ISSN: 2633-4828 Vol. 5 No.4, December, 2023

International Journal of Applied Engineering & Technology

SELF-DRIVING CAR USING IMAGE PROCESSING AND YOLO V5

Sunita Ugale[1], Dinesh Chandwadkar[2] and Sakshat Erande[3]
[1,2,3] K K Wagh Institute of Engineering Education and Research, Nashik, Maharashtra, India
[1]spugale@kkwagh.edu.in,[2]dmchandwadkar@kkwagh.edu.in and [3]sakshatetc08.ss@gmail.com

ABSTRACT
This research work on the topic "Self-Driving Car using Image processing and YOLO V5 model" aims to provide a relaxed driving experience to human driver. In this paper, we propose a prototype of a self-driving car that is capable of driving on its own and detecting an obstacle, traffic signal, and stop signs. A camera module is attached at the top of the car along with a Raspberry Pi board which sends the images from surroundings to the processing unit for the processing which then predicts one of the directions of movement of the car and it also has ultrasonic sensors for monitoring the obstacles in the nearby environment to prevent any possible collision. For Sign Detection there are two methods, one is using CNN but problem with CNN is that image must be perfectly present in front of camera which is not the case with self-driving car. For avoiding this problem we are proposing YOLO V5 which is fast in response and has more accuracy than CNN. Consequently, the car moves in the desired direction by itself without any human interference.

Index Terms – Artificial Intelligence, Image Processing, Self-driving car, YOLO V5

INTRODUCTION
Car crashes have been one of the biggest reasons for fatalities and it claims a staggering 1.3 million casualties a year. Driving at high speeds, talking on the phone, drunk driving, and not following the traffic rules are the major causes of these accidents and the causes of fatalities are growing day by day which has now become a major concern. The traffic safety awareness programs seem to work very little to not at all in some cases. Human error cannot be eliminated but with the intervention of technologies like Artificial Intelligence and Machine Learning, it can be minimized. Right from a simple collision detection mechanism to a full-fledged self-driven car, the technology has come a long way.

The concept of a Self-driven car used to be a thing of imagination but now it has become so much of a reality that every major automobile company is investing hugely in it to make it cheaper and more accessible to people. The idea behind the self-driving car is to reduce human errors using various technologies like Image Processing and Machine learning [1]. These technologies are expected to complement the slower reaction time by humans which can minimize the possibility of accidents. Another useful feature of a self-driven car is that it can help in maintaining good traffic flow.

Along with this, self-driving cars can also be useful in managing the parking space issues by parking themselves properly using the parking space markings and alignments which is not the case with parking done by a human. Often humans park their vehicles in a way that takes up much space.

DESIGN METHODOLOGY

Hardware
The basic model of any self-driving vehicle consists of a radar to detect nearby vehicles, a front-facing camera to recognize the lanes on road, other cars, traffic signals, trees, etc., and an integrated system that controls the speed of the vehicle. It also consists of a long-range ultrasonic sensor to prevent a collision. The camera captures the information from a real-time environment and sends it to the learning algorithm and every movement of the vehicle happens accordingly. The schematic arrangement is shown in Fig 1.

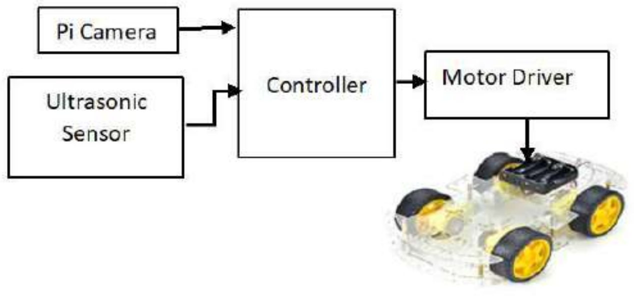

FIG. 1 Self driving car controller block diagram

Pi camera, is a great gadget to capture time-lapse, slow motion with great video clarity. The dimension of the camera is 25mm x 23mm x 9mm, which connects to controller via one of the small sockets on the board itself and uses the dedicated Camera serial interface.

Various distance sensors can be used for sensing obstacles and taking immediate decisions regarding either stopping or lowering the speed of vehicle. For a real-time system, Lidar sensor or ultrasonic sensor can be used for detecting obstacles that are ahead of the self-driving car. The distance between the car and the upcoming solid surface (cars or obstacles/pedestrians) can be monitored. Once it decreases beyond a certain level, the controller will give signals to the motor driver to stop the car.

The controller produces a value, 'curve' by image processing which indicates the extent of turn that road takes. Value of ‘curve’ is positive or negative depending on right/left turn. According to this value, motors have to be driven i.e., to turn towards the right side, speed of motor connected to the right wheel is increased. This operation is controlled by the motor drivers like L298N. Using such a motor driver speed of a motor can be varied by PWM pulse. Also, the direction can be controlled to move the car in reverse direction.

Software

Colab is a short form for Google Colaboratory. It is a product from Google Research. With Colab anybody can write and execute python code on the browser. It is especially well suited for machine learning, education, and data analysis. Colab requires no setup to use and is a hosted Jupyter notebook service which, while providing free access to computing resources like high-performance GPUs (Google, 2017).

We have used this platform for training of own custom dataset (traffic sign) detection algorithm YOLO V5. We used it because it will give fast and accurate results than local machine GPU while training.

VS Code is used for simulation purpose. We have done Lane Detection simulation, Traffic sign detection simulation on local machines.

Makesense.ai is used for creating a bounding box around traffic signs. After creating a bounding box for each image it will generate a text file for each dataset image which contains the name of the class and four coordinates of the bounding box.

Lane Detection

We have used two methods for lane detection. The first one is using CNN (Convolutional Neural Network) and the second one is using image processing.

CNN is a Deep Learning algorithm that takes images as input, assigns importance which is also called as weights to various aspects of the image, and be able to differentiate various images from the others. These weights are changeable and they get modified during the process of learning when the neural network gets trained.

There are different layers in CNN and each circle represents a neuron that has a certain activation level. Links have certain weights and biases. Based on these weights and biases, the neurons in the last layer get lit up. The last layer represents the images from which the desired image is to be identified. So, if an image is detected, its corresponding neuron lights up and for others, the activation level is zero.

In the training process, the images and corresponding expected outputs, i.e., which neuron in the output layer is supposed to light up, are fed as an input. Accordingly, the weights get modified to produce the correct output. There is a function called 'cost function' which represents the current activation of neurons in the last layer and expected activations. The whole process is actually to minimize this cost function using 'gradient descent'.

As we minimize the cost function, activations in the output layer get closer to what is expected and the neural network gets better trained.

Lane detection using CNN: To detect a lane using deep learning, the car has to be driven first on the track using a remote control to collect images and the corresponding values of speed and curve. CNN figures out the rules based on input and outputs that are fed during the training process. Now, that rules are known (i.e., model is trained) the algorithm can take decisions based on the input and the rules that have been learned. When the car is first moved on the track, it captures thousands of images and also the corresponding values of steering angles get stored. Now, this data is used for the training purpose of a neural network.

The weights get modified during the training process as discussed earlier. As an output, we get the value 'curve' for the image as an input. This input is fed to a car using a camera. The images are read and processed by various functions of the OpenCV library. The pixels of the image get stored using the NumPy array. These pixels are given as input to the first layer of CNN. Accordingly, the weights get trained to produce corresponding output.

DRAWBACKS

- The model needs to be trained by driving a car on track using remote control. So, the extra accessory has to be designed.
- The process of training is tedious in terms of time complexity as well as space complexity. It involves the processing of tremendous data which has been fed before operation.
- Training data set has limitations. So a model may not take the correct decision for any other random track. Hence the system is not dynamic.
- Processing has to be done before running as well as in real-time. So, the system can't make decisions all by itself in real-time.
- There is no limitation to the type of roads/turns that a driver might encounter, so a model can never be trained perfectly; which is a threat to safety.

To eliminate some drawbacks of Lane detection using CNN we have used Lane detection using Image Processing.

Lane detection using image processing

The lane detection using image processing algorithm is shown in Fig. 2.

Image Capturing: The input data is a color image set captured from the camera installed on the moving vehicle. It is mounted exactly at the central line of the vehicle. It takes images of the region in front of it which includes other vehicles on the road, lanes, signboards, and sometimes obstacles on the road. The controller unit to which the camera is connected captures images in real-time up to the speed of 60 frames/second. These images are saved in the controller memory. The lane detection algorithm reads these images from the memory and processes them.

Conversion to Grayscale: Converting the images into grayscale makes it easier to retain the color information. The edge detection of the images becomes harder and takes up more time for processing. The captured images might contain a lot of unnecessary information made up of many colors due to different materials of the road, shadows, potholes, aging of the asphalt. Due to this, the color of the road will change from place to place. Therefore, the captured images are converted into grayscale. Processing of the grayscale images takes minimal time [2]. 8-bit single-channel grayscale is formed by the 24-bit color images, three channels. The image can be converted through averaging method also, where,

$$Grayscale = (R + G + B) / 3$$

Where R, G & B are the weights of primary color

The problem with the averaging method is that, the result is not satisfactory since the colors have different wavelengths and hence have their contribution to the image formation. To avoid this, we take average according to their contribution. Therefore, we use the luminosity method. where,

$$Grayscale = 0.3 \times R + 0.59 \times G + 0.11 \times B \qquad (1)$$

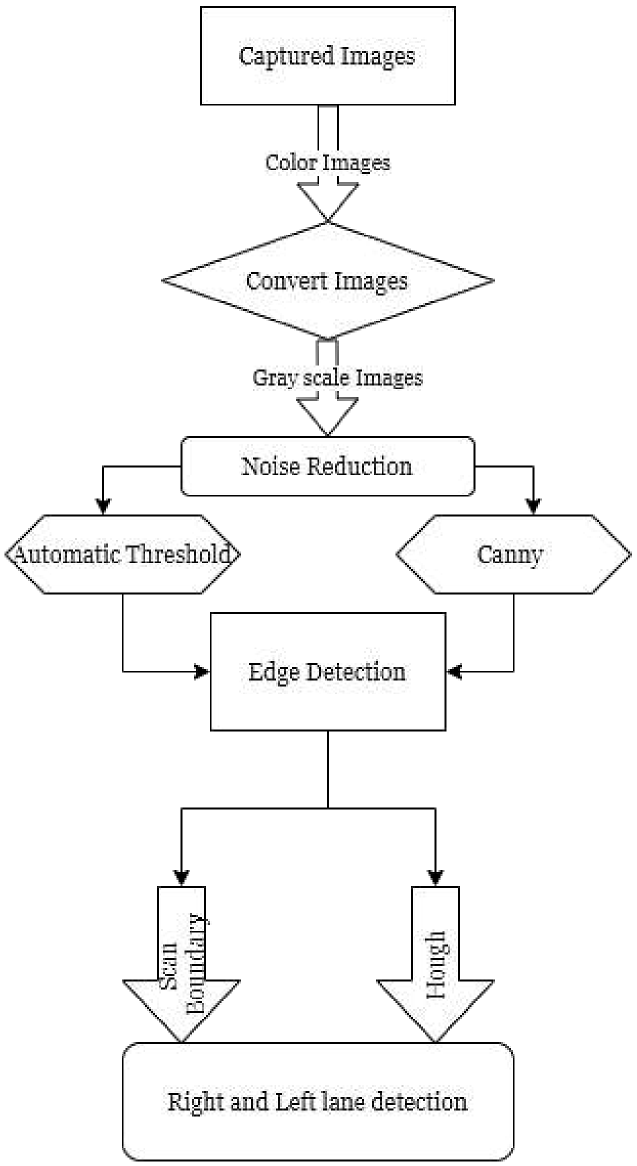

Fig 2. Lane detection algorithm

Noise Reduction: The presence of noise in the images can affect the processing speed of the data. Hence the noise should be either reduced to a level where an algorithm is tolerant enough to function properly or it should be eliminated. The noise will also hinder the correct edge detection. The noise on the lanes is mostly because of the shadows from objects in the surrounding. This noise is reduced by removing the strong shadow boundaries. The basic idea is that shadows have a distinct boundary. This shadow boundary is removed and an original image is reconstructed. The shadow edge image is obtained by applying the edge detection method on both the invariant image and the original image. The image that is constructed by selecting the edges is shadow-free that exists in the original image but is absent in the invariant one and then eliminating the edges from the original image by the application of the pseudo-inverse filter.

Edge Detection: The sharp contrast between the road surface and the marked lines is the lane boundaries. This sharp contrast appears as edges in the images. Hence edge detection is a very important part of finding the lane boundaries. Edge detection also reduces the training time efficiency; in case the outline of the road is obtained properly from the image [2].

Among all the edge detector algorithms canny edge detector is the best and we have used the same in proposed method. This algorithm selects the thresholds automatically. However, this produced edge information that was far from original with a little change, the algorithm produced desired results. The amount of non-edge pixels of the highest and lower threshold is set to an optimum value. This process can be represented by the equation.

$$T_1 = \min_{\substack{x=0,1,....m \\ y=0,1,....n}} \{f(x,y)\} \, and,$$

$$T_2 = \max_{\substack{x=0,1,....m \\ y=0,1,....n}} \{f(x,y)\} \qquad (2)$$

Line Detection: The line detection is done by standard Hough transform with limited search space (Refer Fig 3).

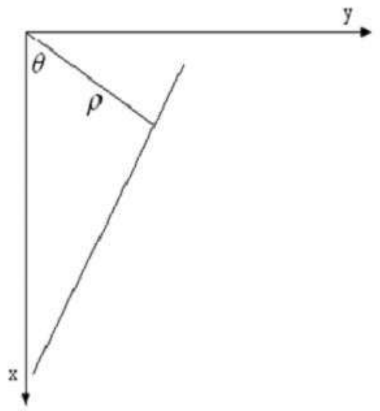

Fig. 3. Hough Transform searching methods

$\rho = x\cos\theta + y\sin\theta$ Over the range of -90° to 90° for θ and $\rho = \pm\sqrt{\#row^2 + \#cols^2}$

(3)

Any line which is not present in the region of interest then it is neglected. For example, if a horizontal is detected in the middle of a lane, then it probably isn't part of the lane. The restricted Hough transform is limited to the search area to 45° on both sides. The input image is divided into two halves to get both sides of the image of the lane. Then each of the left and right sides of the image is searched separately to the angle of 45 degrees for the dominant line which represents the lane marking. The horizon is calculated using both the Hough lines and projecting them at their intersection. This horizontal is called horizon.

Lane boundary-scan: This step takes the Hough lines, edge images, and the horizon as the input. The scan starts at the intersection of Hough lines and the image boundary at the bottom of the frame. Once the intersection is found, it is taken as the starting point for the left and right sides (refer Fig 4). From the initial point, the search begins with some pixels towards the center of the lane and then goes on to look for the first pixel until it reaches a certain number of pixels after the highest range.

For starting condition, the search will be from the left or right-most border, as depicted in below figure. Hence the highest range is the border itself [2].

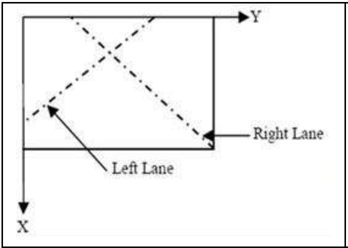	
Fig 4. a) Image coordinates	**b)** Actual Image

Lane Detection using Image Processing: Using image processing, data can be processed in real-time. Moreover, it can be employed on any road since the decision is taken based on live data. The basic concept is 'pixel summation' and is shown in Fig. 5.

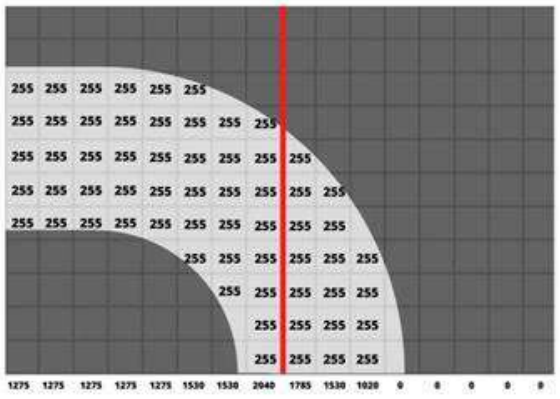

Fig. 5. Pixel Summation

The first stage in image processing is image thresholding (refer Fig 6). In image thresholding colored image is converted into a black and white image for detecting edges of the road.

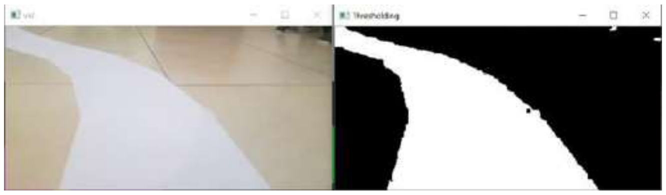

Fig. 6. Image Thresholding

The camera gives a front image of the road but from the front view, we get a solid angle not the original angle for road for getting the original angle of the road. We have used a technique called image warping. To get the top view of the image, 'image warping' is performed. (refer Fig 7).

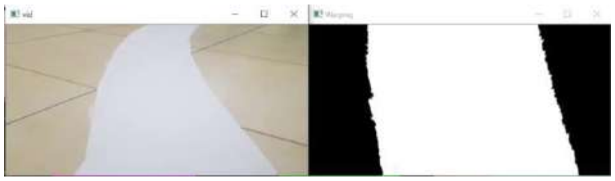

Fig. 7. Image Warping

After image warping we performed a histogram (Refer Fig 8) for finding the curve from the center of the road.

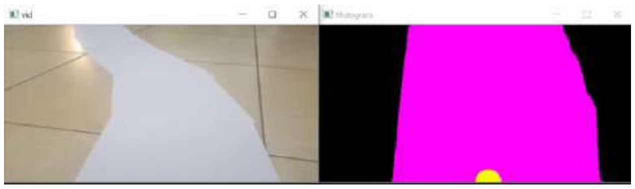

Fig. 8. Image Histogram

After image histogram we have done curve optimization for getting the curve value. We get a different value for a different curve. For the right curve we get the positive value of curve (Refer Fig 9), for the left turn we get the negative value of curve (Refer Fig 10) and for the straight line we get zero as the value of curve.

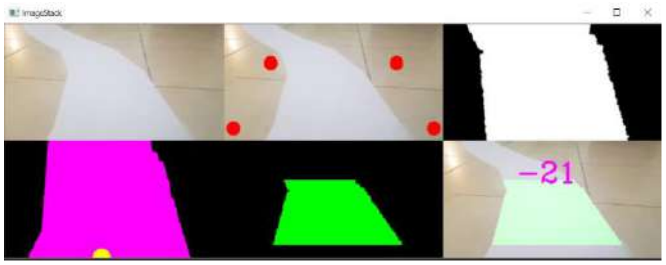

Fig. 9. Negative Curve Value for Left Turn

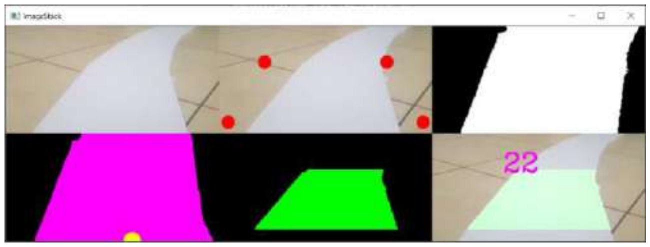

FIG. 10. Positive Curve Value for Right Turn

After getting the curve value, it is fed to controller for giving appropriate actions to the motor driver for controlling the motor for proper turn and acceleration [3].

Traffic Sign Detection

A self-driving car should detect the signs present on the road and take action accordingly. Various signs like speed limit, stop, speed breakers are very essential to be recognized and to be acted upon. So, in this particular phase, we have worked more on sign detection. To make the car to detect the traffic signs present on the road by itself by the input given from the camera and take the necessary action according to sign. For stop sign car must stop, for speed breaker sign car must decrease its speed for some time and then it will again be running at actual speed.

There are many ways of detecting traffic signs like using CNN, R-CNN, YOLO, etc. But in this scenario CNN cannot work efficiently because the problem with this technique is we have to put a sign exactly ahead of the camera which is not the case with self-driving car. Sign can be anywhere in the frame [3] [4].

To avoid this problem, we have used YOLO (You Look Only Once) algorithm. In this research work we have used Yolo Version 5 (V5) which comes with amazing features as it works on 140 fps which can be used for a real-time system. It comes with 80 inbuilt classes. As we have to detect the traffic sign we have used our customized dataset of traffic sign for training [5].

TESTING AND RESULTS

To understand the YOLO algorithm, first, we need to understand what is to be predicted. In Yolo, we predict the name of the class along with bounding boxes around the desired object. For that purpose, we have to do the following things.

- Creation of dataset
- Training of custom dataset
- Testing of a custom dataset

Following are the detailed steps for performing traffic sign detection using YOLO V5

1. First of all, we need to create our dataset. For that we use google extension for downloading a bunch of images. We have downloaded around 100 images for each sign and we have designed this system for 4 traffic signs.
2. After downloading the images we need to create bounding boxes around the signs in the whole image and for that, we used online software which will create a text file for an individual image that contains four coordinates of the bounding box of the targeted image, type of class which is shown in Fig 13.
3. We made two folders, one for the training dataset which contains around 85 images and a text file for each image, and another folder for the validation dataset which contains around 15 images along with a text file for each image.
4. We have used the Yolo V5 repository on Github for the training of custom datasets.
5. For training purposes, we have used Google Colab which comes with an online GPU for performing tasks very fast.
6. The dataset was trained with batch sizes of 10 and 150 epochs with YOLO V5 inbuilt weight file and dataset image size of 640 pixels.
7. After training, the trained file is generated which has an extension as .pt which is nothing but a trained file of our custom dataset.
8. In the training of our dataset, we got an accuracy of around 91% which can be further increased by increasing the number of epochs.
9. After training, we got some good results for steady images but that is not the case with a self-driving car. We need to recognize traffic signs for live camera i. e. for a real-time system. We run this algorithm on the local system having a controller with camera.
10. A .pt file which is generated after training was used for object detection.
11. For implementing the system on a local machine we installed certain python programming language libraries like matplotlib, NumPy, OpenCV-python, Pillow, PyYAML, scipy, torch, torchvision, tqdm, tensorboard, seaborn, pandas, coremltools, onnx, sci-kit-learn, Cython, pycocotools, albumentations and thop (Glenn-Jocher, 2020). After installing all the required library the python program was run on controller for recognizing traffic signs using a live camera.

12. After performing all the steps we get a result that has a live camera with sign detection with a bounding box around traffic sign which is shown in Fig. 11, 12, and 13.

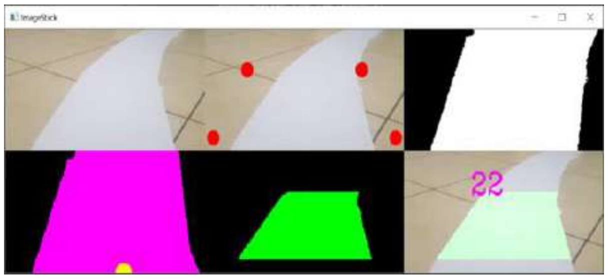

Fig 11. Result of lane detection

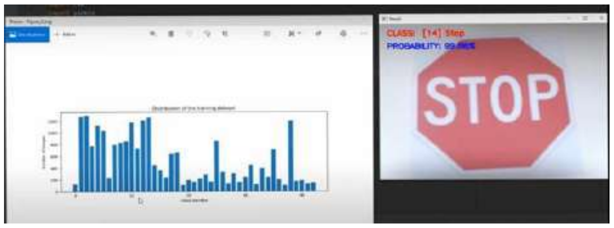

Fig 12. Traffic Sign Detection Using CNN

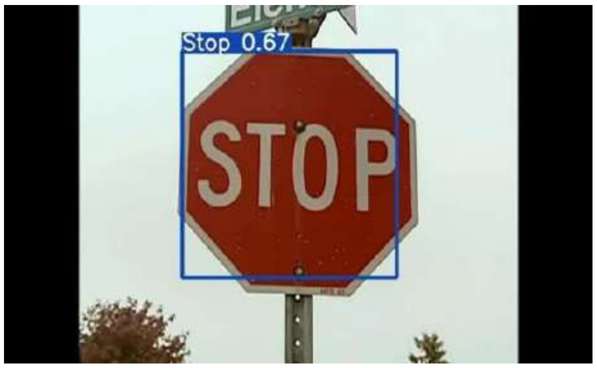

Fig 13. Traffic Sign Detection using YOLO V5

CONCLUSION

In this paper, a practical approach to make a prototype of a self-driving car is proposed. There are two ways for lane detection one is using CNN and the other one is using image processing. After taking a look at the drawbacks of lane detection using CNN, we can use Lane detection using Image Processing.

For Sign Detection also we have used two methods one is using CNN and other using YOLO V5. For CNN method image must be perfectly present in front of a camera. This is not the case with a self-driving car. To avoid this problem we have used YOLO V5 which is fast in response and has more accuracy than CNN.

With the help of Image Processing, YOLO V5, and Deep Learning we developed a successful model which performed as per requirement. Thus, the prototype was successfully designed, tested, and implemented.

LIMITATIONS AND FUTURE WORK

Like Tesla Motors, we also recommend drivers not to rely fully on a self-driving mode of car.

This Self-driving car cannot work satisfactory in following conditions:

- Foggy weather
- Improper illumination

- Invisible Lane marking of the road

A more robust system can be made using Nvidia's Jetson nano controller which is powerful enough to work with bigger-sized vehicles. The system can be equipped with GPS to make the driving experience even easier and reliable. There is some risk factor associated using the PWM technique for the steering control mechanism. The front wheels start spinning sometimes which consumes high power and it is also unsafe at corners. If we could design an advanced system that would eliminate this issue, it would make the system reliable and also it would make the overall design attractive and prevent the causes of fatalities.

REFERENCES

[1] Jain, A., 2018. Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi, and Arduino. International Conference on Electronics, Communication And Aerospace Technology, pp. 1630-1635

[2] Othman O Khalifa, Othman O Khalifa, Md. Rafiqul Islam, Abdulhakam.AM. Assidi, Aisha-Hassan Abdullah and Sheroz Khan., 2011. Vision-Based Road Lane Detection System for Vehicles Guidance. Australian Journal of Basic and Applied Science, Volume 5, pp. 728-738.

[3] Jamel Baili, Mehrez Marzougui, Ameur Sboui, Samer Lahouar, Mounir Hergli, J.Subash Chandra Bose, Kamel Besbes., 2017. Lane Departure Detection Using Image Processing Techniques. International Conference on Anti-Cyber Crimes, Volume 2, pp. 238-241.

[4] Xiong Changzhen; Wang Cong; Ma Weixin; Shan Yanmei., 2016. A Traffic Sign Detection Algorithm Based on Deep Convolutional Neural Network. International Conference on Signal and Image Processing, pp. 676-679.

[5] Aleksa Ćorović, Velibor Ilić, Siniša Đurić, Mališa Marijan, and Bogdan Pavković 2018. The Real-time Detection of Traffic Participants Using YOLO Algorithm. Telecommunications Forum, Volume 26, pp. 1-4.