EMVIA FINAL PROJECT

SELF-DRIVING CAR

Project Members:

1) Nikhil Divekar.
2) Omkar Prabhu.

Introduction:

- Significant progress in machine learning and machine vision over the recent years have paved the way for setting up the firm grounds for autonomous transportation. Self-driving car is the most growing domain in this field.
- Many big companies like Google, Tesla, Ford are working on building and testing different types of autonomous vehicles. Several US states have already passed the law to fast-tack this process of testing and deploying autonomous cars.
- However, despite the tremendous progress, there are instances of incorrect/unexpected behavior by autonomous vehicle and can lead to dangerous conditions like car crash, highway pile-up ultimately leading to possible loss of human life and damage to infrastructure.
- Generally, the autonomous cars are integrated with various sensors and actuators to react to stimuli. However, in this project we will focus on machine vision or image processing aspect of self-driving car. We use camera in the front of vehicle as well as back to extract the data and controlling the car according to the gathered data. Some powerful cameras like SuperCam3 4K stereo camera are being used at industrial level to get the high-resolution image and work according to that. This data can be sent to the embedded system like Jetson in our case for processing and manipulating the image according to our needs.
- The important modules of our project include lane detection, vehicle detection, road sign detection and pedestrian detection. We have integrated all of these modules together using multithreaded program so that every thread runs on different processor.
- In lane detection, we are finding the lanes in front of us and then giving the feedback for steering control for directions.
- In vehicle detection, the vehicles in the given frame are detected and if any vehicle is detected just in the front in the same lane, then speed control is performed.
- Road sign detection module comprises of detection of signs and behaving according to it. For demo purposes, we will be using stop sign detection.
- Pedestrian detection gets the number of pedestrians in a frame and their location for emergency stopping or wait at pedestrian crossing.

Lane detection module:

- The main motive of this motive is to detect the lanes through the front camera, analyze their orientation and giving feedback to the steering wheel as per the direction, whether left, right or stay on the course and also gives the angle through which steering wheel should be rotated. The basic algorithm used is using the Hough line transform.
- The working methodology used is explained below. First the video is converted to grayscale and HLS format. Mostly the lanes are yellow and white in color so we use masks to segregate the yellow and white color from the image. So, we also used bitwise or to get only yellow-white frame from the images. Next, we applied gaussian blur to remove the noise and then apply canny edge detector. Next, we set the region of interest right in the front of car. So once, we got the lines, we were also interested with the intersection. We applied mathematical principles to the point of intersection and also the slope of the lines. Slope can be used to eliminate unwanted lines (like direction arrows) which are linear but not necessarily edges of the lanes. We use HoughLinesP API to get the lines and plot it on the frame.
- Next, we have two horizontal fixed lines to get the midpoint of the lane edges. The midpoint is found by using the intersection of the two intersection points.
- Next, we get feedback comparing the midpoints on the two horizontal lines. The right or left is decide with the position of one midpoint respect to the other. The angle to turn is decided with respect to the midpoints.
- Also, when we have just one lane detected on just one side, then we have the code which keeps the vehicles aligned to the direction of the lanes.
- Also, to minimize the number of lines we have the code in place which only detects the line closer to the midpoint.

Vehicle detection:

- The main motive behind this module is the detection of vehicle and then acting according to the location of the vehicle detector, we decide whether to accelerate or decelerate. We used the Haar cascade algorithm to detect the vehicles. The vehicles when detected are marked with circle around them for visual perspective.
- The main working principle in the vehicle detection is that it uses pattern matching into smaller ROIs of the images to get the pattern it is looking for. detectMultiScale is the most important function.

- So, for the training purposes, we initially used the Matlab. In Matlab we created three folders namely positive, negative and test video. We converted the videos into frames initially. Next, we divide the frames into either positive or negative images depending on the whether it finds the object it is looking for in the frame. Next, in the positive folder, we have to mark the ROI manually. Next there is Matlab script where we input the session name, positive and negative folder to generate the xml file.
- The training cascade parameters like which cascade classifier is used and also the number of stages are the most important consideration. The co-ordinates and the radius of the circle are the results we get and then we get the circle around the vehicles.

Road-sign detection:

- The main motive of the module is the detection of the road signs and then acting according to the signs detected. We used the Haar cascade algorithm to detect the road signs.
- We decided to the create different xml for every different sign. For the demo purposes, we have decided to implement stop sign detection.
- So, it displays the "stop sign detected" whenever it finds the stop sign. Also, it displays "Stop now" when reached the stop point. This xml is passed as the second classifier.

Pedestrian detection:

- This is the last module we have implemented. Since there is no test data of pedestrians in the demo videos. We have implemented this video on another video.
- We have used HOG descriptor/SVM detector for detection of the pedestrians. Similar to the vehicle detection, the function called detectMultiScale is something very important to consider. We can tweak the parameters to get the better performance.
- It also outputs the number of pedestrians as well as their location in the image.

Integrating all the modules together:

- Integrating all the modules together was a difficult task. We have decided to use multithreading to get the faster throughput. Also, we set the attributes of three threads to point their affinity to the three cores. This way we are implementing SMP processing and load balancing.
- We used pthread to distribute the tasks among the thread. We have divided the services amongst the threads. So, all the modules above have their own unique thread.

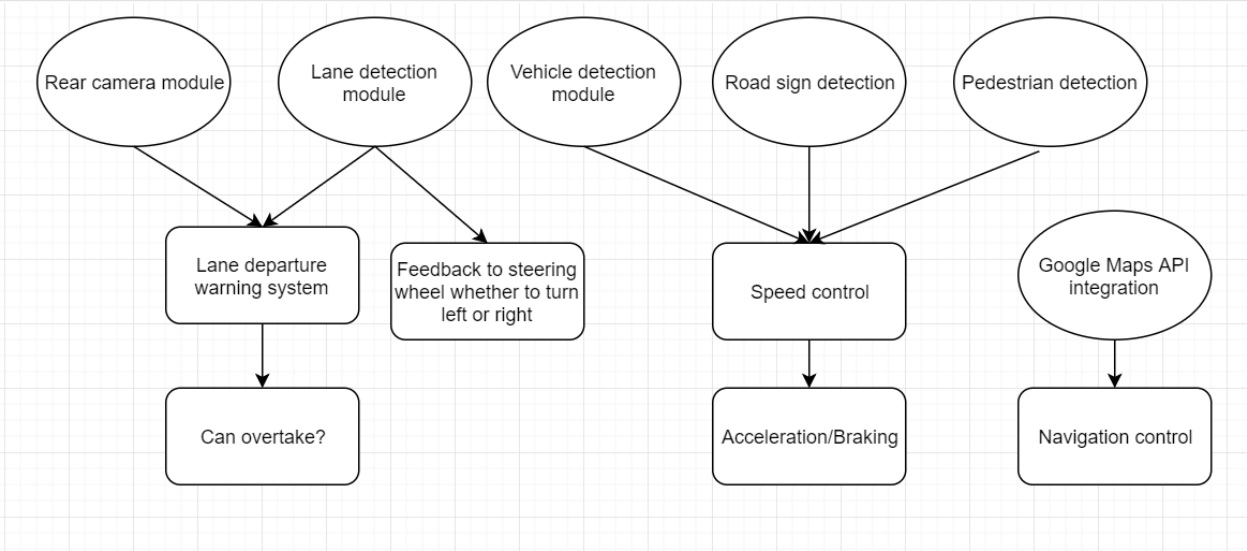
Machine Vision algorithms used:

1) Hough Line transform.
2) Image masking.
3) inRange algorithm
4) Bitwise operation on images.
5) Region of Interest.
6) Drawing circle and lines to get the midpoint mathematically.
7) Resizing Image.
8) Putting Text.

Machine Learning algorithms used:

1) Haar Cascade.
2) HOG descriptor.
3) Histogram analysis.
4) Pattern matching.
5) Writing MATLAB script to generate the xml file.
6) Training images.

Functional Block Diagram (with Future Scope):



Requirements:

Minimum:

1) Lane detection.
2) Vehicle detection.
3) Road sign detection.
4) Pedestrian detection.

Target:

1) Steering wheel feedback according to the lane detection system.
2) Speed control with vehicle detection.
3) Properly detecting road sign and acting according to the same.

We were able to cover all our minimum and target requirements.

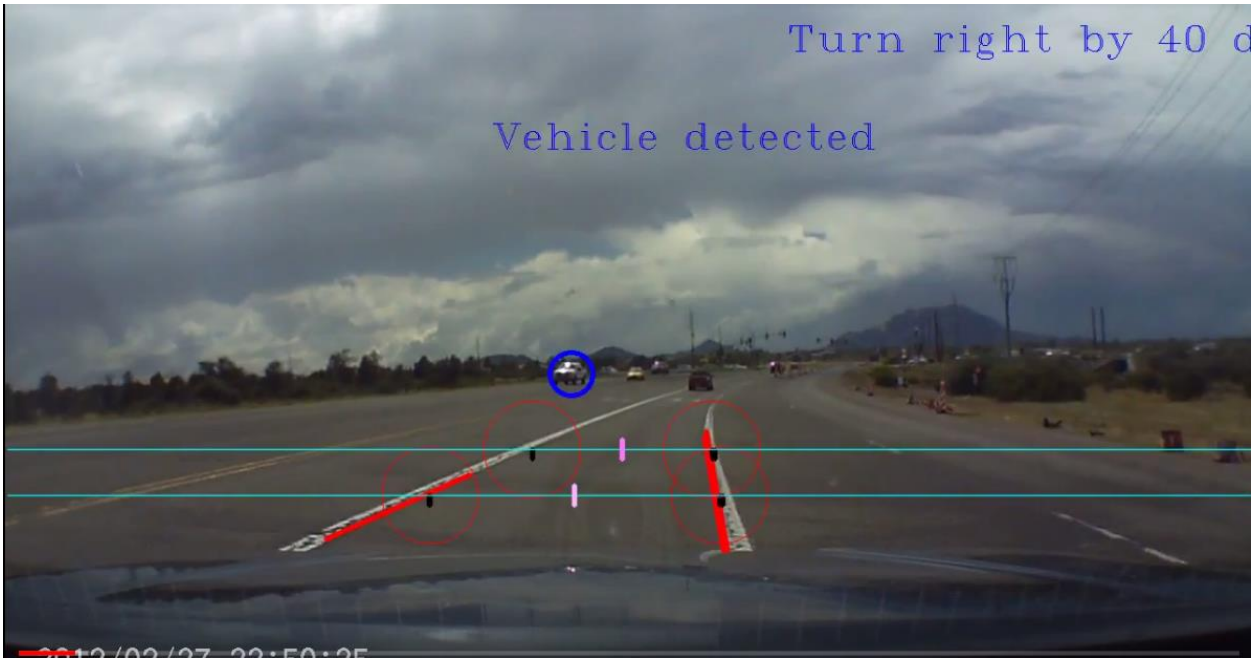
Future Scope:

1) Lane departure warning system.
2) Integrating the Google Map APIs for navigation.
3) Rear camera module to detect the vehicles behind and detect whether it’s ok to change the lane.

Screenshots:

In the above figure, we can see that lanes as well as vehicles are getting detected along with the required orientation of the vehicle (Stay on course).

In the above figure, we can see that lane is getting detected even on the curvature.



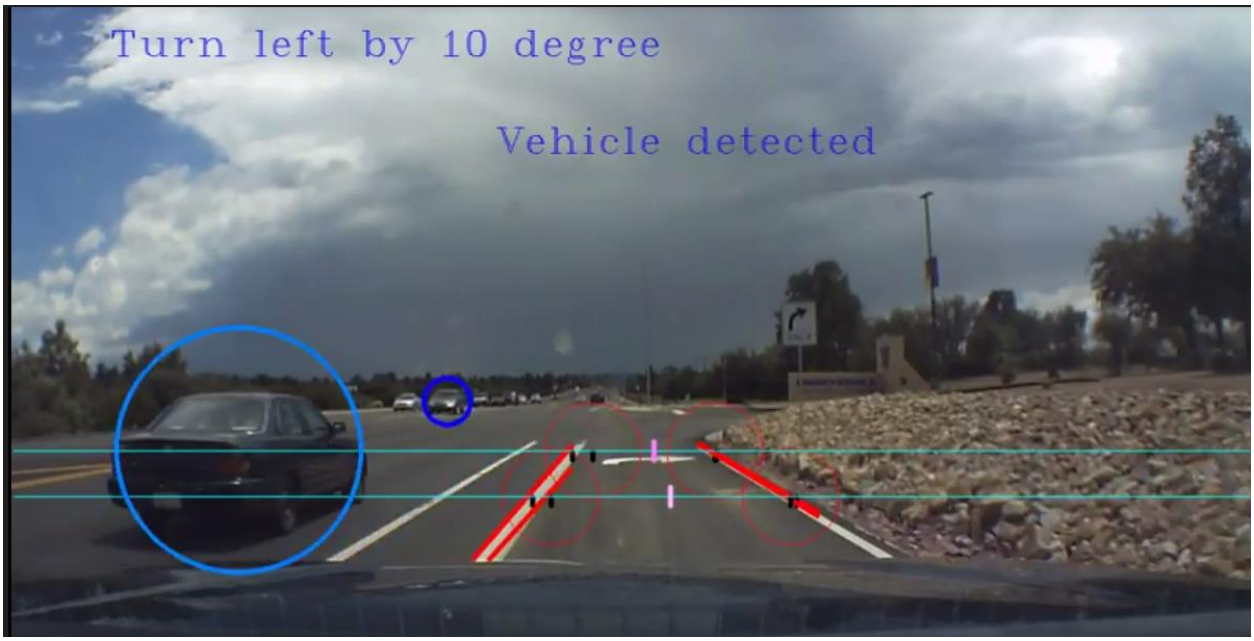
In the above figure, we can see the feedback given to the steering feedback based on the directions the lanes are curving. This depends on the position of midpoints marked in pink. Also, we get how much we need to turn i.e by 40 degrees. This number is proportional to the distance between two midpoints.

The above figure shows the example of vehicles getting detected along with the lanes. Also, text says moving vehicle to the left by 10 degrees. Also, since there is vehicle right in the front of our

vehicle, we get the text saying that “Vehicle detected in front, Thus, decelerating” and also shows the approximate timing in which it will decelerate.

The above figure shows there are quite few noise while taking the sharp turn and giving the amount of turn required. This is achieved by taking into consideration the two closest lines.

This figure shows midpoint getting reset when no lanes are detected.



The above image focusses on how properly the lanes are detected and not the right turn sign, even though it is kind of line also of the same color. We were able to achieve this using the slopes to reduce the number of lines.

This video also suggests that we were able to detect edges properly even during the sharp turn.

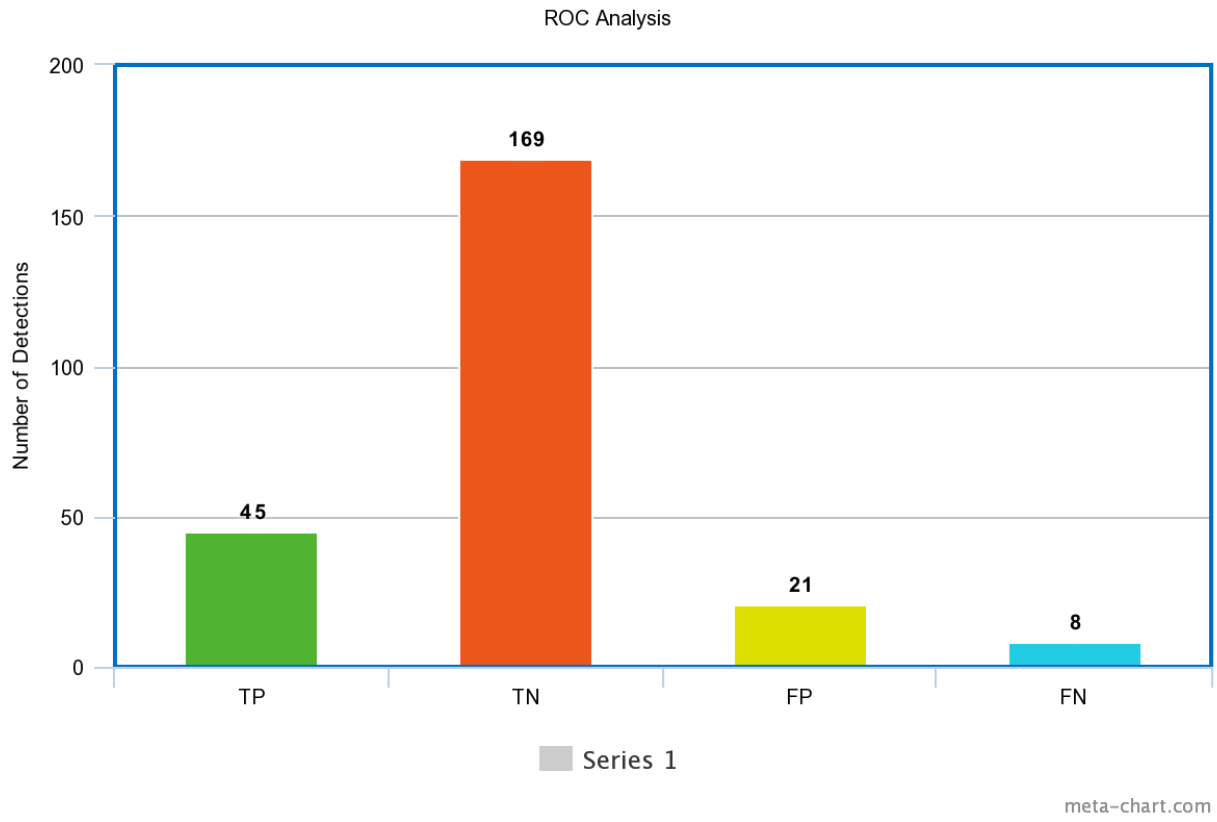
This image suggests that we were able to detect the yellow as well as white lines.

The above figure is the proof that it only detects the yellow lanes but not the yellow grass patch or irregular edges of road.

Analysis:

We performed the confusion matrix analysis on the vehicle detection. So basically, we got the values for True positive, True negative, False positive, False negative. We got the values object by object rather than frame by frame, which we feel is more accurate. We created the bar graph to display these values and we got the expected results. This is also called as ROC analysis.

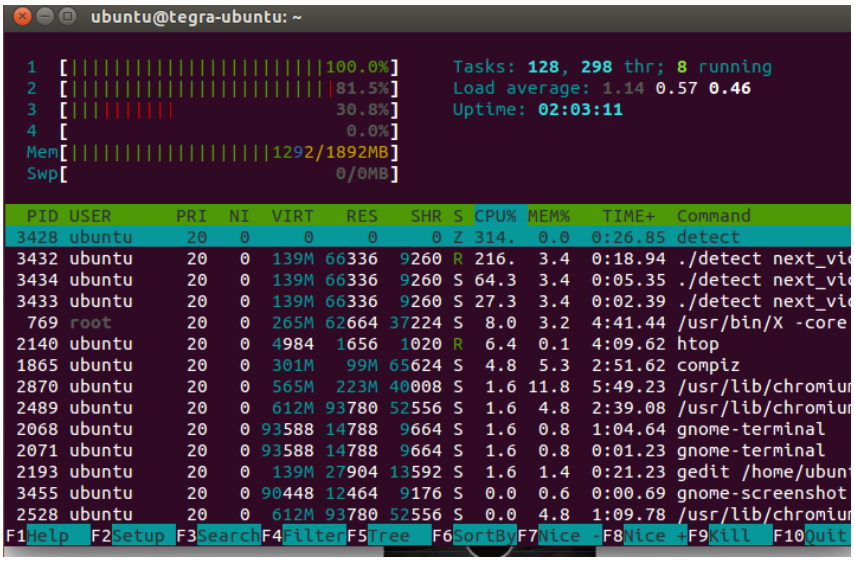
The bar graph we plotted for the given values is as given below.



As seen in the bar graph above, we see that we get large number of true negatives followed by true positives. Next comes false positive and then false negative. This is the expected result and we thus feel our vehicle detection is efficient one (Can be made more efficient with more number of training data).

CPU Usage:

We are using the core affinity assigned to each thread using their attributes. Hence basically, we have assigned every thread to the particular core. So we saw the load getting balanced across three cores. The htop command output is as shown below.



As shown, the process is getting distributed across three cores. This is known as load balancing or SMP processing.

Conclusion:

Overall, I think that we were pretty successful with the Self-driving car basic version. Lane detection, vehicle detection, road sign detection as well as pedestrian detection has the efficiency greater than 80%. Steering feedback is giving perfect output as to the direction but turn angle can be calibrated in the better way. Also, speed control according to vehicle detection had efficiency of almost 50% and can be improved. Stopping according to stop sign was very efficient and detected the stop sign correctly and also indicated “Stop now” at correct point.