T.C

BURSA TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

AUTONOMOUSLY MOVING CAR USING IMAGE PROCESSING AND ARTIFICIAL INTELLIGENCE MODELING

ONUR DURMUŞ **AHMET EMİN KÜÇÜK**

LICENSE THESIS

ELECTRIC AND ELECTRICAL ENGINEERING

CONSTULTANT

ASSOC. PROF. CEMAL HANİLÇİ

JUNE 2020

BURSA

APPROVAL PAGE

Onur DURMUŞ and Ahmet Emin KÜÇÜK, who graduated from the Faculty of Engineering and Natural Sciences, Faculty of Electrical and Electronics Engineering, numbered .... with their thesis titled "Autonomously moving car using image processing and artificial intelligence modeling". successfully presented before the jury members with signatures.

Thesis Advisor :	**Assoc. Prof. Cemal HANİLÇİ** **Institute Name**	**....…….**
Co-Consultant:	**Doç. Dr. Adı SOYADI** **Institute Name**	**....…….**
Jurors :	**Yrd. Doç. Dr. Adı SOYADI** **Institute Name**	**.....**

PROJECT SUPPORT PAGE

In accordance with the articles 9/2 and 22/2 of the Graduate Education and Training Regulation published in the Official Gazette dated 20.04.2016; According to the criteria determined by the Institute of Science, an appropriate report was taken to this undergraduate thesis using the plagiarism software program subscribed by Bursa Technical University. This thesis was supported by Bursa Technical University Scientific Research Projects Unit ..... numbered project.

Due Date: 15/06/2020

Thesis Defense Date: 20/06/2020

PLAGIARISM STATEMENT

In this thesis, we declare that all the information and results presented in visual, auditory and written form are obtained by us in accordance with the academic and ethical rules, and that we have stated all the results and information in the thesis that are not specific to this study by citing the source.

Ahmet Emin KÜÇÜK & Onur DURMUŞ

Signatures

PREFACE

In this thesis study carried out under the roof of Bursa Technical University, this project, which we have developed based on the use of artificial intelligence that has been spreading rapidly in many sectors and automobile sectors with the technology that has developed recently, has been carried out on the autonomous use of cars.

We have a duty to thank our consultant teacher, Associate Professor Doctor Cemal Hanilçi, who guided us in thesis and who gave us their help, and our families who offered us their support during this time.

CONTENTS

ABBREVIATIONS

YOLO: You Only Look Once

CNN: Convolutional Neural Network

R-FCN: Region-based Fully Convolutional Network

R-CNN: Regoion Convolutional Neural Network

SSD: Single Shot Multibox

HSV: Hue,Saturation,Value

OpenCV: Open Source Computer Vision

RGB: Red,Green,Blue

FIGURE LIST

AUTONOMOUSLY MOVING CAR USING IMAGE PROCESSING AND ARTIFICIAL INTELLIGENCE MODELING

Onur DURMUŞ Ahmet Emin KÜÇÜK

BURSA TEKNİK ÜNİVERSİTESİ

JUNE 2020

ABSTRACT

The production of an autonomous car consists of three important stages. The first is artificial intelligence, the second is image processing, and the third is design. In the system to be used embedded, the most appropriate artificial intelligence modeling is done by taking into consideration the performance of the system and the conditions under which the autonomous car will operate. In addition, since image processing is fundamental, it is aimed to work integrated with artificial intelligence modeling.

However, it is not clear whether the design and modeling will fulfill the expected performance. Because the basic microprocessor selection in this system that will be embedded in the car may be insufficient for this modeling. In order to answer this issue, necessary engineering and analysis should be done. By using various artificial intelligence models, modeling suitable for the microprocessor to be used is selected. In this thesis, firstly information will be given about image processing and artificial intelligence modeling and then it will be combined with image processing by training neural networks. And it is planned to compare these models and create the most suitable software. Necessary tests of the selected software will be carried out. The most appropriate design will be made for this software and the materials used and the system will be aimed to work embedded in the car.

1. INTRODUCTION

1.1. Introduction and Purpose of the Project

Image processing has been using for the object detection in recent years. Particularly, it was accelerated with improving of the technology. Hereat, the machine learning algorithms was came into use by the image processing, which is the deep learning so we can say that deep learning is subset of the machine learning. A deep learning algorithm was manufactured being founded on the human brain. It has several neuron layers and the model trains using by these layers for the prediction. Deep learning can use at the wherever we want, such as weather, medical, army. Besides that, there are another options for the prediction. One of them is the Haarcascade Algorithm. It has broadly the same operating logic with the deep learning algorithm. Image processing in target and target location detection in the project being discussedalgorithms are used. Selecting OpenCV for image processing. The main reason is that OpenCV is quite fast in this regard.

At this Project, the line tracking and traffic signs detection were examined, which are the commonly used properties by the image processing at the autonoms cars and purpose of this project that be aimed providing a software concerned this topic. The specified target tarfik signs were detected and the data set was created and the data set was introduced to the training and identified using the haar cascade algorithm. The Ross Kippenbrock method, which includes specific filtering, histogram synchronization, and perspective image warp, was used during the detection phase of road lanes. Raspberry pi mini computer was used for image processing and the necessary distance measurement and engine riding operations were carried out with the Arduino card and serial communication protocol.

2. IMAGE PROCESSING SYSTEMS

2.1 Introduction of the Image Processing Systems

Image processing is a method developed to digitalize the image perform some operations, used to obtain specific images or extract some useful information from that. It is a type of a signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Out of all these signals, the field that deals with the type of signals for which input image and the output is also an image is done in image processing. So that deals with the processing of images, as it name suggest.

2.2 Sense of the Image Processing

There are two different type of processing; one of them is analog image processing and the other one is digital image processing. Analog image processing is done in analog signals. It includes processing on two dimensional analog signals. In this type of processing, the image are manipulated by electrical means by varying electric signal. The common example is the television image. An other example is analog or visual image processing techniques can be used for hard copies such as photocopies and photographs. Other type of image processing technique is digital image processing. Digital image processing is deals with developing a digital system that performs operations on an digital image. Digital image processing has dominated over analog image processing with the passage of time due its wider range of applications.

2.3 Usage Areas

Image processing is used in many fields in technology. Since digital image processing has very wide applications and almost all of the technical fields are impacted by digital image processing. Digital Image processing is not just limited to adjust the spatial resolution of the everyday images captured by the camera. Applications of Digital Image Processing;

1- Image sharpening and restoration,
2- Object Detection & Artifical Neural Network
3- Mediceal field,
4- Remote sensing,
5- Transmission and encoding,
6- Machine/Robot vision,
7- Color processing,
8- Pattern recognition,
9- Video processing,
10- Microscopic Imaging.

3. OBJECT DETECTION WITH IMAGE PROCESSING & DEEP LEARNING

In this section, the decision was made to combine previously used image processing methods with artificial intelligence. Because in the autonomous car to be created, a system is needed to process the traffic signs, line tracking and traffic lights in the environment every time the camera sees it. It was compared in the artificial intelligence algorithms considered.

3.1 Object Detection & Segmentation

There are many methods for object detection and classification methods. Some of those ; YOLO, R-CNN, SSD, DNN etc. If these are mentioned a little bit, they all have their own methods, advantages and disadvantages. The comparison garph is shown in Figure 3.1.

3.2 Modern Convulotioanl Detection & Segmentation

Detections:

- R-FCN
- FASTER R-CNN
- YOLO
- SSD

Segmentation:

- Mask R-CNN
- SegNet

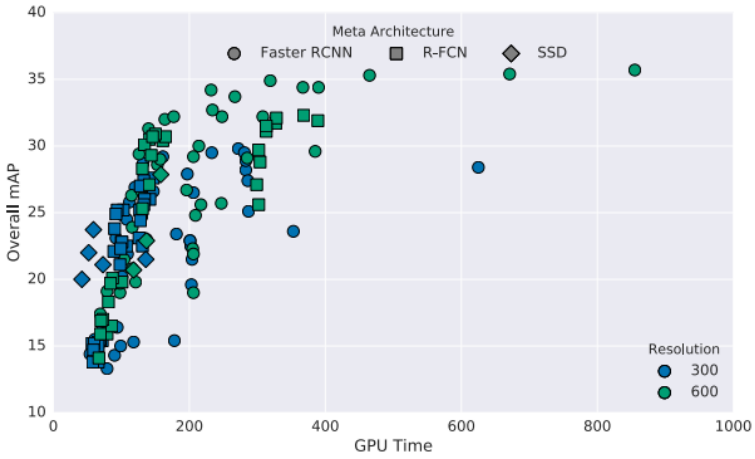


Figure 3. 1

3.3 Faster R-CNN

Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Compared to previous work, Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy.[1][2][3]

- Proposal Generator – Box Classifer
- Good performance but longest run time
- Multi-task loss

3.4 R-FCN

Fast R-CNN computes the feature maps from the whole image once. For every ROI, no more feature extraction is needed. That cuts down the process significantly as there are about 2000 ROIs. Following the same logic, R-FCN improves speed by reducing the amount of work needed for each ROI. It is shown in Figure 3.2.[1][2][3][4][20]

- Addresses translation-variance in detection
- Good balance between speed & performance

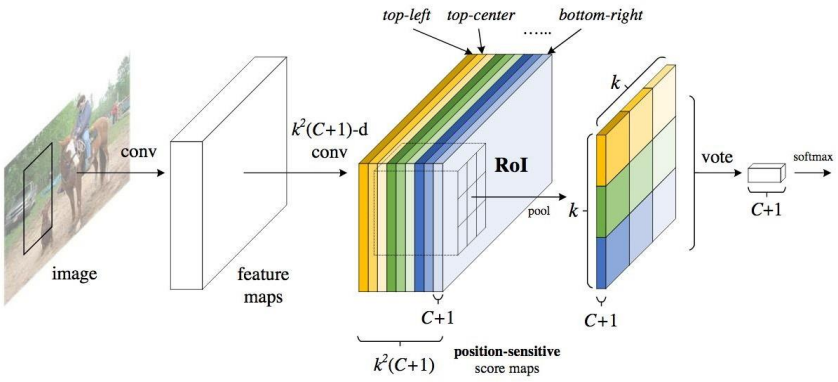


Figure 3. 2

3.5 YOLO (You Only Look Once)

In traditional computer vision approaches, a sliding window has been used to search for objects at different locations and scales. Since this is a very expensive operation, it has been assumed that the aspect ratio of the object is usually constant. YOLO approaches the object detection problem in a completely different way and differs from other neural networks. Transmits the entire image over the network only once. SSD is another object detection algorithm that transmits the image as a deep learning network once, but YOLOv3 is much faster than SSD while achieving very similar accuracy. While it achieves a better result in real time, it seems to be much more successful than the others as FPS. Trainig results are shown in Figure 3.3 and Figure 3.4.[21][11][12]

- Super Fast (21 – 155 FPS)
- Finds objects in image grids at parallel
- Only slightly worse performance than Faster R-CNN

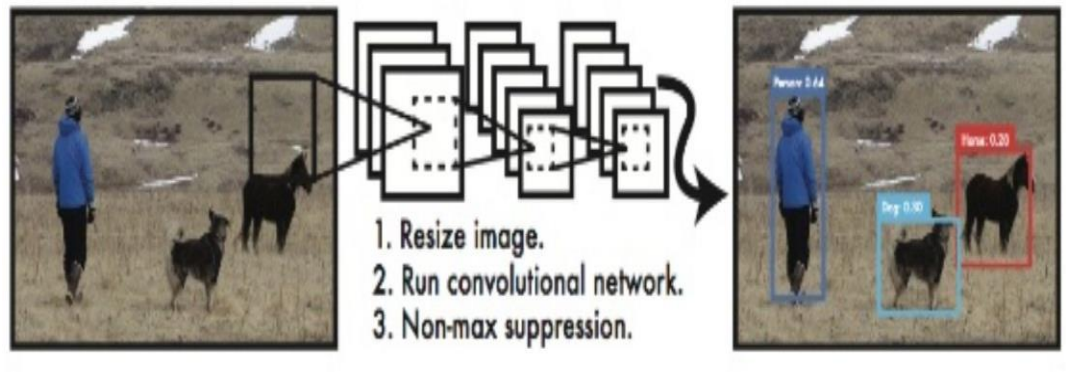


Figure 3. 3

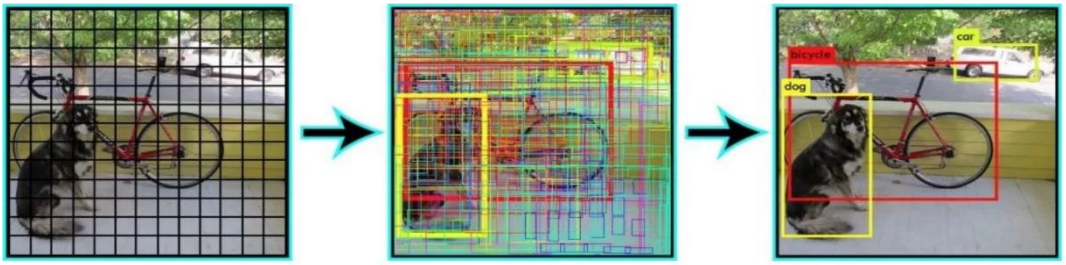


Figure 3. 4

3.6 SSD (Single Shot Detector)

SSD is designed for object detection in real-time. Faster R-CNN uses a region proposal network to create boundary boxes and utilizes those boxes to classify objects. While it is considered the start-of-the-art in accuracy, the whole process runs at 7 frames per second. Far below what a real-time processing needs. SSD speeds up the process by eliminating the need of the region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN's accuracy using lower resolution images, which further pushes the speed higher. According to the following comparison, it achieves the real-time processing speed and even beats the accuracy of the Faster R-CNN. SSD algorithm is shown in Figure 3.5. Comparison of SSD algorithm against to other deep learning algorithms is shown in Figure 3.6.[10][12][22]

- End to end training like YOLO
- Predicts category scores for the default set of default bounding boxes using small convolutional filters applied to feature maps
- Predicts from different feature maps of different scales

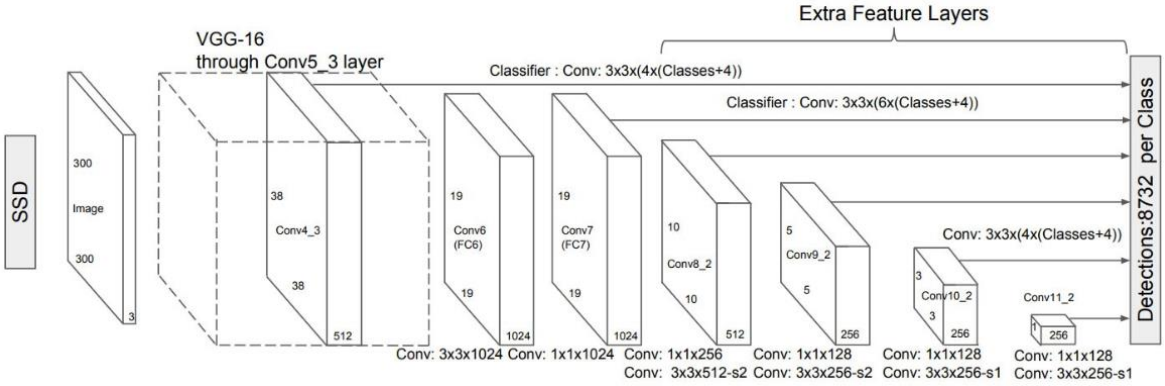
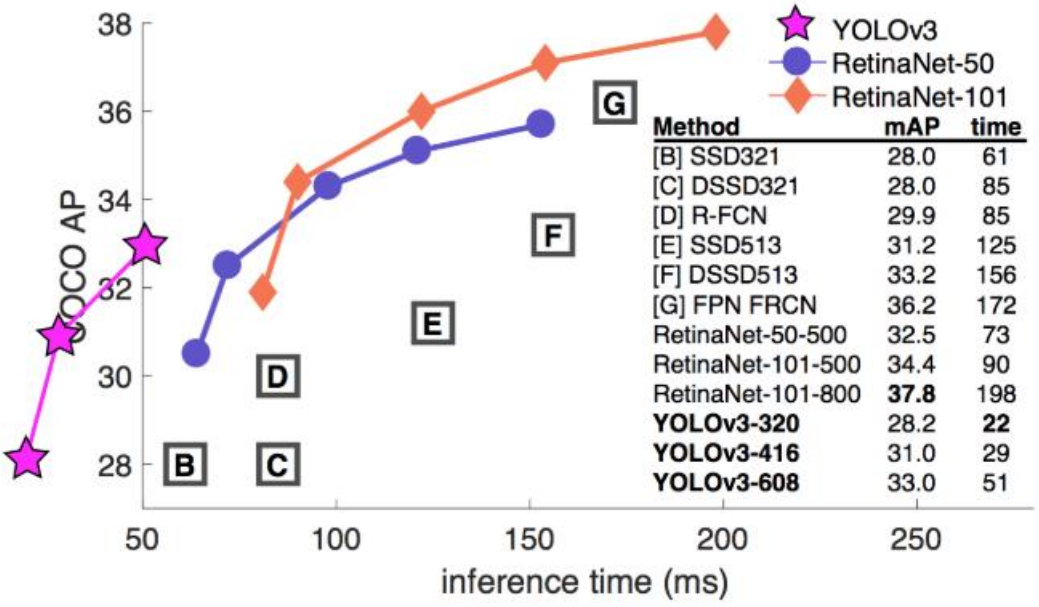


Figure 3. 5

3.7 Comparing Performances (YOLOV3, SSD, FASTER R-CNN)



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 3. 6

That performance data, according to COCO datasets. According to that graph, YOLOv3 was found to be a more reliable choice for faster and reliable performance and better FPS in real-time systems. Here is a example for YOLOV3 obect detection. It is shown in Figure 3.7.[24][25][26]

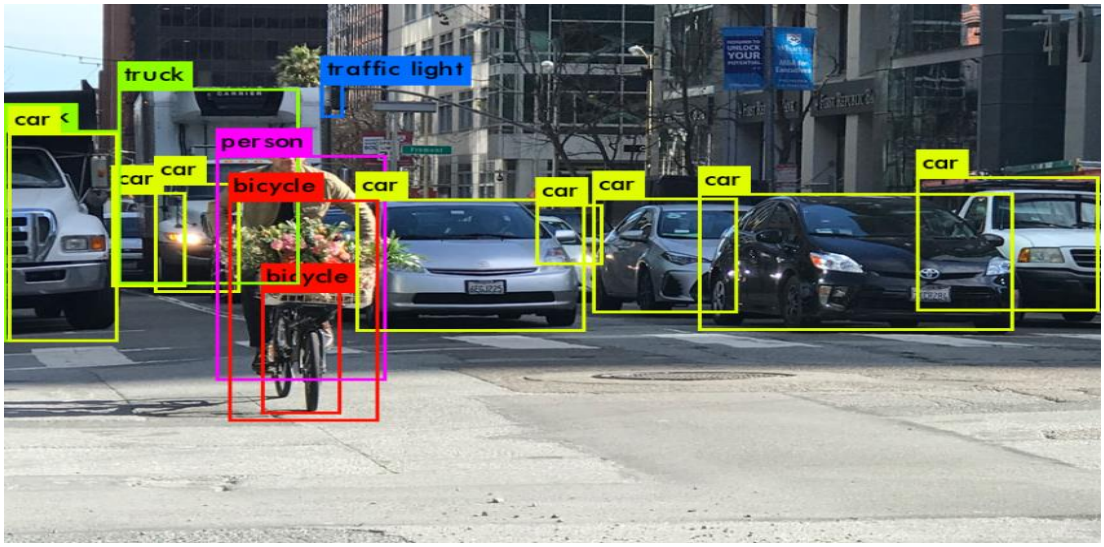


Figure 3. 7

3.8 Why YOLOV3-Darknet ? How It Works ?

A completely different approach is used. A single neural network is applied to the full image. This network splits the image into regions and estimates the bounding boxes and possibilities for each region. These bounding boxes are weighted by predicted probabilities. Yolov3 model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.[25][26]

3.8.1 Bounding Boxes

The network predicts 4 coordinates for each bounding box, tx, ty, tw, th. If the cell is offset from the top left corner of the image by (cx, cy) and the bounding box prior has width and height pw, ph, then the predictions correspond to Figure 3.8:[25]

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

Figure 3. 8

During training that use sum of squared error loss. If the ground truth for some coordinate prediction is tˆ* that gradient is the ground truth value (computed from the ground truth box) minus its prediction: tˆ* − t*. YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more thanany other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold that ignore the prediction. That use the threshold of .5. Unlike that system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness. Shown in Figure 3.9.[26][25]

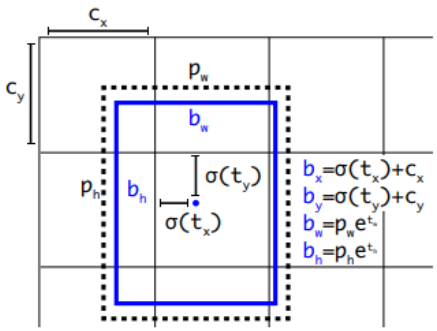


Figure 3. 9

3.8.2 Features Extractor

It uses a new network for performing feature extraction. That new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. That network uses successive 3 × 3 and 1 × 1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so that call it 'darknet53'. Shown in Figure 3.10.[24][25][26]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3. 10

3.9 YOLOv3 Modeling Stages

As a first step, the dataset was created. Due to the need for a lot of photos, a video of the object we focused on was shot. As we know, the video consists of sequential pictures. The video is divided into images with a simple python code. Thus, dataset was created in Figure 3.11 and 3.12.

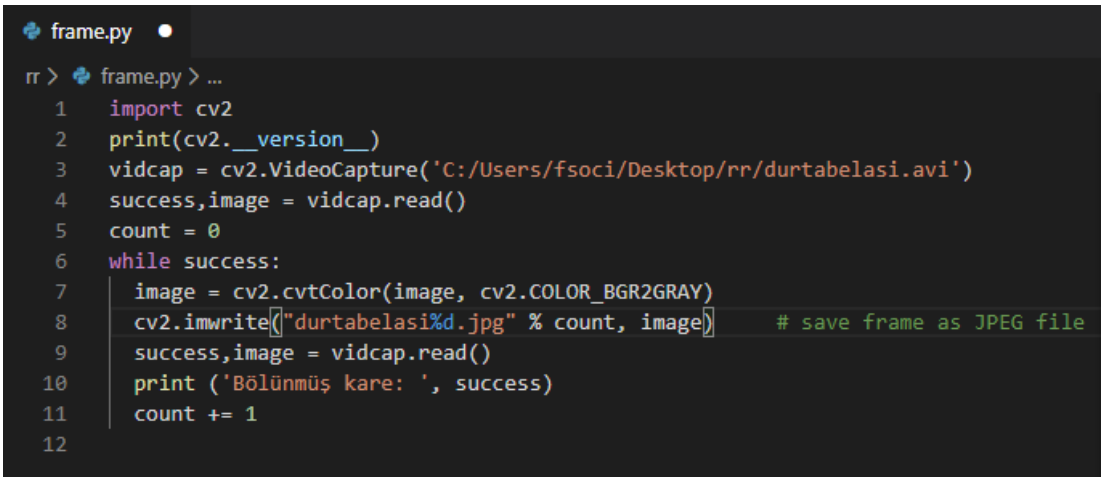


Figure 3. 11

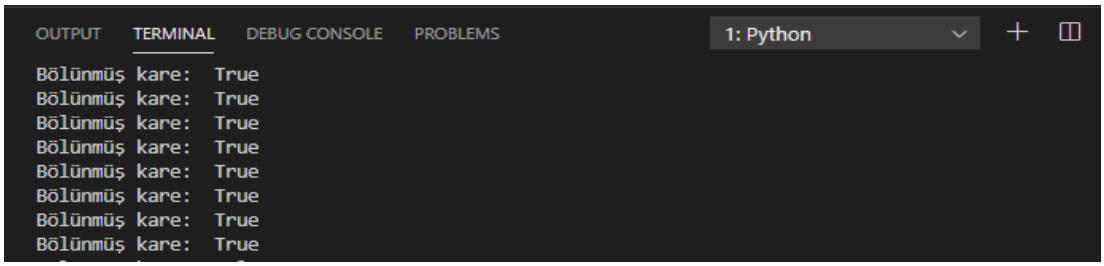


Figure 3. 12

After the code was run, the video was divided into frames. Shown in Figure 3.13.

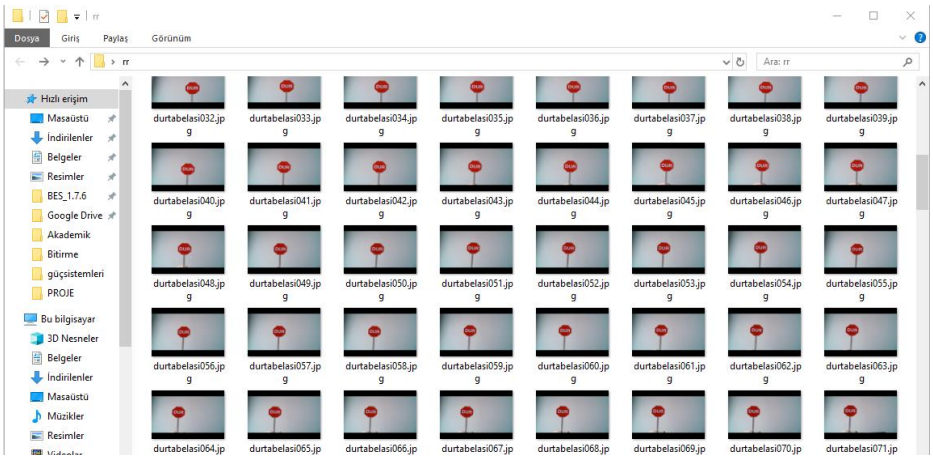


Figure 3. 13

After this step, the label process required for YOLOv3 modeling was started. The "YOLO-Annotation-Tool-master" program was used to do this. Objective here is to determine the coordinates through pixels by taking the target object in the picture into the frame. Thus, the pixel coordinates on each picture will be written to a ".txt" file as output and put into training for use in "YOLOv3" training. Shown in Figure 3.14.

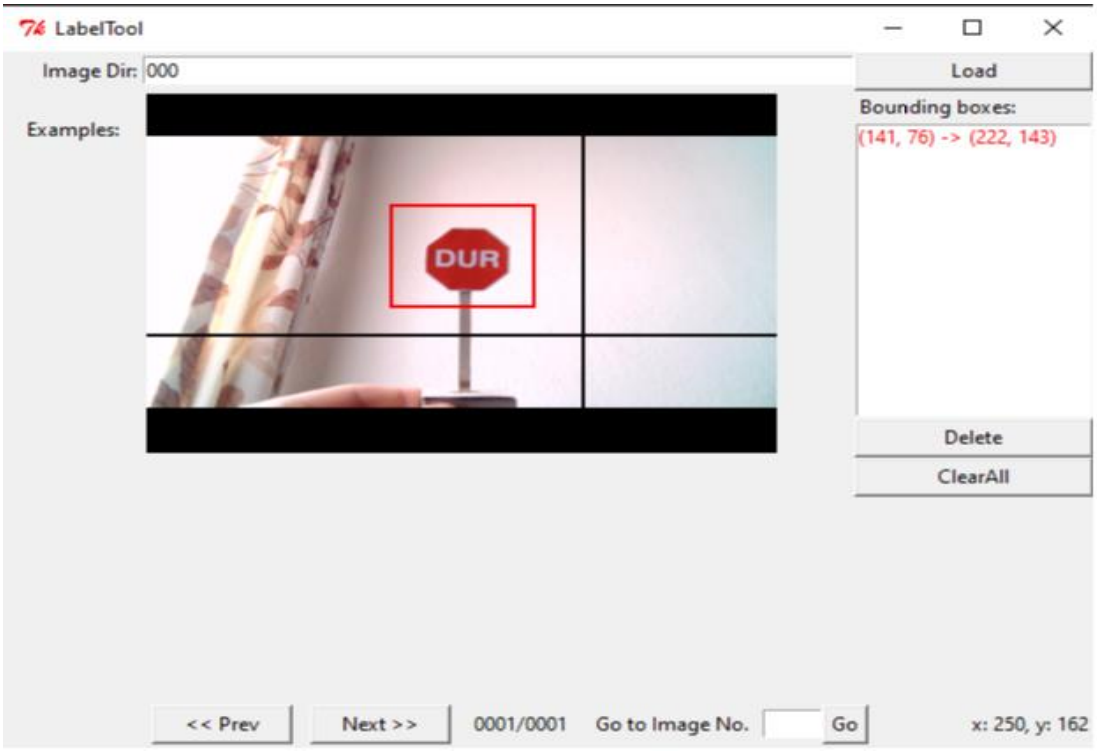


Figure 3. 14

As shown in the right section in the picture, the coordinates of the 'Bounding Boxes' are the coordinates of the red rectangle. Then, when i click next button, that program saves coordinates in txt file. It is shown in Figure 3.15.

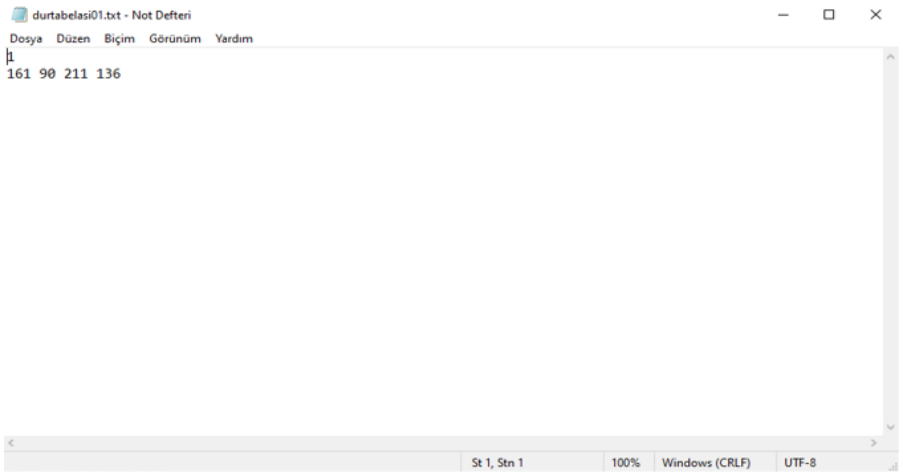


Figure 3. 15

We then 'convert' to bring the txt file to the format that YOLOv3 understands. It is shown in Figure 3.16.

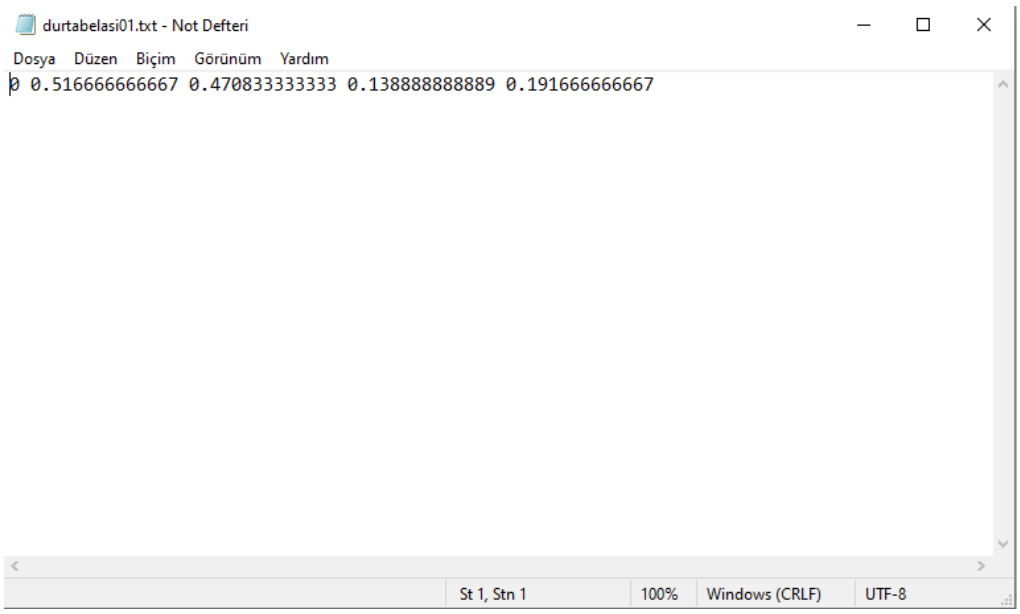


Figure 3. 16

Then, a train.txt file with shortcuts for this converted format of each image was collected, and a randomly generated test.txt file that will be used for testing was created. It is shown in Figure 3.17.

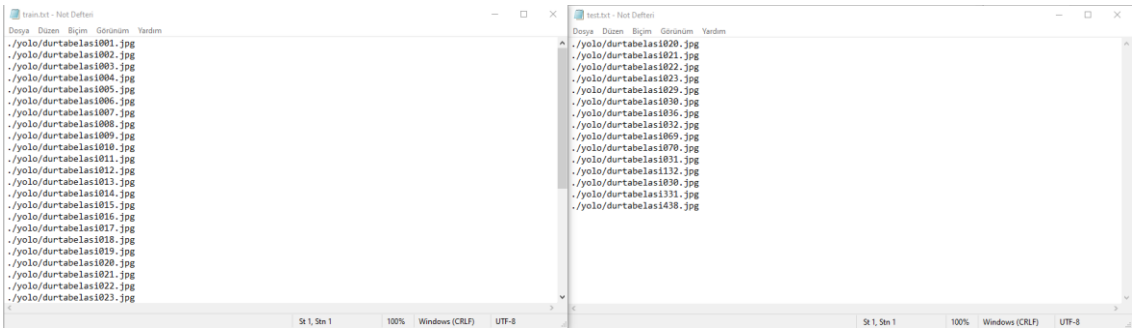


Figure 3. 17

Thus, the test.txt and train.txt files are ready for training. Other criteria continued in the continuation of the training. Now the file with the .names extension, our .cfg file, and .data files have been created. Saving with .LF extension. The file with the .names extension is used to name different object classes trained during the training process when recognized. It is shown in Figure 3.18.

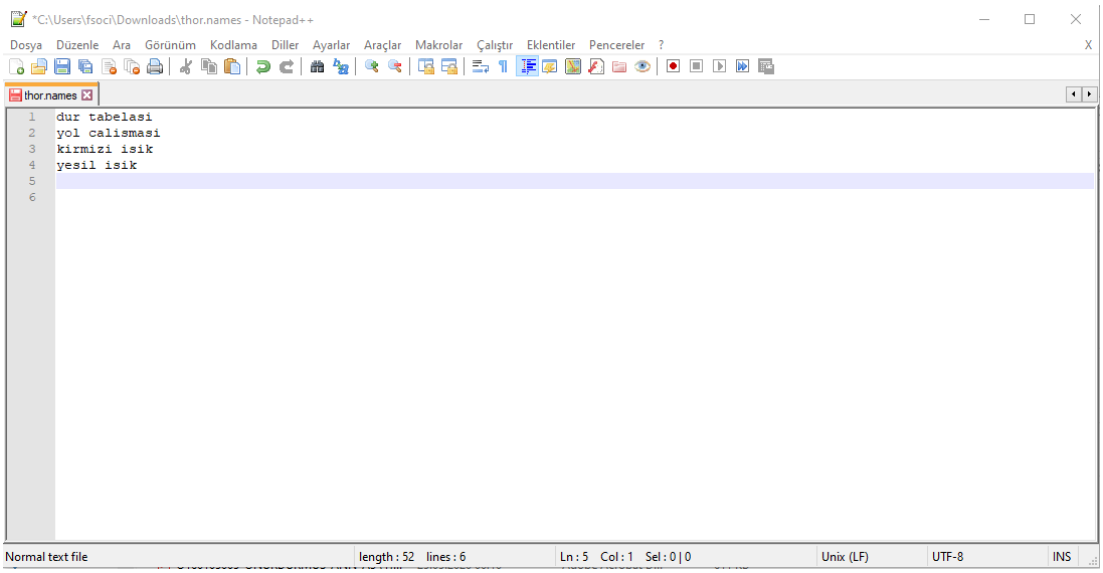


Figure 3. 18

The file with the .data extension is used to identify folder paths for the files that will be used during the training process. It is shown in Figure 3.19.

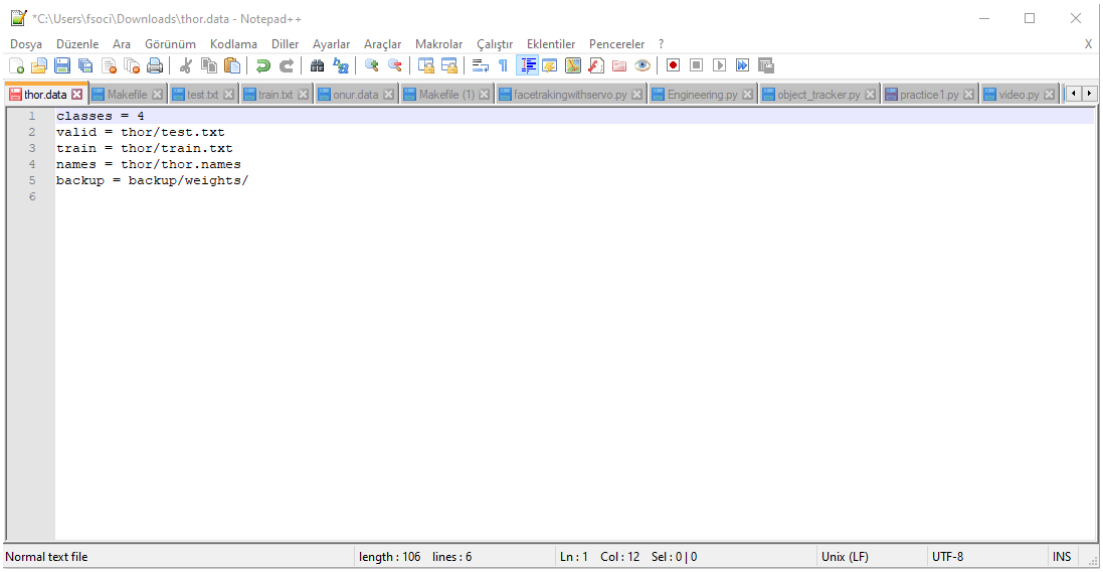


Figure 3. 19

The files are ready for training. At this stage, **'Google Colaboratory**' was used because there was not enough video card power for training. After entering Google Colabrotary, we click on the 'Notebook Settings' tab on the 'Edit' tab and select the 'GPU' tab. It is shown in Figure 3.20.

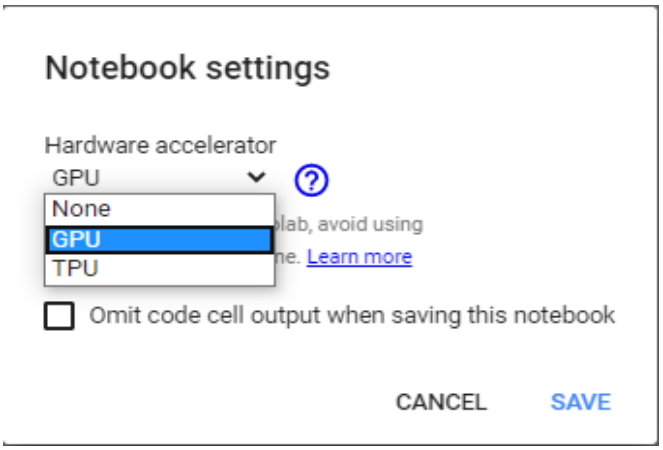


Figure 3. 20

Next, we press the 'Mount Drive' option to connect Google Drive to Google Colabrotary. It is shown in Figure 3.21.

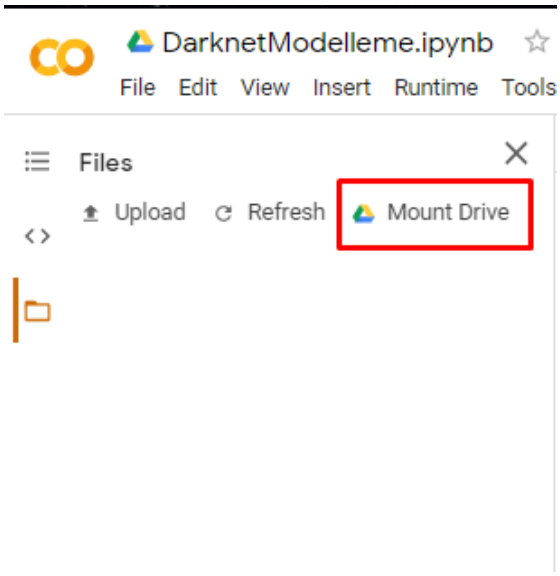


Figure 3. 21

The 'Darknet- Alexey' training folder was first taken via Google Colab. It is shown in Figure 3.22.

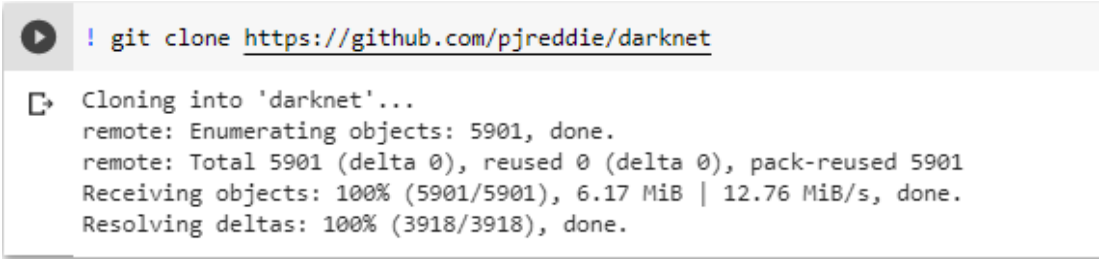


Figure 3. 22

Then for training, the values in the red tiles were converted to '1' in the 'MakeFile' file in the 'Darknet-Alexey' training kit, which was downloaded for image processing, gpu use and use of neural networks. It is shown in Figure 3.23.

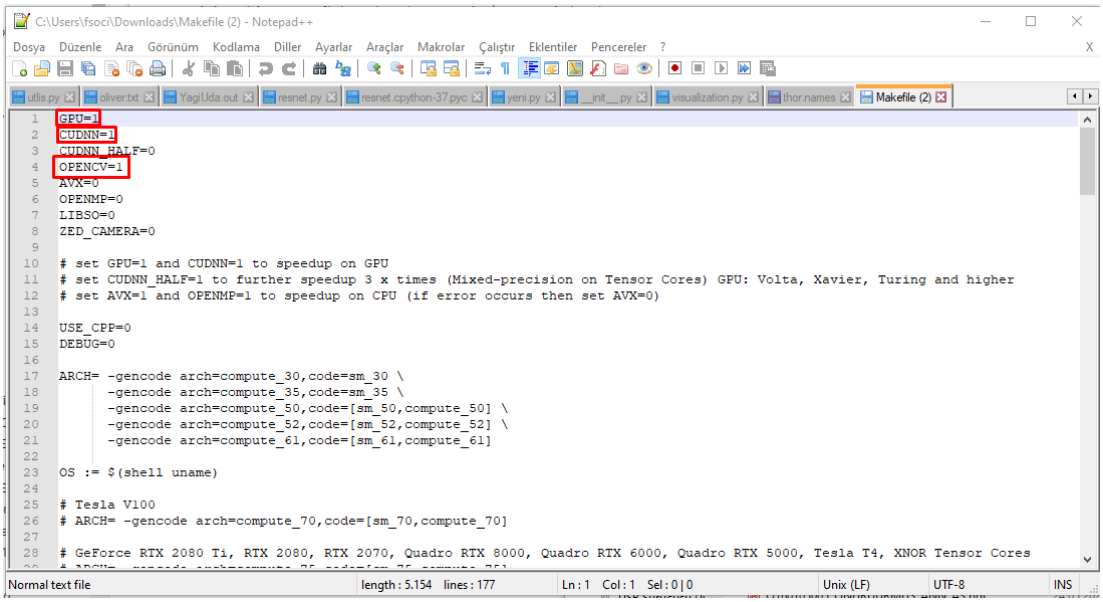


Figure 3. 23

For the modelling to be used in the training, 'darknet53', which is highly optimized and high accuracy, was used. It is shown in Figure 3.24.

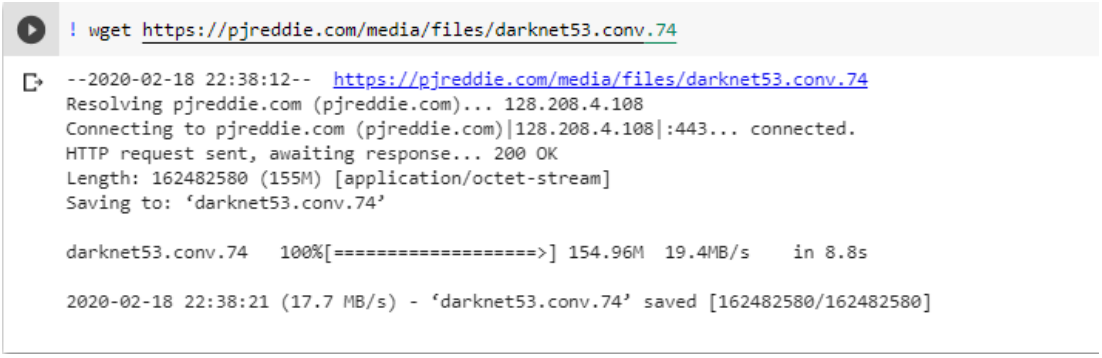


Figure 3. 24

And the file with the .cfg extension to be used was yolov3tiny.cfg. This is due to the fact that fps is faster in terms of performance, even though it has less accuracy than other CFG files. The edited MakeFile file and the training folder were compiled. It is shown in Figure 3.25.

```
%cd /content/drive/My Drive/DarknetEgitim/darknet
!ls
!make clean
!make
```

Figure 3. 25

The model is now ready to be trained. It is shown in Figure 3.26 and 3.27.

```
! ./darknet detector train yolo/thor.data yolo/thor.cfg darknet53.conv.74 dont_show
```

```
thor
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16  0.150 BFLOPs
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32  0.399 BFLOPs
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64  0.399 BFLOPs
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128  0.399 BFLOPs
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256  0.399 BFLOPs
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
   13 conv    256  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 256  0.089 BFLOPs
   14 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   15 conv     27  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x  27  0.005 BFLOPs
   16 yolo
   17 route  13
   18 conv    128  1 x 1 / 1    13 x  13 x 256   ->    13 x  13 x 128  0.011 BFLOPs
   19 upsample            2x    13 x  13 x 128   ->    26 x  26 x 128
   20 route  19 8
   21 conv    256  3 x 3 / 1    26 x  26 x 384   ->    26 x  26 x 256  1.196 BFLOPs
   22 conv     27  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x  27  0.009 BFLOPs
   23 yolo
Loading weights from darknet53.conv.74...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
```

Figure 3. 26

```
12254: 0.968226, 0.624477 avg loss, 0.000004 rate, 7.807458 seconds, 16256 images
 Loaded: 0.000056 seconds
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.462031, GIOU: 0.429511), Class: 0.48
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj: -
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj:
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.546962, GIOU: 0.476411), Class: 0.63
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj: -
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj:
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.467262, GIOU: 0.445301), Class: 0.61
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.427156, GIOU: 0.427156), Class: 0.80
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj:
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.418096, GIOU: 0.382331), Class: 0.61
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj: -
 v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: -nan, GIOU: -nan), Class: -nan, Obj:
```

Figure 3. 27

The value to be noted in the completion of the training is 'avg loss'. When the Average Loss value drops to about 0.6, the training has been terminated. It is shown in Figure 3.28.

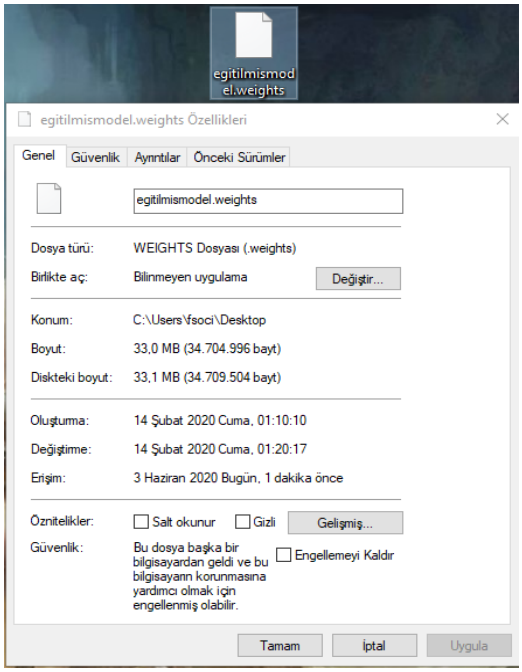


Figure 3.28

Then, the trained model was saved to enter the test stages after the required number of iterations. It was then operated with the code found inside the opencv library.

The training model to be tested in Opencv's 'dnn' mode is entered as a parameter in the .cfg, .names, files, code that is involved in the creation. It is shown in Figure 3.29.

```
test.py
Darknet-Test > test.py > ...
1   import cv2
2   import numpy as np
3   import time
4
5   # Load Yolo
6   net = cv2.dnn.readNet("egitilmismodel.weights", "thor.cfg")
7   classes = []
8
9   with open("thor.names", "r") as f:
10      classes = [line.strip() for line in f.readlines()]
11  layer_names = net.getLayerNames()
12  output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
13  colors = np.random.uniform(0, 255, size=(len(classes), 3))
14
15  cap = cv2.VideoCapture(0)
16
```

Figure 3.29

The code is ready to run in real time.

Test results shown in Figure 3.30, Figure 3.31, Figure 3.32, Figure 3.33

1)

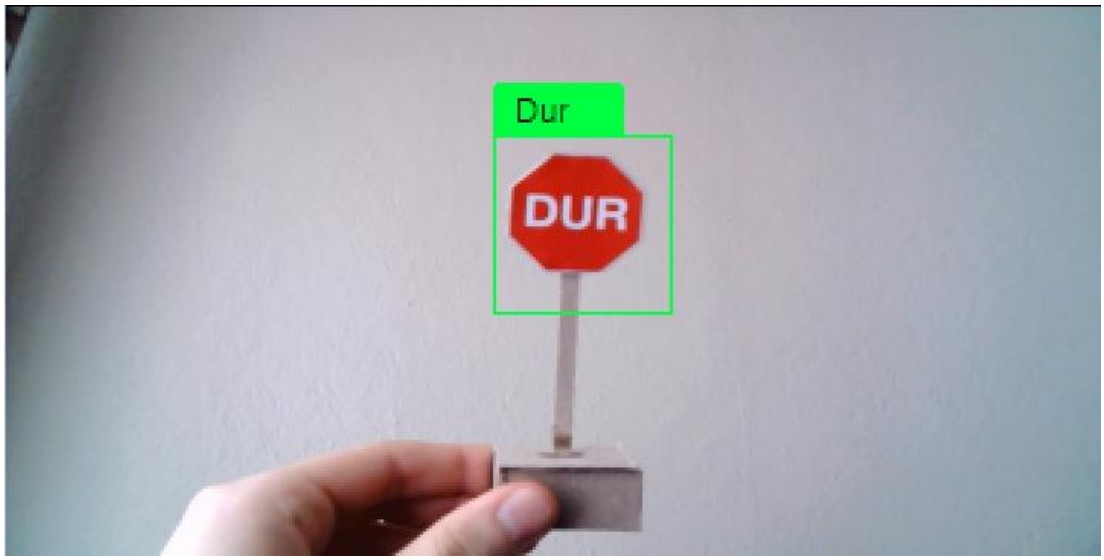


Figure 3. 30

2)

Figure 3. 31

3)

Figure 3. 32

4)

Figure 3. 33

RESULT

Unfortunately, we could not actively use the selected and built YOLOv3 modeling in the autonomous car. The reason is the Raspberry Pi Model 3 B + card, which is the basic mechanism of the autonomous car, was insufficient in real-time image processing. Very low FPS values (1.1-1.3 FPS) were obtained and a healthy study could not be achieved.

4. HAARCASCADE AND OBJECT DETECTION

4.1 What is the Haarcascade Classificator and It's Operating Logic

After learned of the main Opencv algorithims, talk about detection of the objects and following them in a photos. You can find it in the Opencv library, which name is Haarcascade Classificator. Haarcascade Classificator constituded by "Paul Viola" and "Micheal Jones" to resolve this problem. In the simplest form the Haarcascade Classificator is that, the objects introduce in the computer, which we want to detect of them. After that, this object is tried to find out by searching on the similar frames or video frames.

For the training, the code needs some positive photos, whose have the wished object and code needs some negative photos, whose have not wished object. The code is produced some target values by checking sum of the black pixels values with sum of the white pixels values using with the features, which are defined as in the Figure 4.1 to searching inside the positive pictures. [28]

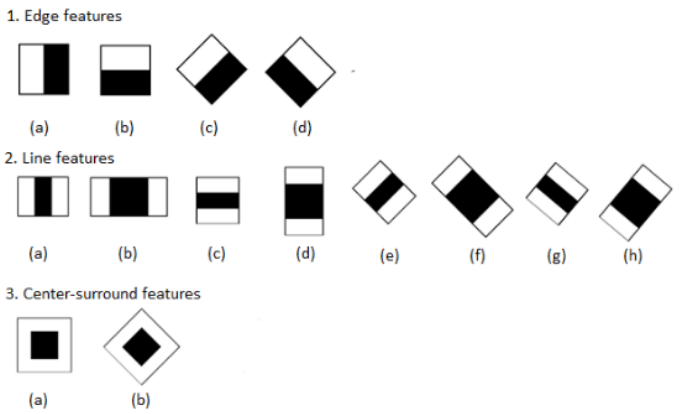


Figure 4. 1 Features

These frames called feature are called weak classifiers. The reason of that they are not a true classificator shape of unaided. An object has many weak classifiers and it means that, the wanted object is there at the point of the sum of weak classifiers. The simplest working form of the classifier is as below**.**

The features are combed on the positive photos as follows.

Figure 4. 2 Combing of the Photo

It can say that brightness ratio of the cheeks is more than brightness ratio of the nose in above figure. At the same time, there is same thing that the eyes place is more black than showing the white place.

Figure 4. 3

The values are produced with passing over the photos, whose have the wanted object. For instance, many dark and luminous features will be created in the areas, such as mouth, nose, forehead, hair in face combing. The target values are created by each of these. And this process is repeated at other stages by changing the frame sizes. There is no wanted object in the negative photos, so many without using features are sifted in this situation. The using features are defined with choosing the object in positive photos. It is very significant that there are many positive and negative photos to attain the expected results.[28]

The speed is too significant at the real time image processing. In the Haarcascade Classifier, the images are taken the integral intead of calculating sum of the pixel values one by one, so it will be removed the large processing power from the computer.

Also, the cuopling areas are combed as they have been defined before instead of combing of each features again and again during the detection of object, so it will be removed the large processing power from the computer. There are some values to take referances of classifier, whose are “min hit rate” and “max false alarm rate”. The classifier works to arrive these values at each training epoch.

After the training, the code will be created a file extension with .xml and this file is used by the Opencv library for object detection.

4.2 Why Did We Use the Haarcascade Classificator ?

There are many algotirhms for the object detection, such as Yolo, R-CNN, SSD. These algorithms are given good accuracy rate. We had decided to use the Yolov3 algorithm as it has been mentioned before, but nevertheless we had taken too low FPS value, which value is “1”. The FPS means that, the image transferred on the screen at one second. As we know, Yolov3 algorithm is used the CNN construction and this construction has many steps, such as convolution layers, max pooling, hidden layers. Generally the deep learning algorithms are used with GPU processor to take good results but our project had a CPU instead of the GPU. Taking into account all of these, we was came up against by a speed problem. Therefore, Fps was very significant value for object detection and this value did not sufficient for real time object detection. It might be amount of 15. Comparison of the Object Detection Algorithms are shown in Figure 4.4.

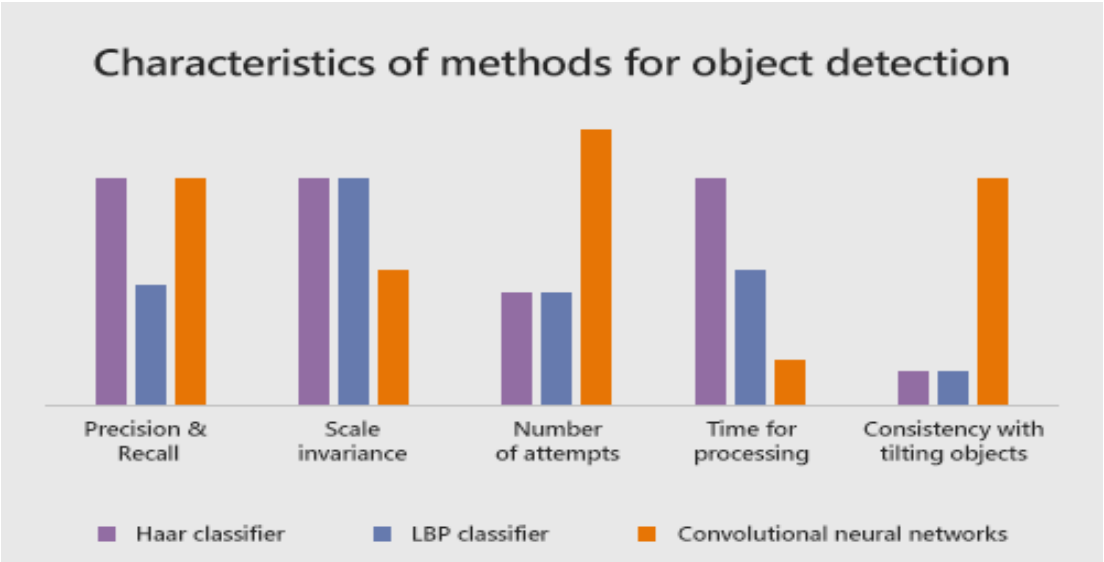


Figure 4. 4 Comparison of the Object Detection Algorithms

Using the Haarcascade algorithim was finded acceptable because of these reasons, whose had given in the Figure x. The processing time of Haarcascade Classifier is quicker than the other deep learning algorithms so it was decided to use for real time object detection in this project.

5. CONSTITUTION OF HAARCASCADE SOFTWARE

5.1 Data Collection and Separating Datas

We can detect many objects using by Haarcascade Classifier, such as face, car, hand. As ıt has been mentioned, the code is needed some positive and negative photos for detections. If the code has many photos, the results will be good evenly. The positive photos have the wanted objects. If these photos has different brightness and different angles, the results can be more effective. There might be thousands of datas for a good classifier but ıt can be constituted less datas. There are 4 objects for detection in this project, which are “red light”, “green light”, “stop traffic sign” and “roadwork traffic sign”. The code is consisted amount of 850 positive photos and 400 negative photos for each object forwhy the code is created just one .xml file.

Principally, the folder was constituted, which name is “dasar_haartrain”. The .xml file will be constituted in this folder and it has positive and negative folders for positive and negative photos.[28] This step is shown in Figure 5.1.

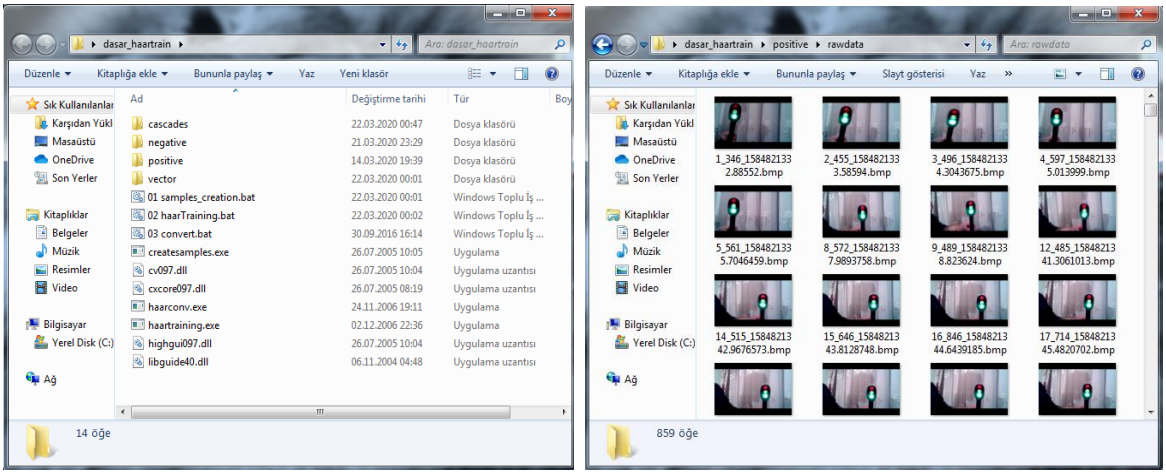


Figure 5. 1

5.2 Preparing for Trainig

At this stage, the each photo was resized for less overhead. To make this process, the app was used to resize, which name is “objectmaker.exe”. This app enable to determine the coordinates of wanted object lines in photo. This step is shown in Figure 5.2.

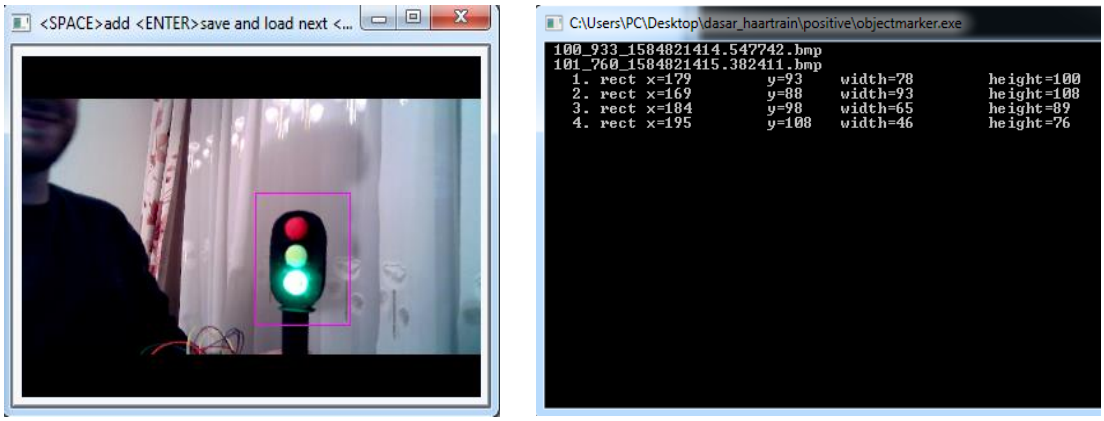


Figure 5. 2

As it is seen above, the pink rectangle was drawn using by “object maker.exe” to take less image area. Thus, it is caused less training time. In the right side of the picture, there are some coordinate values, which was producted by pink rectangle. Each photo is the 2-D and it can be shown “x” and “y” coordinates. “X” values are represented the width of photo and “y” values are represented the height. The “rect x” and “rect y” values are upper left-hand corner coordinate of pink rectangle. “Width” is the rectangle’s width and “height” is the rectangle’s height.

After this process, there was constituted a “info.txt” file, which has the file paths and the coordinates for the training. Additionally, there was constituted a value preceded by coordinates, which value is equal to “1”. It means that how many wanted object are in photo. If the photo has more than one wanted object, this value can be that how many wanted objects are at this photo. Photos will be cropped with a opencv function according to the coordinates values, which were given in the left side of “info.txt” and the process will be continued via these cropped photos.[28]

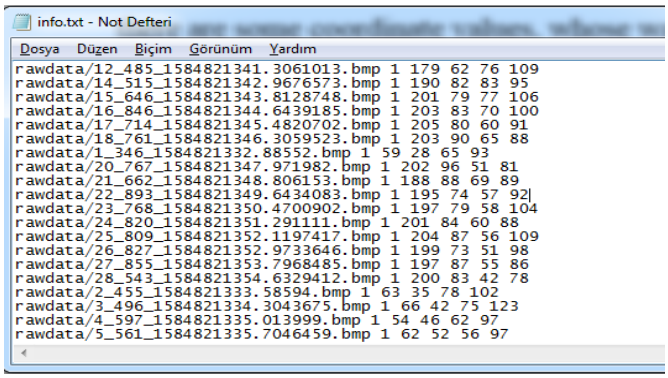


Figure 5. 3 File Path and Coordinates of Positive Photos

As it has been mentioned before, the code needs to negative photos, which have not the wanted object. There is a folder named “negative”, which has amount of 400 negative photos. This step is shown in Figure 5.4.

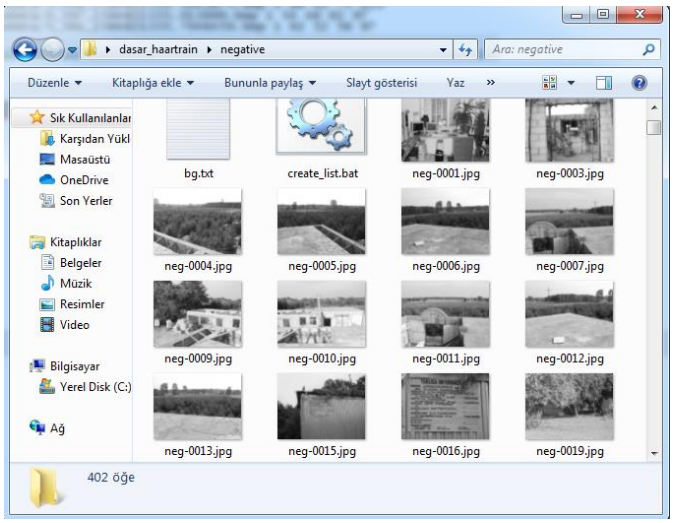


Figure 5. 4 Negative Photos

There is same images extensions for negative photos as in positive photos all negative photos should be same size. All extensions of negative photos was located one under the other in the “bg.txt”. Barely it has not coordinates values as in positive photos. This step is shown in Figure 5.5.

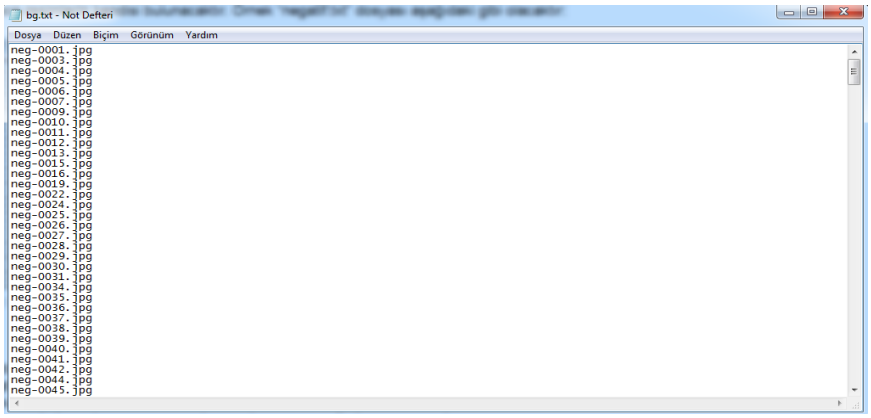


Figure 5. 5 File Path of Negative Photos

The classifier needs thousands of positive photo to detect effectively. The classifier will be given good results which is involved in the event 2000 positive photos and 800 negative photos because the classifier runs the positive photos above of the negative photos.

The classifier requires convert the photos as gray scale to use at the Haarcascade Classifier and it requires to create a vector file with croped the peace which involved the wanted object. There is one file in the “dasar_haartrain”, which name is “createsamples.exe”. This file takes the txt file with the name and address path of the positive images as parameters, the number of positive images, the name of the vector file to be created, and the ratio values that the samples will be sized. There are two options to send these parameter values at the “opencv_createsamples.exe” file. It can run with command window or creating a “.bat” file, which uses command window’s codes. Creating a “.bat” file is more sensible because the code will be repeatedly used training of the classifier. The “dasar_haartrain” folder has this “.bat” file and ıt has many values, such as “.vec”, “-num”, “-w –h”, “-info”.[28] This step is shown in Figure 5.6.

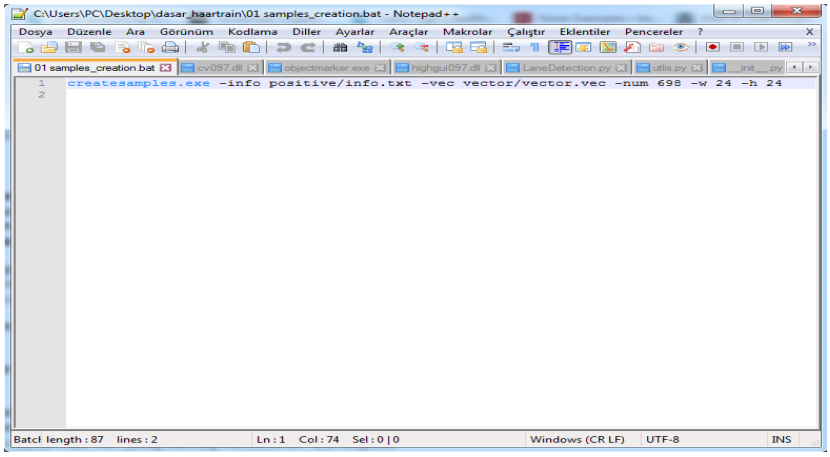


Figure 5. 6 Vector Creation File

*-info positive/info.txt: it ensures that take the txt file as we constitude.

*-num: It is number of positive photos.

*-vec: It is name of vector file, which will be constituted.

*-w 24 –h 24: This part is cropped photo again, which cropped before. The reason of this process decreasing the process time. This step is shown in Figure 5.7.

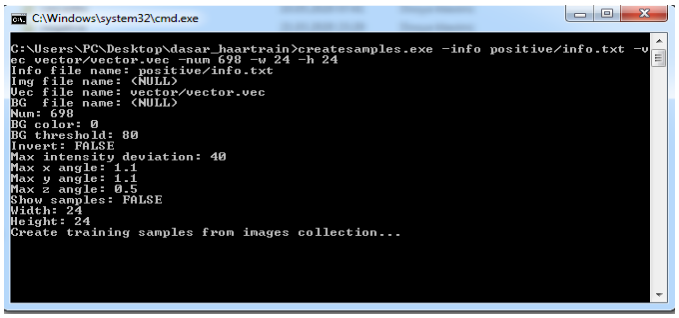


Figure 5. 7

When we double clicked the .bat file as we constituted before, this window will be created and it will be closed automatically.

5.3 Training Process of Haarcascade Classifier

This training can make one .exe file, which is at the opencv library. This file's name is "haartraining.exe". This exe file works to arrive defined values, which are "minhitrate" and "maxfalsealarmrate" amount of defined processing time. The positive photos are combed with the different frames at the each process. When it arrived the defined process value, the ".xml" file will be created to find the objects. The code needs a .bat file which has many parameters, such as "-data Cascade", "-vec vel.vec", "-bg negatif.txt" , "-numPos", "-numNeg", "numStages", "-minHitRate", "-maxFalseAlarmRate", "-mem", "-w 24 –h 24" in this process.[28] This step is shown in Figure 5.8.

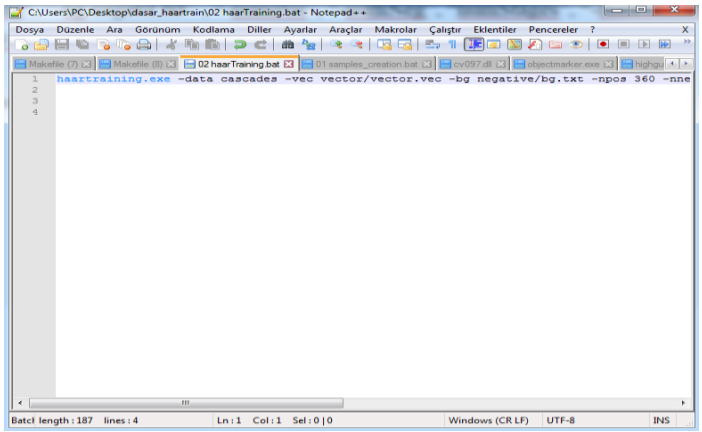


Figure 5. 8 HaarTraining .bat File

*-data Cascade: In this section, the name of the xml file that the classifier obtains at each stage, the xml file where the parameters are kept, and the folder where the last xml file was constituted is created.

*-vel el.vec: This section is defined vector file, which obtained with positive photos.

*-bg negatif.txt: It shows us the background. The code is taken the negatif .txt file, which of the positive photos will be processed.

-numPos: This section uses to express how many positive photos will be used at the training stages. This section is not the same sum of the positive photos number in vector file because vector file will be processed the positive photos amount of negative photos at the background. Therefore, a formula was developed to use this section that numPos<=(Number of Postitve Samples in the vector file-Nunber of Negative Samples)/(1+(Stage Number-1)(1-minhitrate))). The numPos value was defined according to this formula.[28] This step is shown in Figure 5.9.

*-numNeg: This section is defined how many negative photos used for this classifier.

*-numStages: This section is indicated the how many stage classifier invloves to reach the result.

*-minHitRate: It is given hit rate, which is used on each stage and generally it is taken amount of 0.995.

*-maxFalseAlarmRate: It is given acceptable fault rate while recognizing objects. If this rate is too large, the classiffier can detects without wanted objects. If it is too small, the classifier will be not found when the wanted failing perfect.

*-mem: It mains the memory size for training.

*-w 24 –h 24: This is dimension of the vector file.

After arranged the parameters, the code will be started after double clicked .bat file.

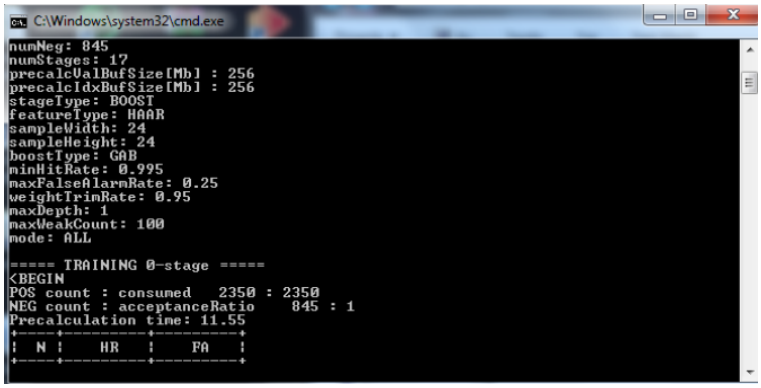


Figure 5. 9 Training of Haarcascade Classifier

After completed the training section, the classifier will be constituted a ".xml" file within the "dasr_haartrain" folder. Residual the code can define the wanted objects under favour of this .xml file. This step is shown in Figure 5.10.

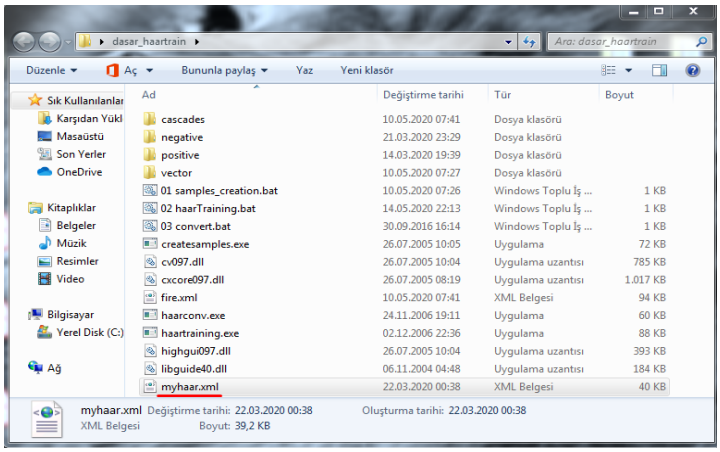


Figure 5. 10 Creating of .xml File

5.4 Get Used of the .xml File and Procure of the Results

After the .xml file was constituted, it will be used by the code which has the opencv algorithms. Code will be created a rectangle when it was detected a wanted object. This rectangle's coordinates are involved of wanted object coordinates so the user can follow the wanted object coordinates at the real time. This code has 4 different .xml files for each wanted object. The code is able to run these files at the same time for 4 different wanted objects in the each circle. When the car sees the red light with the help of the camera, it waits until the traffic light turns green. When traffic light turns the green, car begins to move through the road. When car sees the stop sign, it stops. The results are shown in Figure 11.

Figure 5. 11

6. LINE DETECTION

6.1 What is the Line Detection and It's Popular Detectors?

In image processing, **line detection** is an algorithm that takes a collection of edge points and finds all the lines on which these edge points lie. There are many methods fort he line detection, such as hough transform, convolution-based technique, deep learning algorithms, discrete line parameterization, Ross Kippenbrock method. These methods can use for the different tasks. In our Project, we use these method for the lane tracking.

6.1.1 Hough Transform

The Hough transform is often used in computer vision or image processing field to find imperfect instances of geometrical shapes (lines, circles, etc) [9][10]. These shapes are found by carrying out a voting process on a parameter space. In this space, each point corresponds to a distinct object. At the end, local maxima identify the most probable shapes. For example, a straight line y = mx + b can be parameterized by the point(m,b) . We iterate over all pixels in the input image and through all allowed values for one of the parameters. Having assigned these values, we can solve for the second parameter and then cast a vote for this line in the parameter space (also called the accumulator). The drawback to this approach is that vertical lines give rise to unbounded values for the parameters. For computational reasons, it is more practical to parameterize the line in terms of its polar coordinates ρ and θ [10][11][12].

The parameter ρ represents the distance between the line and the origin while θ represents the angle of the vector from the origin to the closest point on the line. Using these parameters the equation of a straight line becomes:

$$\rho = y \sin\theta + x \cos\theta$$

The idea of the algorithm is to look for lines having a particular orientation θ , then iterate over all image points (x,y) and compute the ρ parameter, that is the distance from origin to that line. We can keep an accumulator to count the number of points that have been reported for the line (ρ,θ) and then look at local maxima to find the best fitting lines [5][6][7][8].

The Hough transform is very efficient on random noised images and is able to detect even heavily damaged lines. This is because the mathematical model used is not

strictly followed, but rather applied to generate statistical indicators which are interpreted relative to each other and the most probable objects are identified. Interpreted relative to each other and the most probable objects are identified.[9][13][14][15]

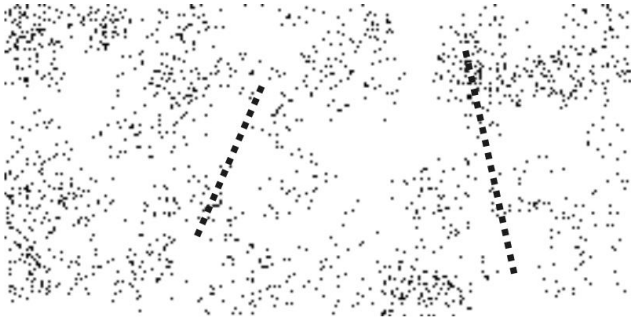


Figure 6. 1 Input Image

Figure 6. 2 Output Using Standard (r ,q)Hough

Transform

6.1.2 Discrete Line Parameterization

The Discrete Line Parameterization. This takes into account the fact that in a digital image we only have a restricted set of possible lines. Each line can be represented by two parameters *Xo* and dx (Fig. x). It is a close analogy to the previous (ρ,θ) parameterization: *Xo* and ρ both reflect offsets while dx and θ represent angles and orientation. Based on this we ca proceed similarly to Hough transform and iterate over all points in the image and vote the discrete lines that pass through them.

The advantage of this method comes from the fact that it uses only integer numbers, thus the unknown parameter can be coined more accurately, discarding some of the false lines that were being picked by the more loosely Hough transform method.

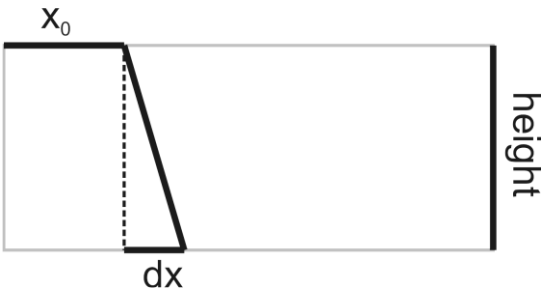


Figure 6. 3 A discrete line representation

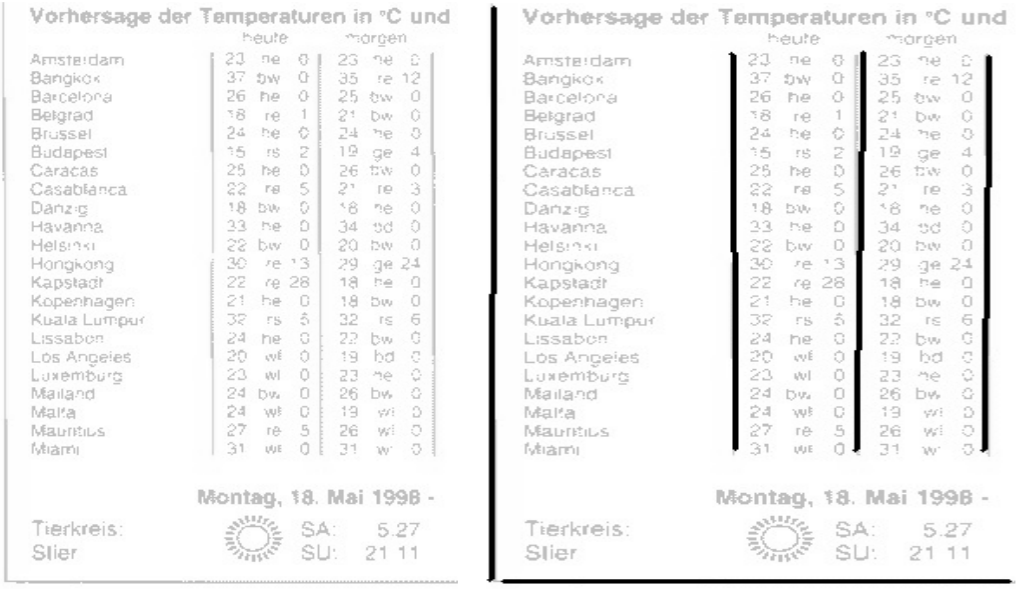


Figure 6. 4

6.1.3 Convolution Base Technique

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of ‘multiplying together’ two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values. In an image processing context, one of the input arrays is normally just a gray level image. The second array is usually much smaller, and is also twodimensional (although it may be just a single pixel thick), and is known as the kernel. The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of theimage. (Note that implementations differ in what they do at the edges of images, as

explained below.) Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together.

The line detection operator consists of a convolution kernel tuned to detect the presence of lines of a particular width n, at a particular orientation θ. Figure 1 shows a collection of four such kernels, which each respond to lines of single pixel width at the particular orientation shown.

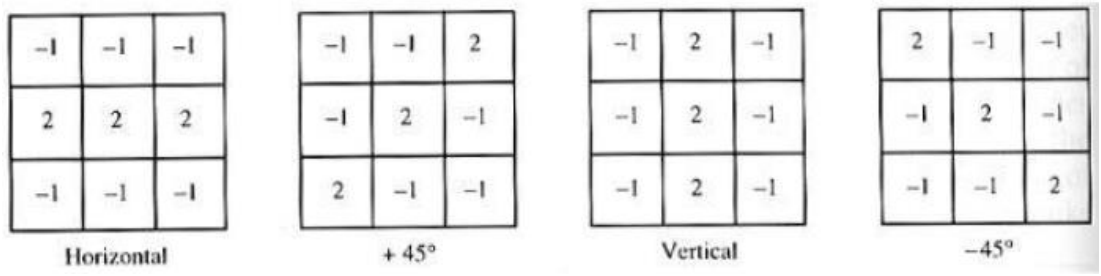


Figure 6. 5 Four line detection kernels which respond maximally to horizontal, vertical and oblique (+45 and - 45 degree) single pixel wide lines.

These masks above are tuned for light lines against a dark background, and would give a big negative response to dark lines against a light background. If you are only interested in detecting dark lines against a light background, then you should negate the mask values. Alternatively, you might be interested in either kind of line, in which case, you could take the absolute value of the convolution output. In the discussion and examples below, we will use the kernels above without an absolute value. Ifi denotes the response of kernel i, we can apply each of these kernels across an image, and for any particular point, if Ri > Rj for all j = i that point is more likely to contain a line whose orientation (and width) corresponds to that of kernel i. One usually thresholds Ri to eliminate weak lines corresponding to edges and other features with intensity gradients which have a different scale than the desired line width. In order to find complete lines, one must join together line fragments, e.g., with an edge tracking operator.

6.1.4 Fitting a straight line

Lawrence Roberts has proposed the Roberts edge detecttion technique for detecting the edges within an image in 1965. It is a simple and computationally. It efficient approach. measures the spatial gradient of an image. The pixel value at that point in the resultant image characterizes estimated absolute magnitude value of the spatial gradient of the inputted image at that point. It takes input image as gray scale image and produces edges involving in that image. The main disadvantages of this technique are that it can't detect that type of edges which are multiplies of 45 degrees and it is not symmetric. The Robert operator contains the pair of 2x2 convolution masks which are illustrated in Figure x. One mask is just to other rotated by 90 degrees. This is true because there are few single pixel width lines in this image, and therefore the detector is responding to the other high spatial frequency image features (i.e. edges, thick lines and noise). (Note that in the previous example, the image contained the feature that the kernel was tuned for and therefore we were able to threshold away the weaker kernel response to edges.) We could improve this result by increasing the width of the kernel or geometrically scaling the image. To fit a single straight line to data, we must fit

$$y = mx + h$$

where m is the gradient and h is the intercept with the y - axis. This is a very common fitting problem and the simplest is least squares fit. If we have n points yi, xi, being point on the line, then if we define the square error as

$$e^2 = \sum_{i=1}^{n} (y_i - (mx_i + h))^2$$

and we get the standard solution by minimizing e^2 by setting

$$\frac{\partial e^2}{\partial m} = 0 \text{and } \frac{\partial e^2}{\partial n} = 0$$

which has the effect of minimizing the vertical distance between the points and the line as shown in figure x

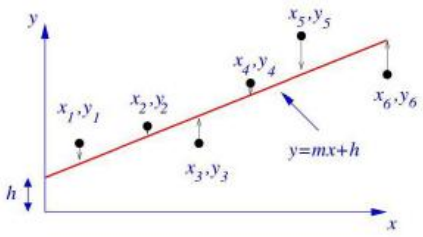


Figure 6. 6

The works very well for a single line, but if there is more than one line, things get rather more complicated and, as shown in figure 3, the simple least squares simply gives the best average line single line which is usually wrong. Least-Square only works if you have a single line, or are able to segment out a segment of the image that contains a single line. We need to look for something a lot more general than this.[16][18]

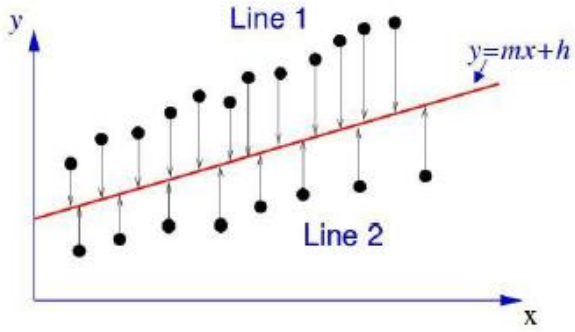


Figure 6. 7

Figure x. Least square fit two lines of
data point by a single line

6.1.5 Ross Kippenbrock Technique

This technique was developed by Ross Kippenbrock. Tenchnique is consisted four steps. These steps are shown in Figure 6.8.

1.Undistort
2.Warp
3.Isolate Lanes
4.Curve Fit
5.Final Image

Figure 6. 8

6.1.6 Removing distortion

As we know, cameras use lenses because of this the image is distorted. The software can fit more of surrounding into a smaller frame. The typical type of distortion is shown in Figure 6.9.

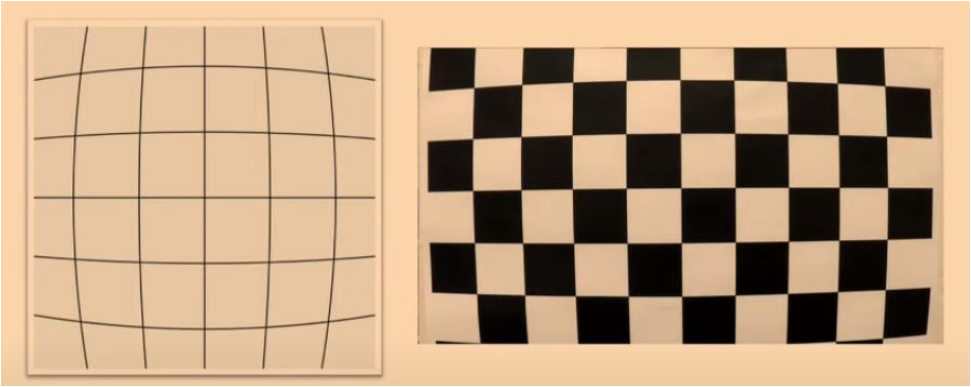


Figure 6. 9

The OpenCV has a number of functions that will help us do this. There were used pictures of chessboard at various perspective for the calibration. OpenCV finds those corners in the image using this fine chessboard corners function. If this returns that it is found those points then we add them to a list of these corner points. The software takes those image points that you found on the chessborad and it is going to return a set of distortion coefficient which will the transform the image based on like series of differrent distortions also there is a camera matrix so there is some linear algebra behind the scenes. The function's output is shown in Figure 6.10.

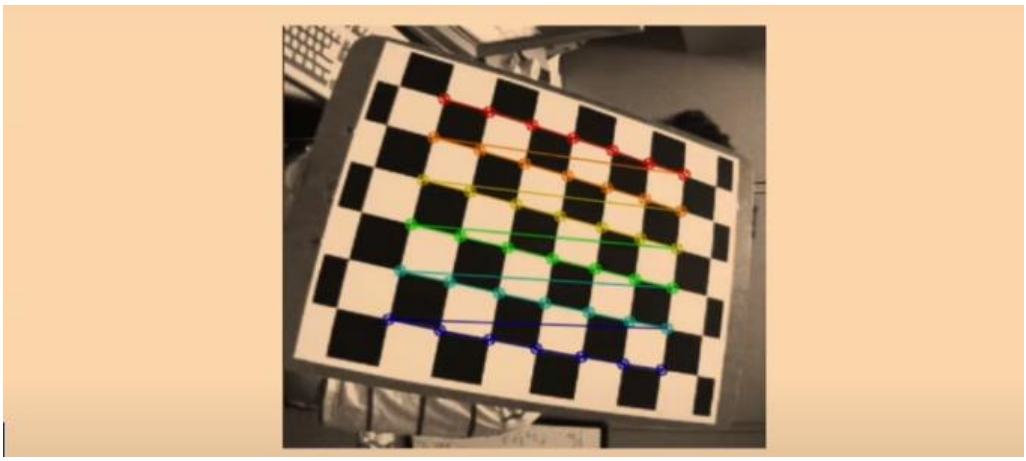


Figure 6. 10

6.1.7 Warping images

The software should be get bird's-eye view to get more healthy results. For fitting curves and finding lane lines it's going to be easier to find those lines if the camera looks down on subject instead of looking out on. Therefore, sofware have to has a sets of points from the original image so those are kind of lanes going off to infinity. These points are warped into straight lines so this is just a straight road and those lines should end up straight when the software is warping the image. The original image and warped points are shown in Figure 6.11.[21][23][27]

Figure 6. 11

Figure x. Original Image and Warped Points

The sofware has the perpective transform function which returns another matrix more linear algebra then software takes the outputs of that martix and apply this warp perspective to original image with using that matrix and the result of these steps is shown in Figure 6.12.

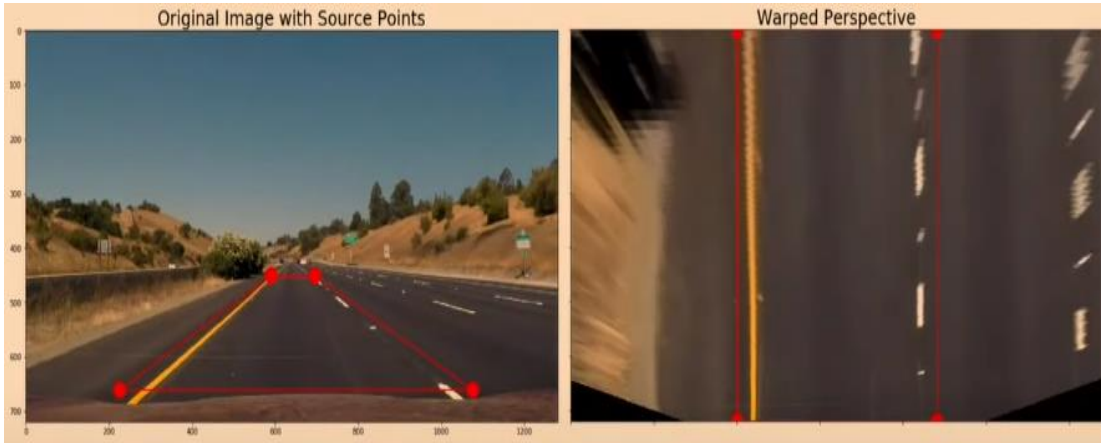


Figure 6. 12

6.1.8 Finding the lane lines

There is going to be two approach that the software will take one is color selection and the other one is going to be edge detection. The color selection and edge detection are shown in Figure 6.13.

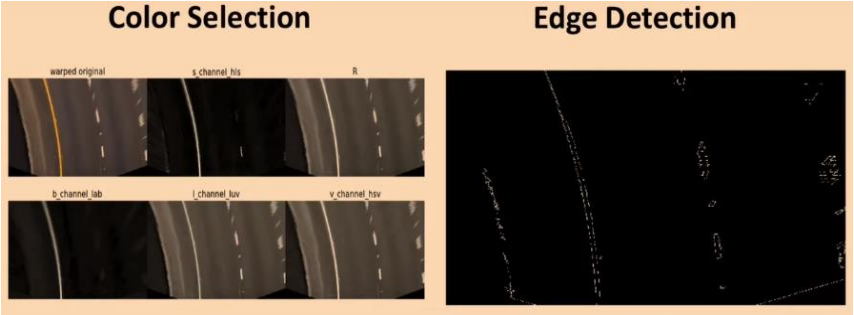


Figure 6. 13

Figure x. Color Selection and Edge Detection

For color selection the software is going to first kind of look at what an image inside of a computer so taking a look at the first ten pixels of an image just zooming on that it can see that it's just series of these little squares. The exaple is shown in Figure 6.14.

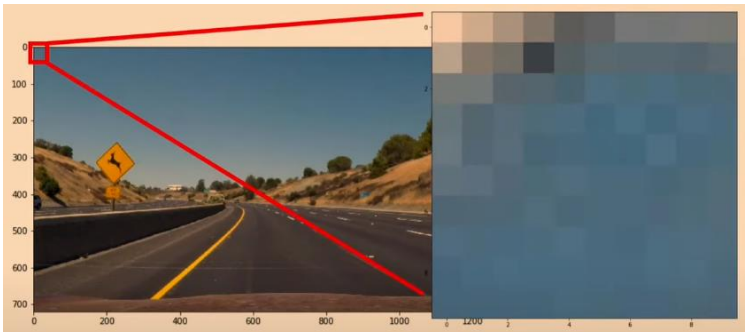


Figure 6. 14

Each squares have three values in a list. This do means something one of them is a red value green value and a blue value. These go from 0 to 255 in most images because they are 8-bit images. The max value can be 256 in a 8-bit. These values are shown in Figure 6.15.

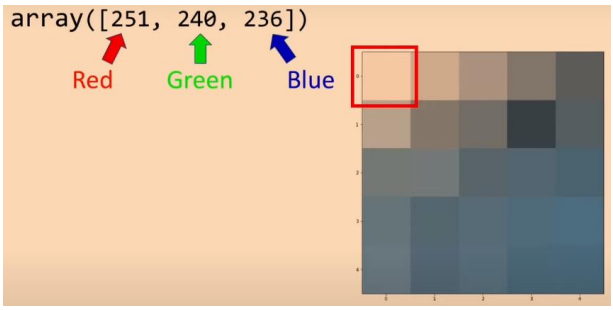


Figure 6. 15

Couple others are hue/saturation light and value color spaces so those they just take RGB colors and them into this other representatiton of that same image and then there is a number of other ones. This one is called lab it uses lightness and then there's two axes for the color and these are going to prove pretty hepful. These members are shown in Figure 6.16.

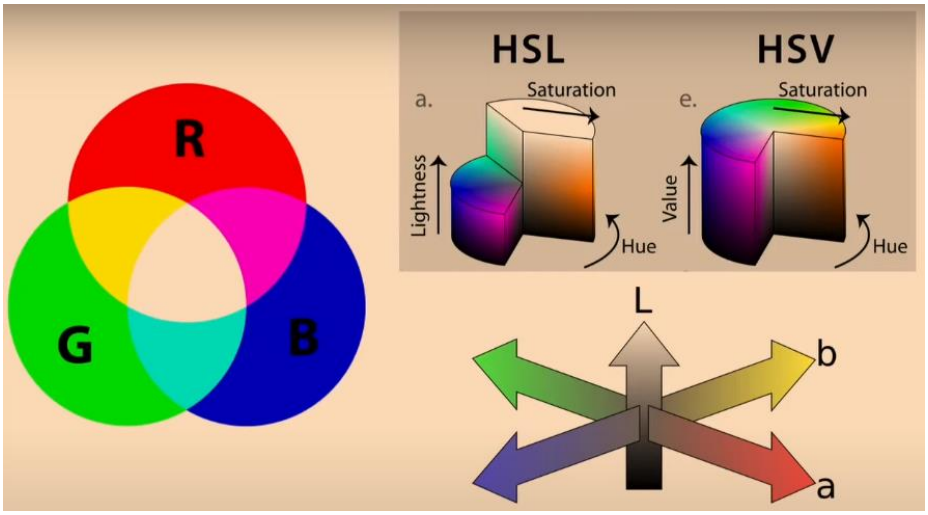


Figure 6. 16

Canny edge detection which just uses that same Sobel system but applies a couple nice tricks. That throws out pixels aren't part of the edge and then also thresholds the image to try to connect edges that weren't connected before so the software going to use to produce our final guess. The canny detection is shown in Figure 6.17.

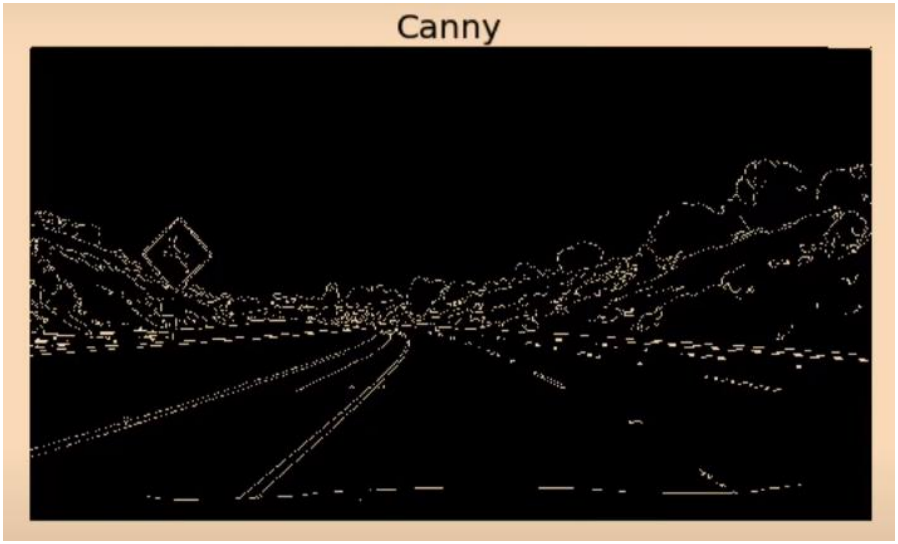


Figure 6. 17

6.1.9 Curve fitting

There are two methods for curve fitting. Method one is can be applied on any image and method two needs a previous fit. It runs quite a bit faster so the software tries to use method two or possible. If sofware does not find a fit, it kind of fall back to this first method which is little slower but little more robust. These two methods are shown in Figure 6.18.[21]

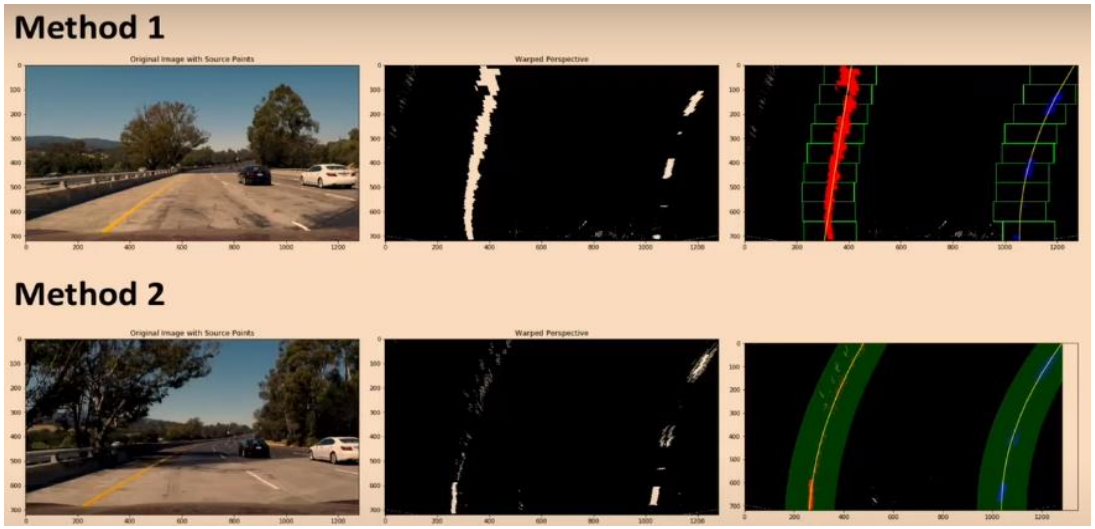


Figure 6. 18

For method one the software takes a histogram of the bottom half of the image and wherever that histogram peaks that the line starts. The y-value is just the sum of the pixels at x-location. And two maximums are where those lines start. This is a good indicator of how close it can be between a false value and a real value. In Figure 6.19

the lane that is to the right of ours and the sofware is detecting that two lines. Throwing out some of these extraneous points would be probably pretty useful fort his algorithm.

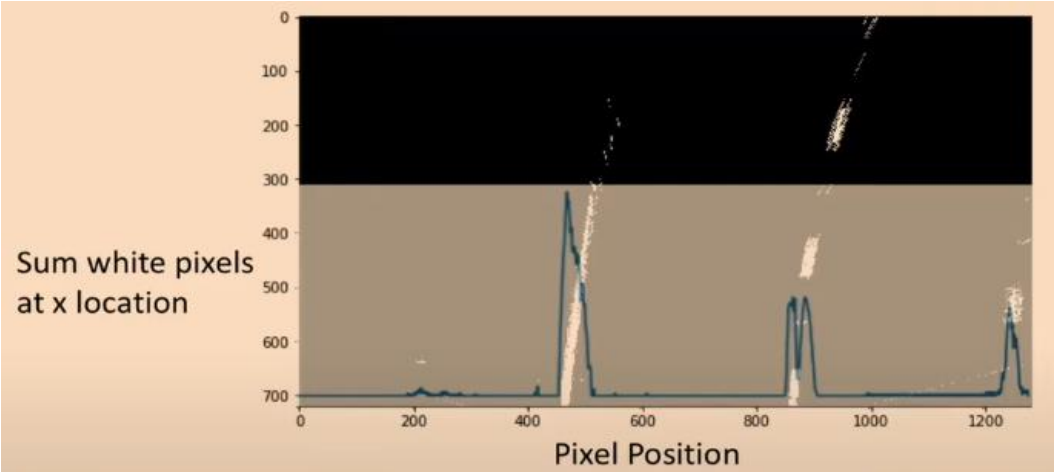


Figure 6. 19

Therefore, the sofware has left start and right start points. The software is going to draw a box around those starting points and any of those pixels from the binary image that fall in that box. The software puts them in a big list of x and y values. For the next one software will just add a box based on the average of previous line. This step is shown in Figure 6.20.[21][23]

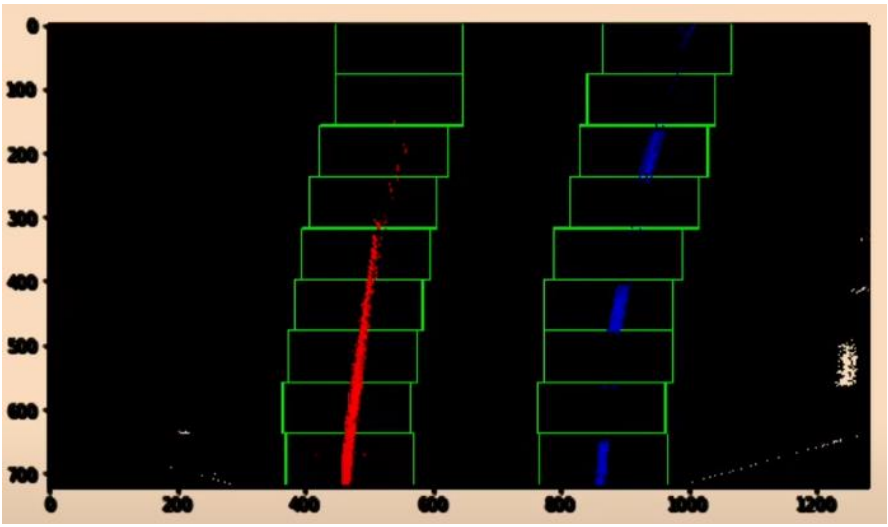


Figure 6. 20

The end of this step, the software has a big list of all the Y values and all the X values of the pixels. Then the software runs a numpy polyfit second-order polyfit for those pixels and that gives us the curve fit in pixel space. This curve fitting is shown in Figure 6.21.

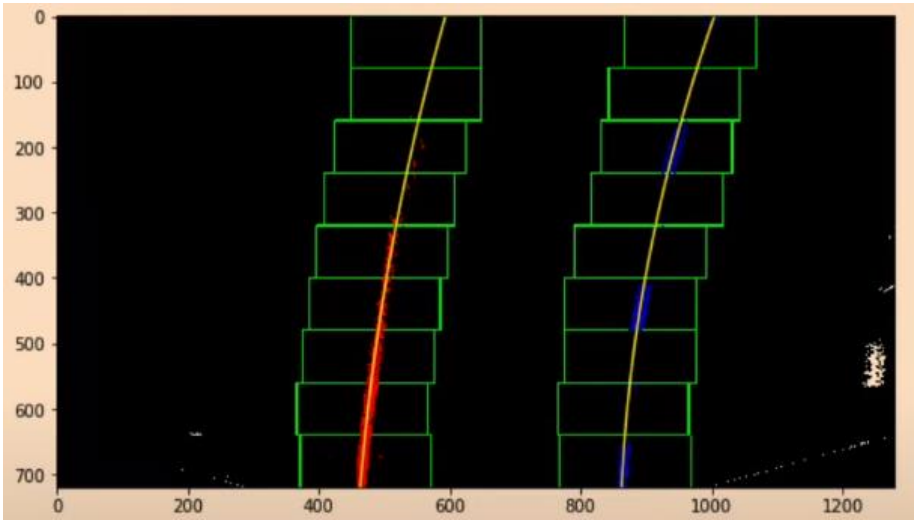


Figure 6. 21

The software has a radius of curveture so going to help the car to guess. The software is find and then calculates the radius of curvature. The calculating is shown in Figure 6.22.[21][23][27]

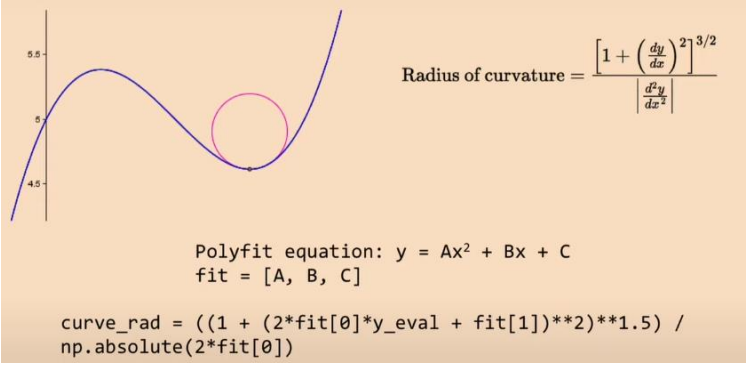


Figure 6. 22

The software were taken two curve fits and it found plot those on the image using polylines function and then fills in space between those lines with the green color. This drive area is shown in Figure 6.23.

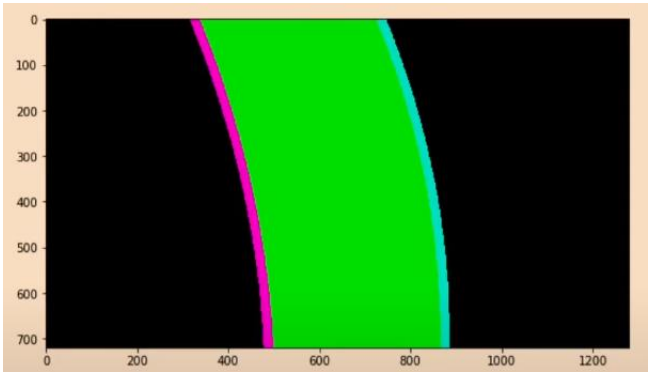


Figure 6. 23

At the end of these steps, the software needs a unwarped image because it has bird's eye view. To get a nice overlay, OpenCV is presented us a addWeighted function. The unwarped image is shown in Figure 6.24.

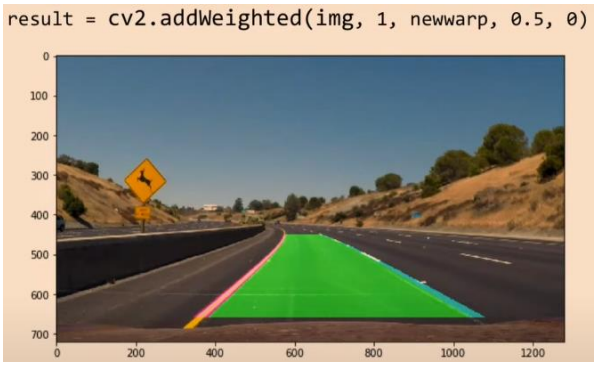


Figure 6. 24

Finally, the software is thrown some text. This put text function gives chance to print the radius of curveture and offset away from the right side of the line. The final image is shown in Figure 6.25.

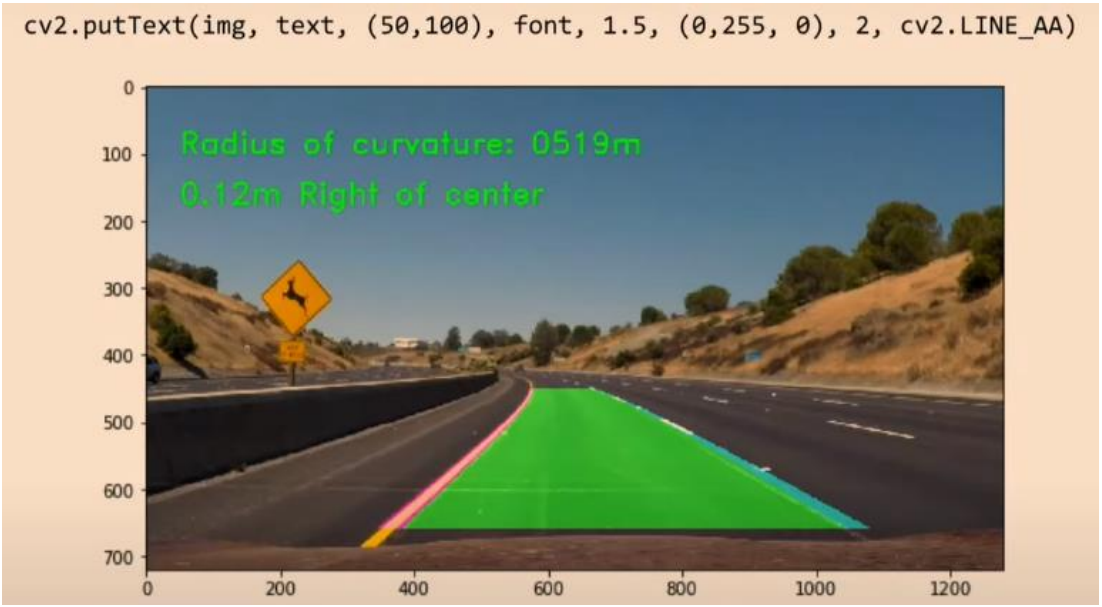


Figure 6. 25

6.2 Which Method Did We Use and Why Did We Use This Method?

In this project, we had many options ahead of us. The necessary criterions were considered, while the method selection was made. The main criterion is the accuracy. When we consider the accuracy, the Ross Kippenbrock method gave the best result against the other line detection methods. When we used the other methods, they were finding lines but results were not stable enough. The second important criterion is that

the method should not be affected by environmental conditions. When we consider this criterion, Ross Kippenbrock method did not affect by environmental conditions against the other line detection methods. This is because ross kippenbrock method has functions such as color filtering and edge sensing histogram synchronization. All criterions were considered and it has been decided to implement Ross Kippenbrock method.

7. CONSTITUTION OF CURVED LANE DETETCTION SOFTWARE

7.1 Distortion Correction

Distortions can constitute on the image due to the camera lens and these distortions should remove from the image for more healthy results. Distortions were prevented using by “undistort()” function. This function results are shown in Figure 7.1.

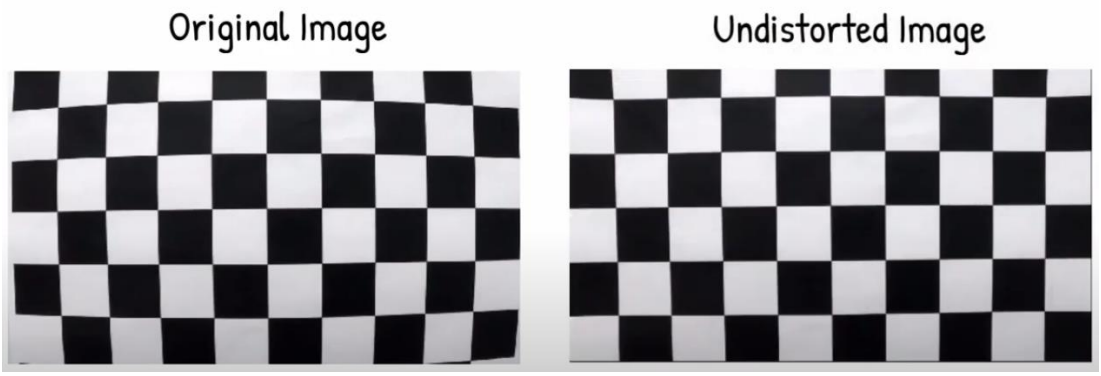


Figure 7. 1

7.2 Color Filter and Edge Detection

The software has white and yellow color filters for the lane colors. As we know, the road lines generally are white or yellow colors.

7.2.1 HSV color filter

RGB color space is the color space we use in general. Three main components are used in this color space. Image R, G, B i.e. red, green, blue color codes are defined on. Each pixel receives intermediate values based on these color codes. So we can get a color picture.

HSV color space defines color with HUE, Saturation, and Value. Although RGB also used a mixture of colors, The HSV also used color, saturation and brightness values. Saturation refers to the brightness of the color when determining the viability of the color. For example, if you want to The brightness value is zero while the color and saturation values for the black color in hsv space can take any between 0 and 255.

In white, the brightness value is 255. Accordingly, it is more convenient to use hsv color space when we want to distinguish an object of a specific color in any computerized vision/image processing application. Because instead of filters that we will use for threshold value in rgb color space, we can only determine the threshold value with the Hue component here. We can get clearer colors. In 1978, alvy was identified by Ray Smith to create a structure more similar to the human vision system than rgb space. As can be seen in Figure 1, the H value varies from 0-360°, while the self values of the colors change. However, if H is selected as constant and other values (S-V) are changed from 0-100, different saturation and brightness values of the same color are obtained. Because of this feature, HSV is often preferred for color-based separation. The transformation between the two spaces is carried out with a nreonsical connection. HSV color space is shown in Figure 7.2.

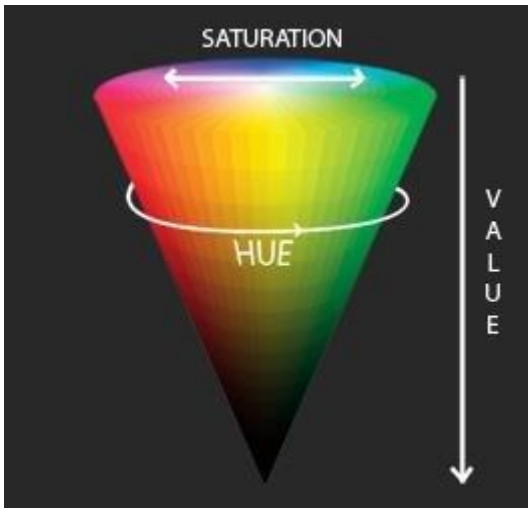


Figure 7. 1

We used specific formulas to switch from RGB color space to HSV color space. The point to note here is that rgb values should be reduced to 0-1. After each pixel is divided by 255, the following formulas and HSV values can be found.

First, color codes in the pixel values of the image are retrieved [1].

R' = R/255

G' = G/255

B' = B/255

Cmax = max (R', G', B')

Cmin = min (R', G', B')

Δ = Cmax – Cmin

Calculation for Hue value:

image::2.png[picture 2]

Calculation for saturation value:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Calculation for value(Brightness):

V = Cmax

Images constitute three colors, which are “red”, “blue” and “green”. HSV color space gives to us more healthy results. Because of this reason the captured image was filtered by the HSV color filter. HSV color filter is shown in Figure 7.3.

Figure 7. 2

7.2.2 Canny edge detection

Developed by John F. Canny in 1986. It has a multi-stage algorithm. The Canny Edge detection algorithm works very precisely, so it can also detect unwanted noise in the image as an edge. The first step to do this is to remove unwanted noisein the image by applying "Gaussian Filter".Filtered and corrected image glides with the "Sobel"

kernel for obtaining derivatives in horizontal directions (Gx) and vertical direction (Gy). For two images in horizontal and vertical directions, the edge and direction are located as follows. The gradiant aspect always stands at the edges. It is rounded vertically, horizontally and in four angles in two diagonal directions.[16][17][18][19]

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

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

Figure 7. 3

A complete scan of the image is performed to remove unwanted pixels that do not create edges after the slope size and direction are removed. For this, the maximum of space is checked in the slope direction area of each pixel.

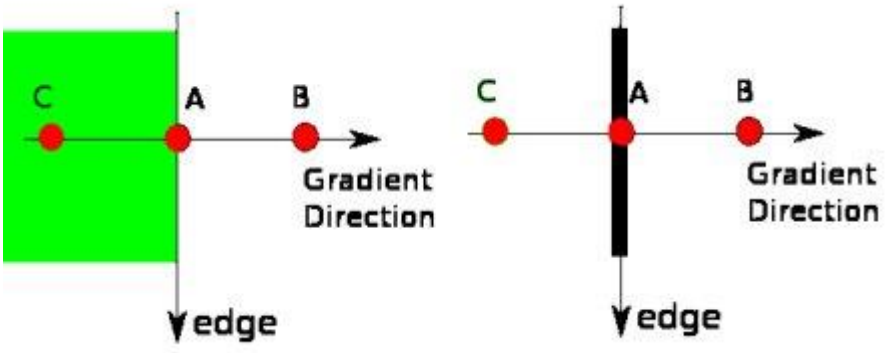


Figure 7. 4

If you describe it with the image above, the "A" point is a point on the edge. Gradiant Direction comes to the side."B" and "C" are in the Gradiant direction. Therefore, point B and C are checked to find out if point A has created the local maximum. If it creates it, the next stage will be switched. In short, a thin-edged Binary (black and white) picture is obtained at this stage.

At this point, it is decided whether all edges are really edges. Two threshold values "minVal" and "maxVal" are required. The edges larger than the Density Gradian maxVal are considered to be the edge. The edges smaller than the Density Gradian minVal are certainly not considered edges. The edges that fall between these two threshold values are considered or not edged, depending on their connections. If the edges in between are connected to the exact edge definition, i.e. the edges higher than the matching maxVal, the edge is accepted. Otherwise, they will not accept the edge. If we explain an example, you're going to have to take a say.

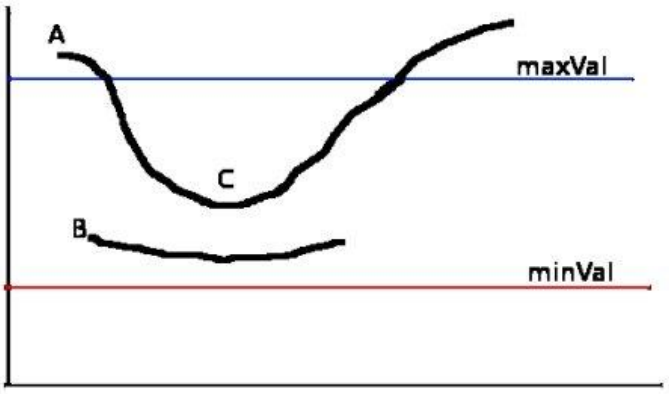


Figure 7. 5

The picture above shows two edges compared to these two edges minVal and MaxVal;

The exact edge is because part A is larger than maxVal.

The Part C is between two values but is attached to the exact edge the exact edge of the room is accepted because it continues.

The B part is between two values, but any certainty the exact edge is not accepted because it is not connected to the edge. After this stage, the Canny edge process was completely finished and the desired edges were found in the picture.

The color filter was applied together with edge detector. At this project, the “Canny” function was used for the edge detection. There are several edge detection functions but Canny was given to us more clear results. After the edge detection, the HSV and Canny images were combined together using by “bitwise_or()” funtion to take more clearly results. Opencv library presents to us all of these functions. Canny and combined images are shown in Figure 7.7.

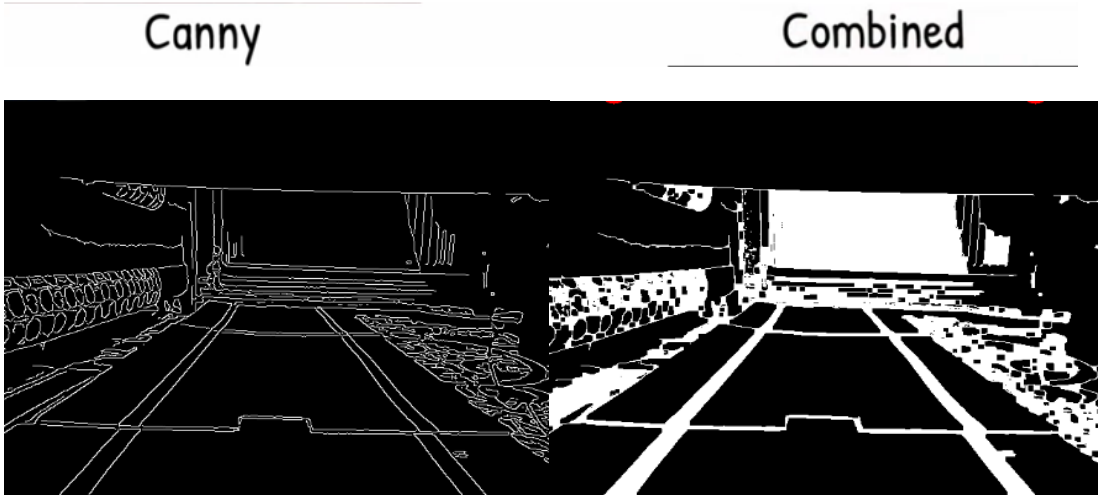


Figure 7. 6

7.3 Perspective Correction

It is a projection technique developed to visualize the effect (image) in the eyes of the observer depending on the location of the perspective object in a two dimensional plane. The purpose of this method that correction effect, which will constitute due to departure of object. The angle that we look can change even though the image constituted.

For example, we can correct a picture taken in perspective as if it were just taken from the front, or adjust the shot angle as if it was a different point. Algorithm is frequently used in character recognition applications, sitting in certain patterns, during applications such as license plate recognition and face recognition during normalization.

The perspective correction needs 4 point as an input for perspective transform. There are many matrix problems, which will solve at background of the software. At this project, we was taken those from our “valTrackbar()” function. This trackbar has some initialized coordinates for starting based on these values points were determined our warp perspective.

Those 4 points were constituted by our “drawPoint()” function, which has circle function of Opencv library. These 4 point and track bar are shown in Figure 7.8.

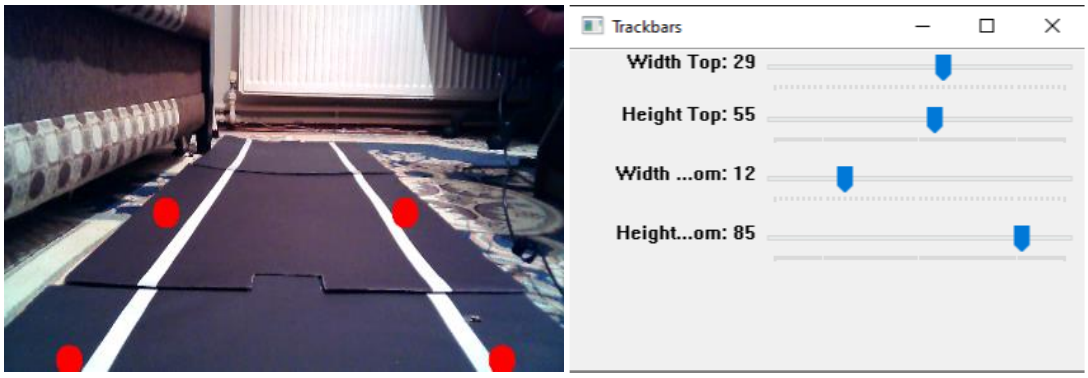


Figure 7. 7

Under favor of this track bar, we can move these 4 points through the different locations. These locations should be as close as possible to the lane lines to get more clear results. After these sections, the warp perspective method was applied to our combined image, which was of Canny and color. The warp perspective result is given in Figure 7.9.

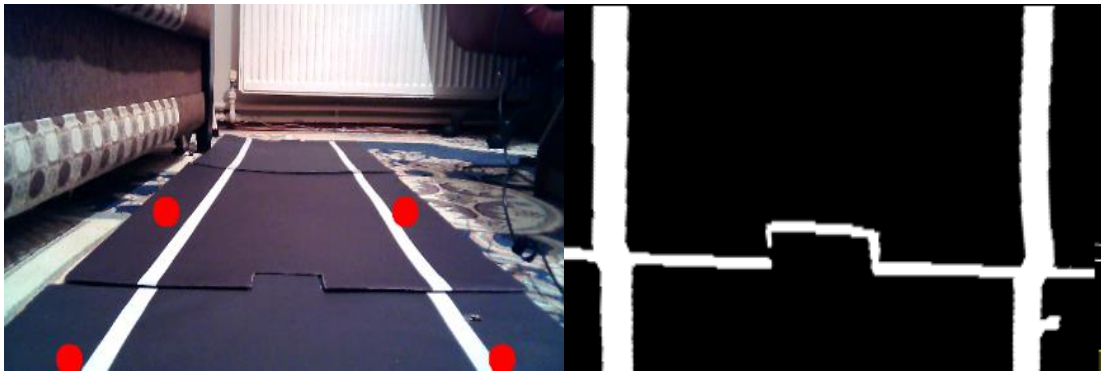


Figure 7. 8

7.4 Histogram Peak Detection

Histogram peak detection allow us to find out to the left lane line and right lane line pixels, where they start. The histogram has two peak points for this project, which are the right lane line and the left lane line. There are two histogram examples generated from two images for lane line detection in Figure 7.10.

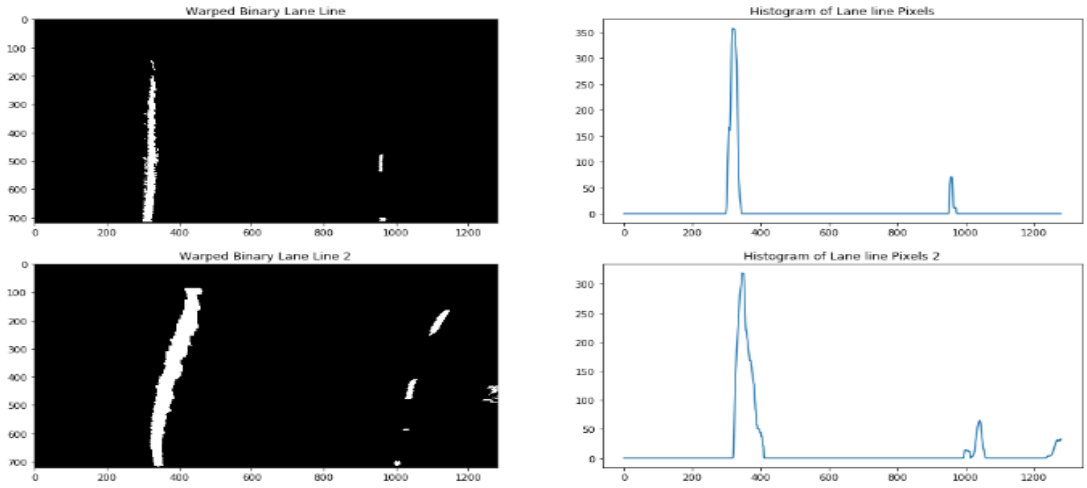


Figure 7. 9

These peaks of locations are used to find out a starting point to each lane line. The sliding window search technique was used for finding lane lines. This technique is searched repeatedly all the way from ascending order. The white pixels are detected and added to a list, while it was scanning. If the window has adequate range detected pixels, the following window is going to constitute centred in their mean position. By this means we can follow the road of the pixels throughout the image.

After the detected pixels to each lane line, each mead points of these windows were marked and the curve was fitted using by second order polynomial funtion. The line was generated, which is the best impendence result of where the lane line is. The results are shown in Figure 7.11.(red color is left lane line and blue color is right lane line)

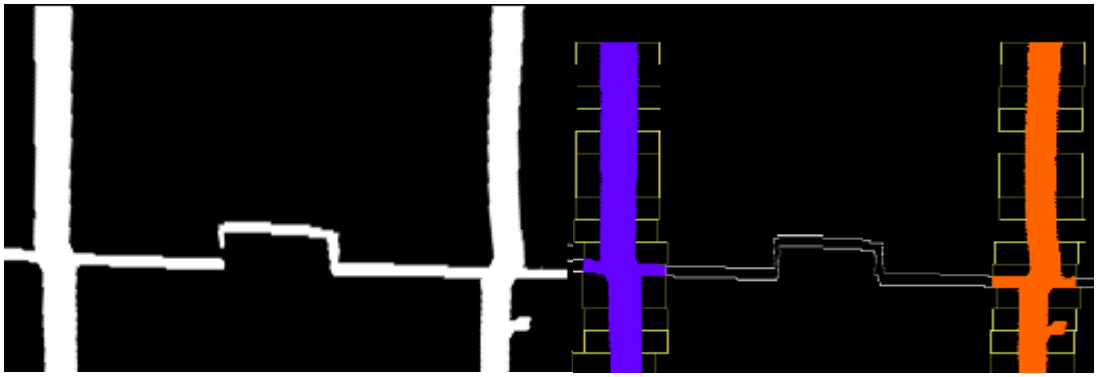


Figure 7. 10

The software has “polyfit()” function to get second order polynomial and it has a “getcurve()” function to show impendence results. The software has “drawlanes()” function to draw the points like a line. Another results are shown in Figure 7.12. which the search area highlighted and filled.

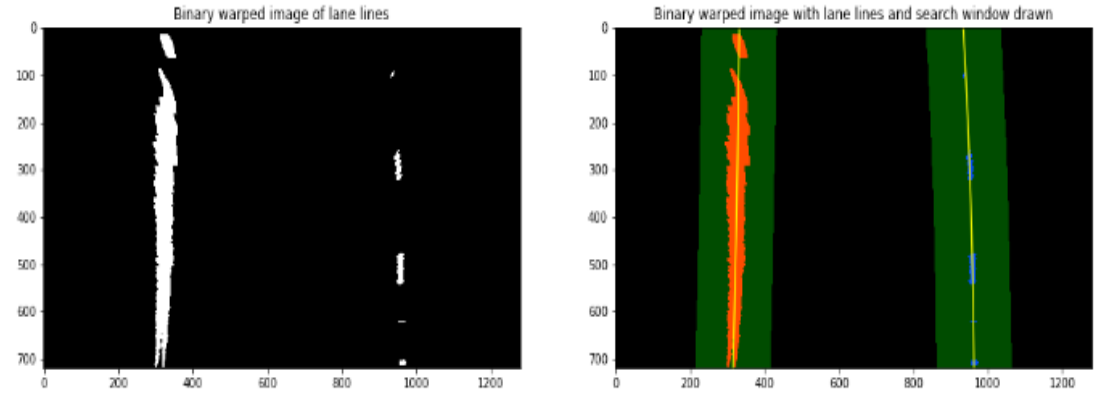


Figure 7. 11

7.4 Overlay to Original Image

At the last step, overlay it back to the original image along with the curveture value. The sofware has a “inv_perspective()” function to basically reverse of what we have done before to apply it back to our overlay original image. The overlay to original image is shown in Figure 7.13.

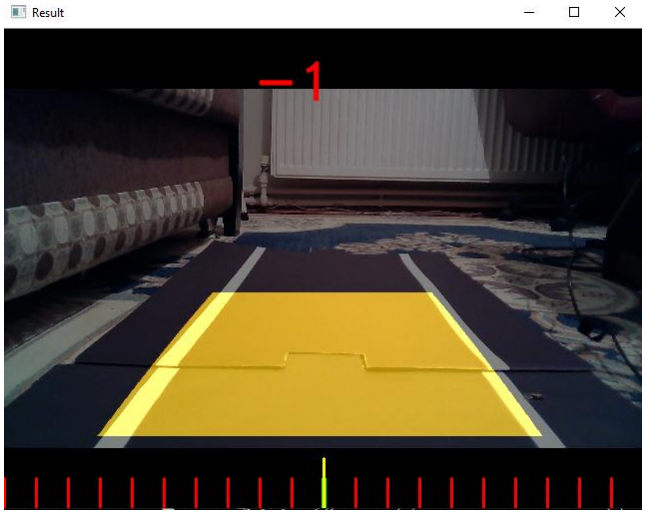


Figure 7. 12

After that the values were averaged again based on our new curve value, which is our lane curve so we have some smoother results. After this section, all of the determined values were defined using with the “putText()” function so that we can visualize curve value. The visualize function was used which is “stackImages()” that are allowed us to stack all the images together at the one screen. The first one is the original image. The second one is the undistorted image then we have the warp points where we are getting all our points in third one. The fourth one is the color image and the fifth one is the Canny image. The sixth one is the combined image, which constituted color image and canny image. The seventh one is that the warp perspective and from that warp perspective the histogram initial boxes were taken using the peak points. These boxes were drawed on top of them based on the mean position of the previous box and this allows us to follow the curve. The center points of these curves were taken and using these center points the poly_fit() function which is the second order polinomial function was applied. By this means that we can drive the radius or

the curvature of this curve. After that we can do the mean of them and we can get one final result. The red color number is the mean of our curvature in the ninth image. The servo motor is moved according to the value of this number. If this number is negative, the car will have started going away throug the left side from the road. . If this number is positive, car will have started going away throug right side from the road, thus car can move properly without stray away from the road. At the end the overlayed image was constituted that back to our original image. The outputs are shown in Figure 7.14.

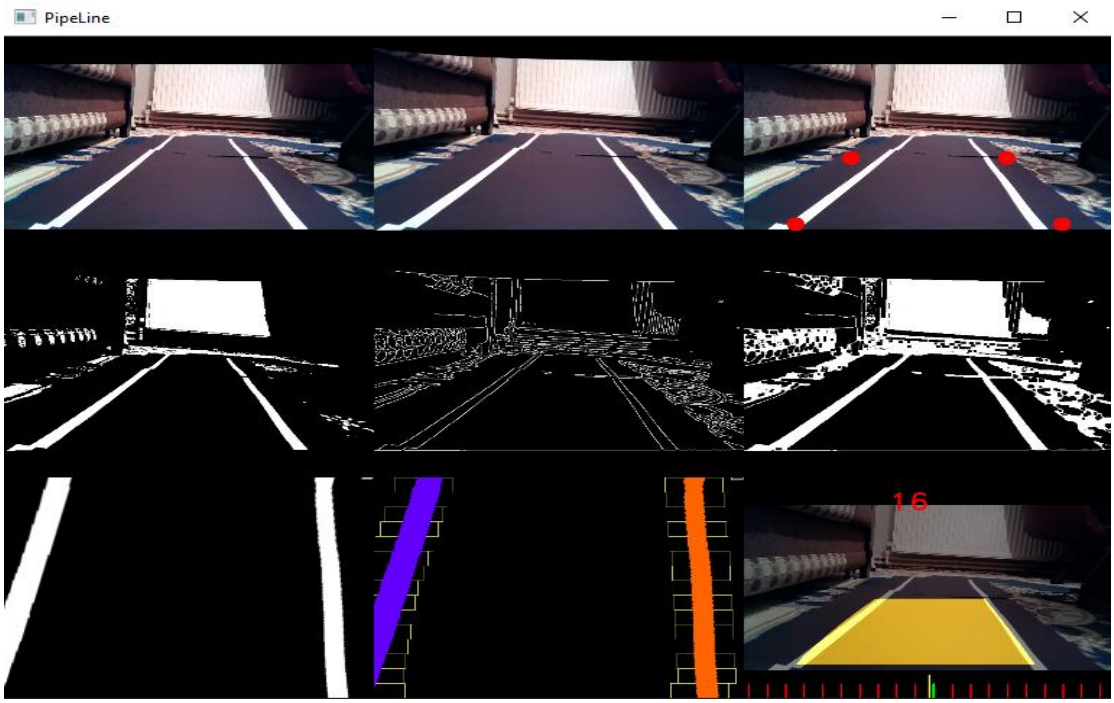


Figure 7. 13

8. DESIGN & PRODUCTION

This stage includes the results of the autonomous car, drawing stages using SolidWorks and then being printed via a 3D printer. SolidWorks is a CAD software used to create 2D and 3D solid models in a simple, fast and efficient way. For that reason, we used that program.

8.1 Autonomous Car Features

There are two different structures on the wheels of the car. Front wheel, dc motor free wheels with a free turn were used. The purpose of the front mechanism was designed with the aim of directing the car with servo motor control. The rear wheels have a dc motor, and they have a design to carry out the commands necessary for the car to go. They are shown in Figure 8.2 and Figure 8.3.

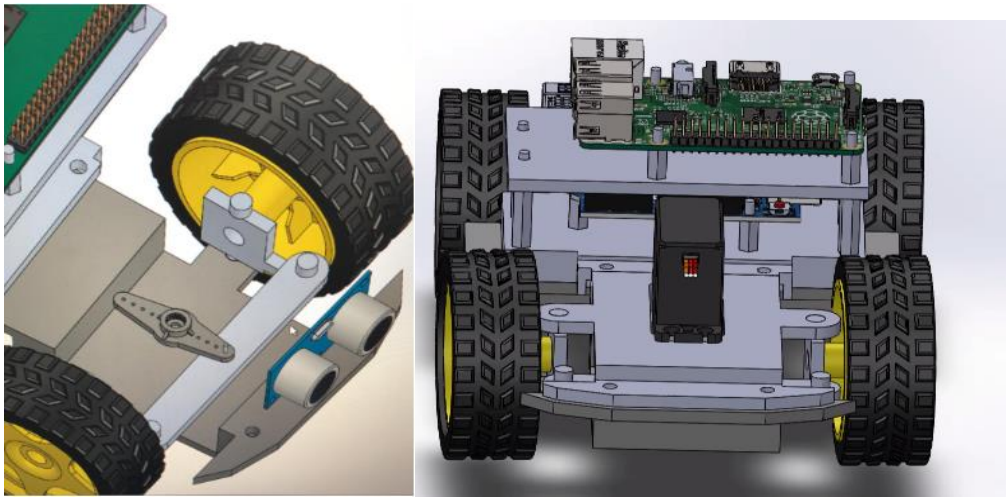


Figure 8. 1

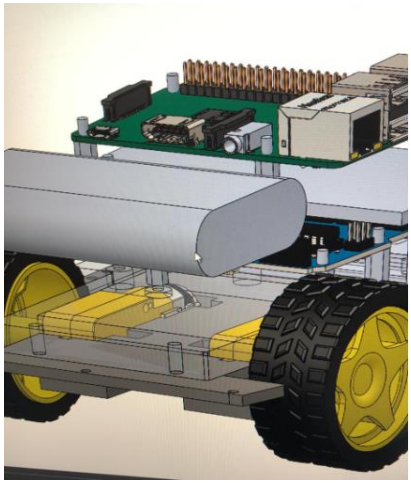


Figure 8. 2

The interior design of the autonomous car has been designed in accordance with the purpose of all electrical and software resources to have a certain order. The basic control cards are; Raspberry Pi and Arduino have a folded structure considering certain factors (eg temperature, short circuit etc.). In addition, there is a breadboard circuit board just above the rear wheels. The purpose of the Breadboard is to drive the motors using the LN298D motor driver, which will be mounted on the rear wheels. Thus, narrow space is used in the most efficient way. Design is shown in Figure 8.4 and 8.5.

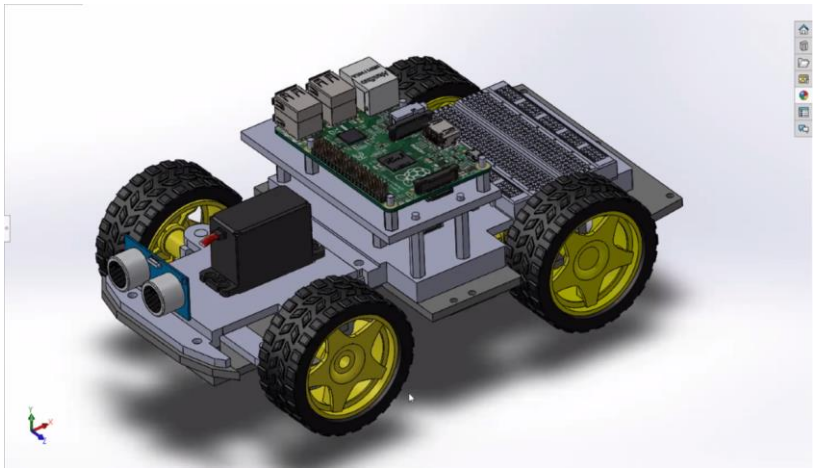


Figure 8. 3

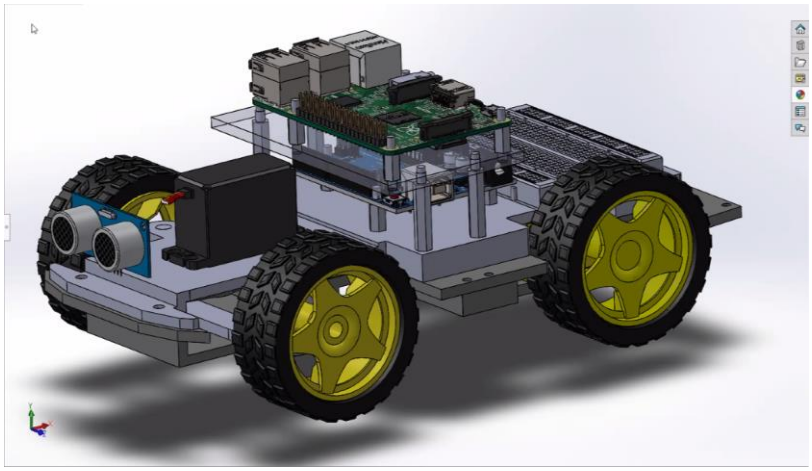


Figure 8. 4

The exterior design is designed in a structure similar to the 'CyberTruck' style in order to achieve a stylish and beautiful appearance and to keep the electronic components inside without too narrow space. On the front of the autonomous car, a circular grille was used for the HC-SR04 distance sensor. On top of the autonomous car, a z-shaped piece was designed to fit the design of the camera. Luggage design was designed for the powerbank, which will be used to feed the electronic components inside, at the rear of the autonomous car. The completed design is shown in Figure 8.6, Figure 8.7, Figure 8.8, Figure 8.9, Figure 8.10 and Figure 8.11.

Figure 8. 5

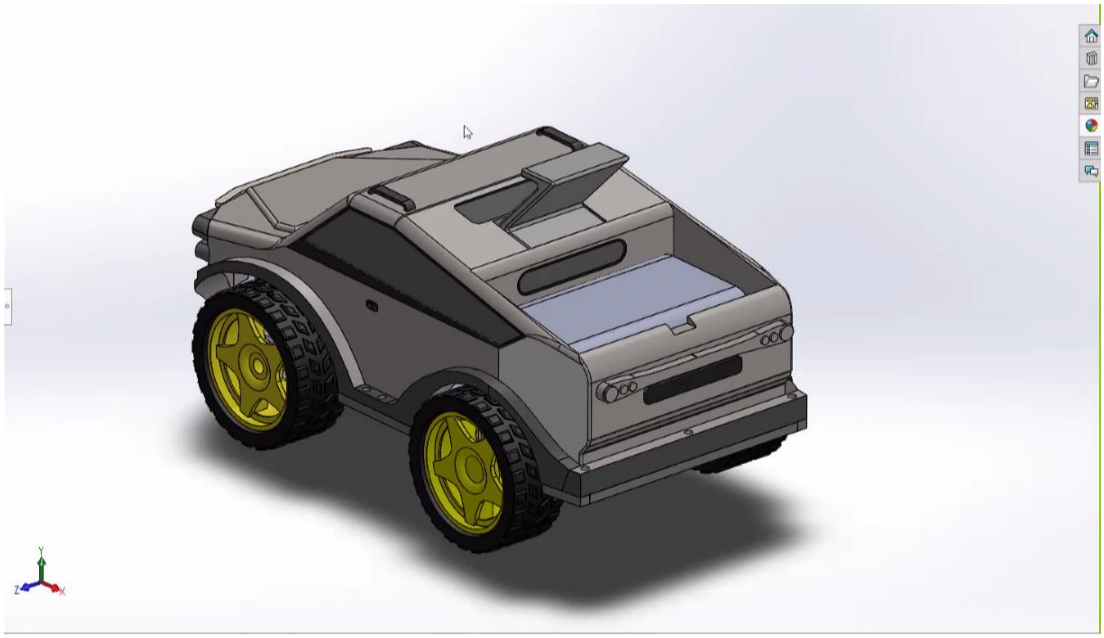


Figure 8. 6

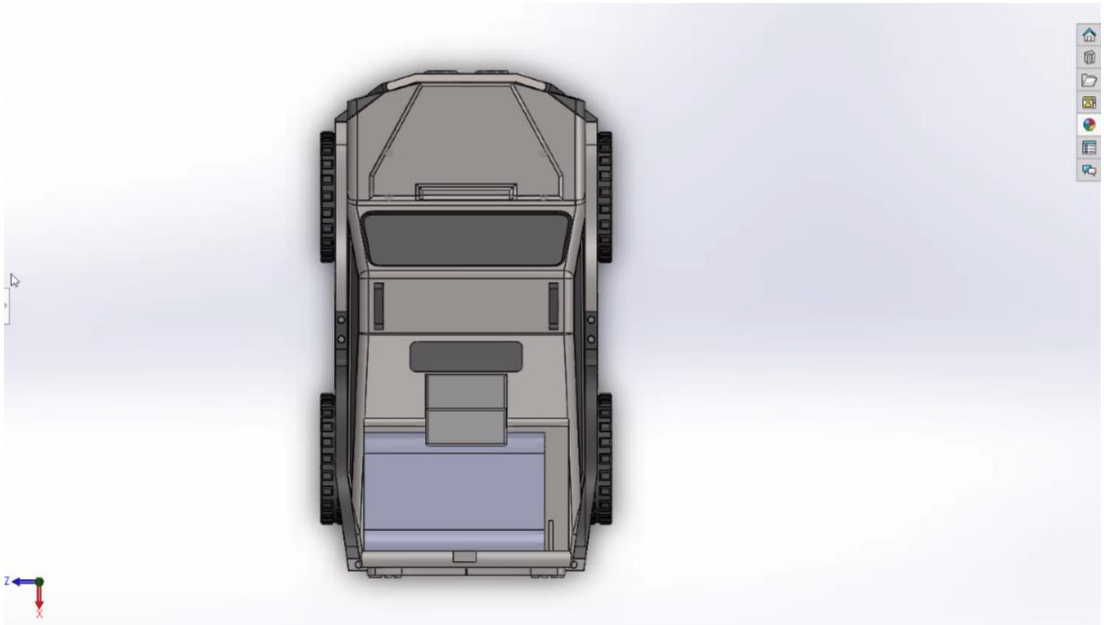


Figure 8. 7

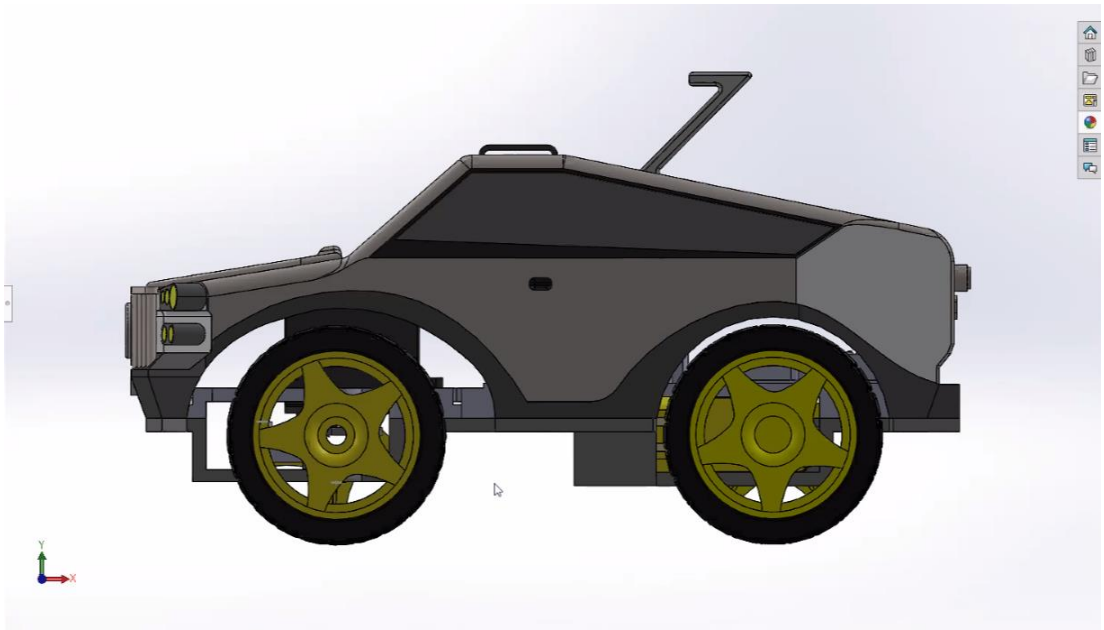


Figure 8. 8

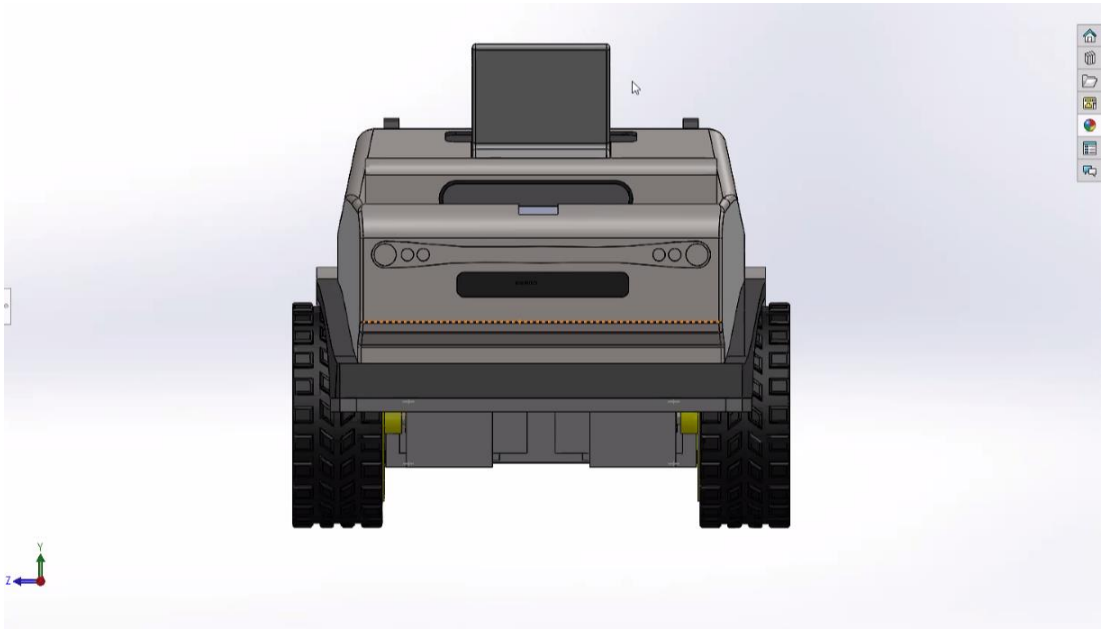


Figure 8. 9

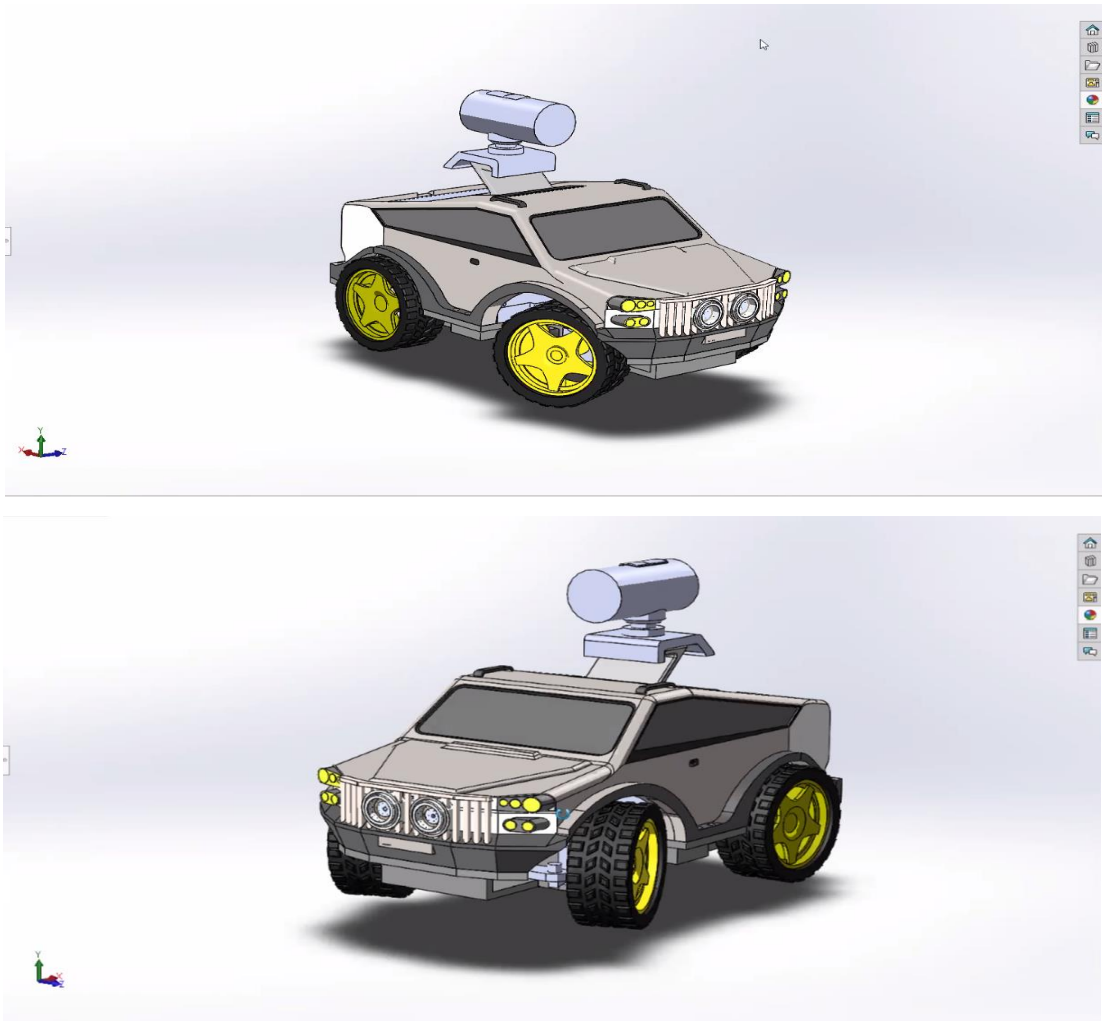


Figure 8.11

In the production phase, we produced the autonomous car using a 3D printer. 3D printer stages are shown in Figure 8.13, Figure 8.14 and Figure 8.15.

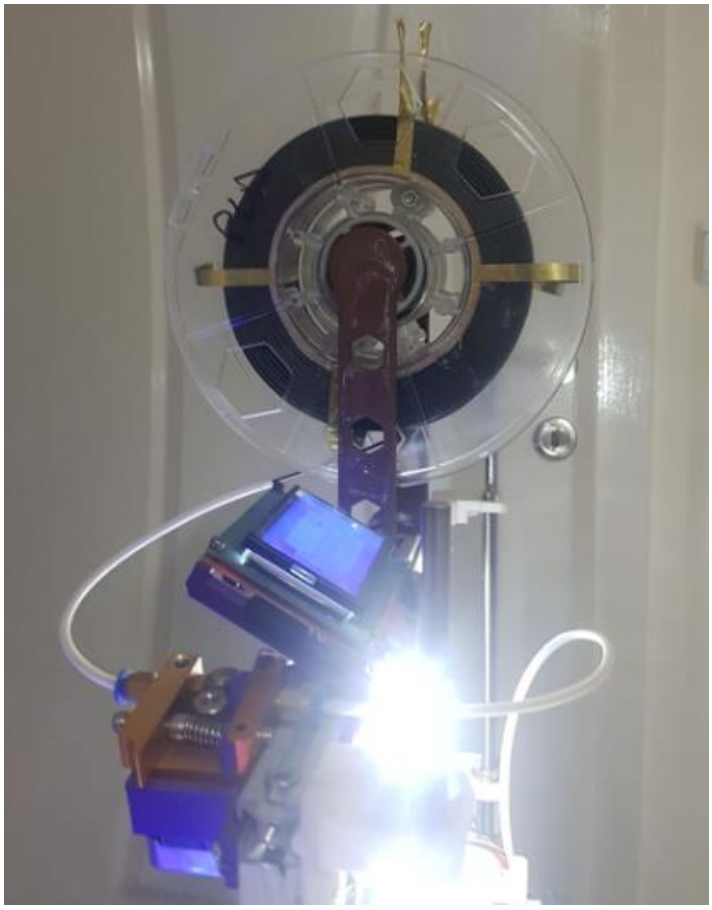


Figure 8. 10

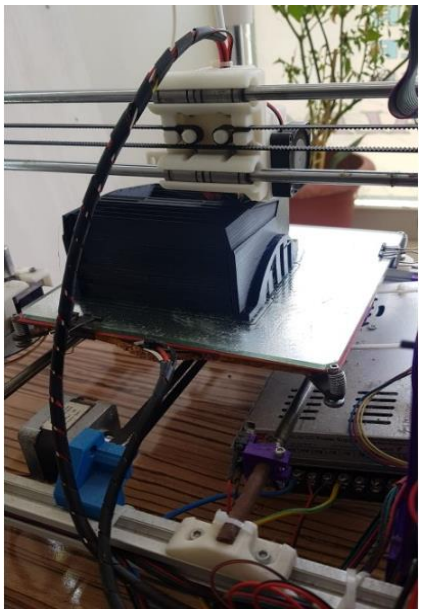


Figure 8. 11

Figure 8. 12

8.2 MOUNTED OF THE AUTONOMOUS CAR

Figure 8. 13

Figure 8. 14

Figure 8. 15

9. TESTS & RACEWAY

The raceway was created by the interlacing of black cardboard. Road strips were created by pasting white tape on the runway. It is shown in Figure 9.1.

Figure 9. 1

In the code, the car moves 5 seconds after it stops when it sees the stop sign.

Figure 9. 2

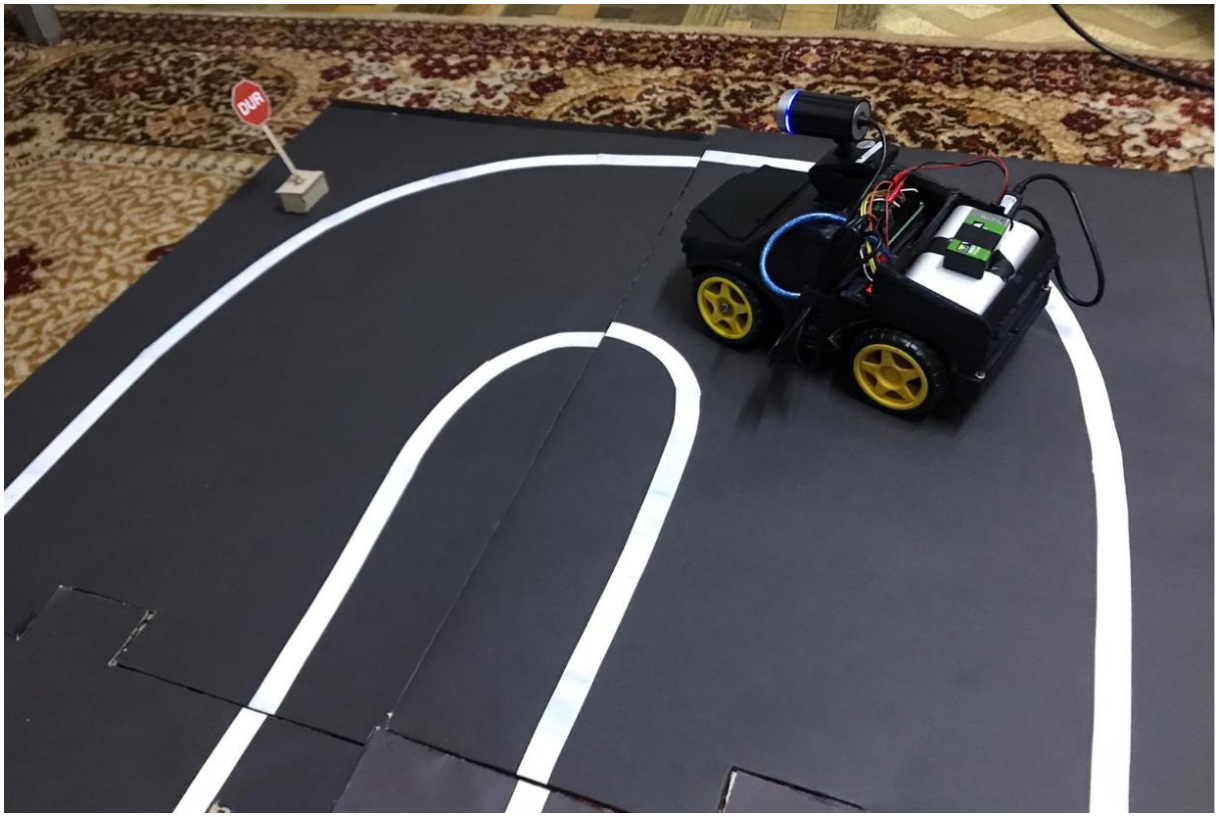


Figure 9. 3

When the car sees the red light, the red light continues to stop until it turns green. In the created circuit, with the help of a button, the red light can be turned off and the green light can be lit.

Figure 9. 4

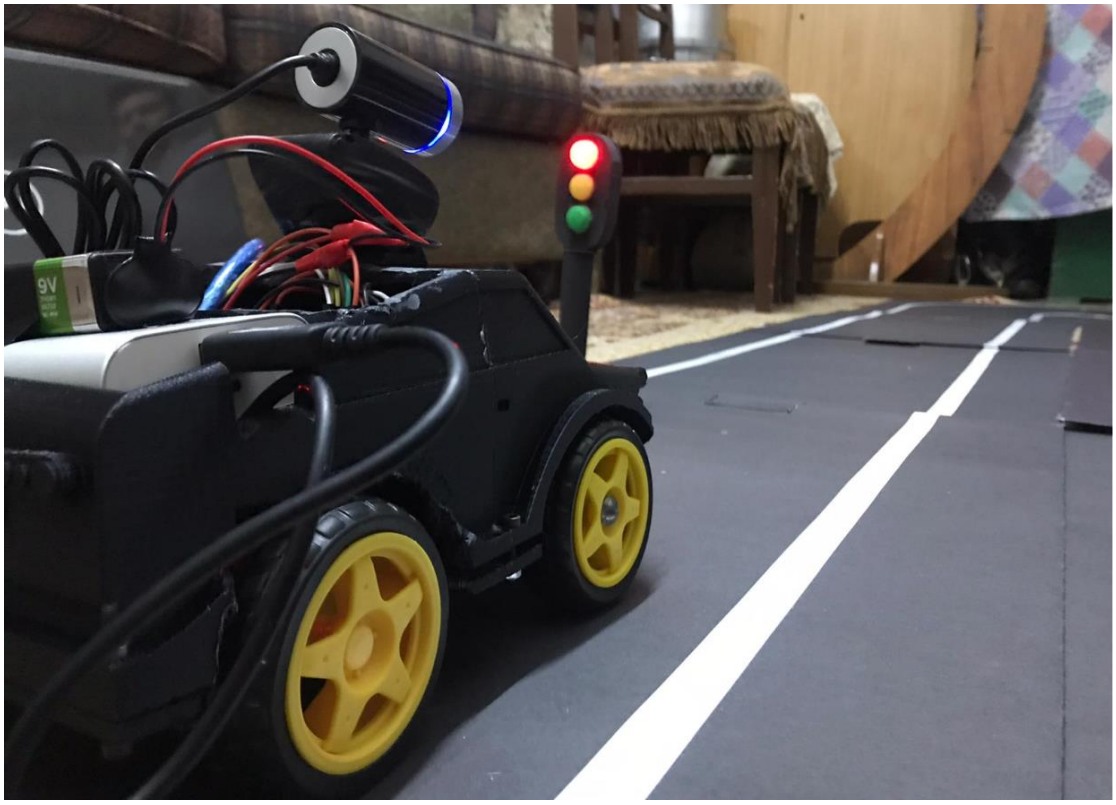


Figure 9. 5

Figure 9. 6

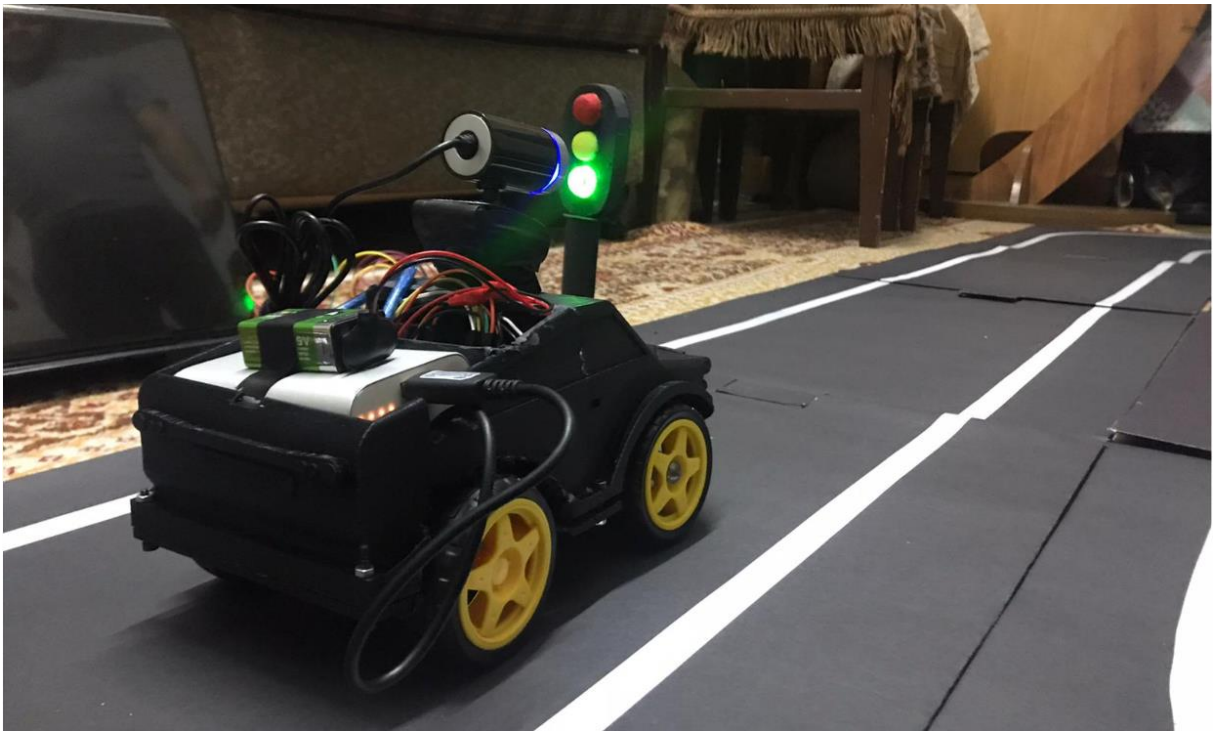


Figure 9. 7

In the lane tracking system, the green line under the frame represents the need to go left or right. When the car turns to the right, the green line moves to the right of the yellow line fixed to the middle of the frame, and when the car turns to the left, the green line moves to the left of the fixed yellow line. When the green line is in line with the green line, it represents that the servo motor should not be moved and continue in the straight direction. The results are shown in Figure 9.8, Figure 9.9, Figure 9.10, Figure 9.11, Figure 9.12 and Figure 9.13.

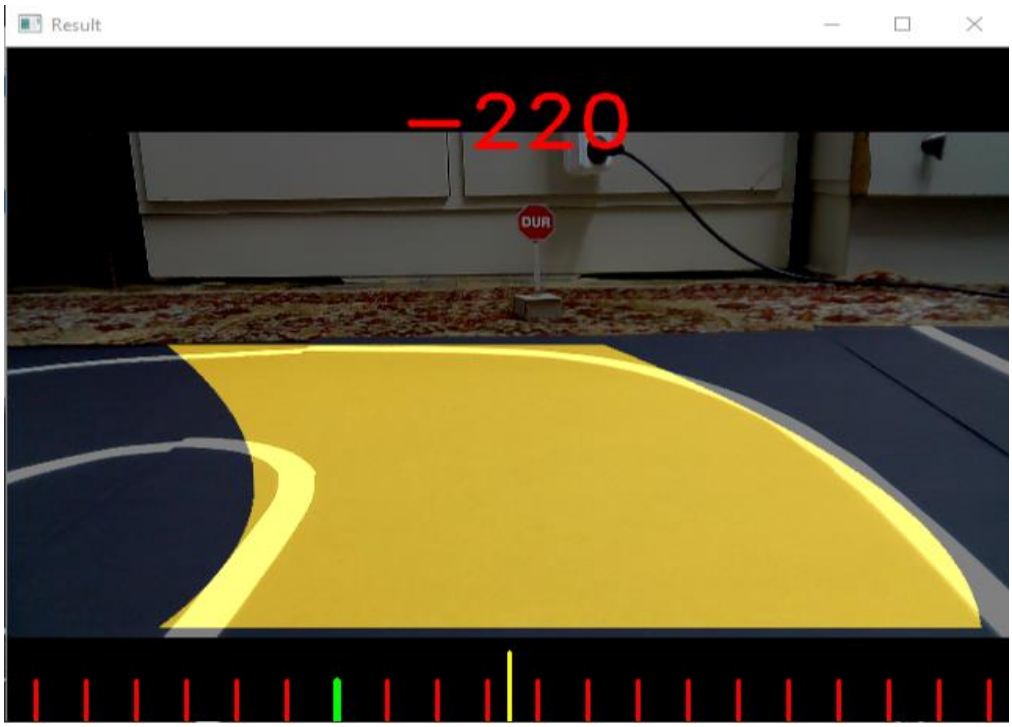


Figure 9. 8

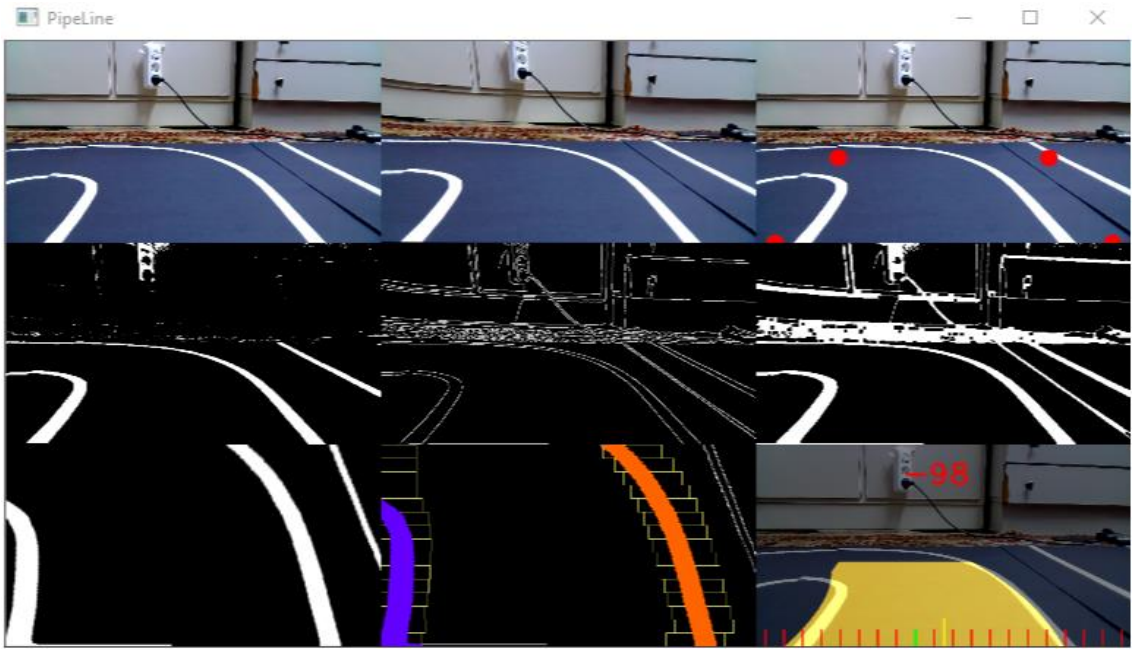


Figure 9. 9

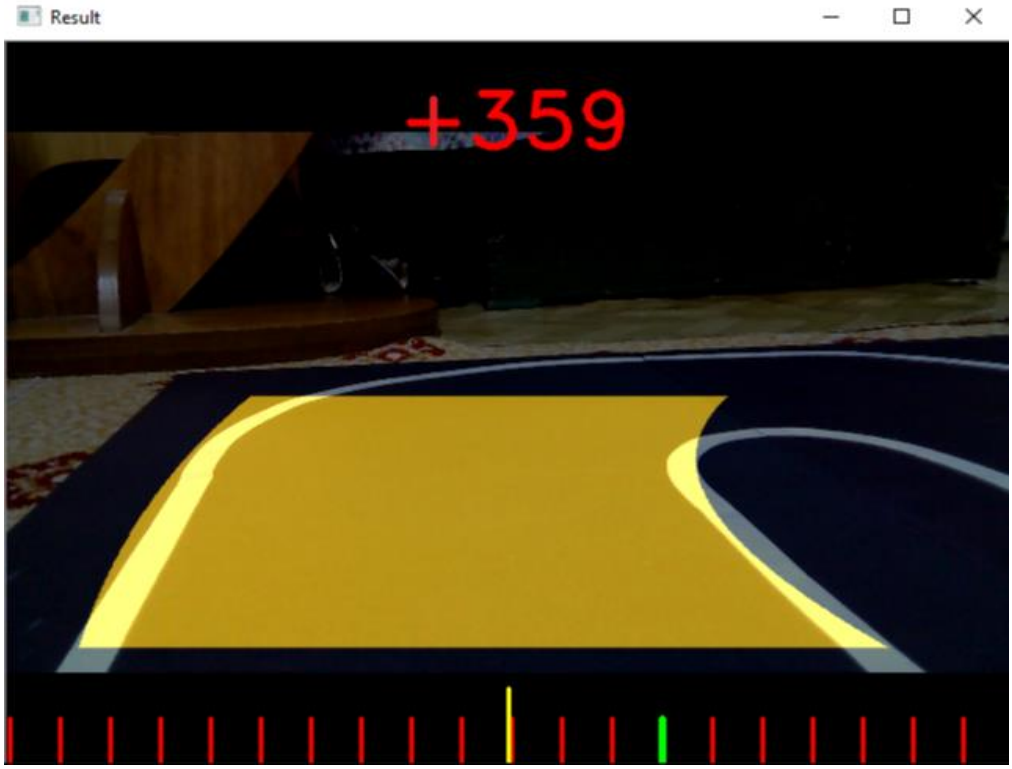


Figure 9. 10

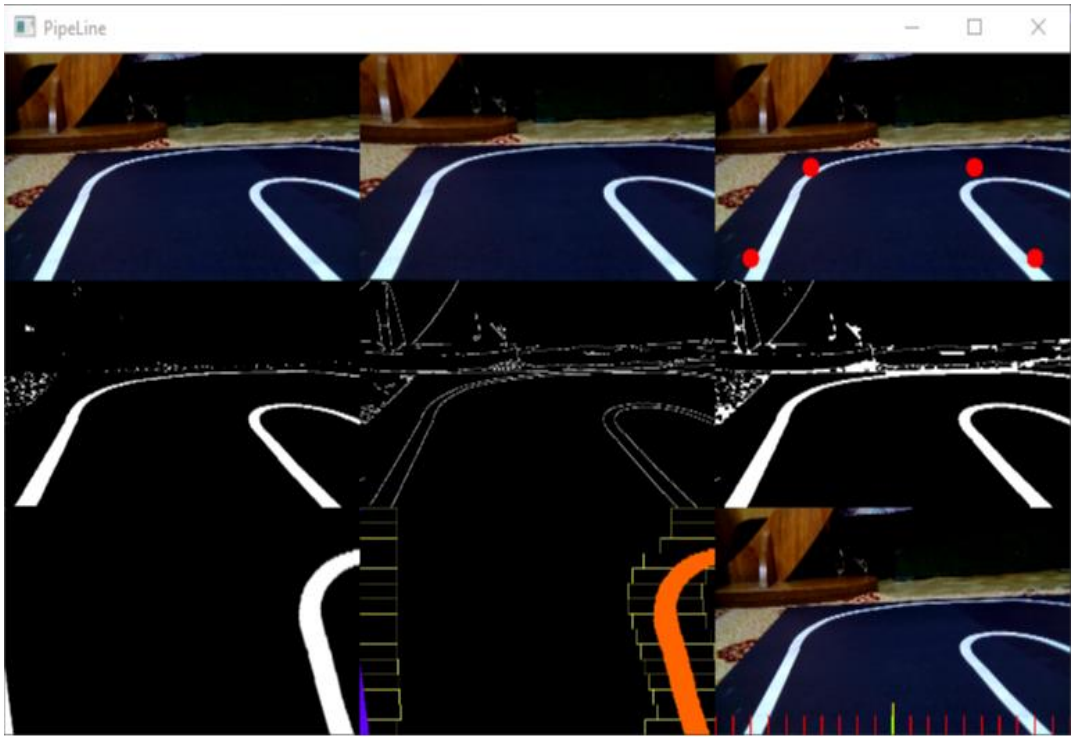


Figure 9. 11

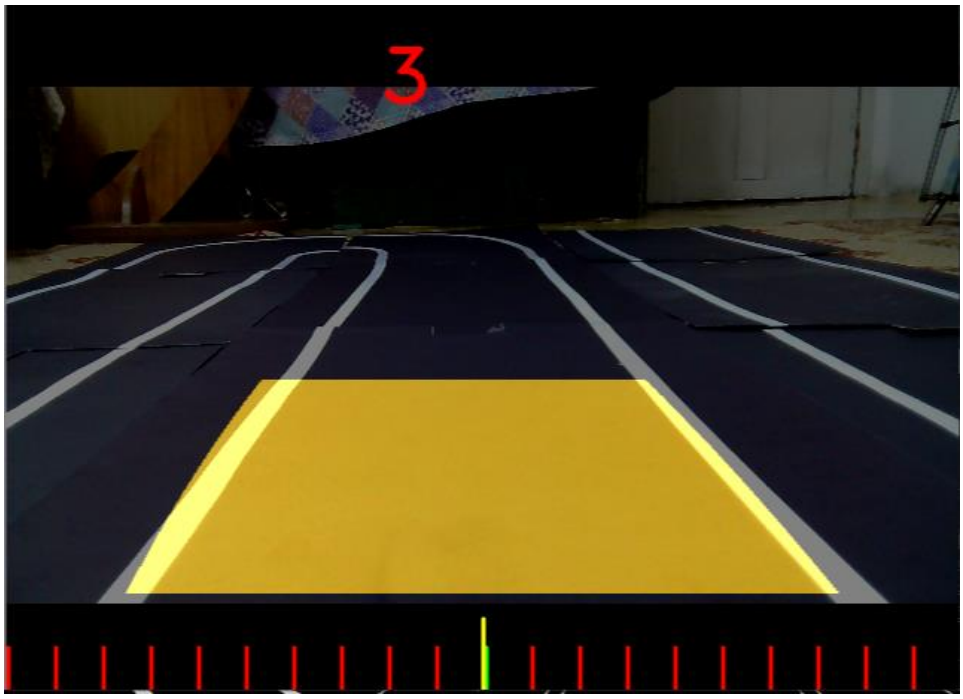


Figure 9. 12

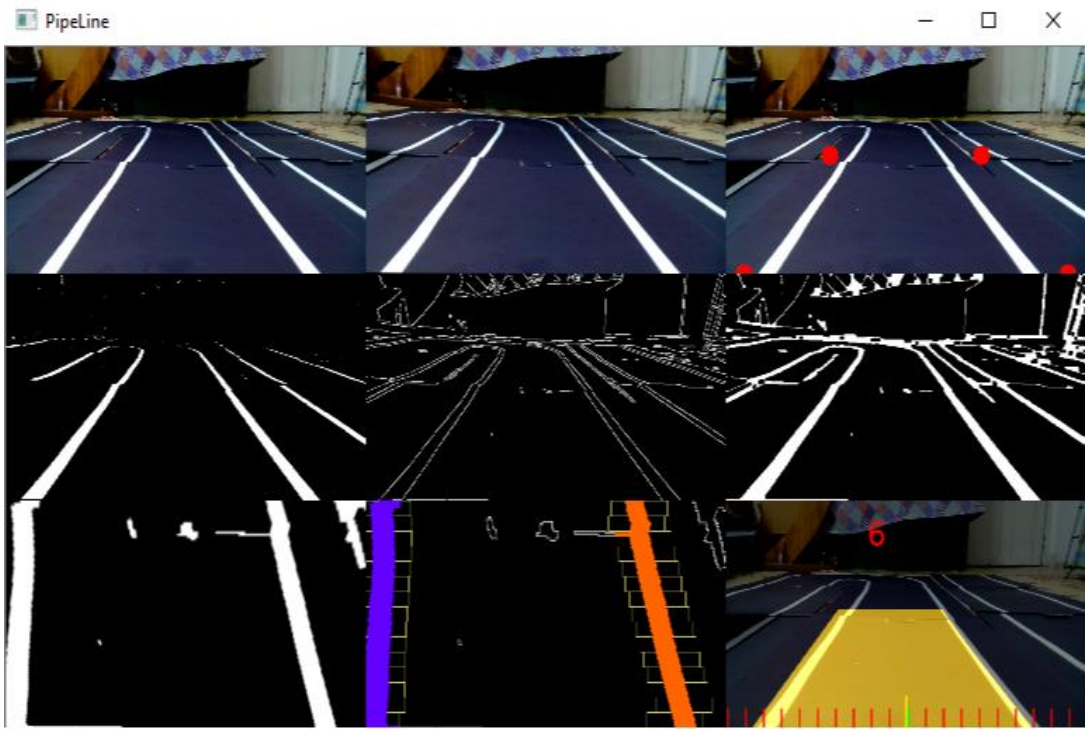


Figure 9. 13

10. CIRCUITS DIAGRAMS

10.1 Raspberry Pi – Arduino Diagram

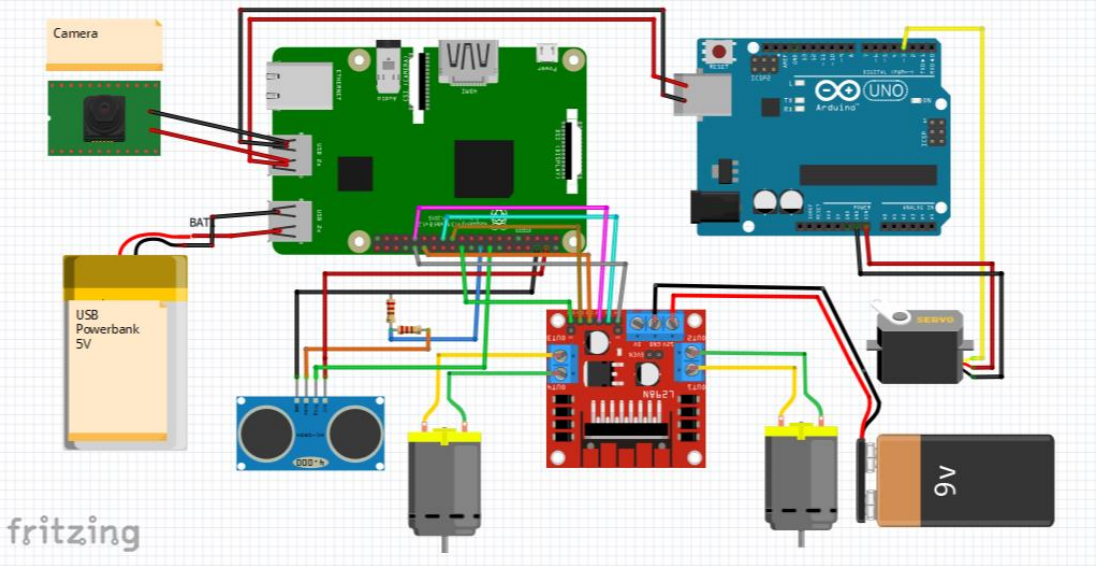


Figure 10. 1

10.2 Traffic Light Diagram

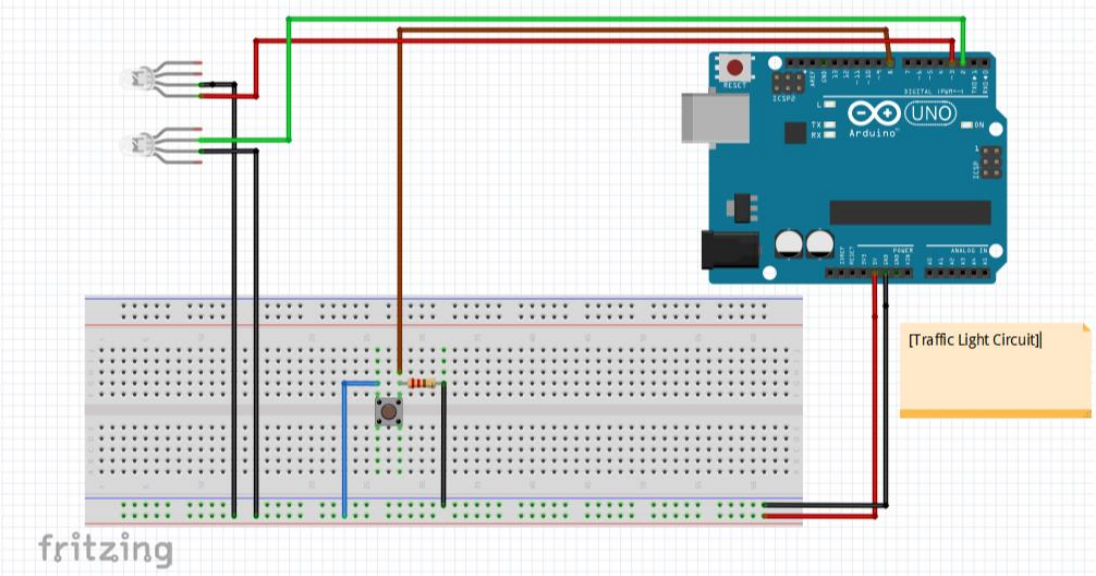


Figure 10. 2

REFERENCES

[1] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Insideoutside net: Detecting objects in context with skip pooling and recurrent neural networks. arXiv preprint arXiv:1512.04143, 2015. (FASTER R-CNN)

[2] A. Shrivastava, A. Gupta, and R. Girshick. Training regionbased object detectors with online hard example mining. arXiv preprint arXiv:1604.03540, 2016. (faster)

[3] A. Shrivastava and A. Gupta. Contextual priming and feedback for faster r-cnn. In European Conference on Computer Vision, pages 330–348. Springer, 2016.

[4] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollar. A multipath network for object ´ detection.

[5] Duda, R. O. and P. E. Hart, Use of the Hough Transformation to Detect Lines and Curves in Pictures, Comm. ACM, Vol. 15, 1972, pp. 11–15

[6] W. E. L. Grimson and D. P. Huttenlocher, On the sensitivity of the Hough transform for object recognition, IEEE Trans. Pattern Anal. Mach. Intell., Vol. 12, 1990, pp. 255–274

[7] J. Illingworth and J. Kittler, The adaptive Hough transform, IEEE Trans. Pattern Anal. Mach. Intell., Vol. 9, 1987, pp. 690–698

[8] J. Illingworth and J. Kittler, A survey of the Hough transform, Comput. Vision Graphics Image Process., 44, 1988, pp. 87–116

[9] W. Niblack and D. Petkovic, On improving the accuracy of the Hough Transform, Mach. Vision Appl,. Vol. 3, 1990, pp. 87–106

[10] https://arxiv.org/pdf/1611.10012.pdf (Page 10)

[11] Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. (Figure 3.3, Figure 3.4)

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C.Berg [arXiv][demo][code] (Mar 2016)

[13] Sklansky, J., On the Hough Technique for Curve Detection, TC Journal, Vol. 27, 1978

[14] N. Kiryati, Y. Eldar, & A.M. Bruckstein, A probabilistic Hough transform, Pattern Recognition, Vol. 24, 1991, pp. 303–316.

[15] D.H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, Pattern Recognition, Vol.13, No.2, 1981, pp. 111-122

[16] S. Das, “Comparison of various edge detection technique”, International Journal of Signal Processing, Image Processing and Pattern Recognition, vol.9, no.2, (2016), pp.143-158.

[17] J. Canny, “Line Detection by Hough transformation ” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 8, pp. 679-698, 1986

[18]] R. C. Gonzalez, R. E. Woods and S. L. Eddins, “Digital Image Processing Using MATLAB”, Pearson Education Ptd. Ltd, Singapore, (2004).

[19] M. Juneja and P. Singh Sandhu, “Performance Evaluation of Edge Detection Techniques for Images in Spatial Domain”, International Journal of Computer Theory and Engineering, vol. 1, no.5, (2009), pp. 614-621.

[20] https://towardsdatascience.com/review-r-fcn-positive-sensitive-score-maps-object-detection-91cd2389345c (Figure 3.2)

[21] http://www.intmath.com/applications-differentiation/8-radius-curvature.php

[22] https://arxiv.org/pdf/1506.02640.pdf

[23] Test images and videos come from the udacity open source self-driving car repo: http://github.com/udacity/self-driving-car

[24] https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[25] https://pjreddie.com/darknet/yolo/ (Figure 3.6)

[26] https://github.com/pjreddie/darknet

[27] https://www.nytimes.com/interactive/2016/12/14/technology/how-self-driving-cars-work.html

[28] talhakoc.net/opencv-haarcascade-siniflandirici-egitimi