Traffic Information Detection

Donghao Qiao, Jiayuan Zhou and Farhana Zulkernine

January 9, 2020

Traffic Information Detection

Donghao Qiao[1], Jiayuan Zhou[1], and Farhana Zulkernine[1]

[1] Queen's University, Kingston ON K7L3N6, Canada
{d.qiao, jiayuan,zhou, farhana.zulkernine}@queensu.ca

Abstract. In this paper we built a model which contains two submodules: lane detection and vehicle detection. Our lane detection model is based on a heuristic approach to detect lanes. It can be broken down into three steps: Image preprocess, Lane edge points identification, and lane cure generation. As for the vehicle detection, we applied YOLO series algorithms which are fast, accurate and can be used in real-time detection.

Keywords: Deep Learning, Computer Vision, Lane Detection, Vehicle Detection, YOLO.

1 Introduction

In recent years, with a number of technology breakthroughs in the world, one of the Artificial Intelligence (AI) branches self-driving vehicle is becoming closer to our lives. In this paper, we implement the lane detection and vehicle detection with Python3, which are the first and a very important step for auto-driving and road-safety alert.

2 Dataset

We recorded a real road video to test our model. The resolution of the video is 1280x70 and the duration is 16 seconds with 30 frames per second.
We used the pre-trained YOLO model and YOLOv3 model to detect vehicles. The pre-trained YOLO model was trained on two datasets, PASCAL VOC 2007 and 2012. The pre-trained YOLOv3 model was trained on COCO 2014 dataset. Table 1 shows the details of the datasets. The COCO 2014 dataset YOLOv3 used has 117,263 images for training and 5,000 for testing.

Table 1.

	#Classes	Size
PASCAL VOC 2007	20	9,963
PASCAL VOC 2012	20	11,530
COCO 2014	80	122,263

3 Implementation

Our model consists of two submodules 1) lane detection, 2) vehicle detection. For every captured image, the model will detect lane and vehicles at the same time using two submodules, respectively, and then integrate the detected lanes and vehicles together as an output. In this section, we introduce the two submodules, lane detection model and vehicle detection, respectively.

3.1 Lane Detection

Our lane detection model is based on a heuristic approach which is the Advanced Lane Finding project from Udacity [5] to detect lanes. In general, our approach can be broken down into three steps:

1) Image preprocess: We first preprocess captured images to remove redundant background information and reserve lane information.
2) Lane edge points identification: Then applied a sliding window approach to identify lane edge points.
3) Lane cure generation: Finally, we use polynomial regression to generate lane curve based on identified lane edge points.

Following we introduce the details of each step.

Image Preprocess. Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

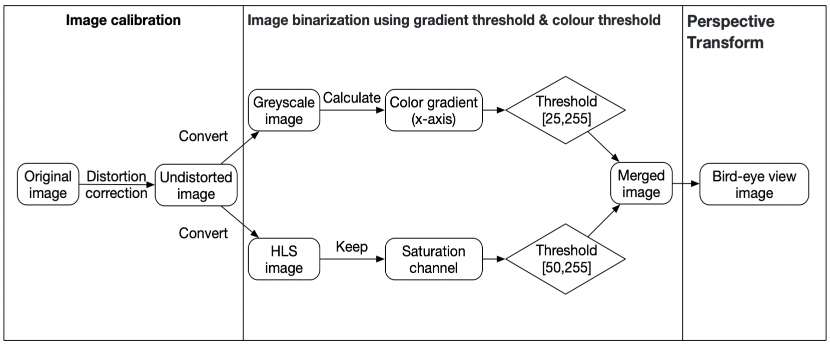

Fig. 1. The flow of the image preprocessing process.

Fig. 1. shows the flow of the image preprocessing process. Due to the optical design of lenses, the original camera captured images will be shown in a distorted way. For example, the left image of Fig. 2. shows a chessboard which is shown in a distorted way. In order to calibrate the image, we use the calibration function provided by Python OpenCV library to calculate distortion coefficients and transform the original distortion image into an undistorted image. The right image of Fig. 2. shows the undistorted/corrected chessboard.

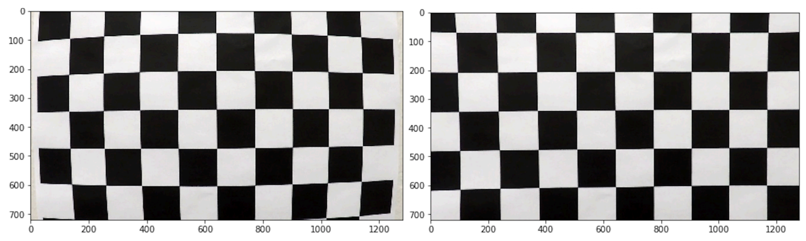

Fig. 2.

Then we remove the redundant background information using a threshold-based binarization method. We first greyscale the undistorted image and calculate the gradient of RGB value along the x-axis of the image and set 25 as the threshold value to binarize the image. Fig. 3. shows the image before and after the gradient-based binarization.

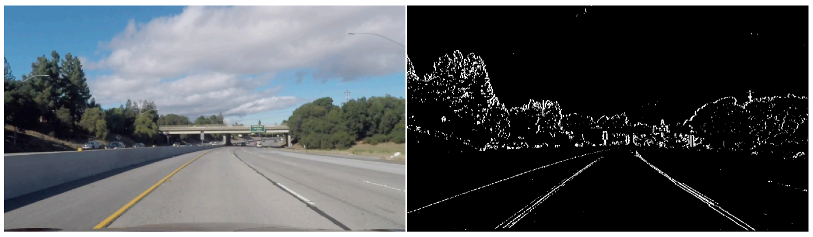

Fig. 3.

In order to further capture lane information which is removed by the above gradient-based binarization. We first convert the undistorted image into HLS (i.e., hue, saturation, lightness) image and only keep the saturation information of the image. After that, we set 50 as the threshold value to binarize the image. Fig. 4. shows the image before and after the color-based binarization.

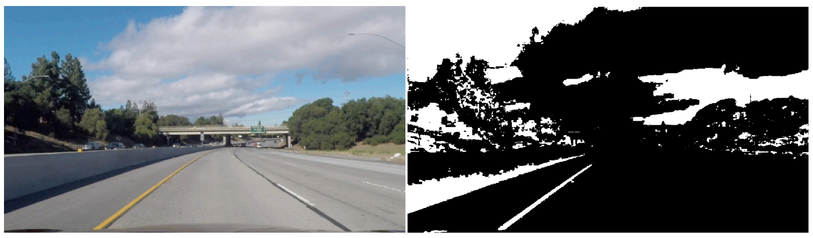

Fig. 4.

We can find that the gradient-based binarization keeps more lane information in terms of the shapes of lanes and the color-based binarization keeps more lane information in terms of the points inside lanes.
To reserve the most lane information, we merge two binarized images (see the left image of Fig. 5.). Since we are only interested in detecting the lanes in front of the camera, we mask the other area of the image (see the right image of Fig. 5.).

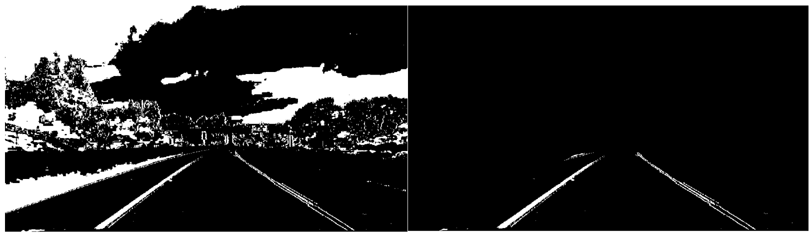

Fig. 5.

In order to better identify lane curve points and regress lane curves, we finally covert the masked image into a bird-eye view (see Fig. 6.).

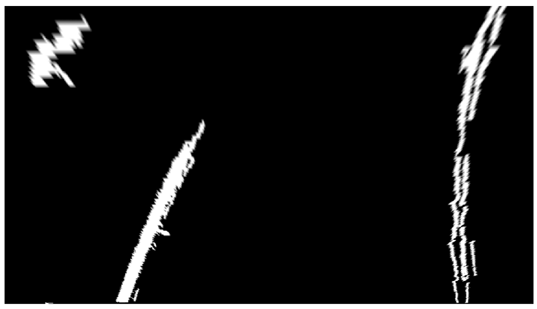

Fig. 6. Bird-eye view of the image.

Lane edge points identification. First, we use the histogram to detect the lane locations. We plot the white pixels density of all columns of the lower half image as shown in Fig. 7. From the histogram we can see two obvious peaks which are the location of the left and right lanes. Since the density (i.e., the value of y-axis) on the location (280, 0) start spiking until (450,0), we consider the x-axis coordinate of left lane is between those two points. Similarly, the x-axis coordinate of right lane is at around (1100,0).

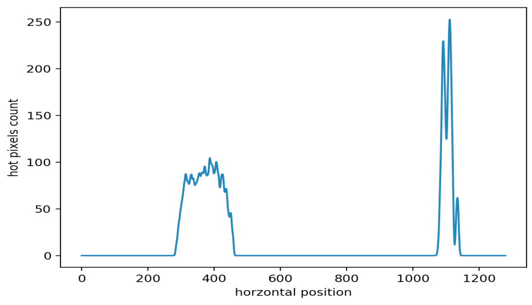

Fig. 7.

Secondly, in order to detect the lane curve points from the bird-eye view image, we apply a sliding window approach as shown in Fig. 8. The shape of the sliding windows are 68*120 pixels, and each lane has 10 sliding windows.

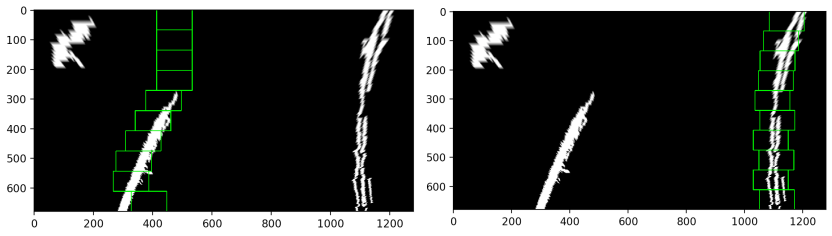

Fig. 8.

Lane curve generation. Finally, we apply polynomial regression on the identified lane edge points to generate lane curves (see the left image of Fig. 8.). Since the car always drives in the middle of the closest two-lane curves under the safe circumstances, we mask the area between two fitted lane curves with green color (see the right image of Fig. 8.).

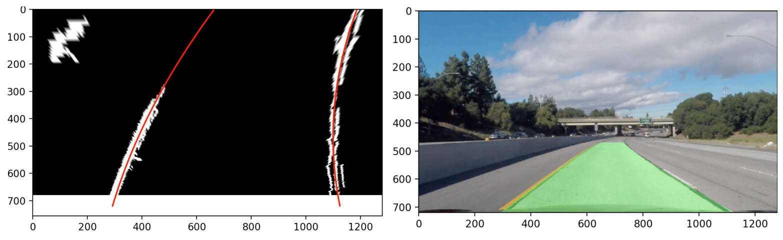

Fig. 8.

3.2 Vehicle Detection

We used YOLO series algorithms which are fast and accurate for vehicle detection. Different from regional proposal algorithms like R-CNN, YOLO does not require plenty of time to generate region proposals. Considering we want to use our model in real-time detection, it is essential to detect the objects fast and accurate, so that the car can run on the streets safely. In the following, we are going to introduce how we used YOLO and YOLOv3.

YOLO: Unified, Real-Time Object Detection. The procedure of how YOLO detects objects is shown in Fig. 9. Different from the regional proposal methods, YOLO divides the input image into S*S grids, if the center of an object is in a grid, then that grid is responsible for predicting that object. For example, the center of the dog is located in the second column and the fifth row, so that grid is responsible for detecting the dog. Each grid will predict number of B bounding boxes and 1 confidence. The confidence is equal to Pr(Object)*IOU_{pred}^{truth} the first part reflects how likely the box contains an object and the second part intersection over union (IOU) reflects how accurate is the bounding box. In this model, the image was divided into 7*7 grids, and each grid predicted 2 bounding boxes. We used the bounding boxes with high confidence scores (greater than 0.3) to predict vehicles.

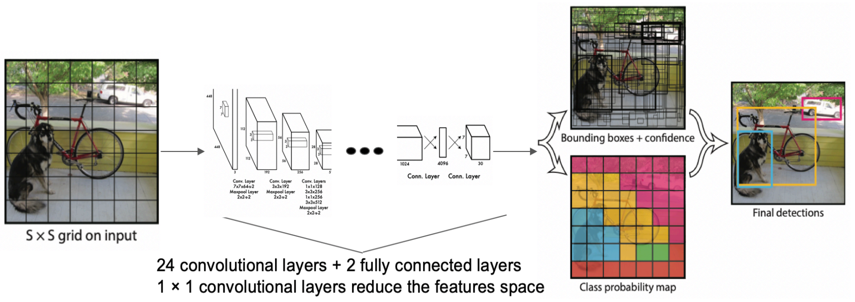
S × S grid on input
Conn. Layer
Conn. Layer
Bounding boxes + confidence
Class probability map
Final detections
24 convolutional layers + 2 fully connected layers
1 × 1 convolutional layers reduce the features space

Fig. 9. YOLO Model [1].

The YOLO detection network is shown in Fig. 10. and we applied Tensorflow to build the vehicle detection network. The network of YOLO has 24 convolutional layers and 2 fully connected layers. The learning rate is 0.1 and the threshold of the confidence is 0.1 as well. The output shape of the network is 7*7*30, because the images are split into 7*7 grids, each grid predicts the position(x, y, w, h) and confidence of two bounding boxes and the probabilities of the twenty classes, so 30 outputs: 5 outputs per box [4 box coordinates + 1 object confidence], times 2 boxes, and pluses 20 classes' probabilities.

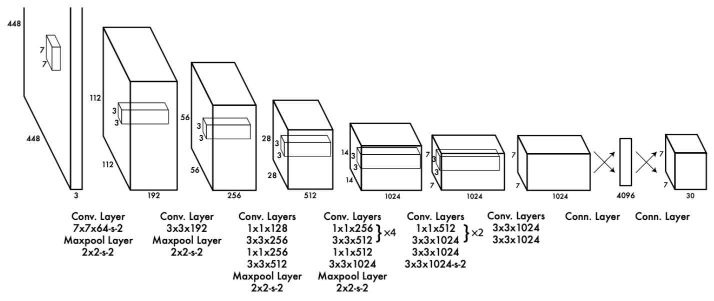
448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Fig. 10. YOLO Network [1].

YOLOv3: An Incremental Improvement. Although YOLO is very fast and can obtain satisfactory results, it still has many disadvantages. Each grid can only predict one object, and YOLO struggle with small objects. Therefore, we decided to switch to another algorithm. From Table 2 and Table 3 below we can see that YOLOv3 is much faster and has higher mAP and AP than YOLO and YOLOv2, so we tried YOLOv3 to detect vehicles.

Table 2. Comparing the accuracy and speed among YOLO, YOLOv2 and YOLOv3.

	Train Dataset	mAP	Testing Dataset	mAP	FPS
YOLO	VOC 2007&2012	63.4	VOC 2007&2012	57.9	45
YOLOv2	VOC 2007&2012	77.8	VOC 2007&2012	73.4	59
YOLOv3	COCO 2014	NA	VOC 2007	78.6	78

Table 3. Comparing the accuracy and speed between YOLOv2 and YOLOv3.

	Train Dataset	Testing Dataset	AP	FPS
YOLOv2	COCO 2014	VOC 2007&2012	21.6	59
YOLOv3	COCO 2014	VOC 2007	33.0	78

Fig. 11. shows how the authors of YOLOv2 enhance the performance of YOLOv2 step by step, and the mAP on dataset VOC2007 was increased from 63.4 to 78.6. Comparing to YOLOv2, YOLOv3 used a different network which called Darknet-53 as shown in Fig. 12. The network is mainly composed of 3*3 and 1*1 filters with skip connections like the residual network in ResNet, so the network can have 53 layers. The learning rate is 0.001, and the threshold is 0.7. The output layer of the network has 255 nodes: 85 outputs per anchor [4 box coordinates + 1 object confidence + 80 class confidences], times 3 anchors. The model was trained on COCO 2014 which has 80 classes, 117,263 images from the train and validate COCO sets, tests on 5000 images, and can get higher AP than YOLOv2, and SSD.

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Fig. 11. The path from YOLO to YOLOv2[2]

The weights and configure file can be downloaded from DarkNet website https://pjreddie.com/darknet/yolo/, then we applied OpenCV to build the network for vehicle detection. We tried the pretrained Tiny YOLOv3 with Darknet-19 first, because it is much faster than YOLOv3, but comparing to YOLO, the performance of Darknet-19 did not improve too much. Finally, we used YOLOv3 with Darknet-53, YOLOv3 could detect more vehicles and smaller cars.

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

Fig. 12. YOLOv3 Network [2]

4 Results

We finally test the model on a video that has 485 frames, the detection speed is around 2.5 FPS on our own PC. One frame of the results is shown in Fig. 13. From the output videos of both models, we can see that the lane detection can detect the lane in front of the car, and it can also detect the curve lane. The model is robust, although there is a line in the middle of the lane, the model did not influence by that noise.

Compared the results of YOLO and YOLOv3, we found that YOLOv3 performs much better than YOLO which is the same as we expected. They have the same speed in detection, whereas YOLOv3 can predict a higher number of cars, and can also predict the very small car at the end of the road.

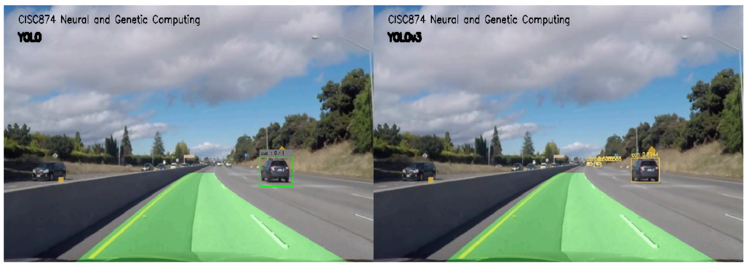
CISC874 Neural and Genetic Computing
YOLO
CISC874 Neural and Genetic Computing
YOLOv3

Fig. 13. One frame of the results.

To evaluate the recall of the results, we saved the images for both models every second, and we got 16 images from each model. We counted the cars which have the same direction as ours (on the right lanes in the images), and the correct detections. Then, we got the recalls for each model as shown in Fig. 14. We can see that the recall of YOLOv3 is much higher than YOLO.

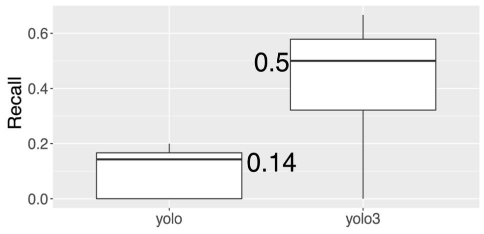

Fig. 14. Recall of the models.

5 Problems and Discussions

The first problem is the detection process consumes a very long time and cannot use in real-time detection. This is because we tested the model on our own PC instead of using the GPU, and detected the lane and vehicles sequentially, it costs twice the time to do the detection.

Secondly, we do not evaluate the result of lane detection. We are still working on calculating the IOU of the lane. We also want to detect multiple lanes with CNN so that can accelerate the model with GPU.

Finally, we used YOLOv3 to detect vehicles, whereas we did not train the model with our dataset. The model was trained on COCO 2014 which has 117,263 training images, and around half of the data is vehicles such as cars, buses, and trucks [4]. The median of the recall of our model is 0.5, and the detection results are not as good as we expected. It cannot detect the vehicles in the left lanes.

References

1. Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection.
2. Redmon, J., and Farhadi, A., 2016. YOLO9000: Better, Faster, Stronger.
3. Redmon, J., and Farhadi, A., 2018. YOLOv3: An Incremental Improvement.
4. Lin, T., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C., and Dollar, P., 2015. Microsoft COCO: Common Objects in Context.
5. https://github.com/udacity/CarND-Advanced-Lane-Lines