Controlling a sliding contact on an electric vehicle with computer vision and AI

Axel Sondh

Björn Johnsson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6122
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

Emission from road vehicles is a massive problem and contributes to climate change on our planet. One solution that people are turning to is electrical propulsion instead of fossil fuel. There are, however, problems with putting big batteries on road vehicles. They are expensive to build, they require rare minerals, and the process of creating batteries emits plenty of greenhouse gas.

To reduce the need of big batteries, Elonroad is creating a way of charging road vehicles while driving. This works by putting rails in roads and sliding contacts underneath vehicles. For this to work the sliding contacts and the rail needs to stay aligned while driving.

In this thesis the problem is solved by controlling the sliding contacts position with use of a camera, machine learning and a controller. The proposed structure is to use a pre-trained neural network called *MobileNet* together with a custom neural network to estimate the position of the sliding contact. The estimated position is then used as input to a PID controller that controls the position of the sliding contact with a motor.

Keywords: Machine learning, Computer vision, Electric vehicle, Electric road

Acknowledgements

We would like to thank our supervisor Johan Grönqvist and examiner Kristian Soltesz at the department of Automatic Control at LTH for the valuable help during this thesis. A special thanks goes out to our supervisor Johan, his knowledge in machine learning and control theory helped us greatly. Another thanks goes out to Leif Andersson at the department of Automatic Control for the feedback on writing a report in LaTeX.

We would also like to thank Dan Zethraeus and his coworkers at Elonroad for answering our questions and creating a welcoming atmosphere.

Contents

1. Introduction **10**
- 1.1 Background 10
- 1.2 Problem Description 10

2. Machine Learning Theory **12**
- 2.1 Artificial Neural Networks 12
- 2.2 Training neural networks 14
- 2.3 Convolutional Neural Networks 17
- 2.4 Pre-trained networks 18
- 2.5 Data Augmentation 20
- 2.6 Tensorflow and Keras 20
- 2.7 Control Theory 20

3. Method **22**
- 3.1 Tools and equipment 22
- 3.2 Collecting and processing data 23
- 3.3 Training neural networks 27
- 3.4 Testing neural networks 28

4. Results **31**
- 4.1 Metrics 31
- 4.2 Simulations 34
- 4.3 Real life process 34

5. Discussion and Conclusion **37**
- 5.1 Data 37
- 5.2 Neural network 38
- 5.3 Hardware 41
- 5.4 Future possibilities 42
- 5.5 Conclusion 44

Bibliography **45**

Appendices **47**

A. Histograms **48**

Nomenclature

AI - Artificial Intelligence

ML - Machine Learning

ANN - Artificial Neural Network

CNN - Convolutional Neural Network

API - Application Programming Interface

FPS - Frames per second

IR - Infrared

RGB - Red, Green, Blue

FLOPS - Floating point operations per second

GPU - Graphics Processing Unit

1

Introduction

1.1 Background

Climate change is affecting our world today and starting to cause more and more problems for the people living in it. CO2 emissions from transportation is a big part of the problem and is responsible for a quarter of the total amount of CO2 emissions in the world. 74.5% of the transportation emissions comes from road vehicles [Ritchie, 2020]. Electrifying cars will reduce the CO2 emission by using renewable energy as a source of fuel [SMMT, 2019].

A problem with electric vehicles is the need for a battery. To be able to travel long distances today, a large battery is needed. Batteries require rare-earth elements, that only a few miners in the world have access to. These rare-earth elements also produce toxic waste during extraction which, if not stored correctly, can have a negative effect of the surrounding environment. The weight of batteries causes strain on the tires of the vehicle, which possibly generate more particle emissions. In addition, the production of batteries also generates emission [Wikipedia, 2020c] [Maughan, 2015]. Elonroad is developing a system that allows vehicles to charge while driving using a electric rail installed in the road. By doing this the big batteries could be swapped with smaller batteries, while maintaining, or even extending the available range. [Elonroad, 2020].

The rail in the road is divided into 1-meter-long sections. By splitting the rail into smaller sections different potential can my applied to each one. Under the vehicles, three sliding contacts are installed with approximately 80cm between them. One of the contacts will in this way always be connected to ground and one with the positive potential, thus closing the circuit.

1.2 Problem Description

To make conductive charging possible, the vehicle being charged needs to have contact with an electric rail on the road. Since the car might be moving at high speeds, it is a challenge to make sure the sliding contacts under the car remains in contact with the electric rail on the road. To solve this problem the sliding contacts have been mounted on a belt that can be moved sideways by a motor. The problem that remains is for a controller to know how the sliding contact should be moved to make the distance between the electric rail and the sliding contacts as small as possible.

This thesis aims to solve the problem with the help of a camera underneath the car. The images captured by the camera are going to be analyzed by an AI, estimating the distance between the electric rail and the sliding contacts. This distance is then to be used as an input to a controller controlling the motor. To make this study feasible, considering time and scope, the following limitations are put into place.

- Elonroad want to examine the use of a camera to solve the problem, because of this, no other options will be investigated.
- The dataset collected for the training and evaluation of the network is only collected in a laboratory environment. Therefore the impact of many factors, such as the weather, will be limited.
- Some investigation will be done with different lightning, but mostly the same light will be used.
- The project will aim to evaluate the concept. Therefore, the hardware limitations and the ability to deploy the solution will not be considered.
- The test rig used in the project has only one sliding contact, since they would all be controlled by the same motor this isn't considered a significant difference.

2

Machine Learning Theory

The concept of artificial neural networks is not in any way a new one. The underlying concept was presented already in 1943. However, the lack of computational power at the time meant that the concept would not become significant for many years to come. [Wikipedia, 2020a]

The evolution of computers has now led us to a place where it is possible to use neural networks in a reasonable way. Today's supercomputers are able to do 100 billion times the number of FLOPS (floating point operations per second) than the supercomputers in 1956 [Routley, 2017]. Because of this, artificial neural networks have become a hot topic in the world of computer science, and beyond.

One can often hear words like "Artificial Intelligence (AI)", "Machine Learning (ML)", "Deep learning" and "Neural Networks". In figure 2.1 the relationship between these is visualised. The term Artificial Intelligence refers to the entire field and includes everything that mimics humans. This can be anything from a code using some if statements to a neural network.

Machine Learning is more specific, it refers to machines being able to learn by training itself to solve a problem. This is included in the term artificial intelligence and can be done using, for an example, neural networks. Neural networks are a tool that is often used when practicing Deep Learning. A way for a machine to learn using a layered structure of neurons, mimicking the human brain [Bini, 2018] [Zhang et al., 2017].

2.1 Artificial Neural Networks

An artificial neural network (ANN) is constructed from three different types of layers: Input layer, hidden layer and output layer, see figure 2.2. The input layer receives the initial data for the network, one or several hidden layers is where all the computation is done and the output layer produces the result for the given inputs [Albawi et al., 2017]. Each layer contains a specific number of neurons, which are represented as circles in figure 2.2. A neuron is a node which holds a number, this number is called the neurons "activation" [Nielsen, 2015].

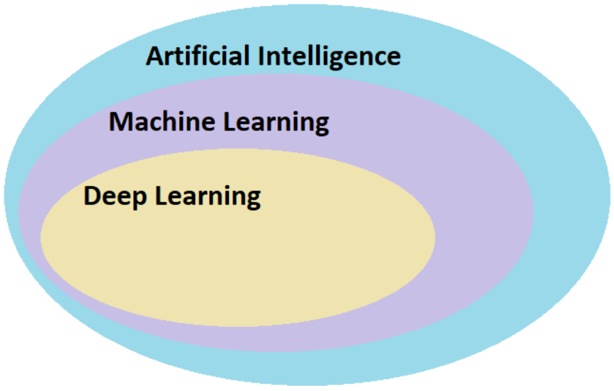

Figure 2.1 The relation between concepts often talked about in this area of work.

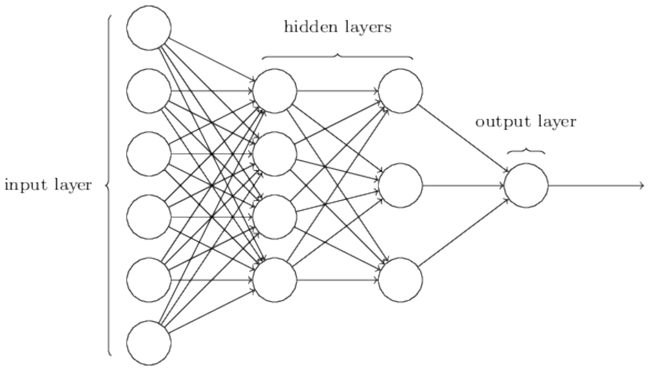

Figure 2.2 The structure of a neural network with an input layer, two hidden layers and an output layer. Each arrow represents a connection with its weight and each circle represents a neuron with an activation. Figure from [Nielsen, 2015].

If the input data is an image the number of neurons in the input layer corresponds to the number of pixels in the image. In a grayscale image the number inside the neuron represents the light intensity. This number is often normalized from 0 to 1 where 0 represents black and 1 represents white. If the image is a RGB-image (red, green, blue) each pixel has three neurons. One neuron for each color, representing the intensity of that particular color. For example, a 180x180 image have 32400 (180×180) neurons in the input layer for a grayscale image but 97200 ($3\times180\times180$) neurons for a RGB-image [Nielsen, 2015]

Between each layer there is a bunch of connections, represented in figure 2.2

as arrows. In a fully connected neural network each neuron is connected to every neuron in the previous layer. Each connection in the neural network has a weight w_{nk} assigned to them. This weight is specific to each connection. Weights are assigned so that the activation in specific neurons have more influence over the activation in specific neurons in the next layer [Nielsen, 2015].

The way a neural network operates is that the activation in all neurons in one layer determines, with the help of the weights, the activation in each neuron in the next layer. When an image is sent through the network the activation of the input neurons is multiplied with each connection weight to determine the activation in the connected neurons. Observe that the weight is associated to the connection between two specific neurons, meaning that each neuron in the next layer will have a different connection to each neuron in the previous layer. This is presented mathematically in equation 2.1, with A_1 being the activation in the first neuron in the next layer, a_n activation in previous neuron and w_{n1} the weight for that connection.

$$A_1 = a_1w_{11} + a_2w_{21} + a_3w_{31} + ... + a_nw_{n1} \tag{2.1}$$

The sum of the weighted activation from the previous layer might result in a value outside of the normalized range of 0 to 1. An activation between 0 and 1 is usually preferred to avoid uncontrolled growth of the values, making the problem less robust. A solution to this is to squish the numbers down to the range 0 to 1 with the help of normalization or an activation function $\sigma()$, see equation 2.2 [Nielsen, 2015]. An example of an activation function is shown in figure 2.3.

$$A_1 = \sigma(a_1w_{11} + a_2w_{21} + a_3w_{31} + ... + a_nw_{n1}) \tag{2.2}$$

In addition to keeping the size of the values in control, activation functions introduces a non-linearity to the systems. These non-linearities make the system more complex and able to solve problems that can´t be solved using only linear combinations [MissingLink.ai, 2020].

2.2 Training neural networks

The following theory is describing the methods and networks used in this specific project. Because of this, some of the theory might not be true for general training of neural networks.

When initializing the neural network for the first time, the weights are assigned randomly. Training a neural network is the same as adjusting the weights so that the output layer produces the result that is desired [Nielsen, 2015].

In order to train the neural network, samples of known data are sent through the network. This data is called training data. The network makes a prediction for each sample and compares this to the known correct answer. The comparison between the known answer and the prediction is done using a loss function. This function

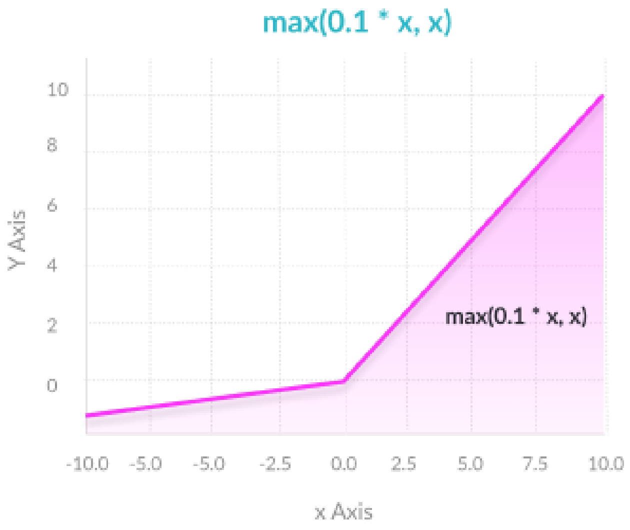

Figure 2.3 LeakyRelu activation function. Figure from [MissingLink.ai, 2020].

determines how far from the correct value the prediction is. If the value (loss) of the loss function is low the network is making good predictions, but if the loss is high, the network makes a bad prediction [Nielsen, 2015].

To improve the network's predictions, lowering the loss, the weights for all connections need to be adjusted. The network does this with the help of backpropagation [Nielsen, 2015]. Backpropagation, or "backward propagation of errors", calculates the gradient of the loss function with respect to the weights. The gradient tells us how fast and in what direction the loss changes when the weights are modified [HECHT-NIELSEN, 1992].

Common issues when training neural networks

The objective of training a neural network is to generate a model that performs well on the training dataset and new data which it has never seen before. The model should learn to generalize from the training data and apply it on new data. Too much learning could cause the network to become overfitted for the problem, making it particularly good at predicting on the training dataset but poor at predicting on new data. The reason for this could be that the training dataset is not big or varied enough, making it impossible for the network to set good generalized weights. [Roelofs et al., 2019].

To identify if a network is becoming overfitted one could monitor the training process by evaluating the training dataset and validation dataset continuously. The

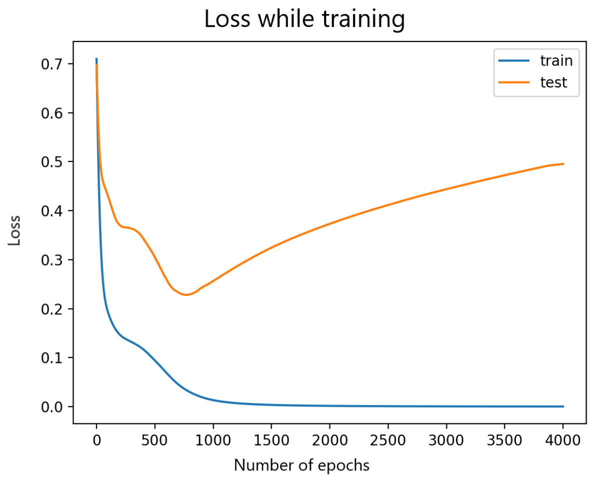

Figure 2.4 A neural network overfitting, where test is the loss of the validation dataset, figure from [Brownlee, 2018].

validation dataset is a dataset which the network has never been trained on. Plotting the performance curve of both these datasets during training show a certain pattern when overfitting. If the network is getting overfitted the validation loss will increase after an amount of training. This can be observed in fig 2.4. A lower loss represents better performance [Roelofs et al., 2019].

There are many ways to decrease overfitting, one way is to use dropout layers. A dropout layer randomly sets some weights to 0, if dropout is set to 0.2 then 20% of the weights connected to the dropout layer will be deactivated. This means that training will not be done on all weights for every sample of data. Another way to decrease overfitting is to increase the amount of data that network is training on and to make the data more diverse.

When using images as inputs, adding layers to the network might help identify more complex patterns in the images. This makes the network deeper. As an example, the first layer might learn to recognize edges, the second triangles and the third even more complex patterns and so on. For a fully connected network, as the example in figure 2.2, detecting edges and other spatial structures in images is not effective and requires many computational operations [Nielsen, 2015]. In chapter 2.3 a more effective way of doing this is explained.

In deeper networks it turns out that the gradient tends to vanish in the early layers, the vanishing gradient problem. The gradient, as is mentioned above in chapter 2.2, is backpropagating through the deep network causing multiple multiplications which in turn can make the gradient infinitely small. This means that for a very deep network the gradient seems to disappear [Nielsen, 2015]. The solution to this

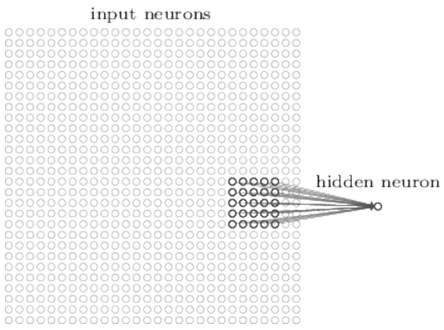

Figure 2.5 In a CNN a small window of neurons on the input image is connected to a single neuron in the next layer, figure from [Nielsen, 2015]

problem can be quite complex, in section 2.4 the pre-trained networks each have their own method to solve this.

2.3 Convolutional Neural Networks

When the input to a neural network is a two-dimensional image, a fully connected neural network might not be the best design. This is because the input layer in such a network will flatten the input image to a one-dimensional vector, removing the original structure of the image. In a convolutional neural network (CNN), the original structure is preserved. This allows the CNN to identify spatial patterns such as edges, shapes and objects more efficiently than a fully connected neural network [Nielsen, 2015].

In a CNN it's helpful to think of each input as a matrix of neurons. Instead of connecting every input pixel to every neuron in the next layer a small window of neurons is connected to one neuron in the next layer, visualized in figure 2.5 [Nielsen, 2015].

The small window is called a kernel and acts as a filter. The kernel is always smaller than the size of the input and produces a dot product of the neurons in the window, which then are added to produce a single value for each patch of the input. The output from the kernel, when applied to the input image, results in a map of values, also known as a feature map. The kernel moves across the image, and depending on how large steps it takes, might overlap a part of the previous windows neurons. [Albawi et al., 2017] This process is illustrated in figure 2.6.

It is common that a convolutional layer has several kernels in parallel for any given input. Each kernel has a value for every input, as seen in fig 2.7. The values for the different kernels are all initialized with a function generating random values. This makes it possible to extract many different features in the same layer. As an example, 32 kernels give 32 different ways of extracting features from an input

[Albawi et al., 2017].

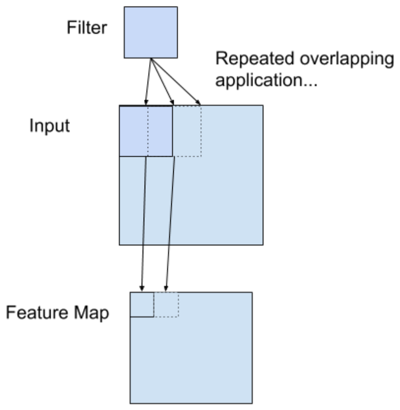

Figure 2.6 A convolutional layer acting as a filter on the input, creating a feature map. Figure from [Albawi et al., 2017].

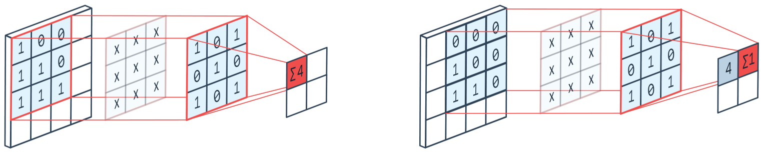

Figure 2.7 An illustration of how a kernel moves over an image generating a feature-map. Figure from [Peltarion, 2020].

2.4 Pre-trained networks

A pre-trained network is a model that has been trained on a large dataset already. The pre-trained network can either be used as is or be adapted to solve other problems than it's initially trained for, this concept is called transfer learning. To solve the problem in this thesis, transfer learning is used by adding a small network to the pre-trained network and training it to solve the project specific problem [Chollet, 2020].

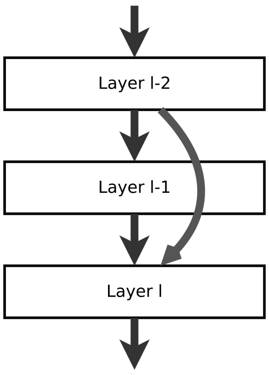

Figure 2.8 Layers in a residual network, illustrating a skip connection. Figure from [Wikipedia, 2020d].

ResNet

A trend in the research community is to go deeper with networks (more layers). However, building a deeper network can cause a problem with the gradient vanishing, as described above in chapter 2.2. The gradient is backpropagating through the deep network causing multiple multiplications which in turn can make the gradient infinitely small. This causes the performance to saturate [George Philipp, 2018].

ResNet is a pre-trained network that aims to keep the depth of the network and at the same time remove the vanishing gradient problem. In order to do this ResNet uses something called "residual blocks" [George Philipp, 2018]. Instead of having each layer only connected to the next layer, residual blocks also have a connection to the layer 2-3 layers away, see figure 2.8. This means that it is possible to skip training a few layers. The residual connection is therefore also called a skip connection. Because of these skip connections it is possible to propagate larger gradients to the first layers, giving the network the ability to train the first layers as quickly as the last layers. In practise this mean that for each data sample, different parts of the network will be trained. In a way, this allows the network to choose the optimal number of layers during training, making it more dynamics. [Sahoo, 2018]

MobileNet

MobileNet is a neural network that is specifically designed for mobile and resource constrained hardware. The goal of MobileNet is to decrease the number of operations and amount of memory significantly, while keeping a high accuracy. MobileNet also utilize the concept of skip connections as described above in ResNet [Sandler et al., 2019].

2.5 Data Augmentation

Adding new training data for an ANN will often improve the performance, for example decreasing overfitting and increase robustness. In a CNN, or another network where the data consists of images, image data augmentation is used to create transformed versions of the images. In a classification problem, for example object detection, the image can be rotated, mirrored, zoomed, etc. A CNN can with this method learn features that are invariant to the location in the image.

When augmenting data, it is important not to transform the images to such an extent that it no longer can be associated to the correct class. For a regression problem, this is especially important. If the purpose is to find the position or angle in an image it is important not to transform the image so that the position of the object is moved. Other augmentation could instead be used as for example, lightning, color, noise, etc. [Perez and Wang, 2017]

2.6 Tensorflow and Keras

In order to work with machine learning it is common to use existing frameworks that help the user with low level operations. One of the more popular solutions is to use Keras together with Tensorflow as backend. Keras provides high-level APIs allowing the user to focus on high-level tasks and getting fast results. Keras itself does not contain the computational backend required to train and run networks. Because of this Keras has to be used together with for example Tensorflow. One advantage of Tensorflow is the flexibility that comes with the frameworks. Tensorflow offers excellent support when deploying trained networks to different hardware. This makes it possible to train a network on a windows computer and transferring it to a Raspberry Pi running Linux based environment without too much of a hustle [Tensorflow, 2019].

2.7 Control Theory

The purpose of control theory is to control systems in an optimized way with good robustness. Control theory has been used for over a hundred years and is still a big field both for research and industry. A controller usually has a task, as an example: to control the temperature in a house. To do this the controller typically has one or several inputs, in this particular case, probably the current temperature, measured by an accurate temperature sensor [Wikipedia, 2020b].

There are different ways of creating a controller, one of the more common structures is PID (proportional, integral and derivative) controller. Equation 2.3 shows the equation for a PID controller. Ki, Kd and Kp are the gain for integral, derivative and proportional parts. I is the integrated error over time. D is the derivative of the error, E is the current error. When a PID controller is implemented, a low pass filter is usually added to avoid undesirable behaviour at high frequencies. [Kristiansson and Lennartson, 2006]

$$\text{Control signal} = Ki \cdot I + Kd \cdot D + Kp \cdot E \tag{2.3}$$

When using control theory together with an AI, a new component with an uncertainty is introduced to the system. In this project the AIs prediction is used as an input to the controller, raising the question whether the prediction is accurate enough. There are ways of estimating the certainty of the AI and as a consequence one could adjust the controller based on this information. [Chryssolouris et al., 1996].

3

Method

Developing an AI is usually an iterative process. The following is a brief walk-through of the different steps, followed by a more thorough explanation of each step. The input to the networks was an image. As output the networks produced a single number representing the distance between the sliding contact and the electric rail on the road.

1. Images were collected with corresponding distance measurements. The data was then processed in a way that can be interpreted by the program that is used for training neural networks. An augmentation of the images was also made.
2. A neural network was constructed with a sequence of layers. When the network was created, the data collected in the previous step was used to train the network.
3. The trained network was evaluated for its purpose to find out if its performance was sufficient for the task.

If the conclusion in the last step was that the network was unsatisfactory at solving its task, some, or all, steps had to be considered for change.

3.1 Tools and equipment

The following is a list of the tools and equipment used in this project.

- Raspberry Pi 4 Model B
- Arduino Uno
- Raspberry Pi Camera Module V2
- Wide lens for Raspberry Pi Camera

- 16mm lens mount for Raspberry Pi Camera V2 [Solarbotics, 2020]
- 74 metal pieces
- 1 54kΩ resistor
- 73 1kΩ resistors
- Computer with the following installed: Python 3, Tensorflow, Keras, Numpy, Pillow, Imgaug.

3.2 Collecting and processing data

Measuring distance to associate to each image

The objective was to create a dataset in which images were associated with a value representing a distance in centimeters. To automate the process of collecting and labeling images a device was created that could measure the distance that was going to be linked to the image.

In figure 3.1 the distance-measuring device is shown. It consists of 73 metal pieces that are approximately 5mm wide. Between each metal piece a resistance of $1k\Omega$ was soldered. Connected to the device was an Arduino.

A measurement was made by applying a voltage of 5V from the Arduino over both the reference resistance, with a known resistance of $56k\Omega$, and an unknown number of $1k\Omega$ resistances, representing a distance. This was made while both the ground-plate and the metal pieces were dragged along the rail. As can be seen in figure 3.2

A closed loop circuit was in this way formed using the rail itself to connect the first pin (counted from the reference resistance) in contact with the rail and the ground-plate. The resulting circuit is shown in figure 3.3 From measuring the voltage denoted as V the resistance was calculated using ohm's law in equation 3.1:

$$V = \frac{R_{measurement}}{R_{measurement} + R_{reference}} \cdot 5V \tag{3.1}$$

The different metal pieces in figure 3.1 where placed in a way so that the distance from the center of one piece was 6mm from the next one's center. From this it was possible to relate a voltage measurement to a distance. This way of measuring distances produces discrete values (with 6mm between each) since the measuring device knows only which pin is short circuited.

Collecting the data

Two different ways of collecting images were used in the project. At first a phone camera was used that streamed its visual feed using the app *IP Webcam* [Khlebovich, 2020] to a computer, figure 3.4. The computer was also connected to the Arduino through a USB-cable to get the distance measurements.

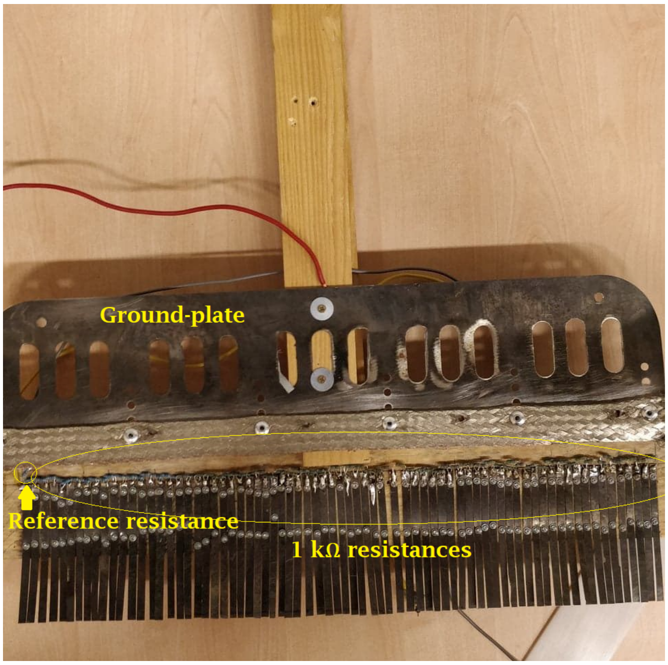

Figure 3.1 The constructed measuring setup, with flexible 5mm metal pieces connected in series with $1k\Omega$ resistance. The ground-plate is also shown in the image. Note that the ground-plate and the metal pieces are attached to different wooden planks.

The second, and final, way of collecting images was using a Raspberry Pi instead of the computer, figure 3.5. In this setup the Raspberry Pi was connected to a Raspberry Pi Camera Module v2 with a wide lens and the Arduino. Both the images and the labels were saved on the Raspberry Pi.

In both cases, the rig where the camera and distance-measuring device were mounted was moved along the electric rail. A python script was used that saved each image in a folder that had a name representing the distance measured. A brief example of how one sequence of images was gathered is presented below:

1. An image is captured by either a phone or a raspberry camera.
2. The Arduino measures distances continuously and communicates with the python script on demand.
3. Depending on the value received from the Arduino the destination folder of the image is decided. E.g. if the value from the Arduino is "3.6" the image

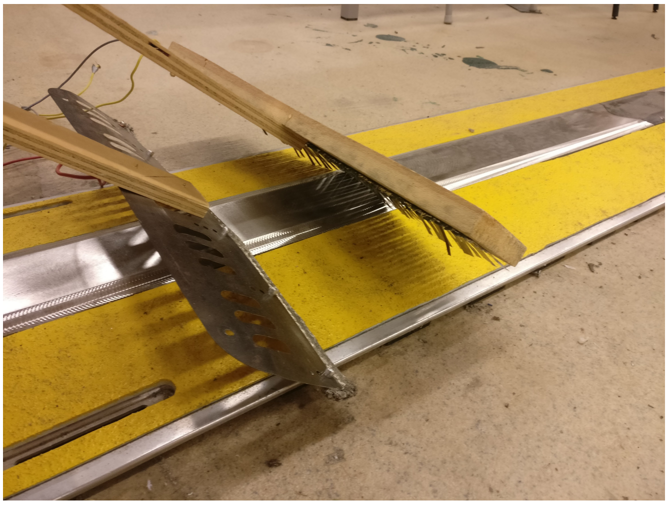

Figure 3.2 The constructed measuring setup collecting distance measurements.

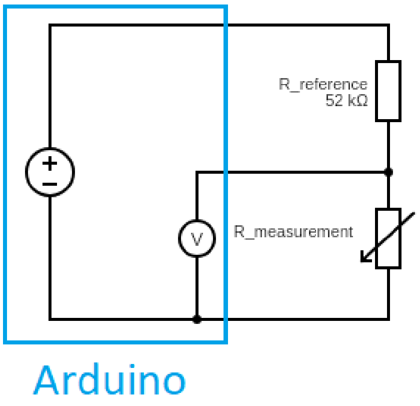

Figure 3.3 Circuit schematics showing how the distance measurements were made. R_reference is a resistor with a fixed size of $56k\Omega$. R_measurement is an unknown number of $1k\Omega$ resistors

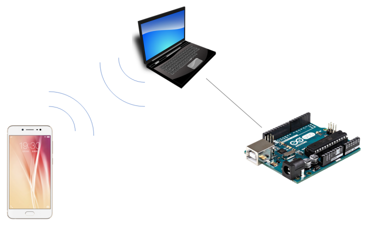

Figure 3.4 The first setup used to collect images and auto-labeling them with the Arduino. Images taken from [Freno, 2020] [Supup, 2020] [Yineri, 2020].

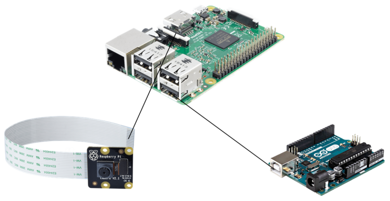

Figure 3.5 The second setup used to collect images and auto-labeling them with the Arduino. Images taken from [Naoppay, 2020] [Supup, 2020].

is saved in the folder "3.6", together with all other images with the same measurements.

4. This was repeated until the desired number of images were collected, usually 300-1000 images. The collection speed was 3-10 images/second.

To make the images more realistic compared to real situations, LEDs were used to shine at the electric rail while the images were collected, to simulate different light settings.

Data augmentation

When all of the images were collected, data augmentation was applied to the images. For this a library called *Imgaug* was used. The following augmentations were applied to all the images.

- **TranslatateY** that moves the image in the Y axis a random amount between -50 and 50 pixels
- **GaussianBlur** that adds blurriness to the image

- **MotionBlur** that adds the effect of movement to the image
- **AddToBrightness** that adds or removes to the brightness of the image
- **Rotate** that rotates the image a random amount between -5 and 5 degrees
- **Spatter** that adds spatter to the image
- **PerspectiveTransform** that distorts the proportions of the image

Preparing images for training

In this project the pre-trained networks MobileNet and ResNet, described in section 2.4 were used. When using MobileNet and ResNet, the images were processed in a way so that every pixel was represented with three values ranging from 0 to 1, when ResNet was used, -1 to 1 when MobileNet was used. Each of the values represented the presence of the colors red, green, blue (RGB).

When training the ConvNet, grayscale images were used. In that case each pixel in an image, was represented by only one value ranging from 0 to 1, representing the light intensity in the image. This way, the images were prepared in a way so that Keras together with Tensorflow could be used to train the networks.

3.3 Training neural networks

The networks were trained using Keras with Tensorflow as its back end. Two main approaches were used to create the networks. The first one was to design the networks, adding all layers, and to train them from scratch ourselves. The other one was to use pre-trained networks with a couple of extra layers added on-top, as described in section 2.4.

During this project, a lot of different networks were tried. Some of them discarded immediately, others were developed further and ended up as parts in the final networks and some of them were used as they were.

When the project was first started, the approach to use networks both created and trained by ourselves, was used. Different numbers of convolutional layers, ranging from 1 to 48, with different kernel sizes were evaluated. The last layer in all models was a fully connected single neuron because the model is solving a regression problem. Between the convolution layers and the output layer, one to three fully connected layers were added, each with 1 to 1024 neurons. In addition, three different activation functions were evaluated in between all the layers. The functions being; Relu, LeakyRelu and linear. When the networks were trained a cost-function had to be used. In this project *mean square error* was chosen, resulting in a smaller penalty for small errors while punishing larger errors more.

A version of these networks with an additional classifier was also tried. This worked by having both a classifier and a regression network one after the other.

First the classifier estimated if the image was taken far out to the right, a bit to the right, in the middle, a bit to the left or far out to the left. The second part of the network consisted of 5 networks each trained only on images from one of the classes just described. All these networks had the same structure as the convolution networks described in the previous paragraph, except that the classifier had 5 output neurons, one for each class.

After a while, pre-trained networks were introduced in the project. In these cases, either MobileNetV2 or ResNet50V2 were used as a base. On top of these, one to three fully connected layers were added, each containing between 1 and 1024 nodes. When training networks, there are several parameters that must be decided.

- Number of epochs - how many times the network is trained on the training data.
- Batch size - how many images that are presented to the network before updating the weights.
- Learning rate - how big the changes of the weights are after each batch.

These parameters were decided by testing different values, as well by hardware limitations. When the networks were trained, dropout layers were placed between every group of layers. All of this resulted in three networks that were considered the best:

- ConvNet - A network designed and trained ourselves. This network had the following structure: 15 convolution layers - maxpool - 4 convolution layers - maxpool - 3 convolution layers - maxpool - a layer with 32 dense nodes and finally the output node. LeakyRelu was used as activation between all the groups of layers. Included in this is also the version with a classifier.
- ResNet50V2 - This network used ResNet50V2 as a base. On top of this base one layer with 32 nodes was added and finally the output node. No activation functions were manually added.
- MobileNetV2 - A network based on MobileNetV2 with one layer with 32 nodes and the output node added on top. No activation functions were added.

For a more detailed explanation of how to train networks using Keras and the different methods used, we refer to [Keras, 2020b].

3.4 Testing neural networks

Testing and evaluation of the networks were made with scrips that loaded a trained network and ran tests on it, generating various plots and metrics. All the scripts described in the section worked by loading a test dataset of images and corresponding

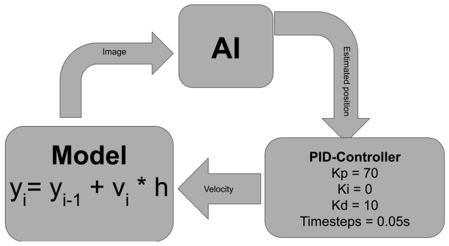

Figure 3.6 The structure of the simulation. The AI was the neural network. The controller was a PID-controller created using python. The model was a simplification of how the real process work, with the same inputs and outputs as the real process.

true values to each image. All the images were then evaluated by the network. When done, the scripts had the predicted and the expected values for each of the images. The following metrics of performance were calculated and presented by the scripts.

- Accuracy was calculated by taking the difference between true and predicted value (error). Both the accuracy for a maximum error of 1 and 2 cm was calculated.
- Plot with all the true and all the predicted values.
- A histogram showing the accuracy of all the different areas of measurement (e.g 0, 1.6, -12.6).

In addition, some more advanced tests were conducted to measure performance on the actual problem, see the following two sections below.

Model of the process

To get a feeling of how the network would perform on its actual task, a simulation environment was created.

The model in figure 3.6 worked by keeping track of a position. As input the model got a velocity using this input the position of the model was updated by multiplying the velocity with a chosen time step and adding the product to the past position. The time step was set to a value considered reasonable, in this case 50ms. An image was then chosen at random from the folder containing images with the smallest difference between the models simulated position and the true values of the images in the folder. In this way the AI could be fed with images in a similar way as the real process with a camera would.

The image was then evaluated by the neural network (denoted AI in figure 3.6), resulting in an estimation of the position. To close the loop, the estimated position was used as an input to the controller. Using this, a desired velocity was calculated and sent to the model.

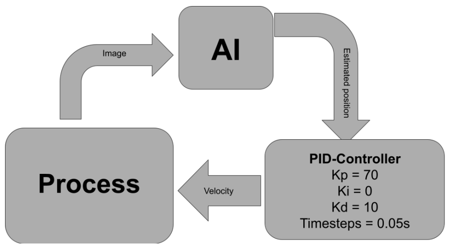

Figure 3.7 The AI was the neural network. The controller was a PID-controller created using python. The process was the motor moving the sliding contact and the camera that captured the images

Test on real process

Lastly, the neural networks were tested on the real setup. These tests were conducted in two ways. In the first one the neural network evaluated images from the camera in real time while the camera was moved sideways over the electric rail by hand. During this a graph was plotted to show the predictions made by the neural network. Since the tests were made in real time, only one network could be evaluated at once. Because of this the experiment had to be repeated three times, one for each of the networks. Since the camera was moved by hand, the movement pattern could not be replicated, hence the inputs to the networks were not the same.

The second test worked by testing the neural networks on the real process, almost in the same way as the simulation. The difference being that instead of a simulation of the process the real one was used. Taking a velocity from the controller and feeding it to the motor, moving the camera sideways. The camera delivered a continuous stream of images to the AI which in turn kept on giving position estimates to the controller. A PID-controller was implemented using python.

4

Results

A lot of different networks were tested to get an understanding of the problem. In figure 4.1 the loss-epoch graph of four different pre-trained is shown. This was done to evaluate which pre-trained networks to proceed with. In this chapter the 3 key networks, described in chapter 3.3, are evaluated.

1. ConvNet - A network designed from scratch ourselves, using 22 convolution layers and 2 fully connected layers.
2. ResNet - A network that uses the pre-trained network ResNet50V2 as its base.
3. MobileNet - A network that uses the pre-trained network MobileNetV2 as its base.

4.1 Metrics

After each network is trained the accuracy is calculated. This accuracy is presented in table 4.1. From this we can tell what percentage of the model's predictions that has an error of 1 cm or less and 2 cm or less.

Plots showing the true and predicted values for all test dataset images for the three models are presented in figure 4.2, 4.3, 4.4. There are also histograms showing the accuracy for all the different groups of images i.e the accuracy of images with true values of "12.6", "0.6", "-13.2" and so on, in appendix A figure A.1, A.3, A.2.

	Accuracy (1 cm)	Accuracy (2 cm)
ConvNet	21%	41%
ConvNet with classifier	89%	98%
ResNet50V2	65%	88%
MobileNetV2	59%	85%

Table 4.1 A table showing the accuracy of the different models with an error of 1 and 2 centimeters or less.

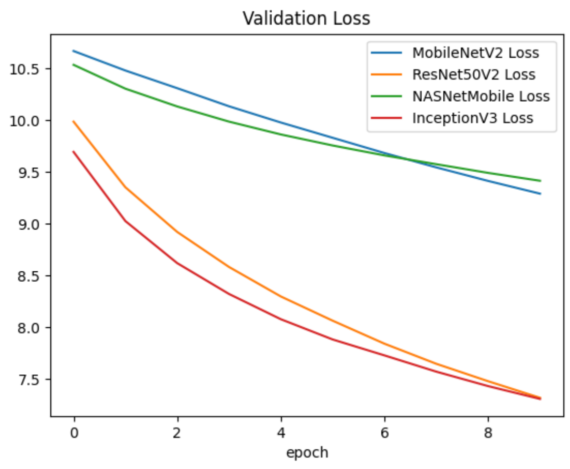

Figure 4.1 MobileNetV2, ResNet50V2, NASNetMobile and IneptionV3 validation loss compared for 10 epochs. The same training dataset and validation dataset was used for all models.

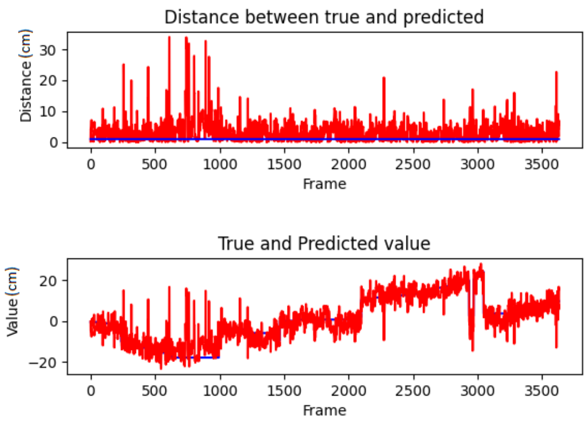

Figure 4.2 This is a plot of the result using ConvNet. The red graph in the top plot shows the distance in *cm* between the predicted value and true value while the blue line is plotting the margin of 1*cm*. The red graph in the bottom plot shows the predicted value and the blue graph show the true value, both in *cm*.

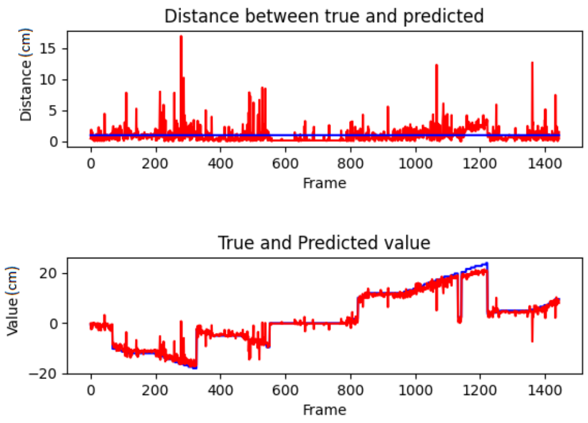

Figure 4.3 This is a plot of the result using ResNet. The red graph in the top plot shows the distance in *cm* between the predicted value and true value while the blue line is plotting the margin of 1*cm*. The red graph in the bottom plot shows the predicted value and the blue graph show the true value, both in *cm*.

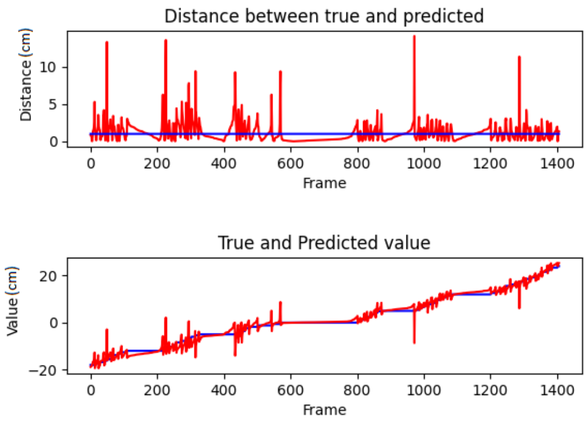

Figure 4.4 This is a plot of the result using MobileNet. The red graph in the top plot shows the distance in *cm* between the predicted value and true value while the blue line is plotting the margin of 1*cm*. The red graph in the bottom plot shows the predicted value and the blue graph show the true value, both in *cm*.

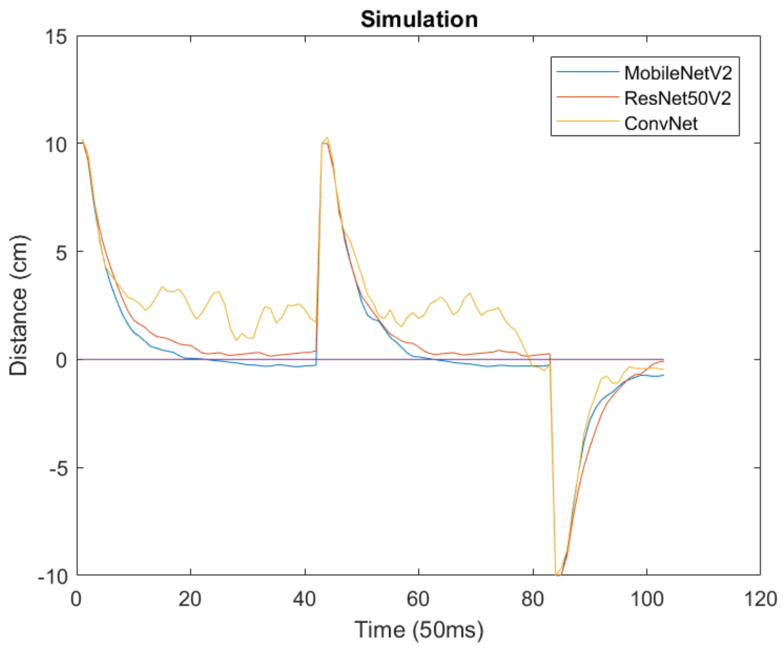

Figure 4.5 Plot of simulated performance of MobileNet, ResNet and ConvNet. The x-axis shows the number of iterations, each iteration represent $50ms$.

4.2 Simulations

Each of the three networks is tested in a simulated environment, as described in chapter 3.4. Figure 4.5 shows the response of all the networks in the same plot to compare performance between them. On the x-axis, the time is presented in steps of 50ms. The y-axis shows the distance from the desired position 0, representing the rail in the road, and the model's position. At time 0 the model starts at +10 cm, the AI and the controller tries to bring the model's position to 0. This is repeated at time 40 (2.5 seconds). At time 85 (4.25 seconds) the model is set to -10 cm. These are simulations of disturbances, one can think of it as someone giving the setup with the camera a push, placing it at the wrong position. Which forces the AI and controller to try to find their way back.

4.3 Real life process

Two tests were made using the real life process with the methods described in chapter 3.4. In figure 4.6 the graph generated when the camera was moved back and forth is shown.

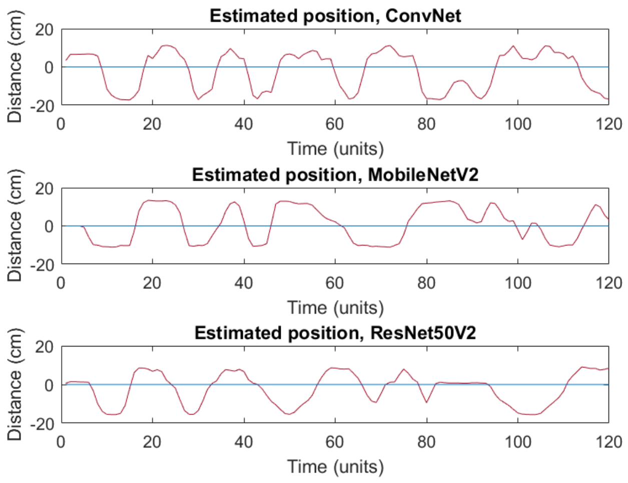

Figure 4.6 Plots generated from all of the networks by moving the setup with the camera sideways. The networks do not receive the same data since the data was generated from three different experiments.

The second test was done with the controller and motor connected, the result from this is presented in figure 4.7. When starting the motor, it first does a calibration, which can be seen in the plots as the dip to -18cm.

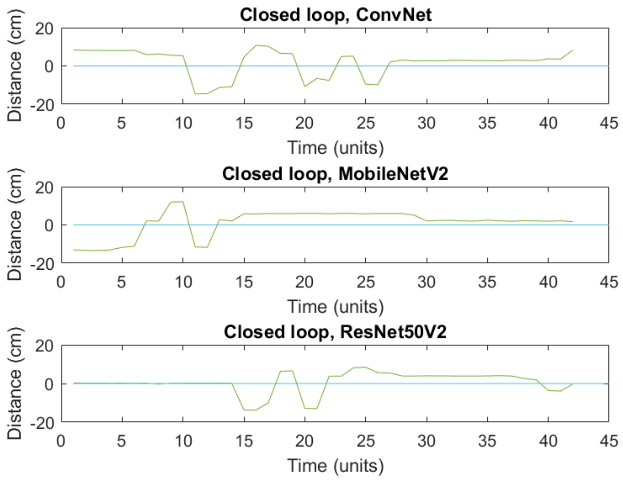

Figure 4.7 The closed loop response from all of the networks. Note that the graphs are generated from three different experiments.

5 Discussion and Conclusion

5.1 Data

The data collection, organization and processing was a big part of the project.

Two different ways of collecting images were used, as described in chapter 3.2. At first a phone was used and later in the project a Raspberry Pi with a camera. The advantage of using the phone at the beginning was to get the project going quickly. By creating the first dataset using the phone, the first networks could be created within the first weeks. It was also a good way of finding out what the requirements for the camera would be. Which was important since it was always known that the phone would be exchanged for a better camera solution later. By doing this we learned, for instance, that a wide-angle lens would be required to see the electric rail when the camera was far out on the side. We also realised that camera resolution was less important than frame rate. Once these requirements were known, the search for a suitable camera could start. After thorough research, the Raspberry Pi Camera Module V2 came out as the best choice since it met all the requirements, speed, resolution, field of view, and was cheap.

A way of automatically labeling images had to be created, since labeling every image with a distance by hand was considered unreasonable. The solution in this project, chapter 3.2, worked okay but had some drawbacks. Some examples are: If two of the metal pieces short circuited by touching, the voltage measured would not be correct. Also, the device could probably not be used at high speed since it is too fragile. Overall, the performance of the measuring device was considered good enough for the purpose of this thesis and it was therefore a good solution. If the product idea was to be developed further another solution would probably have to be used for creating bigger datasets. One way could be to automatically label the images with the AI together with some manual corrections. This could be an effective way to create big datasets fast.

When the first networks were trained and connected to the camera it became obvious that the networks were not robust enough. If the angle of the camera was changed, either up/down or right/left, the AI had a very hard time. This implied that the AI was trained on a to homogeneous dataset and therefore was sensitive to disturbances. The reason this was not discovered in the previous testing was that the test dataset also was homogeneous. To solve this problem more diversity was introduced to the dataset in two ways. One way was to change the angles of the camera as new images were collected. Another way was to generate augmented copies of images using image processing techniques, see chapter 3.2. Augmentation can simulate a lot of conditions and thereby create a more diverse dataset. LEDs were also used to create both simulations of headlights and sun in the images.

5.2 Neural network

Three neural networks were designed and tested in this project, ConvNet, MobileNet and ResNet. Therefore the discussion will focus on these three networks. These three networks are explained in chapter 3.3

Neural networks - Method

All the neural networks have input layers that have the shape of an image and output layers with only one neuron containing a single number. This number represent where in the image the electric rail is located. If the number is close to 0 then it means that the electric rail is in the center of the image.

The input image has different shapes, depending on what network it is fed to. For the ConvNet an image shape of $320 \times 180 \times 1$ is used, the last number representing grayscale. For the pretrained networks an image shape of $224 \times 224 \times 3$ was used, where the last number represents RGB. The main reason behind the shapes is the limiting computing capacity. To be able to train networks with high resolution images in a reasonable time, more computing power is required. Since we did not have access to this computing capacity, we did not attempt higher resolution. For future work it might be worth exploring if a higher resolution produces a better result.

To get the largest field of view possible from the Raspberry Pi Camera, images were captured in 640×360, according to the documentation for the camera. Since none of the networks used images this large, they were reshaped to 320×180, for the ConvNet. Grayscale was used since it reduces the number of trainable parameters by two thirds, reducing the number of computing operations. However, when processing the images for the pre-trained networks a different image shape was used, simply because the networks were designed for that specific input shape.

The problem that has to be solved is a regression problem. Because of this a single output neuron is used as output. It is important to note that we cannot compare this number directly to the known answer when evaluating performance. This is because the network outputs a number, as for example $10.54cm$, but the known answer might be $10.6cm$. If we only compare these numbers, we will have almost zero accuracy for the model. Therefore, a margin is added. Since the electric rail is about 10*cm* wide and the sliding contact about 4*cm*, a 2*cm* margin was considered to be a good margin when evaluating the networks.

The optimal type of neural network for this problem would be a CNN. This is because the input that we used was images and as discussed in chapter 2.3, there are several advantages of using convolutions to find spatial structures in images (edges, objects etc...).

To find the optimal number of layers and other parameters the iterative method, described in chapter 3.3, was used. When using this method, it was important to keep good documentation for all iterations. In the beginning of the project the documentation lacked, which in turn slowed down the process. Since we did not know what had already been tested or which combinations gave good results. When this problem was discovered, the documentation improved. Hence, when retraining the pre-trained models later in the project, the documentation did not slow down the process. For future work, a clear frame for the process and documentation should be established before starting the process of finding a satisfying neural network.

ConvNet looked very promising in the beginning, showing good results on the accuracy when testing it on the non-augmented dataset. It was therefore surprising when it did not perform well at all when testing it on the real process. The network had a high accuracy during specific conditions, and it was clear that the network had been overtrained. As described above in chapter 5.1, a bigger dataset was collected and augmented in order to get a more varying dataset and reduce the risk of overtraining. Another option that was explored was to use other neural network alternatives.

After the disappointing result when using ConvNet, pre-trained models such as MobileNet were explored. When MobileNet was evaluated using the non-augmented dataset the accuracy was worse than ConvNet but it performed much better on the augmented dataset and on the real process. The reason being that ConvNet became overfitted on the dataset much faster than MobileNet. The reason for trying MobileNet was because it had been mentioned in a lot of articles, praising its small size and high accuracy. Because of the good result from MobileNet it was decided to explore other pre-trained models. When choosing which pre-trained models to try, high accuracy and the size of the model were considered the most important parameters. Information about these parameters was fetched from [Keras, 2020a], where different models are compared. From this information four pre-trained networks were chosen and evaluated. The results can be seen in table 4.1. With these results, and from discussing with our academic supervisor, it was decided to also try ResNet50V2 in addition to MobileNet.

Neural networks - Results

From looking at table 4.1 it's obvious that both MobileNet and ResNet perform better than the ConvNet. Although the ConvNet version with a classifier out-performs all the other ones.

At a first glance one would probably choose the ConvNet version with the classifier. But as it turns out this model displays bad performance when it comes to the real process. The network is very good at evaluating the test dataset, which was made before augmentation was introduced to the project. However, when the network was faced with new images in the real process the performance was by far the worst of all the three models. This is a typical sign of overtraining. As it turns out, having a structure with both a classifier and a regression network makes the solution unstable. From this we learned that to make a model that works in practice, the theoretical scores generated with test data are less important. Therefore, more of the testing was made on the real process. The solution using the ConvNet version with a classifier was discarded and the focus was now to find a robust model rather than getting a perfect theoretical score.

There are probably several reasons why MobileNet and ResNet perform better than ConvNet. One of them being that the tests are run with augmented data, which is hard for ConvNet to handle. This is also consistent with the results from the tests on the real process, where ConvNet is very sensitive to disturbances. The reason for this could be that ConvNet is a relatively shallow, sequential, network. Both MobileNet and ResNet are very deep networks with shortcuts and other complex improvements, see chapter 2.4. Another big difference is that the pre-trained models have been trained on millions of images already. This way the networks have already learned to find features in images when they are used in this project. Due to this it is not surprising that pre-trained networks perform better than the ConvNet.

While the overall accuracy of the model is a good measurement of its performance it is also interesting to evaluate the accuracy in different ranges. It is much more important for the model to predict accurately when the electric rail is in the center of the image, which is when the vehicle is being charged. When the electric rail is in either side in the images the accuracy is not as important, only that the network knows if it is left or right and the approximate distance. Therefore, a histogram was plotted for all models with the accuracy for the different parts of the range. This is visualized in appendix A figure A.1, A.2, A.3. It is clear in these plots that MobileNet and ResNet have a much better accuracy when the electric rail is in the center of the image than ConvNet.

The reason that we have a better performance in the center and not homogeneous for all values could be because there are more collected images from the center. This was done on purpose since the values close to zero were considered more important. The effects of this are not studied, it was mostly done on a hunch. There is a risk that it might make the network biased to predict values close to zero. There are also groups of images collected at $\pm 5cm$ and $\pm 12cm$. The main idea behind this was that if the network is able to successfully predict -12cm, -5cm, 0cm, 5cm and 12cm that just might help the overall performance by having some distinct reference points. The effects from creating datasets like this are not known and should probably be evaluated before being used further.

The theoretical accuracy in table 4.1 is not enough to determine if the networks are robust enough for our problem. To be sure which network perform better all three networks were tested on a simulation, as described in chapter 4.2. The results from this are presented in figure 4.5. It is very clear in this figure that the ConvNet does not perform as well as both MobileNet and ResNet. This can probably be explained by the same reasoning as above. The result from the simulation is still not enough to convince us that we have found a working network. Thus two more tests were performed, this time on the real process. The first one was done by sending images through the network in real time and evaluating them, figure 4.6. The second test was to make the network predict positions from images, used in the loop with the motor and controller, figure 4.7.

In figure 4.6, showing the a real time plot when moving the camera, it is possible to see that the ConvNet has a tendency to jump between values as compared to MobileNet and ResNets much smoother graphs. When observing the process, it was very clear that ConvNet had a hard time deciding where the electric rail was, even when standing completely still. At the same time both MobileNet and ResNet had no problem with this.

In figure 4.7, showing the closed loop, it can be observed that ConvNet overshoots more. Its indecisiveness is visualized at approximately time 40, as it starts drifting without apparent reason. In this figure it is important to note that the FPS (frames per second) differs. ConvNet is the smallest network, with the least number of calculations, making the Raspberry Pi able to handle 4 FPS. MobileNet processes images at 2 FPS and ResNet at 1 FPS. The FPS is determined by how many images that can be analyzed by a Raspberry Pi 4B each second. This means that even though ConvNet is working at a much higher FPS, it performs worse than the other. We can therefore conclude that ConvNet is not an optimal solution for this thesis' problem.

As seen in the results, ResNet achieves a marginally higher accuracy but is computationally heavier and is therefore slower. This means that MobileNet is the preferred choice since the difference in speed is much larger than the difference in accuracy. This is especially true with the limited hardware available in this project.

5.3 Hardware

The purpose of this project was mainly to test a concept. Because of this the performance that affected how many images could be analyzed each second was ignored. If the task is to create a solution that works in real time, the hardware has to be carefully chosen.

When the AI is running on the Raspberry Pi, about 2 images can be handled each second, though there is plenty of hardware that is designed for image processing with AI. Some of them are rated to be able to analyze more than 100 images/second when running MobileNet. The camera used in the project (The Raspberry Pi Camera Module v2) is rated to collect a maximum 90 frames per second. Therefore, the next step when advancing the project would be to buy a processing unit able of handling

90 fps to test on the process. After this, another evaluation would be required to see if this solution would be sufficient.

5.4 Future possibilities

The field of AI and control theory is an interesting and changing field. There are a lot of things that could be studied further in order to get the best performance. The following are some of the ideas that emerge during the project but were outside of the scope of the project because of time limitations.

AI structure

There are several systems working side by side in a car. Maybe this could be used by combining information between the systems. For instance, the car's speed and acceleration could be used as an input to the AI. Another change to the input could be to use a series of images as input, or the previous output. The AI could also be used to predict more than the position of the electric rail, like weather and a tired driver.

Finding out required accuracy of the AI

When the network's accuracies were calculated the margins were set to $\pm 1cm$ and $\pm 2cm$. These values were decided from looking at the rail and sliding contact and are not necessarily the accuracy required to control the process. One way of finding the minimum accuracy would be to use the simulation environment. This way one could simulate neural networks with different accuracies to find out what the largest margin is that can be tolerated when controlling.

Kalman filter

If the AI and controller is to be used in a real-life situation one can expect more disturbances to the signals generated by the AI. The choice to use a PID controller is most likely still a good one since it is a simple system, modelled as a single integrator, that is being controlled. However, a filter would most likely be needed to achieve good performance in real-life situations. A moving average filter was introduced at one point during the project but since the system was slow (2 Hz), it was considered to worsen performance instead of improving it. If the hardware would be improved, increasing the speed of the system, the filter should be reintroduced for a more stable performance.

An interesting alternative to consider could be the use of a Kalman filter that estimates the position of the sliding contact. In this way more information than the information given by the AI could be used. Speed of the car, acceleration and angle of the car's wheels are all things that could be taken into account when calculating the position.

Predict the certainty of the network

One way of making the system more effective would be to make a calculation of how certain the AI is at its prediction. This could help by being able to adjust the controller accordingly. If the AI is fairly confident that the prediction is within a small interval the controller can act more aggressive. If the AI on the other hand is uncertain, the controller can take a steadier approach or even skip the frame. To implement this would probably be time consuming and the outcome is not at all certain. Because of the complexity it was decided to skip this in the project. Even though it would be a very interesting topic to examine.

IR-Camera

The datasets collected in this project were all captured in a daylight environment. Something to explore further is how the network would react to a darker environment, simulating driving at night. The Raspberry Pi Camera that was purchased for this project was bought without an IR-filter, meaning that the captured images contain infrared light. A future possibility could therefore be to purchase an IR-light to illuminate the electric rail in darkness, while capturing images. It would then be interesting to see if the already trained network could identify the electric rail in the image. If the network is unable to detect the electric rail, another network, specialized for images at night, would have to be trained. When it gets too dark, we would switch to the network trained on images at night.

Datasets

When training neural networks, the datasets are very important for the outcome. While a good dataset will likely improve the network's performance, a bad dataset may hinder the network from solving the problem. There are some aspects when it comes to the datasets used in this project that would be interesting to investigate and try to improve.

One could examine how the composition of the dataset affects performance. Currently the number of images are not evenly spread over the interval. There are as an example clusters of images at -12, -5, 0, 5 and 12 cm. The reason for doing this was mostly based on a guess without much evidence to back it up. Because of this the effects of creating datasets like this should be investigated further if the project was to continue.

In order to make the handling of datasets easier and to avoid confusion and mix ups between dataset-files a program could be created. This way it would be easier to get an overview of the data, generate different datasets and merge new data as new images are collected.

Controller

The controller was designed to be slow, in order to not overreact on the slow signals from the AI. The parameters were chosen by briefly observing the system while the parameters were changed but not much time was put into this. If the hardware would be improved, yielding a much faster system, the controller should be optimized to find more optimal parameters. One way of doing this could be by modelling the system and then using a *Matlab* module to find the parameters. Another way could be to manually find good parameters by finding the limits of the system by trial and error.

5.5 Conclusion

Overall the project was a success. The goal of creating an AI that can analyze images and work together with a controller was achieved. ResNet achieves a marginally higher accuracy but is computationally heavier and is therefore slower. This means that MobileNet is the preferred choice since the large difference in speed is more important than the small difference in accuracy.

To make this concept applicable to the real problem, in an outdoor environment and at greater vehicle speed, there are some improvements to be made. The first thing would be to upgrade the computing hardware to be able to analyze more frames each second, increasing the FPS. Another improvement should be to create a bigger and more diverse dataset for the network to be trained on. This dataset should contain different lightning, weather conditions etc.

To continue this project, we highly recommend investing in a computer with a Graphics Processing Unit (GPU) designed to handle these kinds of computations. Training the neural network could take more than 24 hours with the current dataset. If one would like to continue with a larger dataset it is almost necessary to have access to a powerful GPU and more RAM.

Bibliography

Albawi, S., T. A. Mohammed, and S. 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`.

Bini, S. A. (2018). “Artificial intelligence, machine learning, deep learning, and cognitive computing: what do these terms mean and how will they impact health care?” *The Journal of Arthroplasty* **33**:8, pp. 2358–2361. ISSN: 0883-5403. DOI: `https://doi.org/10.1016/j.arth.2018.02.067`. URL: `http://www.sciencedirect.com/science/article/pii/S0883540318302158`.

Brownlee, J. (2018). *Use early stopping to halt the training of neural networks at the right time*. URL: `https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/` (visited on 2020-12-03).

Chollet, F. (2020). *Transfer learning and fine-tuning*. URL: `https://www.tensorflow.org/tutorials/images/transfer_learning` (visited on 2020-12-02).

Chryssolouris, G., M. Lee, and A. Ramsey (1996). “Confidence interval prediction for neural network models”. *IEEE Transactions on Neural Networks* **7**:1, pp. 229–232. DOI: `10.1109/72.478409`.

Elonroad (2020). *Elonroad*. URL: `https://elonroad.com/` (visited on 2020-12-02).

Freno (2020). *Laptop cartoon*. URL: `https://www.cleanpng.com/png-laptop-macintosh-macbook-pro-15-4-inch-clip-art-la-118041/` (visited on 2020-12-11).

George Philipp Dawn Song, J. G. C. (2018). *Gradients explode - deep networks are shallow - resnet explained*. URL: `https://openreview.net/forum?id=HkpYwMZRb`.

HECHT-NIELSEN, R. (1992). “Iii.3 - theory of the backpropagation neural network**based on “nonindent” by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 ieee.” In: Wechsler, H. (Ed.). *Neural Networks for Perception*. Academic Press, pp. 65–93. ISBN: 978-0-12-741252-8. DOI: https://doi.org/10.1016/B978-0-12-741252-8.50010-8. URL: http://www.sciencedirect.com/science/article/pii/B9780127412528500108.

Keras (2020a). *Keras applications*. URL: https://keras.io/api/applications/ (visited on 2020-12-07).

Keras (2020b). *Model training apis*. URL: https://keras.io/api/models/model_training_apis/ (visited on 2020-12-03).

Khlebovich, P. (2020). *Ip camera adapter 4.0*. URL: https://ip-webcam.appspot.com/ (visited on 2020-12-11).

Kristiansson, B. and B. Lennartson (2006). “Robust tuning of pi and pid controllers: using derivative action despite sensor noise”. *IEEE Control Systems Magazine* **26**:1, pp. 55–69. DOI: 10.1109/MCS.2006.1580154.

Maughan, T. (2015). *The dystopian lake filled by the world’s tech lust*. URL: https://www.bbc.com/future/article/20150402-the-worst-place-on-earth (visited on 2020-02-08).

MissingLink.ai (2020). *7 types of neural network activation functions: how to choose?* URL: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/ (visited on 2020-12-09).

Naoppay (2020). *Engineering cartoon*. URL: https://www.cleanpng.com/png-raspberry-pi-3-single-board-computer-linux-pi-821319/ (visited on 2020-12-11).

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. URL: http://neuralnetworksanddeeplearning.com/.

Peltarion (2020). *2d convolution block*. URL: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block (visited on 2020-12-11).

Perez, L. and J. Wang (2017). “The effectiveness of data augmentation in image classification using deep learning”. *CoRR* **abs/1712.04621**. arXiv: 1712.04621. URL: http://arxiv.org/abs/1712.04621.

Ritchie, H. (2020). *Cars, planes, trains: where do co2 emissions from transport come from?* URL: https://ourworldindata.org/co2-emissions-from-transport (visited on 2020-12-03).

Roelofs, R., V. Shankar, B. Recht, S. Fridovich-Keil, M. Hardt, J. Miller, and L. Schmidt (2019). “A meta-analysis of overfitting in machine learning”. *Advances in Neural Information Processing Systems* **32**, pp. 9179–9189.

Routley, N. (2017). *Visualizing the trillion-fold increase in computing powers*. URL: https://www.visualcapitalist.com/visualizing-trillion-fold-increase-computing-power/ (visited on 2020-12-02).

Sahoo, S. (2018). *Residual blocks — building blocks of resnet*. URL: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec (visited on 2020-12-03).

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen (2019). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Dover Publications.

SMMT (2019). *New car co2 report 2019*. URL: https://www.smmt.co.uk/reports/co2-report/ (visited on 2021-01-27).

Solarbotics (2020). *16mm lens mount for raspberry pi camera v2*. URL: https://www.thingiverse.com/thing:4342032 (visited on 2020-12-10).

Supup (2020). *Engineering cartoon*. URL: https://www.cleanpng.com/png-arduino-uno-atmega328-single-board-microcontroller-2899186/ (visited on 2020-12-11).

Tensorflow (2019). URL: https://www.tensorflow.org/ (visited on 2020-12-03).

Wikipedia (2020a). *Artificial neural networks*. URL: https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=990982491 (visited on 2020-12-02).

Wikipedia (2020b). *Control theory*. URL: https://en.wikipedia.org/wiki/Control_theory (visited on 2020-12-14).

Wikipedia (2020c). *Environmental aspects of the electric car*. URL: https://en.wikipedia.org/w/index.php?title=Environmental_aspects_of_the_electric_car&oldid=991990816 (visited on 2020-12-02).

Wikipedia (2020d). *Residual neural network*. URL: https://en.wikipedia.org/w/index.php?title=Residual_neural_network&oldid=978101817 (visited on 2020-12-09).

Yineri (2020). *Cartoon phone*. URL: https://www.cleanpng.com/png-smartphone-nokia-x7-00-vivo-icon-smartphone-95329/ (visited on 2020-12-11).

Zhang, L., J. Tan, D. Han, and H. Zhu (2017). "From machine learning to deep learning: progress in machine intelligence for rational drug discovery". *Drug Discovery Today* **22**:11, pp. 1680–1685. ISSN: 1359-6446. DOI: https://doi.org/10.1016/j.drudis.2017.08.010. URL: http://www.sciencedirect.com/science/article/pii/S1359644616304366.

A

Histograms

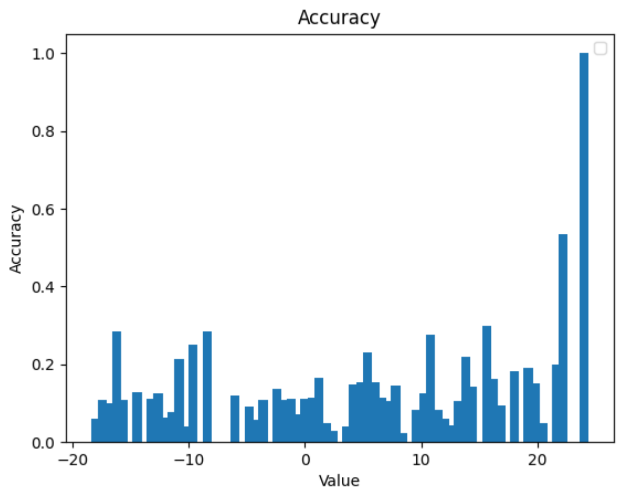

Figure A.1 A histogram show the accuracy over the measurement spectrum for ConvNet.

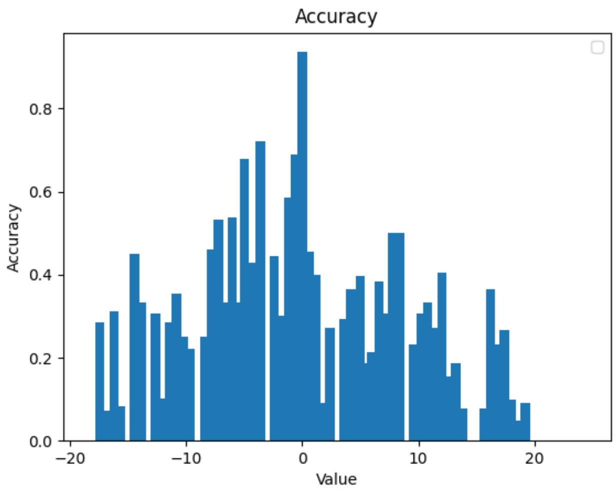

Figure A.2 A histogram show the accuracy over the measurement spectrum for ResNet.

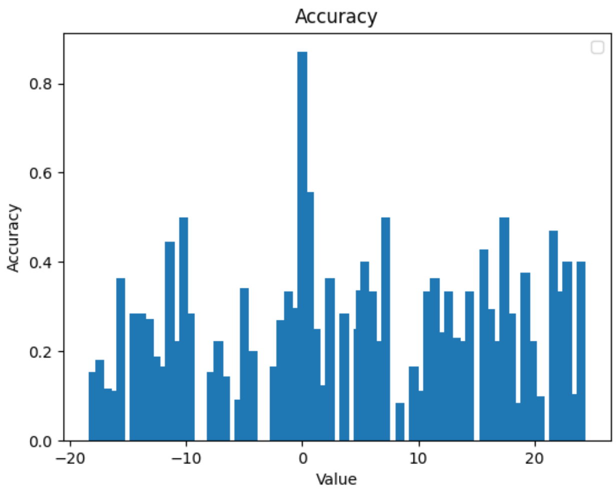

Figure A.3 A histogram show the accuracy over the measurement spectrum for MobileNet.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* February 2021 *Document Number* TFRT-6122
Author(s) Axel Sondh Björn Johnsson	*Supervisor* Dan Zethraeus, Elonroad, Sweden Johan Grönqvist, Dept. of Automatic Control, Lund University, Sweden Kristian Soltesz, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Controlling a sliding contact on an electric vehicle with computer vision and AI

Abstract

Emission from road vehicles is a massive problem and contributes to climate change on our planet. One solution that people are turning to is electrical propulsion instead of fossil fuel. There are, however, problems with putting big batteries on road vehicles. They are expensive to build, they require rare minerals, and the process of creating batteries emits plenty of greenhouse gas.

To reduce the need of big batteries, Elonroad is creating a way of charging road vehicles while driving. This works by putting rails in roads and sliding contacts underneath vehicles. For this to work the sliding contacts and the rail needs to stay aligned while driving.

In this thesis the problem is solved by controlling the sliding contacts position with use of a camera, machine learning and a controller. The proposed structure is to use a pre-trained neural network called MobileNet together with a custom neural network to estimate the position of the sliding contact. The estimated position is then used as input to a PID controller that controls the position of the sliding contact with a motor.

Keywords

Machine learning, Computer vision, Electric vehicle, Electric road

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316	*ISBN*

Language English	*Number of pages* 1-49	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/