Autonomous Vehicle System

Rohan Chopra
University of Texas At Arlington
Rohan.chopra@mavs.uta.edu
http://rohanchopra.uta.cloud

Abstract

This paper focuses on the performance of an autonomous vehicle system developed to run on devices with low computational power, lacking a Graphical Processing Unit (GPU). This system includes lane detection, speed calculation, steering angle prediction and object detection. The lane detector was implemented by following an advanced lane detector algorithm. This included calculating lane curvature, which was used to predict the steering wheel angle. For the object detection part, a pretrained YOLOv3 Model and TinYOLOv3 was used. YOLOv3 is the more accurate version of yolo but ran with 5 FPS on a non-GPU computer. Meanwhile, TinYOLOv3 is smaller, faster and less accurate version of Yolo but ran with 15 FPS on a non-GPU computer. The speed detection was calculated by using pixel per speed logic. Code was written in python 3.7 and used MoviePy and OpenCv[7] to implement certain aspects of the project. The system pipeline was used on a 1240 x 720 HD video input. Time metrics was calculated, time taken to detect an object and render for a single frame and the whole video.

Index Terms:- Object Detection , YOLO, Autonomous vehicle detection , Python , Non-GPU , deep learning

1. Introduction

Lately, autonomous vehicles and self-driving cars have been a topic of interest for many major companies. Since then many organizations have been actively working and researching on improving the self-driving car system. Currently, the world has achieved the 'level 3' of autonomous vehicles called conditional automation. The conditional automation allows a vehicle to drive itself under ideal conditions, a human driver behind the steering wheel. This also comes with certain limitations, such as driving under severe weather conditions, as limited-access divided highways at a certain speed etc.

An autonomous vehicle architecture is composed of collecting data from multiple sensors and feeding to the computer vision and neural network part of the architecture. The computer vision part processes the sensor data and detect multiple features from the input. This input is used to learn about the cars position in the surrounding while learning about its surrounding continuously. After the system is aware of its surrounding, it plans and does computations about certain parameters that need will define how the car will drive. Finally, the car takes control and starts driving itself. This process happens very quickly and is repeated continuously.

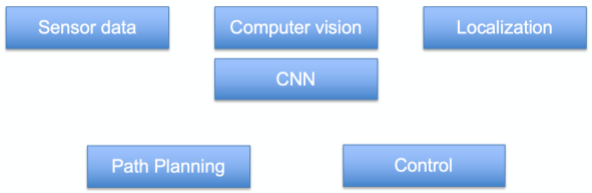


Fig 1. Self-driving car architecture

Many organizations have contributed towards the enhancement of autonomous driving. Since the emergence of deep learning and convolutional neural networks, computer vision and object detection are now being used to detect objects like cars, pedestrians, traffic signals and signs. This can be used to make predictions like lane detection [1], lane correction, lane keeping [3], steering angle prediction, blind spot detection, accident prediction etc.

Fig 2. Images frame passed through an object detector

TinYOLOv3 (an improved You Only Look Once model) being one of the state-of-the-art in DNN-based object detection methods in terms of both speed and accuracy. This is being used for real time object detection in video on embedded computing devices with limited computational power and limited memory [3].

Currently, the state-of-the-art object detection algorithms used in cars rely heavily on sensor output from expensive radars and depth sensors. Other techniques that are solely computer based require immense amount of GPU power and even then, are not always real-time, making the impractical for everyday applications. In this paper, a small autonomous vehicle system is implemented to solve this problem and the results are calculated. The system is amalgamation of multiple techniques (lane detection, speed calculation, object detection and steering angle prediction) to form a single system. This object detection is achieved by using a pretrained You only look once (YOLO) [4] deep learning model. The system was tested on two different versions of YOLO, YOLOv3 and tinYOLOv3. The results were recorded for the performance of the system as a whole.

2. Related work

There has been much work in developing a state-of-the-art object detection algorithm using deep learning and a standard RGB camera. Convolutional networks are the main idea behind building such object detection models. Using CNN and filters decrease the need multiple hidden fully connected layer, improving the time complexity while taking less memory. Also, they allow relational patterns to be drawn from an input which make object detection possible. There has been work on lane detectors to create a state-of-the-art detector model. This project uses these works to implement certain aspects of which directly impact the functioning of the algorithms.

2.1 YOLO

You Only Look Once (YOLO) [4] was developed to create a one step process involving detection and classification. Bounding box and class predictions are made after one evaluation of the input image. The fastest architecture of YOLO is able to achieve 45 FPS and a smaller version, Tiny-YOLO, achieves up to 244 FPS (Tiny YOLOv2) on a computer with a GPU. Confidence here is the probability of the box and is defined as:-

$$C = Pr(Object) * IOU_{truth}$$

The neural network architecture of YOLO contains 24 convolutional layers and 2 fully connected layers. YOLO is later improved with different versions such as YOLOv2 or YOLOv3 in order to minimize localization errors and increase mAP.

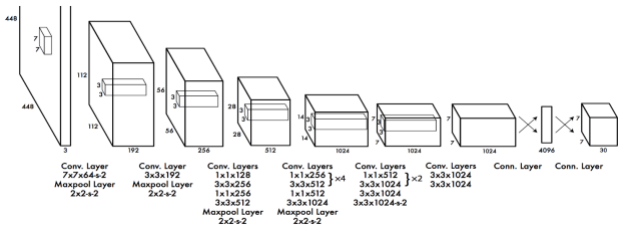


Fig 3. YoloV3 Architecture.

(Source : https://towardsdatascience.com/yolo-you-only-look-once-c63247e59d66)

2.2 Advanced Lane detector

Lane detector has multiple papers [1] [3] published using different methods. The paper referred for this project is called A Much Advanced and Efficient Lane Detection Algorithm for Intelligent Highway Safety [5]. The process includes using histogram and sliding window to find the lanes in an image. The algorithm follows following flow:-

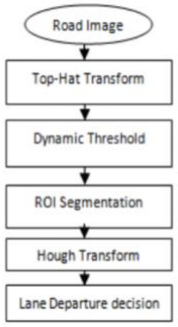


Fig 4. Advanced Lane detector algorithm.
(Source: [5])

2.3 YOLO LITE

This is a more efficient version of YOLO [6] which is faster and developed to run on a non-GPU device with a mAP of 30% on Pascal VOC architecture. The yolo Lite was created by removing certain layers from TinYOLO and performs well on non-GPU devices. Since the goal of this project is to run the system on low computational powered machines, this paper gives a baseline on how the a more optimized version of YOLO would perform under different computational conditions. The flexibility that YOLO has to offer enables it to perform efficiently under different computational conditions. This paper explains how training a YOLO Model to create a custom model can improve its efficiency.

2.4 Realtime embedded Object detection

Realtime object detection [2]refers to detecting objects and processing the frame while giving the output in real time. This process is very important for this project since the whole idea of self-driving car is based on real time object detection.

3. Problem Statement

The goal of my project is to implement, and test multiple autonomous vehicles features like lane detecting, steering wheel angle prediction, object detection and distance calculator as a single system, feeding it an input via RGB based camera. For testing and computing the overall performance of the system under low computational power in the real world.

3.1 Inputs

Used multiple input videos to test the system. The source of the videos are:-

1. Udacity highway video [8]
2. Recorded videos via a Tesla Model 3 dashcam.

The videos used were of 1240x720 resolution following a RGB color scheme.

3.2 Model

The YOLO model is pretrained CNN model which is trained on COCO data set. For this project pretrained YOLOv3 and TinYOLOv3 were used. The models were compiled on my own system and were exported as a native TensorFlow model.

3.3. Lane detection

Lane detection is one of the most important part of an autonomous vehicle system. In order to make decision and drive itself a car needs to be aware of its surrounding. Lane detection not only means detecting the number lanes but also detecting the lane lines so that the vehicle can stay within the lane while driving. This needs to be fast and accurate in order to perform well in real-time.

3.4. Object detection

In order to fully understand the environment, the system needs to be aware of all the objects like vehicle, board signs and other anonymous objects that it may encounter. This is equally important as lane detection. Self-driving cars are required to make fast real time decisions in order to function properly. So the object detection needs to be accurate and fast enough to support real time calculations.

3.5. Steering wheel angle prediction

A self-driving car needs to keep up with curvature of the road. So steering wheel angle needs to predict at every instant to ensure that car stays inside the lane. This system must be accurate and must do future calculations along the road for every 10 seconds ahead of time.

3.6. Speed Prediction

A self-driving car must know the speed and position of the other vehicles on the road. This is required for calculations involving the safety measures. The speed is helpful in stopping or slowing down in case any other car in the vicinity that might affect the car slows downs or decide to come in front.

3.5. Performance as a whole system

The above discussed modules need to work together in real time. The performance needs to be adequate enough for real deployment of the system.

4. Problem Solutions

This project is implemented using Python 3.7 programming language. Yolov3 by Darknet, TensorFlow 2.0 by google, MoviePy and OpenCV[7] libraries were also used. The code was created and executed on a MacBook pro 13-inch running MacOS 10.15 on intel core i5 6th generation with 8gb Ram and no GPU. The project uses two algorithms to implement all the problem statements. Lane detector and steering angle prediction was done using a single algorithm while object detection and speed calculations were done using later.

4.1 Lane detector

To create a lane detector, I used advanced lane detection algorithm [6]. This was done using Open computer vison (OpenCV) python library. The detection algorithm involves multiple processes. The process includes :-

4.1.1 Camera calibration and distortion removal

The first step involves calculation camera calibration matrix. The camera lens is convex so it may make our image distorted so these lines that are straight in real life may not appear straight in the image. So, they need to be corrected. OpenCV[7] library offers a convenient method called findChessboardCorners which will use black and white corner of the chessboard image to create a camera calibration matrix.

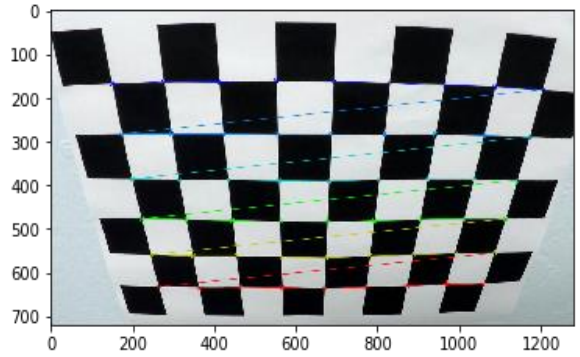


Fig 5. Camera Calibration

4.1.2 Color and Gradient Thresholding

The involves testing out different color schemes to find the best scheme that complements the white lines to display the best separation. HLS thresholding scheme works best with white lines. After that an edge detection algorithm like Sobel or canny operator is used. A higher value of the gradient denotes a white line. A 15x15 Sobel filter was used since it gave the best results.

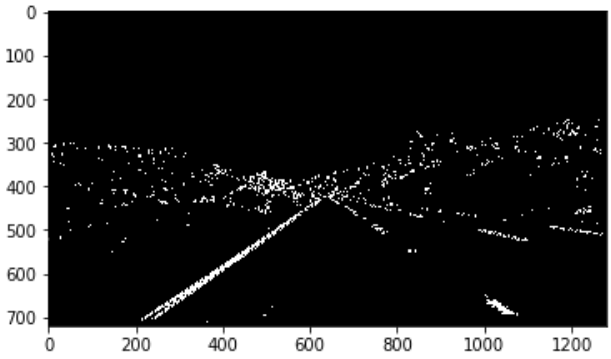


Fig 6. Gradient of an image after HLS thresholding

4.1.3 Perspective Transform

Perspective transform is used to convert the image to a bird's eye view image. OpenCv provides a convenient way to convert a 2d image to a bird's eye view image. The birds eye view is used to determine the coordinates of the trapezoid. A trapezoid is the polygon that is drawn on the rendered image to highlight the detected lane.

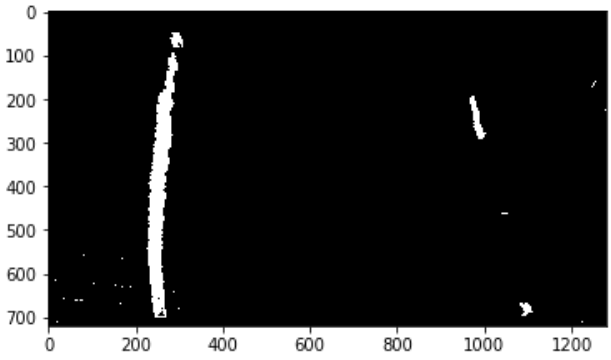


Fig 7. Perspective Transform of an image

4.1.4 Histogram and Sliding window

HOG features are used to find the two highest peaks along an image frame. These peaks signify the two detected white lines. This was further used to calculate the x axis point values for our lanes. The whole process is done over the image using the sliding window technique.

4.1.5 OpenCV for drawing

Finally, using OpenCV library the lane trapezoid is drawn over the image frame. OpenCV provides functions that can be used to draw over an image. These functions require x and y points. They were calculated using the above method.

Fig 8. Final render of the image frame

4.2. Object detector

To create an object detector, I am using Yolov3 neural network model. You only look once (YOLO) is a state-of-the-art, real-time object detection system. It is trained COCO train-dev set with a 0.5 IOU measure.

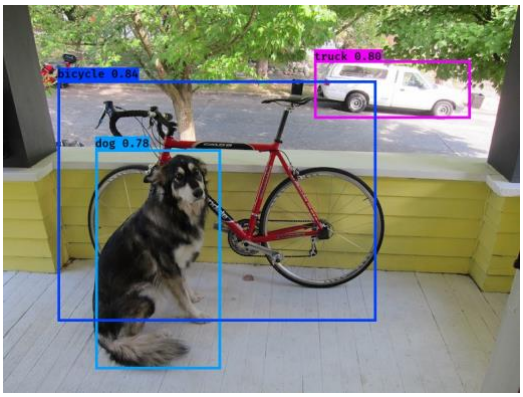


Fig 9. Object detection using YOLO. (Image is provided by YOLO source code)

For this project I used YOLOv3 and TinYOLOv3 models. The two models are compared based on their performance. The performance metrics used were accuracy and detection time.

Using moviepy, each frame of the input image was passed through the YOLO model. The detected objects are drawn over the image frame using the OpenCV library.

4.3. Steering wheel angle prediction

To accurately predict the steering wheel angle sensor data is required. Since, sensor data is not available radius of curvature of the road can be used to calculate the steering wheel angle.

Radius of curvature is calculated using the lane detection algorithm. It is the radius of curvature of the smallest circle that is tangent to left and right lines of the lanes. The calculations are done by fitting a polynomial over the lane lines. The formula used for radius of curvature is

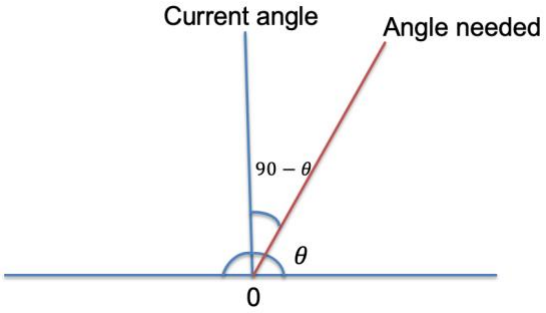
$$\frac{\left[1+\left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

Then using the radius of curvature [8] the formula used to calculate the steering angle is.

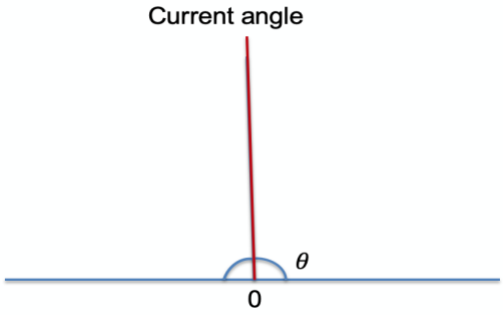
$$\theta^2 = r * (2*d)$$

I took the following assumptions for calculating the steering angle :-

- The steering angle is adjusted every second.
- The steering angle after every adjustment is 90 degree with respect to x axis.
- Negative angle means road is moving left and positive means towards right.



The current steering angle is assumed to be theta (Θ) as 90 degree with respect to x axis, shown by the blue line and the red line is the required angle of the wheel. The current angle theta (Θ) needs to be adjusted by by 90 - Θ if the road is moving towards the right side and 90 + Θ is the road is moving towards the left. After every angle adjustment I assume that the new angle is (Θ) 90 degree with respect to x-axis is the steering angle after it has been adjusted.



4.4. Speed Prediction

To predict the speed of the vehicles the object detection algorithm data is used. If the found objects are vehicles the algorithms works else, it ignores it. A pixel distance is calculated from the center of camera to the detected vehicle. This distance is then multiplied by the speed factor which is an assumption, set to 0.6 for this project. This is then added to the current speed of the self-driving car to give a relative speed of the vehicle.

let's assume that 's' is the number of pixels from the camera to the vehicle and the current speed of the self-driving car is 60 miles/h.
Then the calculated speed is :-

$$Speed = (0.6 \times s) + 60$$

5. Conclusion

To test my project, I ran the project pipeline through multiple different input videos. Multiple log files were created to record the time and accuracy for both of the algorithms. The time taken to detect and object for a single frame and for the whole video was recorded.

The below tables show the average time taken by our algorithms to process 2 of the input videos.

Time taken

Image res – 1280 x 720 – Input1 – 1 min

	Single frame (Avg for 10 frames)	Whole render
Object detector	6.1322 sec	58 min
Lane predictor	2.3725 sec	20 min

Image res – 1280 x 720 – Input2- 6 sec

	Single frame (Avg for 10 frames)	Whole render
Object detector	5.9322 sec	10 min
Lane predictor	1.900 sec	6 min

The results for a single frame object detection took an average of 6 sec to detect an object using YOLOv3 and render the output. It took 2 sec to detect the lane and render the output.
To draw the comparison between the two models I used, YOLOv3 and TinyOLOv3, multiple graphs were plotted that show the time taken to detect objects in the first 10 frames.

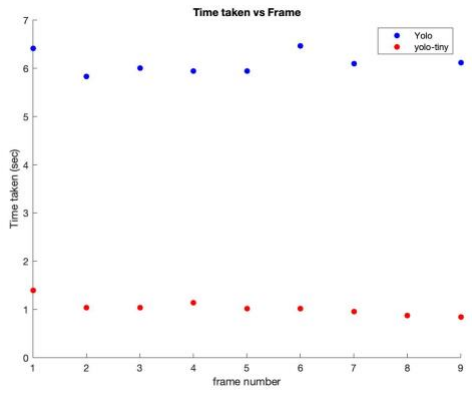


Fig 10. Graph between frame number and time taken to process it

The above graph was extended to create the same visualization for the whole system.

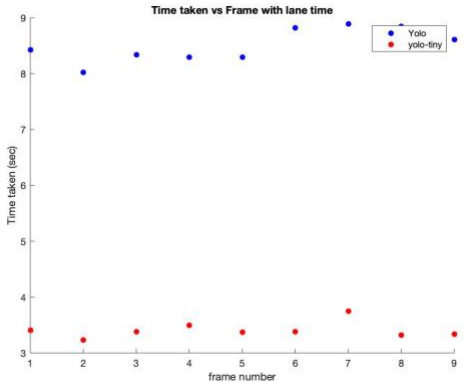


Fig 11. Graph between frame number and time taken to process it for the whole system

The results were quite impressive. The two main metrics that I used were accuracy and time. For a self-driving car to function you need a system to perform accurately in linear time.

5.1 Object detector Analysis

The object detector is accurate using both the YOLO and the TinYOLO model. However, the conventional YOLO outperformed the TinYOLO in case of the accuracy.
Both the models were fast enough for low computational machines. But TinYOLO is faster and more suitable for low computational machines.

5.2 lane detector Analysis

The lane detector was fast enough, and the results were satisfying. But in case of cracks or some other lines on the road the lane detector was not able to properly detect the lines. Instead it detects the cracks as the lane line. So the lane detector needs to be improved before deployment

Fig 12. Lane detector output with error.

5.3 Speed Prediction Analysis

The speed prediction algorithm performed well under the assumptions. This algorithm would work perfectly when the current speed of the car is fed via the sensor.

5.4 Angle Prediction Analysis

The angle prediction algorithm is based on the assumptions I made. This algorithm needs to be compared with real-time sensor data to test its accuracy.

5.5 Result Images

Fig 13. Object detection output

Fig 14. System Output

5.6. Scope of this project

The project tries to combine individual works into a real-world system to test out the performance. The project can be further extended to reach multiple results. Some of them are: -

1. Improving the performance of this kind of system in the real world.
2. Using RNNs (reinforcement learning) to predict the objects future movement thus avoiding accidents.
3. This can also be used to create a tool to remove skepticism that people have over the safety of autonomous vehicles. The low cost-hardware can be s old as a product to demonstrate the working of self-driving cars in their own manually driven car.
4. It can be used to collect real user data.
5. It can be used to improve performance of RGB based autonomous driving in severe weather conditions.

References

[1] Chen, Zhilu, and Xinming Huang. "End-to-end learning for lane keeping of self-driving cars." In 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 1856-1860. IEEE, 2017.

[2] Shafiee, Mohammad Javad, Brendan Chywl, Francis Li, and Alexander Wong. "Fast YOLO: A fast you only look once system for real-time embedded object detection in video." arXiv preprint arXiv:1709.05943 (2017).

[3] Cao J, Song C, Song S, Xiao F, Peng S. Lane Detection Algorithm for Intelligent Vehicles in Complex Road Conditions and Dynamic Environments. Sensors (Basel). 2019;19(14):3166. Published 2019 Jul 18. doi:10.3390/s19143166

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779– 788.

[5] Sharma, Sachin & Shah, Dharmesh. (2013). A Much Advanced and Efficient Lane Detection Algorithm for Intelligent Highway Safety. Computer Science & Information Technology. 3. 10.5121/csit.2013.3106.

[6] R. Huang, J. Pedoeem and C. Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 2503-2510, doi: 10.1109/BigData.2018.8621865.
[7] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc."
[8] http://apps.usd.edu/coglab/schieber/pdf/AutomobileSteering.pdf