# Development of Autonomous Vehicle using Convolutional Neural Network

A Thesis Submitted to University of Bridgeport

In Partial Fulfillment of the Requirements For the Degree of

**Master of Computer Science**

**In the Department of Computer Science**

By

**Gideok Seong**

**May 2019**

Approved by

Supervising Committee:

Supervisor: [illegible]

Jeongkye Lee, Professor of Computer Science and Engineering

[illegible]

Abdel-shakour Abuzneid, Associate Professor of Computer Science and Engineering

[illegible]

Sarosh Patel, Visiting Assistant Professor, School of Engineering

## Acknowledgments

I would like to say that I feel so lucky that I had so much support from Dr. Jeongkyu Lee while I completed this project. He explained so many details that I needed to imagine the blueprint for my thesis. If I did not have his support, this project would not have been as successful. In addition, I am so grateful that I could study abroad with strong support from my family in South Korea. Without their support, I would not have finished my program here. While I have been studying abroad, I have been able to broaden my perspective and I have come a long way. I am grateful that I could meet a variety of people from many different countries and learn more about their fields. I had the chance to learn about a lot of fields such as data mining, machine learning, and big data, which I had not experienced during my undergraduate studies. I am grateful for what I have learned at this school, and I am happy I had the opportunity to complete this project. I would also like to thank my English teacher, Ian Shiach, for his help editing my grammar.

## Abstract

The market for autonomous vehicles (AVs) has been grown up quickly over the past few decades. The reason why AVs have become so popular is because people from all over the world are concerned about driving safety. Worldwide about 1.25 million people are killed each year in traffic accidents, and the introduction of AVs could prevent up to 90% of all auto accidents in the U.S. alone. Accordingly, many companies have been conducting research and field trials, such as Google and Ford. However, it is hard for students to acquire basic knowledge about AV technology and understand how it is developed, because the systems are very complicated, and it is expensive to build model vehicles. Thus, we built a small, simple model system that can help students better understand how AVs work and how to contribute to their development. We use a SunFounder (Smart Video Car Kit for Raspberry Pi) to build a downsized AV car (demonstrator) with lane detection and signal detection capabilities. For lane detection, we use built-in OpenCV functions such as Canny Edge Detection and Hough Line Detection to detect lanes and draw lane lines. For signal detection, we utilize TensorFlow to make a model with a convolutional neural network (CNN) in a Raspberry Pi environment.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

## Introduction

### 1. Background

Since the early 1920s, autonomous vehicles (AVs) have been mentioned, but they have not been a main focus for computer scientists because until recently, they lacked the required computers, algorithms, and camera technology [1]. However, in the 1980s, the development of AVs sharply increased globally [1]. One of the primary reasons was that more than a million people are killed each year in traffic accidents worldwide. If AVs were more fully adopted, a large majority of annual car accidents would be prevented. In accordance with the increasing necessity of AVs, many companies including Waymo (Google) [2], Tesla [3], and Ford [4] have started researching AVs in earnest.

Even though AVs are not ubiquitous yet, AV technology is developing quickly, so we can expect to see more of them on the road in the next few decades. As AVs become more popular in the real world, we need to predict their positive and negative effects. One of the main benefits will be fewer automobile accidents [5], which will prevent a tremendous amount of property damage and passenger injuries. Another benefit is that when AVs are able to drive themselves independently, people will be able to spend their time doing other things, such as read, relax, or catch up on work. In addition, some of the issues caused by human drivers, such as driving while intoxicated, drowsy, or distracted, can be prevented [5]. One drawback to AVs is that they may not be able to make ethical decisions [6]. Furthermore, AVs may not be able to detect all of the objects ahead if many objects are captured by the camera at once [6]. Lastly, in the near future, it may be impracticable to use AVs in some places, such as very large metropolises, because we do not know how AVs will respond to unpredictable situations [6].

We need more people to deal with these issues, but it is hard for them because it is expensive and hard to understand the systems, so we implement a downsized car model system to make it easier for students to understand.

## 2. Objective and Research Questions

The main goal of this project is to develop a downsized AV that can detect lanes and traffic signals. The functions such as Canny Detection, Hough Line Detection, and other algorithms are used to capture the lane lines, so that the AV can reliably stay on a track while driving itself, without staying off the lane lines. Also, a model using TensorFlow, which is used for a deep neural network, is made to implement traffic signal recognition for 43 signal classifications. Our research questions are:

1. *How can we program an AV to reliably stay within lane lines?*
2. *How can we program an AV to differentiate between traffic signals and other objects?*

## *3.* Method

To create the downsized demonstration AV, an off-the-shelf remote-controlled model car from SunFounder is utilized, i.e. Smart Video Car Kit for Raspberry Pi. SunFounder provide access codes to control the motors that drive and steer the AV. The AV features a camera that is fixed to face forward. A straight track with two parallel lane lines that are a constant distance apart is used. German Traffic Image Datasets, which have around 40,000 images and 43 classifications, is used as training images for signal detection.

To successfully answer the research questions, the AV is able to do the following:

- Have a similar design as a real car, with 4 wheels and front-wheel steering
- Control the servos to steer the car
- Use a DC converter to fully provide power to the servo and the Raspberry Pi computer
- Use the camera to capture images frame by frame

# Chapter 2

## Hardware and Software Dependencies

### 1. Introduction to an autonomous car and its components

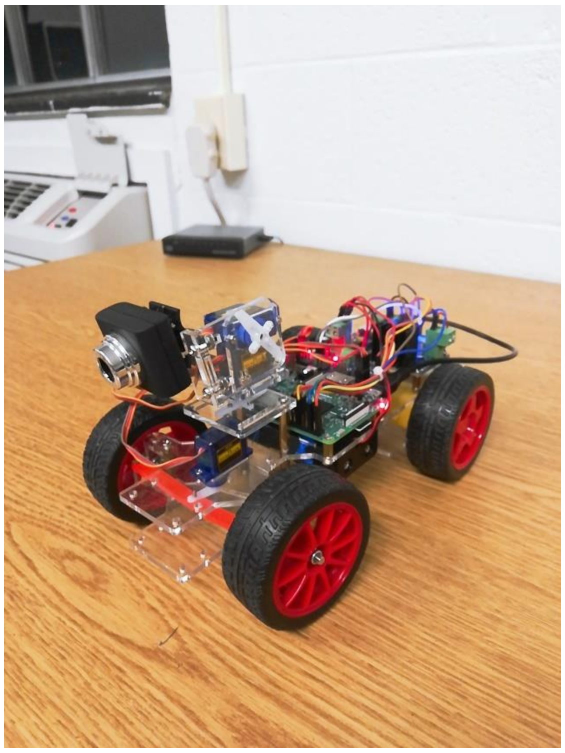


**Figure 2.1. A finished product**

For this project, the SunFounder Smart Video Car Kit (Figure 2.1.) is used to create the demonstrator car. There are many options for remote controlled cars that can be turned into demonstrator AVs; this model is selected because it supports a camera module that can connect to Raspberry Pi, which makes it simple to implement Python algorithms. Raspberry Pi has access to many libraries which are in the Python environment, including OpenCV, TensorFlow, and Numpy. The hardware of the demonstrator is comprised of several main components, such as a DC converter module, a DC motor driver, a servo, a servo controller, and a Raspberry Pi computer. The DC converter module converts the battery output of 7.4V to 5V in order to provide the Raspberry Pi and the servo with power. The DC motor driver, as the name suggests, controls the two DC motors. It allows for the proper voltage and power usage depending on the capacity of motors. The servo controls the direction of the AV, so that the car can turn left and right. The servo can also manipulate movement of the camera, but for this project, we fix the camera to face forward in order to solve our research questions. The servo controller directly manages the servos. In this project, a few predefined Python codes (PCA9685 and motor) are used, which are provided by SunFounder, to steer the AV. The motor code references the PCA9685 code directly to access the servos and manage them. These functions on the Raspberry Pi, as well as some other functions are run to detect lane lines and traffic signals.

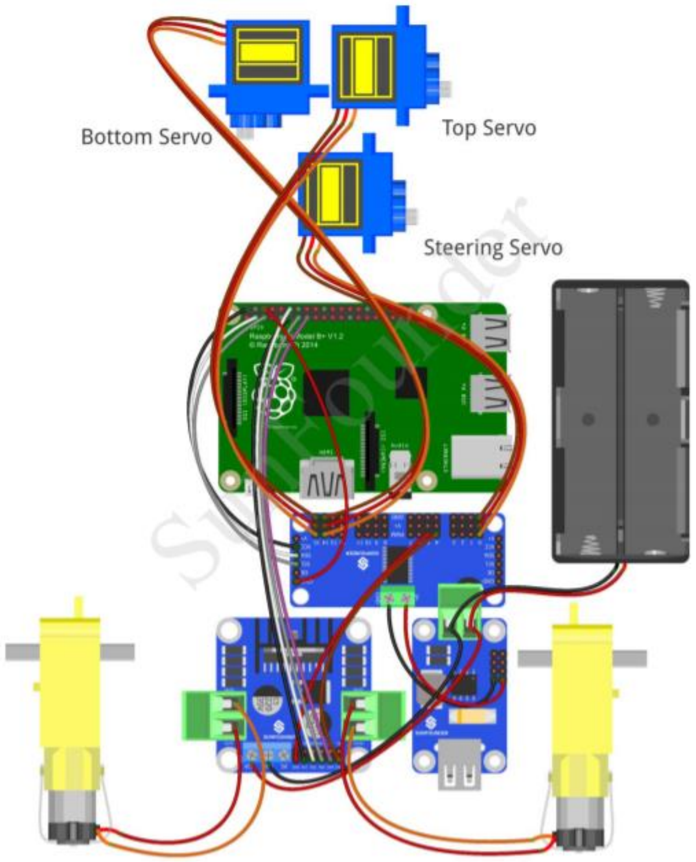


**Figure 2.2. The overall structure of hardware (Source from - https://www.robotshop.com)**

**Figure 2.2.** above represents the overall structure of hardware and **Table 2.1.** below shows the main car components.

| Parts | Name | Qty. |
|---|---|---|
| | Raspberry Pi Model B+ | 1 |
| | 16-Channel 12-bit PWM driver (servo controller) | 1 |
| | L298N DC motor driver | 1 |
| | Step-down DC-DC converter module | 1 |
| | Tower Pro Micro Servo SG90 | 3 |
| | Gear reducer | 2 |
| | Driven wheel | 2 |
| | Active wheel | 2 |

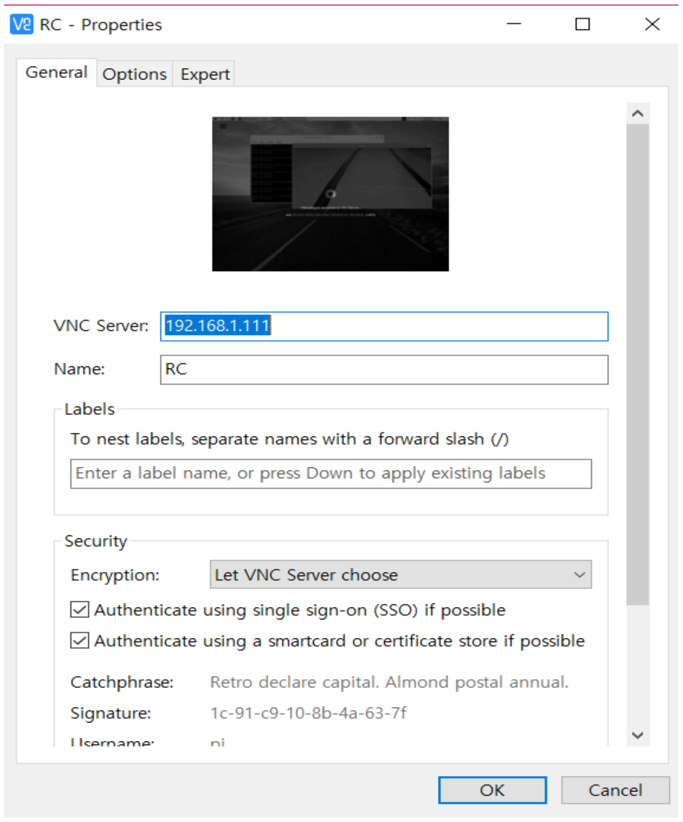
**Table 2.1. Car components (Source from - https://www.robotshop.com)**

## 2. Set up dependencies – Raspberry Pi

In order to use the Raspberry Pi on the AV, a tool called win32DiskImager is used to burn an image of a Raspberry Pi operating system called Raspbian onto an SD card, which is then inserted into the Raspberry Pi. The Raspberry Pi uses the SD card to run the Raspbian operating system. Also, a configuration file on the SD card that allows us to connect to the Raspberry Pi over Wi-Fi is used. It includes the configuration information displayed in the below, including the Wi-Fi ID (ssid) and password (psk).

```
{

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev

update_config=1
network={

     ssid="5EA0DE"
     psk="00340212"
     key_mgmt=WPA-PSK

}
```

Raspberry Pi server has its own IP server address. To access the Raspberry Pi server, a program called VNC viewer is utilized. The following Figure 2.3. shows the settings that is used to connect to the Raspberry Pi server.



VNC Server: 192.168.1.111
Name: RC
Labels
To nest labels, separate names with a forward slash (/)
Enter a label name, or press Down to apply existing labels
Security
Encryption: Let VNC Server choose
☑ Authenticate using single sign-on (SSO) if possible
☑ Authenticate using a smartcard or certificate store if possible
Catchphrase: Retro declare capital. Almond postal annual.
Signature: 1c-91-c9-10-8b-4a-63-7f
Username: pi

**Figure 2.3. Access to Raspberry Pi server**

To access the Raspberry Pi, its IP address is needed. To determine the IP address, a program called Advanced IP Scanner is used. After connecting to the Raspberry Pi, the necessary dependencies are set up such as Python, OpenCV, and Numpy to it.

## 3. Set up dependencies – OpenCV

For this project, OpenCV version 3.3 is chosen. However, this is not the latest version; an older version is used to to avoid compatibility issues. This version of OpenCV is only compatible with Python 3.x versions, so Python version 3.5 is used. The first step to download OpenCV is to expand the filesystem in Raspberry Pi to use all of the available space on the SD card using the command:

```
$ sudo raspi-config
```

From there, “expand file system” from the advanced menu is used. To apply the change, we reboot the server using the command:

```
$ sudo reboot
```

By typing `$ df -h` , we check to see if the disk space has actually increased. There are various dependencies existing to make OpenCV work in Raspberry Pi server. We use update and upgrade packages existing using the command:

```
$ sudo apt-get update && sudo apt-get upgrade
```

Then, we install some developer tools including CMake, which helps configure the OpenCV build process using the command:

```
$ sudo apt-get install build-essential cmake pkg-config
```

Next, we need to download some image I/O packages that are related to loading various image file formats from disk, for example, JPEG, PNG, TIFF, etc:

```
    $ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

Same as the image,  We download I/O, video I/O packages to load video file formats from disk.

```
    $ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-
dev
    $ sudo apt-get install libxvidcore-dev libx264-dev
```

To display images to our screen and build basic GUIs, we use a sub-module named highgui. So as to compile highgui, we deploy GTK development library.

```
    $ sudo apt-get install libgtk2.0-dev libgtk-3-dev
```

There are extra dependencies we use to optimize many operations inside of OpenCV (matrix operations):

```
    $ sudo apt-get install libatlas-base-dev gfortran
```

Next, we download Python environment to be able to compile OpenCV:

```
    $ sudo apt-get install python2.7-dev python3-dev
```

After we download all the dependencies, we get OpenCV version 3.3 from the official OpenCV repository.

```
    $ cd ~
    $ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip
    $ unzip opencv.zip
```

To use full features of OpenCV, we install opencv_contrib as well:

```
    $ wget -O opencv_contrib.zip
    $ https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip
    $ unzip opencv_contrib.zip
```

Before compiling OpenCV on the Raspberry pi, we install pip, which is a Python package manager:

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo python3 get-pip.py
```

At this point, to not conflict with other dependencies, we use virtual environment by keeping only necessary dependencies on different virtual environment:

```
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
```

Then, we update the profile information as follows:

```
$ export WORKON_HOME=$HOME/.virtualenvs
$ export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
$ source /usr/local/bin/virtualenvwrapper.sh
$ source ~/.profile
```

Then, we choose which version of Python will be used when creating virtual environment:

`$ mkvirtualenv cv -p python3` or `$ mkvirtualenv cv -p python2`

To start the virtual environment, we name it as follow:

```
$ workon cv
```

For numerical processing, we use Numpy module:

```
$ pip install numpy
```

After getting into virtual environment, we set up build using CMake:

```
$ cd ~/opencv-3.3.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
    -D BUILD_EXAMPLES=ON ..
```

Before compiling, we need to increase swap space size to speed up to compile. We change in `/etc/dphys-swapfile` and `CONF_SWAPSIZE=1024`

To use new swap space:

```
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
```

Finally, we compile OpenCV as follow: `make -j4`

we finish installing OpenCV on Pi:

```
$ cd /usr/local/lib/python3.5/site-packages/
$ sudo mv cv2.cpython-35m-arm-linux-gnueabihf.so cv2.so
$ cd ~/.virtualenvs/cv/lib/python3.5/site-packages/
$ ln -s /usr/local/lib/python3.5/site-packages/cv2.so cv2.so
```

Now we set up every dependencies for OpenCV, so we test as follow:

```
$ source ~/.profile
$ workon cv
$ python
import cv2
cv2.__version__
```

# Chapter 3

## Lane detection

### 1. Overview of lane detection

Several functions, including Canny Edge Detection and Hough Line Detection, are utilized to create a lane detection algorithm pipeline. First, the original RGB image is converted to grayscale, then a Gaussian blur function is used to suppress noise and non-logical gradients by averaging the values. Next, the grayscale image is used through the Canny Edge function, which detects the edges of the lanes, and converts the grayscale image into a binary image with white pixels for the lane and black pixels for the area outside the lane. Then, to detect specific lanes on the track, we create a function called region_of_interest, which defines the area of the lane lines and get the output that unnecessary data is reduced. Finally, the masked binary image is passed through the Hough Line function, which redraws the lane lines depending on the slope of the edges in the binary image.

## 2. Canny Edge Detection

The main purpose of Canny Edge Detection is to decrease the amount of data in an image, while conserving the main properties to be used for further image processing [7]. There are many edge detection methods, but Canny Edge Detection is one of the standard edge detection functions because of its efficiency to process the data [7]. Canny Edge Detection is designed to develop an algorithm with the following criteria:

**1. Detection:** maximized signal-to-noise ratio

**2. Localization:** detection edges as similar as possible to real edges

**3. Number of responses:** one real edge should match the edge to be detected

Canny Edge Detection has five separate steps:

**Step 1 . Smoothing:** blur the image to reduce noise

**Step 2. Finding gradients:** The edges should be marked where the gradients of the image have large magnitudes.

**Step 3. Non-maximum suppressions:** Only local maxima should be detected as edges.

**Step 4. Double thresholding:** Potential edges are determined by thresholding.

**Step 5. Edge tracking by hysteresis :** Final edges are decided by suppressing all edges that are not connected to a very certain (strong) edge.

A Gaussian filter is used to reduce noises and average nonlogical gradients as the equation (3.1).

$$B = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{3. 1}$$

To find the edge gradients, the equation (3.2) and (3.3), which Horizontal direction is $(G_x)$ and Vertical direction is $(G_y)$ are used.

$$Edge_{\_Gradient}(\mathrm{G}) = \sqrt{G_x^2 + G_y^2} \tag{3.2}$$

$$\text{Angle } (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{3.3}$$

The Figure 3.1 shows how we deal with non-maximum suppressions [8].

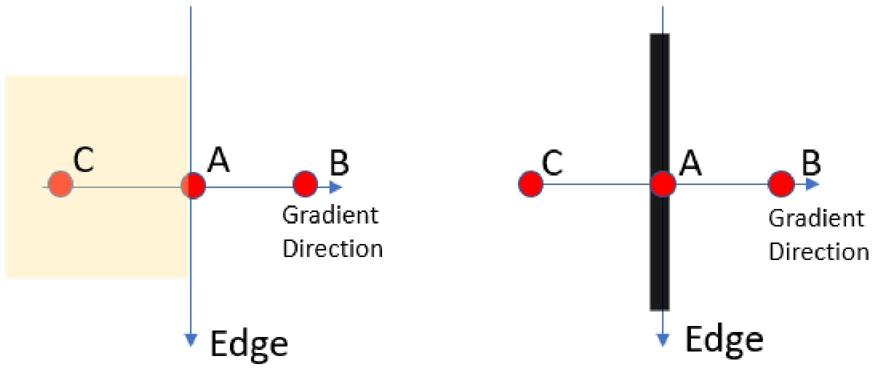


**Figure 3.1. Edge with local maximum**

In the Figure 3.1, only Point A is considered an edge, because it has a local maximum in its neighborhood in the direction of the gradient. As a result, the output is a binary image with this edge [8].

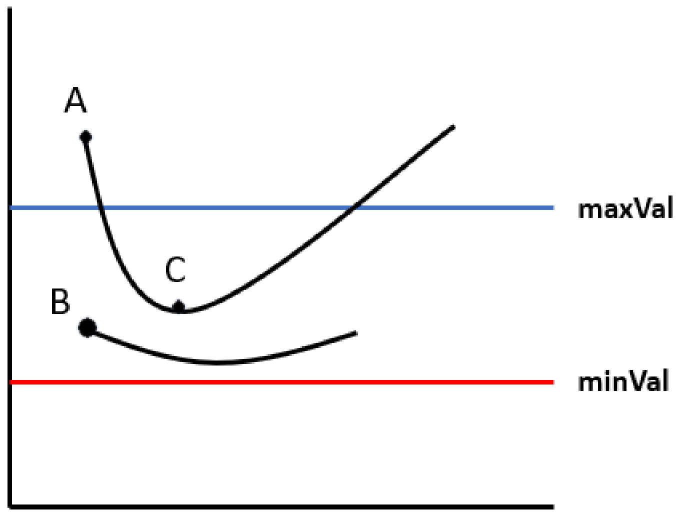


**Figure 3.2. Values between maxVal and minVal**

The Canny Edge Detection function includes two thresholds, the minimum (minVal; 0) and maximum (maxVal; 255) number of pixels. In the Figure 3.2., Point A is considered a “sure-edge” so it is used for edges. Point C is considered a “valid edge” and it is connected to Point A, so it is still utilized as a full curve. However, Point B is not connected to “sure-edge” Point A, so we discard it. The Figure 3.3. displays the edges after applied for Canny Edge Detection function.

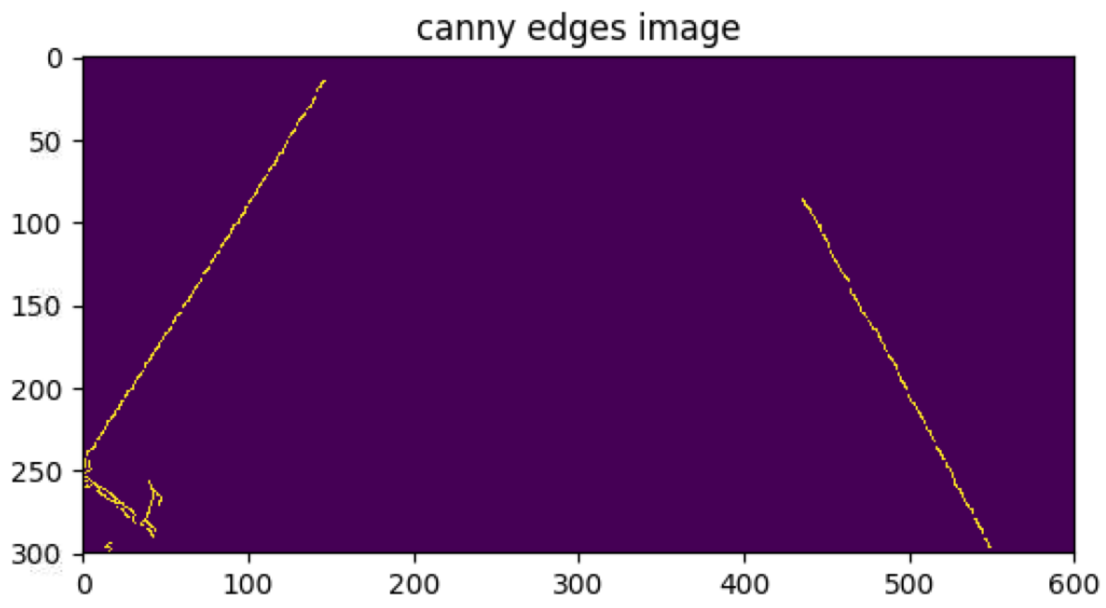
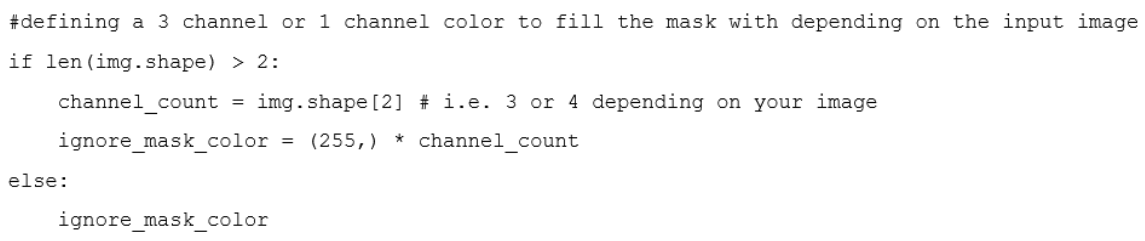


**Figure 3.3. Edges applied for Canny Edge Detection function in lanes. This is an example image that shows the edges after we applied the Canny Edge Detection function.**

## 3. Masked Edge Detection

This function allows edges to be more distinguished by using certain color channels as the Figure 3.4. below and choosing non-zero values in the edge image in a specific area, which is vertices.



```
#defining a 3 channel or 1 channel color to fill the mask with depending on the input image
if len(img.shape) > 2:
    channel_count = img.shape[2] # i.e. 3 or 4 depending on your image
    ignore_mask_color = (255,) * channel_count
else:
    ignore_mask_color
```

**Figure 3.4. Color channels for the mask**

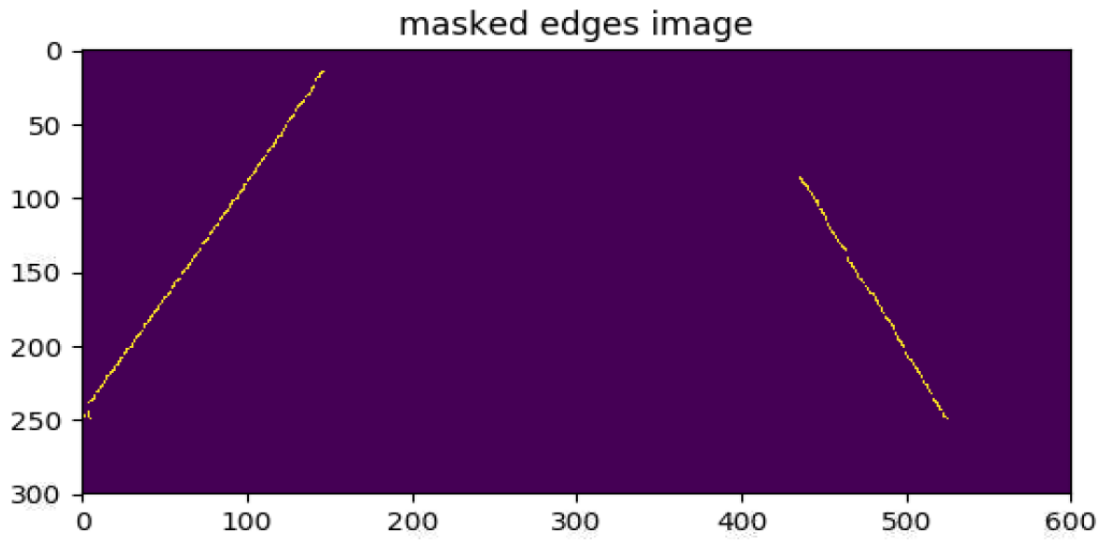
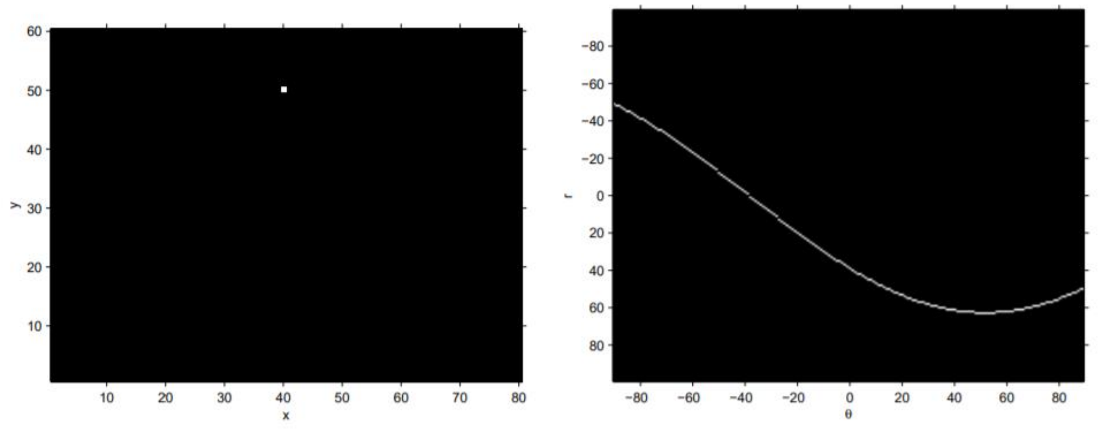


**Figure 3.5. Masked edge image**

The Figure 3.5. above displays the image applied for Masked Edge Detection function in the project. It detects non-values from the edge image, so it removes the area unnecessary.

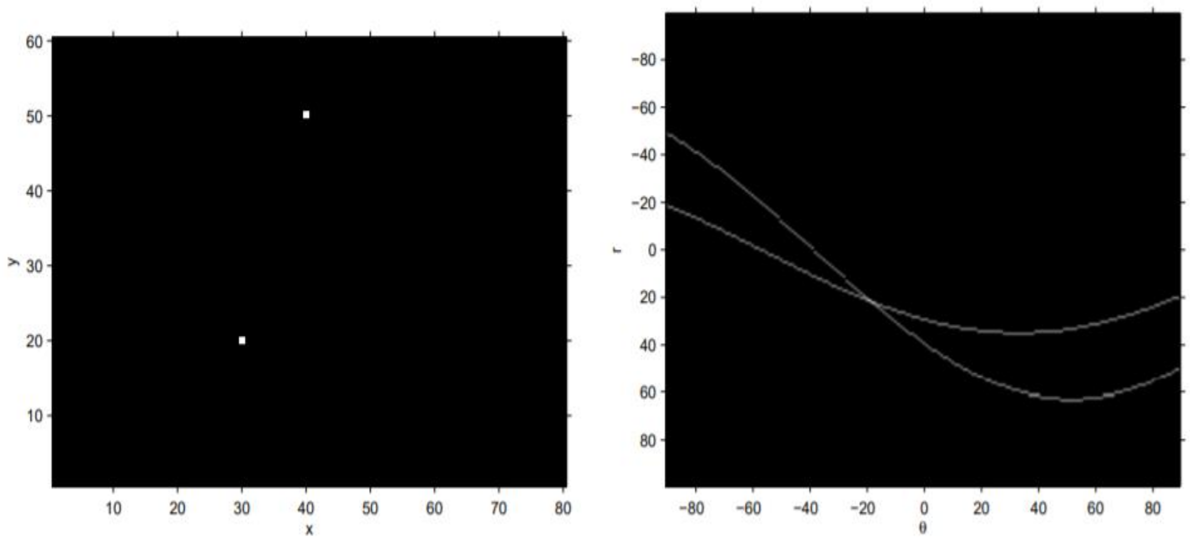
## 4. Hough Line Transform Detection

Paul Hough introduced the method to find lanes in a binary image in 1962. The output of edge detection is vital to reduce the amount of data and detects only necessary edges in an image fairly. However, the result from the edge detector is still an image represented by its pixels. Hough Line Transform theory comes from that if lines and ellipses could be defined by their characteristic equations, then amount of data would considerably be reduced again [9]. General idea for Hough Line Transform is that one edge points in a binary input is represented straight lines in the Hough space as the following Figures 3.6.(a). and (b) [9].



**(a)** Point $p_0$. **(b)** All possible lines through $p_0$ represented in the Hough space.

**(a). the line that the edge point is converted in the Hough space**



**(a)** Points $p_0$ and $p_1$. **(b)** All possible lines through $p_0$ and/or $p_1$ represented in the Hough space.

**Figure 3.6.(b). The intersection that points p0 and p1 share**

In conclusion, each edge point is transformed to a line in the Hough space. Then, the area where most intersection in the Hough space have is interpreted as true lines in the edge map. Thereby, a lot of data can be lessened with high probability of capture of the edge.

## 5. Other algorithms for lane detection

First, we apply for the Grayscale function using cv2.cvtColor(img, cv2.COLOR_RGB2GRAY), which gets only one-color channel, to an original image.

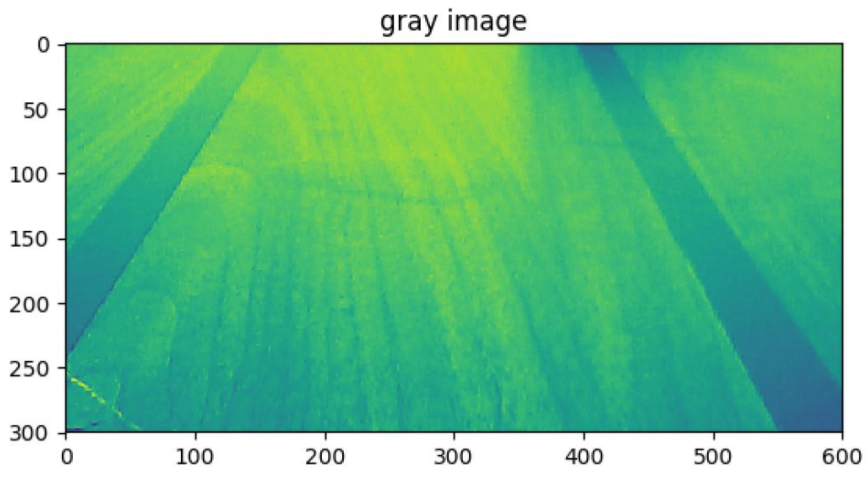


**Figure 3.7. Grayscale image**

The Figure 3.7. shows the output after applied for Grayscale Function. We use the Gaussian Function that is used to reduce noises and average nonlogical gradients.

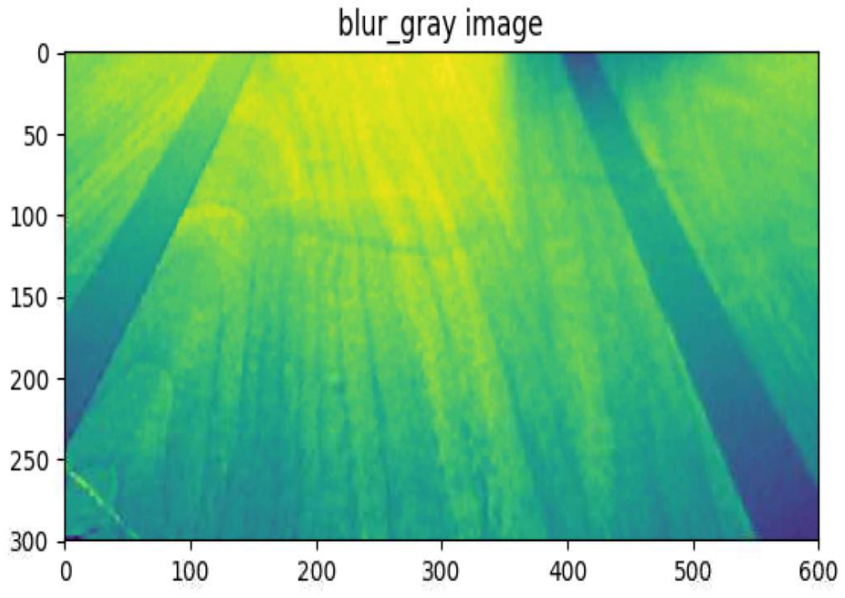


**Figure 3.8. Image applied for Gaussian smoothing.**

We use the Gaussian smoothing to reduce noises and to use Canny Edge Detection. The Figure 3.8 displays the image after applied for Gaussian smoothing.

# Chapter 4

## A model for traffic sign recognition

### 1. Dataset

In this project, for the training, we deploy German Traffic Sign dataset. Training images have a total of 39209 images. A testing image is the frame that the camera takes every second in the main class. Also, the number of labels is 43. About 200 training images each label has the one same label. All of the training images are cropped out by finding only a traffic sign in the background. The size of each training image is different, but it is around (30,30) size. However, to train a model for the training images, we manipulate its image size to (32,32) uniformly. Sample training images look like as the Figure 4.1.

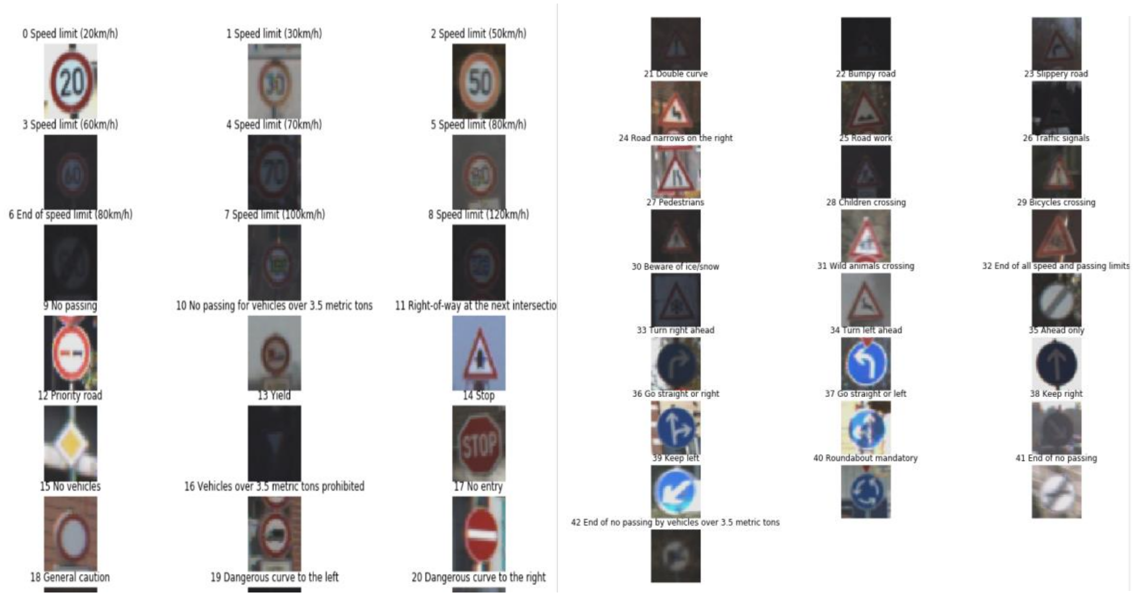


**Figure 4.1. Types of training images with each own label**

## 2. Model Architecture

In this project, for the model, we use convolutional neural network. Convolutional neural network is a kind of artificial neural network that utilizes convolutional layers to filter inputs for valuable information. An input layer, an output layer, and one or more hidden layer consist convolutional networks. The overall structure of the model we implement is as the Figure 4.2.

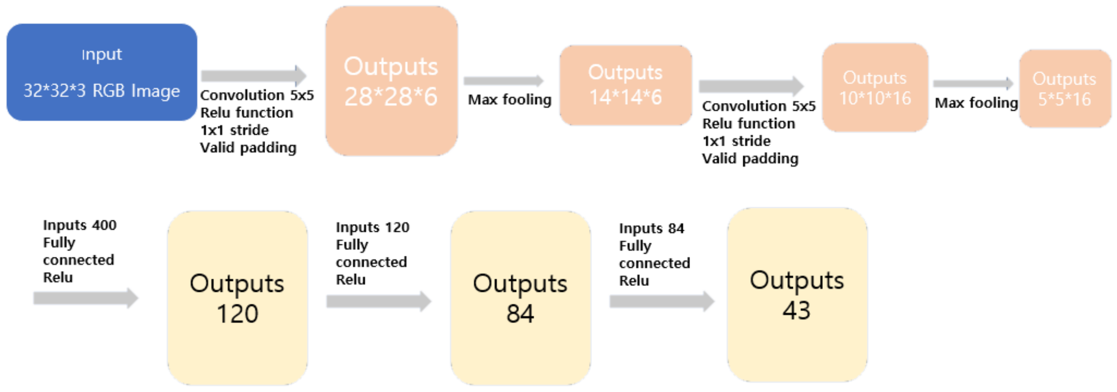


**Figure 4.2. Model structure**

A convolutional network is different than a regular neural network in that the neurons in its layers are organized in three dimensions (width, height, and depth dimensions). Our model has a total of three convolutional layers and one fully-connected layer. The main advantage of using convolutional networks is to find the best feature based on the task by reducing a lot of unnecessary information using a convolutional kernel, which is a filter. The filters are adjusted based on learned parameters to grab the most useful information. Furthermore, a pooling, or subsampling is used for saving a memory by picking up the maximum or average value depending on the type of the pooling in the feature maps. In terms of activation function, we use RELU function, which stands for rectified linear function, since it is good for speeding up the training by getting rid of all negative values while not influencing on accuracy.

### 3. Cross-Entropy

In order to train the model, cross-entropy is indispensable to evaluate the performance of a classification model where output is a probability between 0 and 1 [10]. High probability means low a log loss. Following Figure 4.3. shows the loss in our project and it keeps decreasing depending on the epoch.

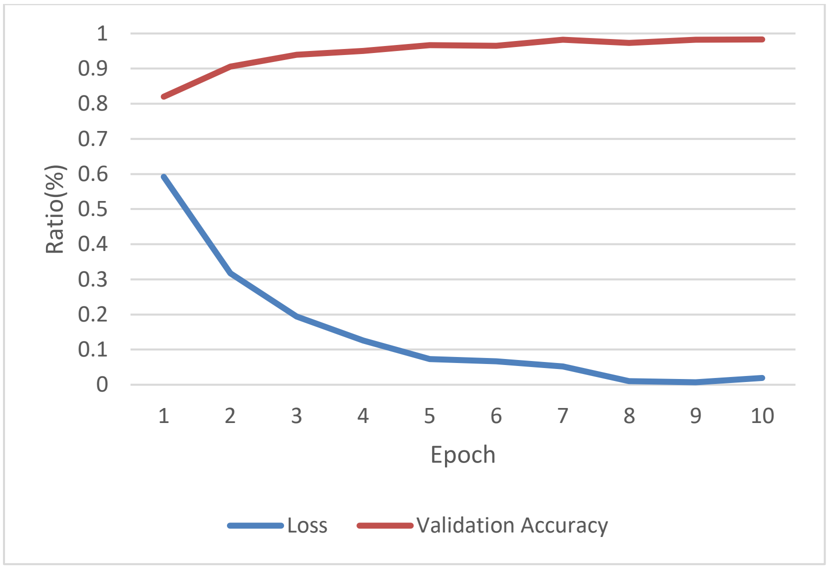


**Figure 4.3. Entropy loss depending on the epoch in the project.**

Entropy loss $\mathrm{H(y)} = -(y\log(p) + (1-y)\log(1-p))$ (4.1)

Using the equation (4.1), we calculate entropy loss. When it comes to multiple classification, we use the equation (4.2)

Cross entropy $\mathrm{H}(\mathrm{y}, \hat{\mathrm{p}}) = -\sum_{c=1}^{M} y_{o,c}\log\left(p_{o,c}\right)$ (4.2)

M – number of classes (dog, cat, fish)

log – the natural log

y – binary indicator (0 or 1) if class label *c* is the correct classification for observation *o*

*p* – predicted probability observation *o* is of class *c*

In case that high log loss is shown, we can tune the parameters that are used for training such as weight and bias. That is the fundamental goal of utilizing Cross-Entropy by checking the log loss and adjusting parameters.

## 4. Optimizer: Adam

Notion 'optimizer' had been emerged to make predictions as correct as possible during the training process. The definitive goal of using optimizer is to minimize the loss function. It keeps parameters updated while training. In our project, we deploy Adam optimizer, or adaptive moment estimation, since Adam is well suited for the problems that are large with regard to data, or parameters [11]. It uses past gradients to calculate current gradients. Adam utilizes the notion of momentum, a method which aids accelerate gradients vectors in the right directions, by adding fractions of previous gradients to the current one.

# Chapter 5

# Experiments

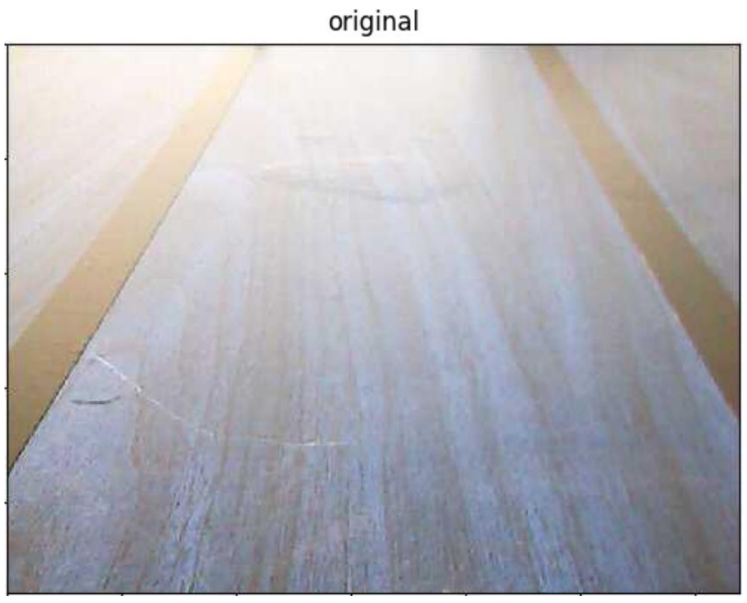
## 1. Lane detection

### 1.1 Project description

Our first research question is, *"How can we program an AV to reliably stay within lane lines?"* To answer this question, we use a downsized remote-controlled car with a fixed-position camera, and we create a track with equally spaced lane lines. We keep the speed of the car equal during testing. We use the Canny Edge Detection and the Hough Line Detection functions for lane detection.

### 1.2 Project experiments

The camera captures too much information because the frame is too large, so we crop the frame in order to focus on the lane lines. Figures 5.1.a and b show the differences between an original image and a cropped image.



**(a) Original image with two lane lines**

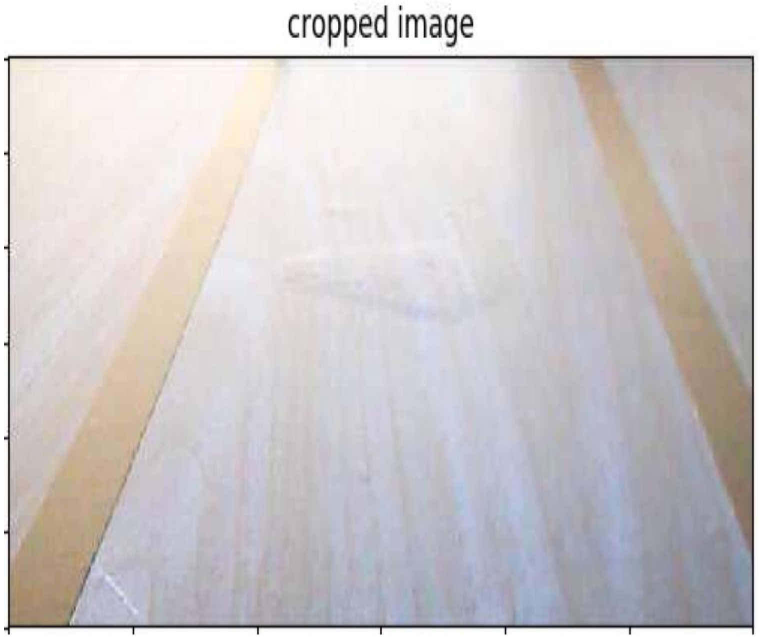


**Figure 5.1.(b) Cropped image with two lane lines**

By cropping the image, we eliminate the unnecessary area, which could have interfered with the output of the experiment. Next, we adjust the kernel value for the Gaussian Blur function (Figure 5.2).

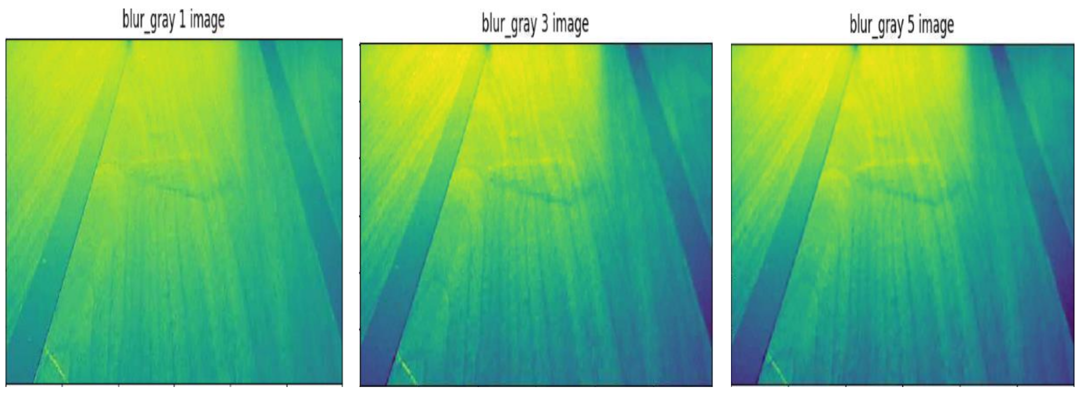


**Figure 5.2. Images with different values of kernels after Gaussian blurry function(value of 1, 3, and 5, respectively)**

Using a kernel value of 3 (Figure 5.2, middle image) shows the clearest lane images after we run the Canny Edge Detection function and Hough Line Detection functions. The Canny Edge function also has parameters to set, which include the high and low thresholds for the colors that are designated part of the lane and not part of the lane. For example, if the low threshold is too low, too many pixels are detected, and the output includes extra incorrect data. The optimal values by fine tuning through trial and error are chosen. Figure 5.3 shows how the output changes as the low and high threshold values change.

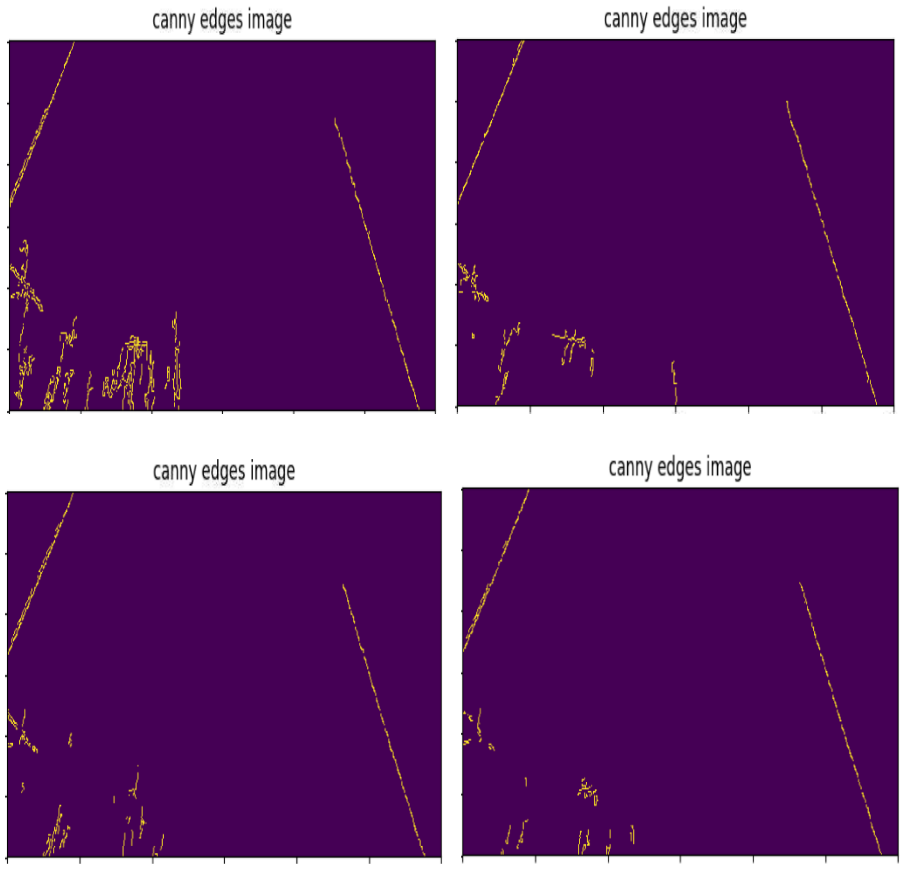


**Figure 5.3. Output images from the Canny Edge function with different high and low threshold values. Each represents (20, 40), (30, 160), (50, 150), (50, 160) minimum and maximum thresholds, respectively.**

The Canny Edge function with a low threshold value of 50 and a high threshold value of 160 displayed a good result that has fewer incorrect edges. Before we run the Hough Line function to detect the lane lines, vertices has to be calculated, which specifies the polygonal area that captures only the lane line edges.

We use the vertices and the output from the Canny Edge function to create a function called region_of_interest, which helps us detect the area within the lane, and ignore the other areas of the image (Figure 5.4).

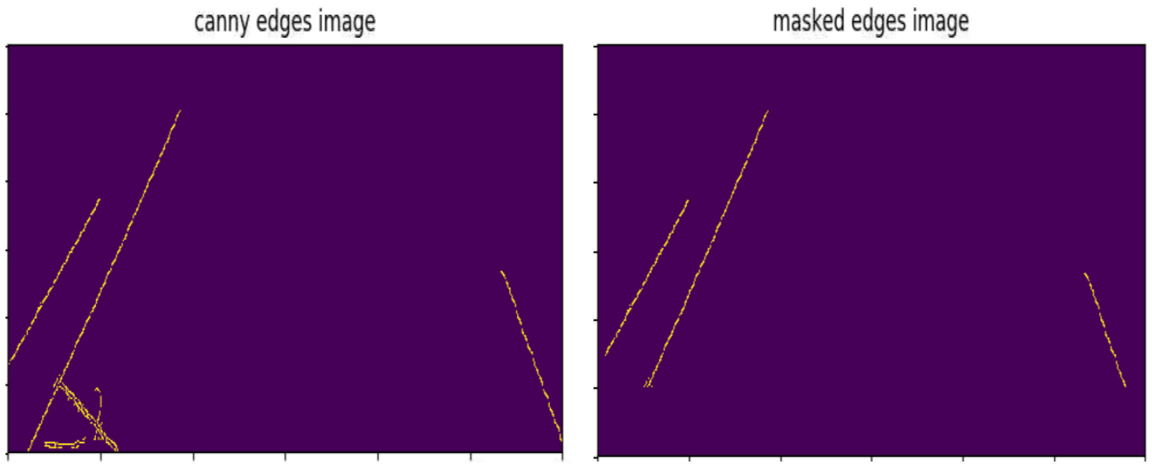


**Figure 5.4. the left (Canny Edge function) the right (region_of_interest)**

After we run the region_of_interest function, the images are free of unnecessary edges.

Finally, we deploy the Hough Line function to draw the lines. Hough Line function uses several parameters such as ρ, which is the distance resolution in pixels of the Hough grid; θ, which is the angular resolution in radians of the Hough grid; threshold, which is the minimum number of intersections in Hough grid cell; min_line_len, which is minimum number of pixels making up a line; and max_line_gap, which is maximum gap in pixels between connectable line segments. Depending on these values, we use the cv2.HoughLinesP function to make the lines with the edges. Using these lines, we implement a function called draw_lines, which detects the x and y coordinates of the points that make up the detected edges and draws the lane lines on the edge image. Using the function called cv2.addWeighted, we redraw the edges of the lane lines on the original image (Figure 5.5).

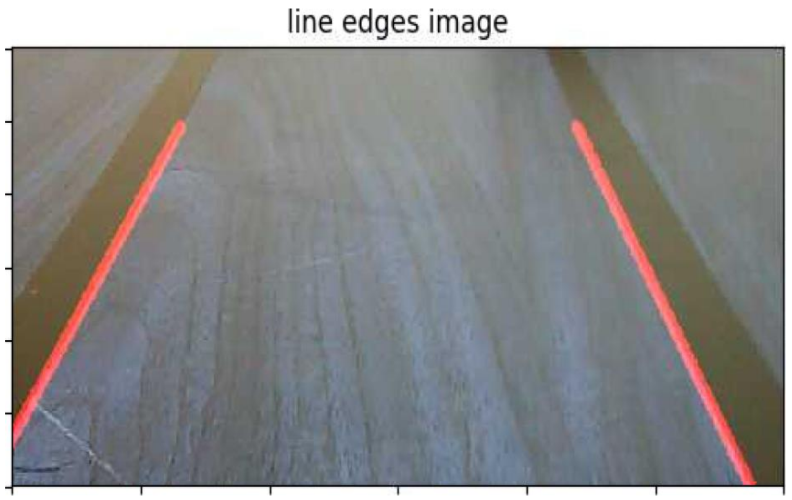


**Figure 5.5. Final detected line edges on the original image**

## 2. Traffic signal detection

### 2.1 Project description

Our second research question is, "*How can we program an AV to differentiate between traffic signals and other objects?"* To answer this question, we use the trained model referred to in Chapter 4. Also, we utilize downsized traffic signals and we assume that traffic signals are always situated in the same position. In addition, we crop the image to a size of 32 square pixels, because the model only accepts images of that.

### 2.2 Project experiments

We use the trained model that had 97% accuracy after the completion of training (Chapter 4). To use the trained model in our project, we use a session concept, which has all of the trained information including the trained images, in the main function. However, during testing, there was an issue; the AV only detects a traffic signal when the signal fits completely within the frame, so the error rates are too high. To deal with this issue, we utilize several frame areas at once. For example, we make four frames that slightly overlap, thus increasing the possibility that the AV successfully captures the traffic signals.

# Chapter 6

## Conclusion

In this project, we have two main research questions: *How can we program an AV to reliably stay within lane lines?* To solve this question, we use multiple OpenCV functions and algorithms, and modify numerous parameters to better detect the lane lines. Through trial and error, we figured out what functions were the most important and most influenced the results. For future work, we want to improve the downsized AV to make it move more like a real AV. Even though the downsized AV did not leave the track, it did not drive straight down the middle of the lane like a real car would. The second research question is: *How can we program an AV to differentiate between traffic signals and other objects?* By designing a model using TensorFlow and training it with a convolutional neural network, we are able to pass frames from the AV's camera through the model in order to detect traffic signals. It allows the AV to accurately detect traffic signals when the frame matches with the training images. Based on that, we could make the car respond to the traffic signals.

I hope that students including undergraduates and graduates who are interested in autonomous vehicles use a downsized AV model like what we implemented to learn about the knowledge of AVs and programming. I also hope that they contribute in this field by learning efficiently and simply.

## References

[1] Bimbraw. K. Autonomous Cars: Past, Present and Future A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology. 12th International Conference on Informatics in Control, Automation and Robotics, 191-198, ISBN: 978-989-758-122-9, 2015

[2] Bansal. M, Krizhevsky. A. and Ogale. A. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. arXiv:1812.03079v1 [cs.RO] 7, 2018

[3] Howell. D, **"**Tesla, Mobileye Rev Up On Future Of Self-Driving Car", 2015, https://www.investors.com/tesla-motors-mobileye-stock-up-on-self-driving-cars-uber/

[4] Hawkins. J. "Ford's self-driving cars are really good, but are they good enough to win?", 2018, https://www.theverge.com/2018/11/15/18096338/ford-self-driving-car-miami-argo-av

[5] Montgomery. D. Public and Private Benefits of Autonomous Vehicles. Securing America's Future Energy, 2018

[6] Litman. T. Autonomous Vehicle Implementation Predictions Implications for Transport Planning. Victoria Transport Policy Institute, 2019

[7] Canny Edge Detection. 09gr820, 2009

[8] OpenCV, "Canny Edge Detection", 2019, https://docs.opencv.org/master/da/d22/tutorial_py_canny.html

[9] Line Detection by Hough transformation. 09gr820, 2009

[10] Kroese. P, Rubinstein Y. and Glynn W. The Cross-Entropy Method for Estimation. Handbook of Statistics. Vol. 31, ISSN: 0169-7161, 2013

[11] Kingma. P. and Ba. L. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. ICLR 2015, arXiv:1412.6980v9 [cs.LG], 2017