trained on to make the system understand the difference. Further, the confidence algorithm can be tuned to decrease the number of false positives, and reduce the delay going from hands off to hands on.

6.4.1 Combining Solutions

In this project, the sensor solution and the camera solution were evaluated as to separate systems. In the future it would be possible to combine both solutions to make a decision based on the output for both systems. In the ideal case it would reduce the drawbacks from both solutions and the functionality would become more robust. The system would not be misled by hanging a weight on the steering wheel as with the sensor solution, or trying to fool it some other way. However, with the current implementation it would be hard to differentiate between the driver holding the steering wheel on the bottom with one hand or if the driver purposely hangs a weight from the steering wheel. In both situations the sensor predicts hands on while the camera is leaning towards hands off.

If the two systems are to work together, a confidence level could be set which takes the predictions from each of the systems and makes a decision. The different predictions from both systems has to weighted to get the best possible prediction. It would be possible to weight the systems differently for different speeds since the sensor based system performs less well during slower velocities. For higher speeds, if both systems have opposite predictions where the camera predicts hands off and the sensor system predicts hands on, the camera should be weighted higher since there may be an object on the wheel which is applying force.

The main problem with a combination of systems is the cost of implementing both a sensor solution and a camera solution. Only installing a camera in a car would increase production expenses for each car built. While the sensor solution relies on already existing sensors, the cost would instead be of extra computational power to run the system on the car's on-board computer.

7
Conclusion

For detection of hands on or off states, machine learning can be used to determine if the driver is holding the steering wheel. The solutions explored are using a camera or the steering gear sensors as inputs to the machine learning algorithm and are running on Raspberry Pi to emulate the computational power in a car for real-time detection. Both the created sensor system and camera solution can detect hands off situations within the regulations of 15 seconds when running in parallel on the Raspberry Pi.

The sensor based solution provides faster updates at 5 Hz and can be further increased as the computational time on the Raspberry Pi is not the limiting factor. The sensor provides a robust output that catches hands off situations within a couple of seconds, but has a delay of up to two seconds when the driver goes back to holding the steering wheel. It has some short sequences of predicting hands off during hands on driving which can be traced to the confidence algorithm that is made for a 1 Hz update frequency instead of 5 Hz. In the end, the system can be mislead by hanging a weight on the steering wheel which is something to look into for a future version.

The camera solution has an average update time of 2.7 seconds and detects the switch between hands on and hands off within 2 samples. In the online version the camera is more robust when holding the steering wheel with 2 hands compared to holding it with 1 hand. The network manages to classify 1 handed driving better in the offline test because of a different neural network structure with parts that are not available in the online migration package. A switch towards a more dedicated C++ neural network package, that includes all parts could increase both accuracy and speed together with a larger dataset is a future improvement.

Bibliography

Groves, AD (1963). *Area of intersection of an ellipse and a rectangle*. Tech. rep. ARMY BALLISTIC RESEARCH LAB ABERDEEN PROVING GROUND MD.

Nussbaumer, Henri J (1981). "The fast Fourier transform". In: *Fast Fourier Transform and Convolution Algorithms*. Springer, pp. 80–111.

Jacobs, Oliver Louis Robert (1993). *Introduction to control theory*. Oxford Univ. Press.

Hochreiter, Sepp and Jürgen Schmidhuber (Dec. 1997). "Long Short-term Memory". In: *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

Bishop, Gary, Greg Welch, et al. (2001). "An introduction to the kalman filter". In: *Proc of SIGGRAPH, Course* 8.27599-23175, p. 41.

Baronti, F. et al. (2009). "Distributed sensor for steering wheel grip force measurement in driver fatigue detection". In: *2009 Design, Automation Test in Europe Conference Exhibition*, pp. 894–897. DOI: 10.1109/DATE.2009.5090790.

Kumar, P. et al. (2013). "Learning-based approach for online lane change intention prediction". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 797–802. DOI: 10.1109/IVS.2013.6629564.

Garinei, A. and R. Marsili (2014). "Development of a new capacitive matrix for a steering wheel's pressure distribution measurement". In: *International Journal of Industrial Ergonomics* 44.1, pp. 114–119. ISSN: 0169-8141. DOI: https://doi.org/10.1016/j.ergon.2013.11.012. URL: https://www.sciencedirect.com/science/article/pii/S016981411300142X.

Bambach, Sven et al. (Dec. 2015). "Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions". In: *The IEEE International Conference on Computer Vision (ICCV)*.

Das, N., E. Ohn-Bar, and M. M. Trivedi (2015). "On Performance Evaluation of Driver Hand Detection Algorithms: Challenges, Dataset, and Metrics". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 2953–2958. DOI: 10.1109/ITSC.2015.473.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Pulver, Andrew and Siwei Lyu (2016). "LSTM with Working Memory". In: *CoRR* abs/1605.01988. arXiv: 1605.01988. URL: http://arxiv.org/abs/1605.01988.

Abraham, Hillary et al. (2017). "What's in a Name: Vehicle Technology Branding Consumer Expectations for Automation". In: AutomotiveUI '17. Oldenburg, Ger-

many: Association for Computing Machinery, pp. 226–234. ISBN: 9781450351508. DOI: 10.1145/3122986.3123018. URL: https://doi.org/10.1145/3122986.3123018.

Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi (2017). "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

Hwang, Kyuyeon and Wonyong Sung (2017). *Online Sequence Training of Recurrent Neural Networks with Connectionist Temporal Classification*. arXiv: 1511.06841 [cs.LG].

Ide, H. and T. Kurita (2017). "Improvement of learning for CNN with ReLU activation by sparse regularization". In: *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2684–2691. DOI: 10.1109/IJCNN.2017.7966185.

Moreillon, Maxime (2017). "HIGHLY AUTOMATED DRIVING – Detection of the driver's hand on and off the steering wheel for ADAS and autonomous driving". In: *7th International Munich Chassis Symposium 2016*. Ed. by Prof. Dr. Peter E. Pfeffer. Wiesbaden: Springer Fachmedien Wiesbaden, pp. 505–525. ISBN: 978-3-658-14219-3.

Sharma, Sagar and Simone Sharma (2017). "Activation functions in neural networks". In: *Towards Data Science* 6.12, pp. 310–316.

Alom, Md Zahangir et al. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. arXiv: 1803.01164 [cs.CV].

Borghi, G. et al. (2018). "Hands on the wheel: A Dataset for Driver Hand Detection and Tracking". In: *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pp. 564–570. DOI: 10.1109/FG.2018.00090.

Cai, Mengmeng, Manisa Pipattanasomporn, and Saifur Rahman (2019). "Day-ahead building-level load forecasts using deep learning vs. traditional time-series techniques". In: *Applied Energy* 236, pp. 1078–1088. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2018.12.042. URL: https://www.sciencedirect.com/science/article/pii/S0306261918318609.

Dutta, Abhishek and Andrew Zisserman (2019). "The VIA Annotation Software for Images, Audio and Video". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: ACM. ISBN: 978-1-4503-6889-6/19/10. DOI: 10.1145/3343031.3350535. URL: https://doi.org/10.1145/3343031.3350535.

Chugh, T. et al. (2020). "An approach to develop haptic feedback control reference for steering systems using open-loop driving manoeuvres". In: *Vehicle System Dynamics* 58.12, pp. 1953–1976. DOI: 10.1080/00423114.2019.1662923. eprint: https://doi.org/10.1080/00423114.2019.1662923. URL: https://doi.org/10.1080/00423114.2019.1662923.

Floridi, Luciano and Massimo Chiriatti (Dec. 2020). "GPT-3: Its Nature, Scope, Limits, and Consequences". In: *Minds and Machines* 30, pp. 1–14. DOI: 10.1007/s11023-020-09548-1.

Sherstinsky, Alex (2020). "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404, p. 132306. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.

2019.132306. URL: https://www.sciencedirect.com/science/article/pii/S0167278919305974.

Zhang, Xunyu and Shumin Jiang (2021). "Application of Fourier Transform and Butterworth Filter in Signal Denoising". In: *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pp. 1277–1281. DOI: 10.1109/ICSP51882.2021.9408933.

United Nations (2017-11-30). *UN Regulation No. 79*. https://unece.org/fileadmin/DAM/trans/main/wp29/wp29regs/2017/R079r3e.pdf.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY