Empirical Comparison of Lane Following and Sign Recognition techniques on a Raspberry Pi car

Eisa Adil
School of Engineering & IT
Manipal Academy of Higher Education, Dubai
eisaadil@outlook.com

Ms. Jimcymol James
School of Engineering & IT
Manipal Academy of Higher Education, Dubai
jimcymol.james@manipaldubai.com

Abstract— **This research paper compares different techniques for lane following and sign recognition, implemented on a Raspberry Pi car, based on the time taken for processing. For lane following, a widely used CNN architecture is compared to a purely Computer Vision based approach; while for sign recognition a classical CNN architecture is compared to more modern object detection algorithms such as YOLO, SSD and R-CNN. Contrary to conventional research work which takes place on real self-driving cars or simulators, this paper emphasizes prototype cars powered by microcomputers that anyone can assemble and implement functionalities on.**

Keywords—Lane Following, Sign Recognition, Object Detection, Convolutional Neural Networks

I. Introduction

In recent years, self-driving cars have gained prominence due to advancements in Artificial Intelligence and increased computational power. Various automobile corporations have invested copiously in the field of autonomous vehicular engineering.

This is due to benefits such as increased affordability, lesser chances of accidents, lesser congestion, increased customer satisfaction, and reduced criminal activity. With an increase in autonomy, cars can be more fuel-efficient; decrease congestion which in turn reduces traffic collisions; provide increased movability for all age groups; relieve passengers from mundane tasks such as navigation and driving; alleviate parking woes; and reduce crime.

However, testing of features such as lane following and sign recognition has posed a challenge on an actual car. Even though simulators are widely used, implementation of features on a microcomputer-driven car can give the self-driving car functionality a more realistic outlook.

Therefore, for this research paper, a Raspberry Pi driven car is assembled (Fig. 1), and the various techniques for lane following and sign recognition are tested, and the time (in milliseconds) it takes to process and compute one frame is recorded. This would help in choosing the right approach while undertaking a lane detection or sign recognition problem for a microprocessor-controlled car.

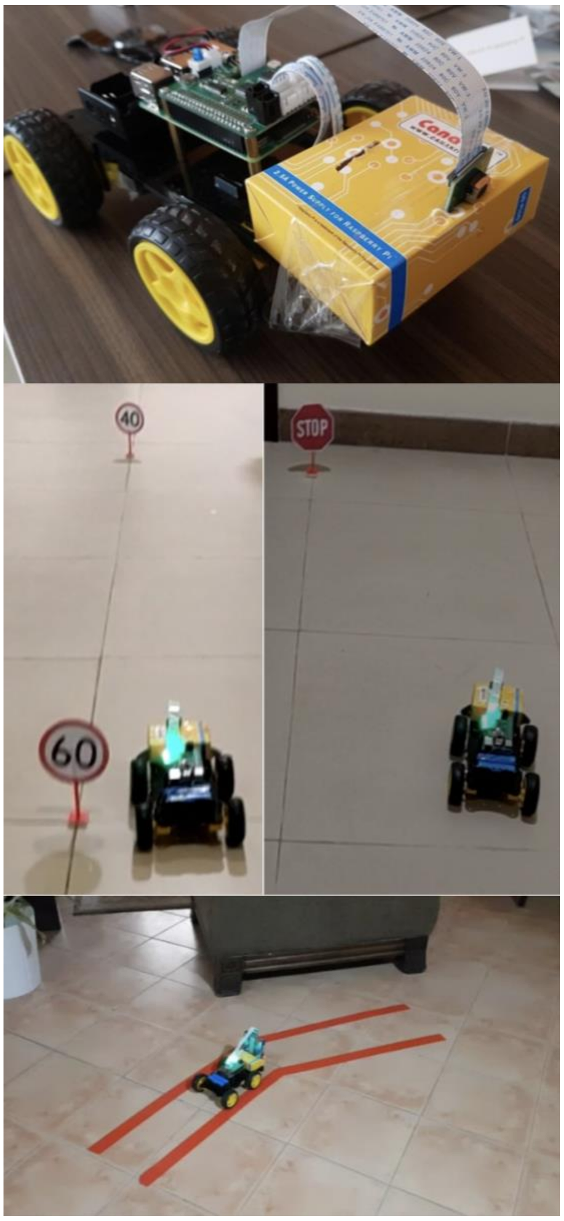


Fig. 1: Assembled Raspberry Pi car with Lane Following and Sign Recognition functionality

II. Literature Review

A. Lane Following

Convolutional Neural Networks (CNNs) are used to derive steering angles from the front camera's input image. This is called an end-to-end system, as intermediate stages such as lane detection, path planning, and control are not explicitly defined [1]. CNNs are especially useful while recognizing images, due to their spatially invariant nature, meaning that the classification of the object can be recognized regardless of shifts in its position within the image. In other neural networks, parameters need to be learned for each pixel, whereas using convolutional filters helps reduce parameters to be learned by scanning the image one group of pixels at a time. Ungurean [2] compares a simple feed-forward neural network with a variety of CNN's such as NVIDIA's PilotNet [1], and their performances are graphed. B. Huval *et al.* [3] evaluates CNN's performance on highway imagery and proposes enhancements to increase accuracy. Alternatively, C. Berger [4] proposes a method in which situation-specific programming is used, for example, an algorithm to have a balanced distance between the left and right-hand-side lanes. A. A. M. Assidiq *et al.* [5] utilizes geometric computer vision techniques to detect lanes by restricting the search area and using hough transforms which group adjacent high-intensity spots together.

B. Sign Recognition

S. Jung [6], proposes a method in which various traffic signs are trained on a LeNet-5 CNN architecture. Color-based segmentation and Hough Transform is applied to propose regions that have a higher likelihood of containing a traffic sign. On the other hand, A. Shustanov *et al.* [7] compares various CNNs experimentally to each other using the TensorFlow GPU library, and combines preprocessing and localization from previous works, to propose a method that classifies 16 traffic signs with very high accuracy. A. Uçar *et al.* [8] classifies the traffic signs after discriminative patches were identified with the CNN AlexNet architecture and Support Vector Machine (SVM). A. De La Escalera [9] describes a proposal for the perception of traffic signs using Computer Vision techniques. This algorithm is divided into two phases -- detection and classification. In the detection phase, the color and corners of the shape of the sign are chosen as features, to propose candidate regions of where the sign is most likely present. For classification, the candidate region's traffic sign is used as the input pattern for a feed-forward neural network (FFNN). Several network architectures with a varying number of layers and nodes are trained and compared.

III. Lane Following

For Lane Detection, NVIDIA's CNN approach [1] is compared to a Computer Vision approach described by S. Jung *et al.* [6]. After necessary adjustments, these approaches are implemented on a Raspberry Pi in order to analyze their performance.

A. CNN Approach (PilotNet)

The CNN approach is widely used in image recognition tasks because the convolution operation captures the two-dimensional nature of images (due to spatial invariance). Since the image is scanned in its entirety by the convolutional filters, lesser parameters need to be learned, which reduces computational expense [1]. The network architecture (Fig. 2) consists of nine layers, including a normalization layer, five convolutional layers, and three fully-connected layers.

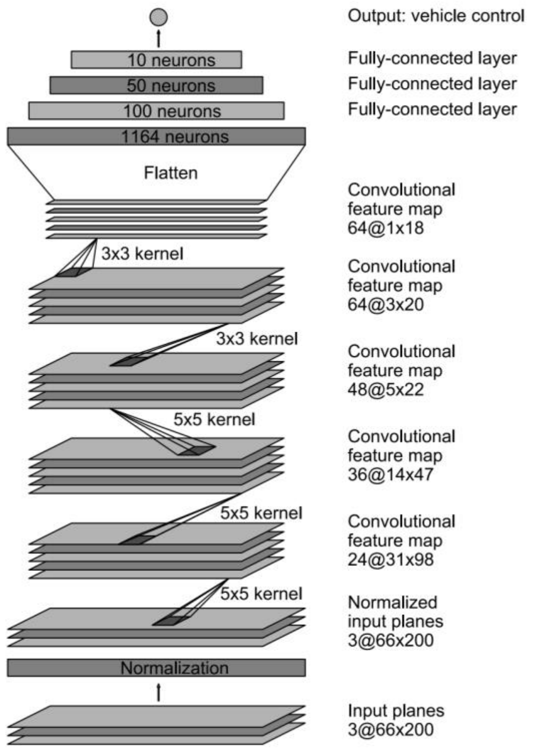


Fig. 2: NVIDIA PilotNet CNN for Lane Detection [1]

It contains about thirty layers in total, not a very deep model by today's standards. The input image to the model (bottom of the diagram) is a 66x200 pixel image, which is a pretty low-resolution image. The image is first normalized, then passed through 5 groups of convolutional layers, finally passed through 4 fully connected neural layers and arrived at a single output, which is the model-predicted steering angle of the car.

Since this is an end-to-end system, it is impossible to determine which layer of the neural network extracts features, and which serves as a controller for the car [1]. The CNN has about 250,000 parameters and 27,000,000 connections [10].

This steering angle that the model has predicted is then compared with the desired steering angle for the video frame, and the error is fed back into the CNN training process via backpropagation. As seen in the diagram, this process is repeated in a loop until the loss (Mean Squared Error) is low enough, meaning the model has learned how to steer reasonably well. Indeed, this is a pretty typical image recognition training process, except the predicted output is a numerical value (regression) instead of the type of an object (classification).

B. Computer Vision Approach

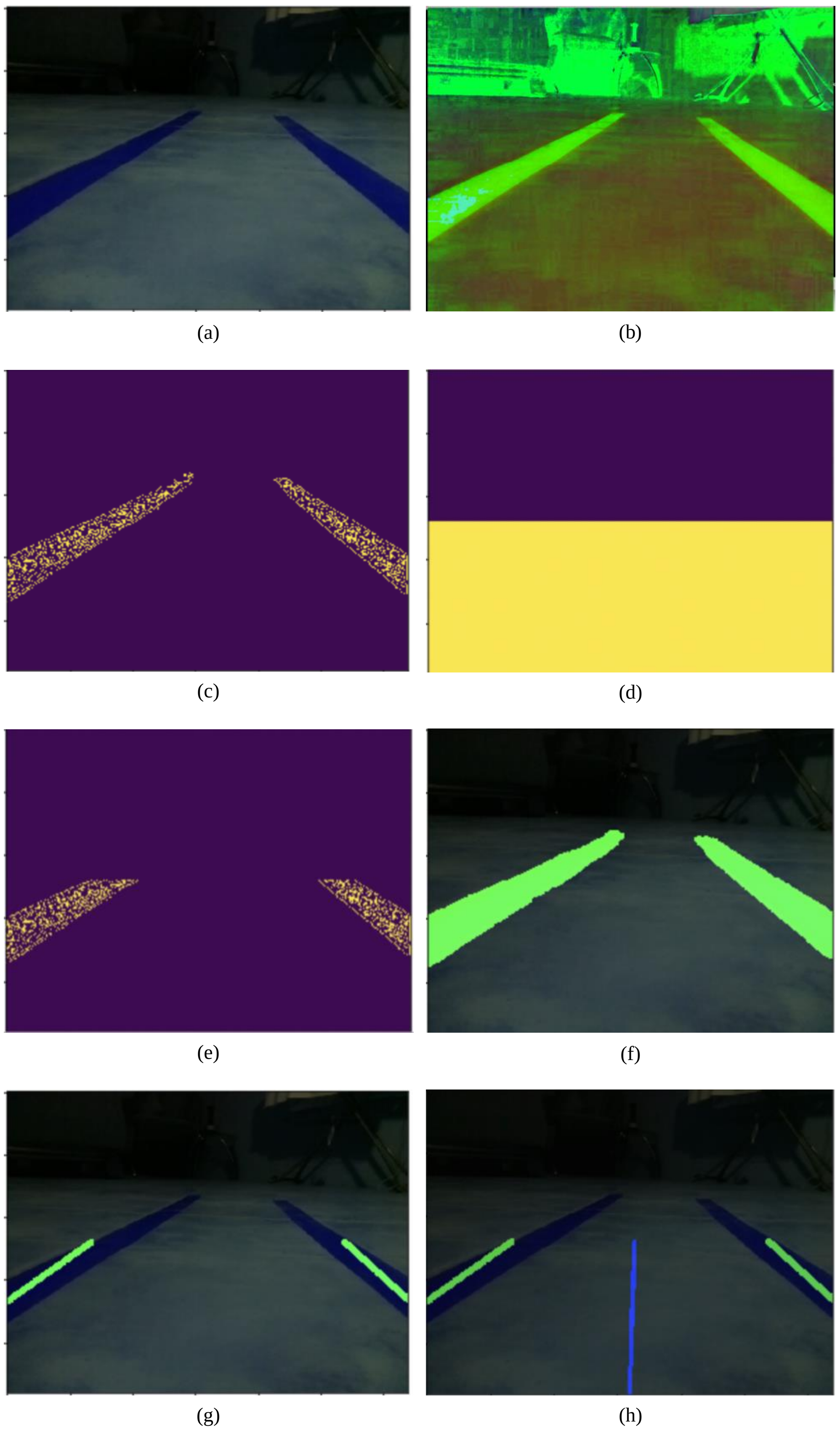


Fig. 3: Experimental process of Computer Vision Approach on Raspberry Pi Car

Our input image is from the camera of the Raspberry Pi car. The lanes are shown in Fig. 3 (a), as blue lanes. However, it must be noted that the lanes are actually red but the OpenCV framework renders the image's red pixels as blue due to BGR format as opposed to RGB.

Assuming the lane lines that the car has to follow are red, the first thing to do is to detect all the red areas on the image. For object detection, the HSV (Hue-Saturation-Value) color space is widely utilized because it works in a wide variety of luminosities. Different thresholds of Hue, Saturation and Value can be pre-determined in order to filter the required color. Red has a very high hue (greater than 160), therefore the thresholds are set accordingly. In Fig. 3 (b), it can be seen that the red pixels of the image are highlighted. This process is called HSV filtering.

Canny Edge Detection is applied for detecting edges in the red mask, in order to obtain a few distinct lines that represent the red lanes [11] as shown in Fig. 3 (c). However, it must be noted that only the lower half of the camera stream is relevant to us, as that's the immediate lane's region. Therefore, we must isolate our region of interest as shown in Fig. 3 (d) and (e). A mask of zero-array is placed to "crop" the top half of the image out. This is very useful in removing objects of a similar color as the lane that are in front of the car, as that may distort the intended lane line.

To find straight lines from a group of pixels, Hough Transform is used. The OpenCV function *HoughLinesP* passes multiple lines through all the red pixels and returns the most likely set of lines, subject to certain minimum threshold constraints, which results in a uniform cluster of line segments as shown in Fig. 3 (f).

Now that we have extracted multiple line segments, we must combine them to form two distinct lines– left lane and right lane. To distinguish between the two groups of line segments, we can use the value of their slopes. All line segments belonging to the left lane line should be upward sloping and on the left side of the screen, whereas all line segments belonging to the right lane line should be downward sloping and on the right side of the screen. Once the line segments are classified into two groups, we take the average of the slopes and intercepts of the two groups of line segments, in order to get the slopes and intercepts of left and right lane lines. The left lane and right lane can be visible in Fig. 3 (g).

Finally, to find the angle that the car must be steered on, the angle between the two lane lines needs to be computed. In Fig. 3 (h), a 90-degree angle indicates that the car should steer forward, whereas 45-89 and 91-135 degree angles indicate left and right turn respectively.

IV. Sign Recognition

First a method proposed by S. Jung *et al.* [6] is detailed. It is a fusion of Computer Vision algorithms such as HSV Thresholding, Canny Edge Detection, and Hough Transform; and a LeNet-5 CNN model. This method is compared to modern object detection algorithms such as SSD [12], YOLO and R-CNN [13]. After necessary adjustments, these approaches are implemented on a Raspberry Pi to analyze their performance.

A. *Filtering + LeNet Architecture Approach*

First, the image is blurred, i.e., its resolution is reduced, in order to decrease the computational time of processing the image, as lesser pixels have to be convolved.

Then traffic sign candidates in the image need to be extracted. This is done using Selective Search, which is used as a preprocess for object detection in various modern algorithms.

Alternatively, we can use HSV Filtering which uses threshold values for expected colors in the object. For example, a stop sign would have a large number of red pixels clustered together, which indicates a high hue value. After using Canny Edge Detection to find the outlines of the objects in the image, we use Hough Transform to detect the objects that have a particular shape, like a stop sign is octagonal. These combinations of functions would give us a viable filter with a reasonable number of bounding boxes to classify, as shown in Fig. 4.

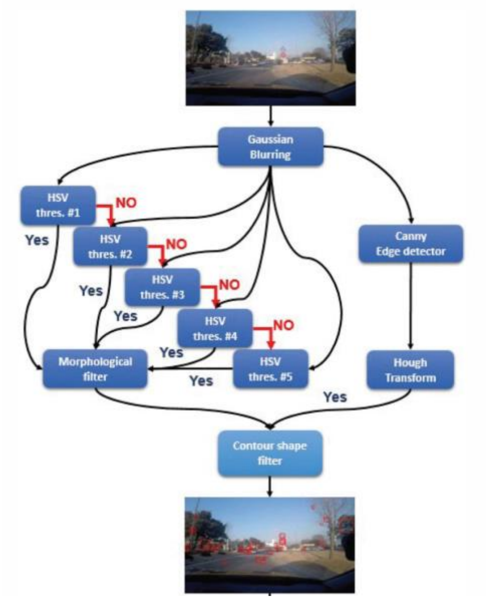


Fig. 4: Extract candidate traffic signs by color and shape [6]

Each bounding box of candidate traffic signs is supplied to a Le-Net 5 CNN model, as shown in Fig. 5. The images are resized into 100x100x3 dimensions containing an image of 100x100 pixels and RGB (x3) channels. This serves as the input image to the CNN, whose output layer has nodes designated for each traffic sign classification in one-hot encoding.

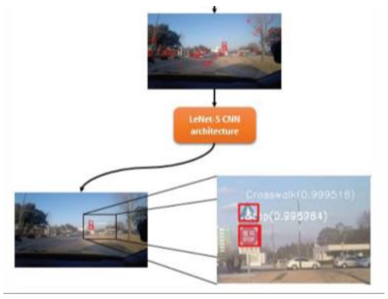


Fig. 5: Input bounded boxes to LeNet-5 CNN architecture [6]

The CNN is constructed based on LeNet-5 model, as shown in Fig. 6. However, S. Jung *et al.* [6] has modified some nodes. In two convolutional layers, a 5x5 size receptive field is used and it strides the input image one pixel at a time (with stride 1). After each convolutional layer, max pooling layer resamples the data and reduces the size of the data by using a 2x2 kernel with stride 2 arrangement.

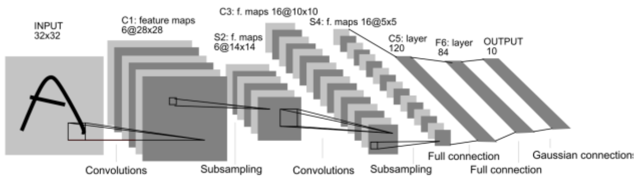


Fig. 6: Internal specifications of LeNet-5 CNN [14]

B. *Modern Object Detection Models Approach*

Object detection is a widely discussed problem in computer vision and deep learning. There are two components in an object detection model – base neural network and detection neural network. An object detection model is usually named as a combination of its base network type and detection network type, for example – “MobileNet SSD” [15].

First, base neutral networks are CNNs that extract features from an image, from low-level features, such as lines, edges, or circles to higher-level features, such as a face, a person, a traffic light, or a stop sign, etc. MobileNet and LeNet are examples of base neural networks. MobileNet is a CNN architecture designed specifically for mobile devices with very limited computing power [16].

Then, detection neural networks are attached to the end of a base neural network, which are used to simultaneously identify multiple objects from a single image with the help of the extracted features. Some of the popular detection networks are R-CNN (Region with CNN features), SSD (Single Shot MultiBox Detector), Faster R-CNN, and YOLO (You Only Look Once), etc.

Original R-CNN method works by cropping externally computed candidate box proposals [17]. These box proposals are computed using an algorithm like selective search, which would generate, say, 1500 regions to work with [18]. A neural network classifier is then run on these regions of interest, to extract a feature map.

Fast R-CNN passes the entire image directly into the neural network classifier for feature extraction, only requiring one convolutional operation per image [17]. Therefore, 1500 regions don’t have to be fed into the neural network for it to work, reducing its computational cost. Both R-CNN and Fast R-CNN use selective search to derive candidate regions, which is a heavy and time-intensive process slowing down the inference time of the network [19].

Faster R-CNN eliminates selective search, which means there are no externally computed box proposals [18]. These box proposals are instead computed directly within the network and are therefore class-agnostic [17].

YOLO (You Only Look Once) [20] differs from the aforementioned approaches in the way that it divides an image into a grid of bounding boxes, and calculates the probability of the class of an object in each bounding box simultaneously. Just like Faster R-CNN, the whole image passes through the CNN only once [18]. Therefore, it is relatively faster but falters when the objects to detect are small, because it can detect only a specific kind of class in a grid [21].

Single Shot Multibox Detector (SSD) [12], though more similar to R-CNN, is different from it because it does not have a separate region proposal step. Hence, calculating boundary boxes and classification of objects takes place in the only step of the process [19]. SSD is faster than R-CNN but comes at the cost of lesser precision [13].

Transfer Learning is the reuse of a previously trained model on a new problem [22]. The base neural network (MobileNet) is generic for all images, and pre-trained weights of the model are available online and can be used for detection of almost any real-world object. Upon supplying a few labelled images of our traffic signs (using the GTSRB dataset), we can train our model to classify a group of traffic signs in only a couple of hours.

V. Car Assembly

The Raspberry Pi car was assembled using the UCTRONICS K0073 kit, which had all the components needed to make a vision based semi-autonomous vehicle. The car was controlled from a laptop using its WiFi module. The camera stream is sent to the laptop, where inference of models and processing is done, and steering angles are relayed back, as shown in Fig. 7. Therefore, the computationally expensive work happens on the laptop only, as the Raspberry Pi has relatively weak processing power.

Microcomputer: Raspberry Pi Model 3B+

Laptop processor: 3.3 GHz Intel Core i7

Laptop graphics card: Intel Iris Graphics 550 1536 MB

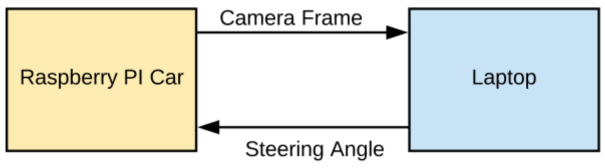
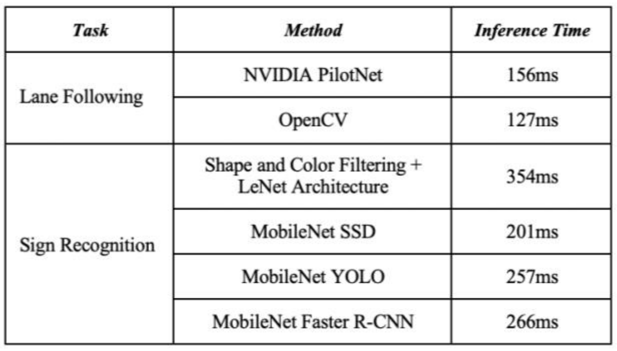


Fig. 7: Communication between Raspberry Pi and Laptop

VI. Experimental Results

The inference times of the aforementioned models and techniques have been listed in Table 1. Inference time indicates the amount of time it takes to process an image received on the laptop from the Raspberry Pi. It does not account for the duration it takes to receive the camera stream or send the steering commands. This is because these things are contingent on the network and particular car model only, whereas these inference times would be relatively consistent on most modern computers. It should be noted, however, that the time it takes for training the CNN models would vary greatly on different machines, particularly due to GPU capability.

TABLE I. Processing Times of Different Methods



Task	*Method*	*Inference Time*
Lane Following	NVIDIA PilotNet	156ms
	OpenCV	127ms
Sign Recognition	Shape and Color Filtering + LeNet Architecture	354ms
	MobileNet SSD	201ms
	MobileNet YOLO	257ms
	MobileNet Faster R-CNN	266ms

VII. Conclusion

The results of the experiment indicate that NVIDIA's PilotNet approach for Lane Following is considerably better than using classic OpenCV approaches. It can also be concluded that modern object detection algorithms such as R-CNN, SSD and YOLO are much faster than the approach described by S. Jung et al. [1], which involves manual filtering of candidate regions and then their classification using the Le-Net 5 CNN model.

Acknowledgment

I would like to thank my supervisor Prof. Jimcymol James for giving me the chance to explore such an interesting field of machine learning and for always pointing me in the right direction, whenever I was starting to get lost. I am also very grateful to Prof. Zaheeruddin Ahmed, who oversaw my writing at Manipal Academy of Higher Education, Dubai and provided valuable insight and resources which made this work possible.

References

[1] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," pp. 1–9, 2016.

[2] D. Ungurean, "DeepRCar: An Autonomous Car Model," p. 67, 2018.

[3] B. Huval *et al.*, "An Empirical Evaluation of Deep Learning on Highway Driving," pp. 1–7, 2015.

[4] C. Berger, "From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation," vol. 5, no. May, pp. 63–79, 2014.

[5] A. A. M. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," *Proc. Int. Conf. Comput. Commun. Eng. 2008, ICCCE08 Glob. Links Hum. Dev.*, pp. 82–88, 2008.

[6] S. Jung, "Real-time Traffic Sign Recognition System with Deep Convolutional Neural Network," pp. 31–34, 2016.

[7] A. Shustanov and P. Yakimov, "CNN Design for Real-Time Traffic Sign Recognition," *Procedia Eng.*, vol. 201, pp. 718–725, 2017.

[8] A. Uçar, Y. Demir, and C. Güzeliş, "Object recognition and detection with deep learning for autonomous driving applications," *Simulation*, vol. 93, no. 9, pp. 759–769, 2017.

[9] A. De La Escalera, L. E. Moreno, M. A. Salichs, and J. M. Armingol, "Road traffic sign detection and classification," *IEEE Trans. Ind. Electron.*, vol. 44, no. 6, pp. 848–859, 1997.

[10] D. Tian, "DeepPiCar—Part 5: Autonomous Lane Navigation via Deep Learning," 2019. [Online]. Available: https://towardsdatascience.com/deeppicar-part-5-lane-following-via-deep-learning-d93acdce6110. [Accessed: 08-Jun-2019].

[11] John Canny, "A Computational Approach To Edge Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–714, 1986.

[12] D. Impiombato *et al.*, "SSD: Single Shot MultiBox Detector Wei," *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 794, pp. 185–192, 2015.

[13] J. Hui, "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and…," *2018*. [Online]. Available: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359. [Accessed: 08-Jun-2019].

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *proc. IEEE*, 1998.

[15] D. Tian, "DeepPiCar—Part 6: Traffic Sign and Pedestrian Detection and Handling," 2019. [Online]. Available: https://towardsdatascience.com/deeppicar-part-6-963334b2abe0. [Accessed: 08-Jun-2019].

[16] A. G. Howard and W. Wang, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2012.

[17] P. Mustamo, "Pirkko Mustamo Object detection in sports: TensorFlow Object Detection API case study," no. January, 2018.

[18] Rohith Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms," 2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accessed: 19-Jul-2019].

[19] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 3296–3305, 2017.

[20] E. Szapiro, "You Only Look Once: Unified, Real-Time Object Detection Joseph," *Ann. Med. Psychol. (Paris).*, 1976.

[21] Sachan Ankit, "Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD – CV-Tricks.com," 2018. [Online]. Available: https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/. [Accessed: 19-Jul-2019].

[22] Niklas Donges, "Transfer Learning – Towards Data Science," 2018. [Online]. Available: https://towardsdatascience.com/transfer-learning-946518f95666. [Accessed: 08-Jun-2019].