University of Dayton
CPS 584 Advanced Intelligent Systems and Deep Learning
Department of Computer Science

Deep Learning Based Driver Assistance System

Prathyusha Muttineni
Ragini Bharadwaj
Hongyu Wu

Instructor:
Mehdi R. Zargham

Dec. 4th, 2019

Table of Contents

1. Introduction

With rapidly developing technology in artificial intelligence, such as computer vision and natural language processing, appearing in the consumer market, more and more emerging intelligent devices are brought to human lives, such as smart home assistants. As artificial intelligence becomes more popular and more versatile, people are putting more attention on AI related technology. Therefore, autonomous driving car, is now a big trend in AI market. However, accompany with tremendous obstacles exiting in real life world, the development of such technology in autonomous driving is facing a big challenging.

Car accident, is one of the major promoter that speed up the development of autonomous driving. According to the statistics, there are 292163 auto accidents in 2019, Ohio. Therefore, to help with this issue, this project is about building a warning system in autonomous driving car for drivers to avoid accidents. Road lane detection is adopt in the first stage of the project, which has been widely explored in the area of autonomous driving. Object Detection is then implemented in the second stage, which will utilize deep learning based algorithm to achieve the goal of real time vehicle recognition. The focus of our project is on detecting if there is any other vehicle trying to cross the lane which can cause high to low intensity accidents and the corresponding warning will be given to the driver to stop or slow down their speed. It will subsume training the network model to detect the surrounding vehicle instructing the machine to take the required steps to avoid negative repercussions.

2. Project Description

In this project we aim to make a road lane and vehicle detection pipeline to mimic lane departure warning system used in self driving cars.

The code is written in Python 3.7. OpenCV and TensorFlow Object detection API called Keras has been used for lane detection and vehicle detection part of the pipeline respectively.

The python file can be implemented without any additional dependencies. For implementing the Vehicle detection part, we used the Keras and copy the python notebook along with the videos folder.

Therefore, the project will accomplish two tasks:

- Detected highway lane lines on a video stream. Used OpenCV image analysis techniques to identify lines, including Hough Transforms and Canny edge detection.
- Built and trained a deep neural network to classify traffic signs, using TensorFlow API. Experimented with different network architectures. Performed image pre-processing and validation to guard against overfitting.

3. Purpose of the Project

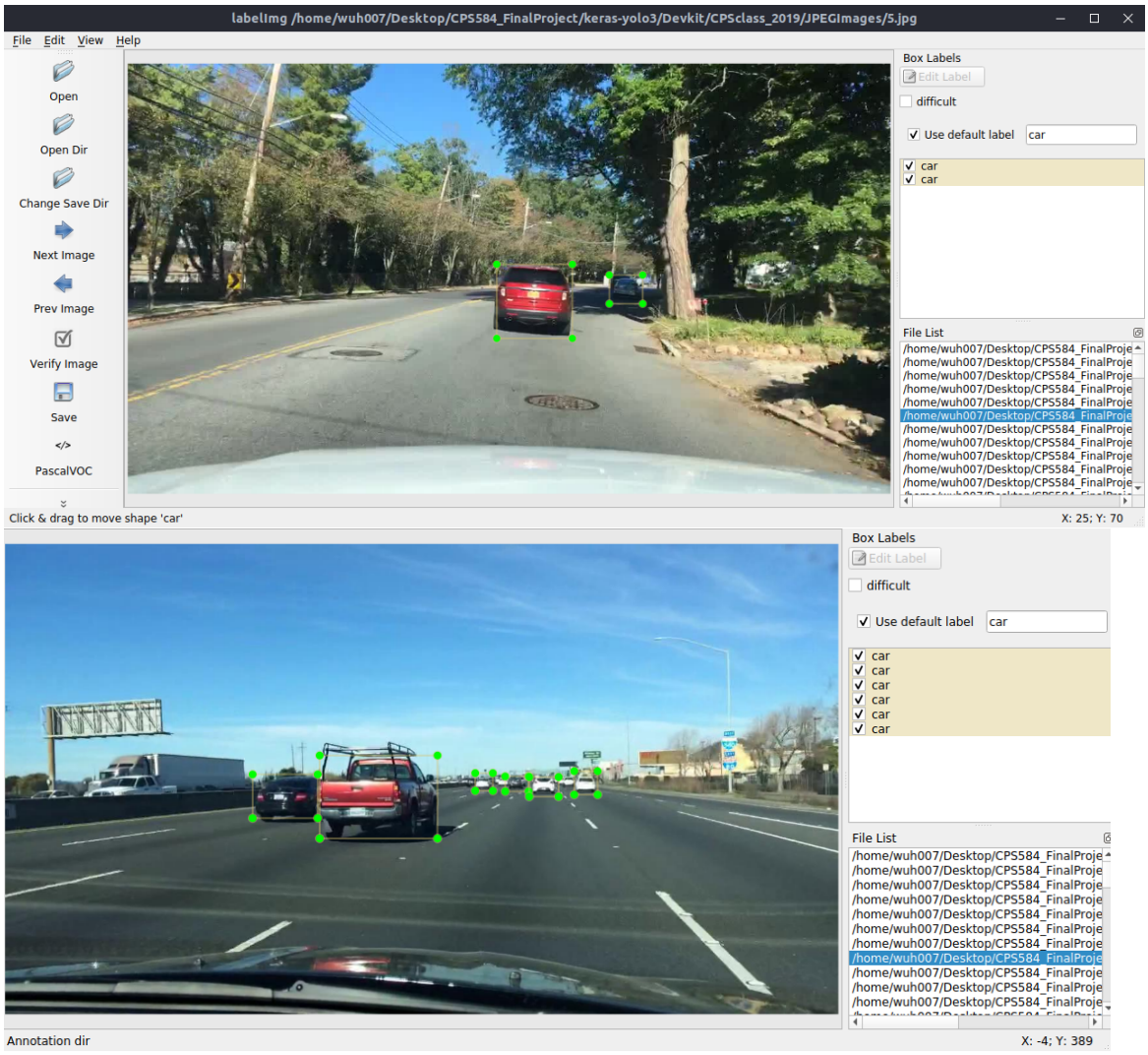
The purpose of this project is to utilize computer vision algorithms and deep learning models that students have learnt, to build and develop an autonomous detection system that will detect the vehicle exiting on the road and road in front of the car, and integrate each other to check if danger happens.

To design an autonomous driver assistance system using deep learning methods, the project includes several fundamental functions described below:

- Color and gradients detection algorithms will be firstly built to detect the existence of road lane in an image. The thresholds for detection varies based upon the color intensity of the road lane, the lighting and texture of the ground, and so on.
- Perspective transformation algorithm will be applied to the output binary image after color and gradients detection, which transform the original image to another plane with a top-down view.
- The road lane will be detected using sliding windows which scan the bird-view image from bottom to top to determine the curvature of the lane.
- The Convolutional Neural Network based object detection algorithm call Yolov3 will be adopt to implement the vehicle detection. This is a deep neural network that has a strong backbone like Imagenet, which can extract the details of each regions of the image and classify the them into specific classes.
- The output location of detected object using object detection algorithm will then be used to compared with the location of the detected road lane at the same time to calculate the distance between vehicle and road lane.
- Threshold for the distance is manually set up so that once the car moving close to the road lane, the warning will be given in real time.

4. Source of Data

The data that are being used to train our object detection algorithm is collected from external resources. The dataset contains around 400 pictures and we labeled the pictures using labelling tool called labellmg, below shows several sample labelled images.



We shot a video by ourselves which we included in our project presentation. This video is our testing data for this project.

5. Survey of Current Method

Gopalan et al used the learning-based approach for lane detection.

Jongin Son et al proposed strategy for functions admirably in different lighting conditions like awful climate conditions and at evening time.

Abdelhamid Mammeri et al has displayed an in vehicle figuring framework. This framework is equipped for confining path markings and conveying them to drivers. It consolidates the Maximally Stable Extremal Region (MSER) technique with the Hough change for identifying and perceiving path markings like lines and pictograms.

6. Proposed Method

In order to accomplish the task of road lane detection, we implement the followings:

- Image Rectification, Color Transforms, Gradient Threshold are the implementation of both color and edge searching techniques to detect the existing road lane in the video.
- Perspective Transform: Perspective transform mean conversion of 3D world image to 2D image. Therefore, this technique helps further analyze the detected road lanes.

In order to accomplish the task of object detection, we implement the following:

- Yolo Framework: The algorithm applies a convolutional neural network to an entire image. The network divides the image into an S x S grid and comes up with bounding boxes, which are boxes drawn around images and predicted probabilities for each of these regions belonging to various classes. In our case, we only have one class which is vehicle, the main idea of the project to detect the existence of vehicle on the road.

In order to combine both road lane detection and object detection algorithms, we implement the following:

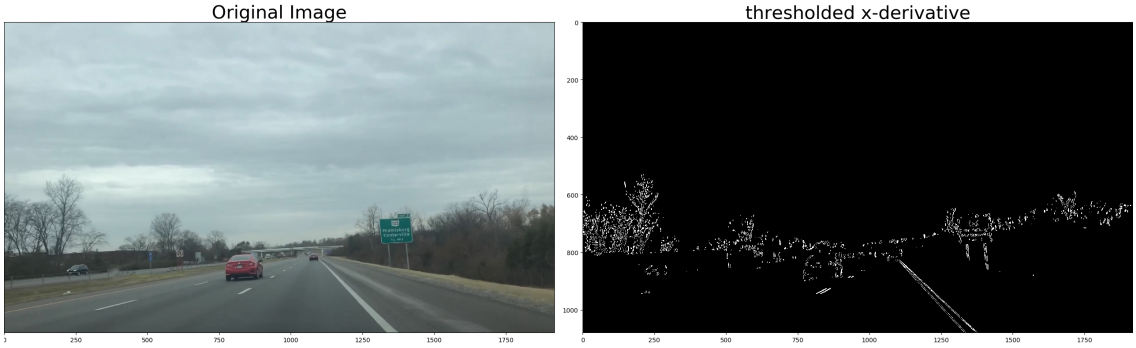
- Distance threshold: We use calculate the distance between location of the detected vehicles and left/right road lanes, and set a threshold based experimental result.

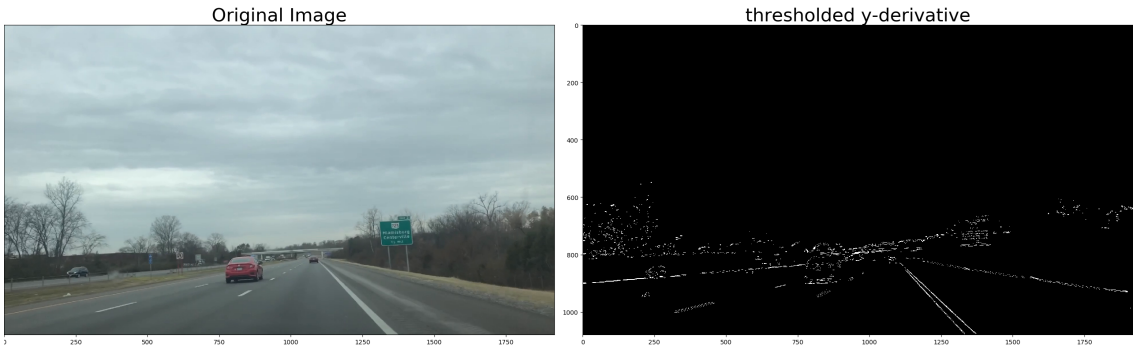
7. Implementation Details

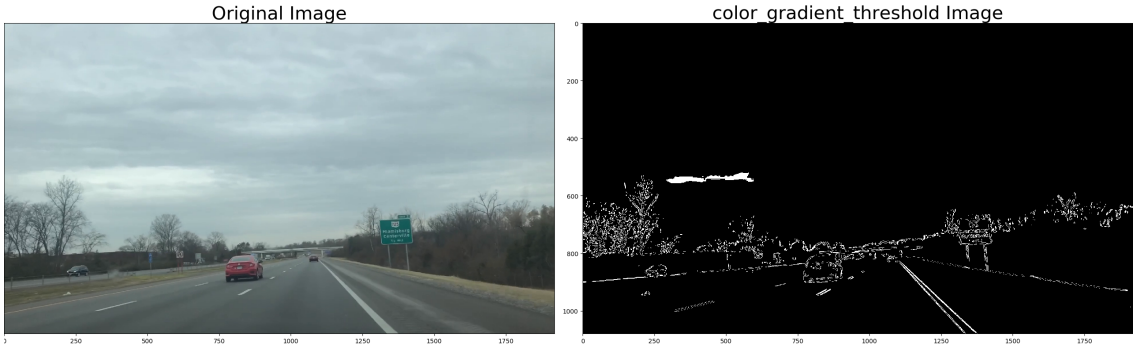
1. Lane detection / tracking:

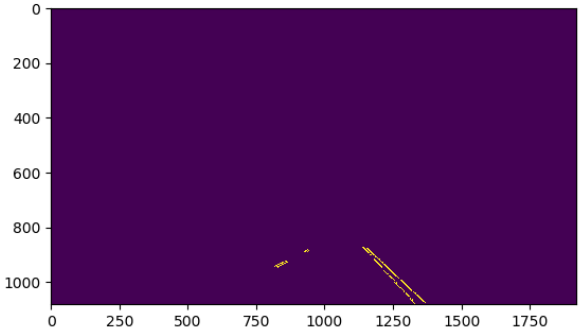
- Use color transforms, gradients, etc., to create a thresholder binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect / track lane pixels and fit to find the lane boundary.

Below shows figures generated in first section.





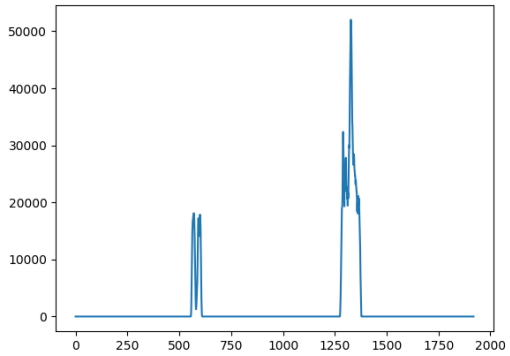


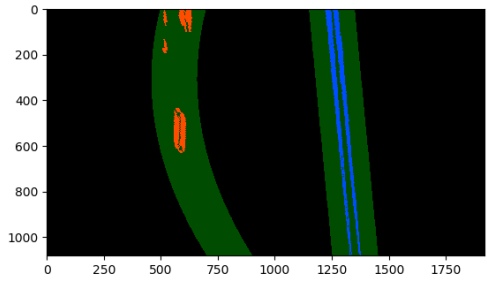


2. Lane status analysis:

- Determine the curvature of the lane
- Compute the vehicle position with respect to center.

Below shows figures generated in second section.





3. Lane augmentation:

- Warp the detected lane boundaries back onto the original image.
- Print the road status into image.

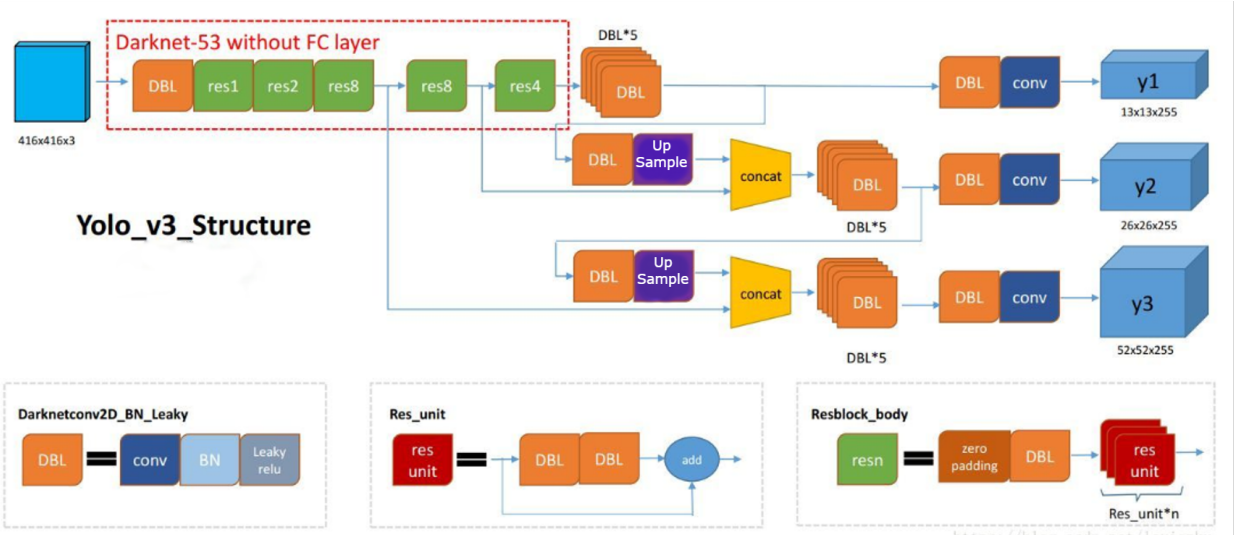
Below shows figures generated in third section.

4. Vehicle detection and warning system:

- Detect the vehicles along with the road lane detection.
- Give warning once danger is detected.

The next page shows figures generated in fourth section, and the structure of Yolov3 is explained.


Miamisburg
Centerville
1 MILE
75
Cincinnati
45



Yolo, is a clever convolutional neural network for doing object detection in real time. The algorithm applies a single deep neural network to an image, and divides the image into multiple regions and classify the class that each region belongs to.

Yolov3 is the up-to-date algorithm of Yolo series, which keeps the advantages of previous Yolo and also makes a few improvement. Several advantages of Yolov3 include:

- Divide and Conquer, the newest Yolov3 remains the essence of Yolo that divides the image into different grids, and makes implementation on each grid. The only difference is the number of grids that the images get divided.
- As an activation function, "Leaky ReLU" is adopt in Yolov3, which works well on fixing the "vanishing gradient" problem. For negative bias, the leaky relu has a small slope to avoid it being zero all time.
- End to end training. Just like previous Yolo, Yolov3 only needs one loss function to cover the entire prediction process.
- The combination of batch normalization and leaky relu. Yolov3 uses this method to achieve the goal of regularization, accelerating convergence and avoiding overfitting.
- Multi-scale training. Yolov3 trains each image in 3 different scales and make predictions across scales.

  Overall, when it comes to the output of Yolov3, for each image, in our project, Yolo will output a scale*scale*3*5+1 = 18 vector. That means, for each grid in each scale, 3 bounding box will be output, and each bouding box contains the 5+1 = 6 metrics, which are x,y axis of center of bounding boxes, width and height of bounding boxes, objectiveness and class score of bounding boxes.

8. Programs Documentation

```
"""This is .py file for performing road lane detection"""

import numpy as np
import cv2

# function for rgb channel color threshold
def rgb_select(img, r_thresh, g_thresh, b_thresh):
    r_channel = img[:,:,0]
    g_channel=img[:,:,1]
    b_channel = img[:,:,2]
    r_binary = np.zeros_like(r_channel)
    r_binary[(r_channel > r_thresh[0]) & (r_channel <= r_thresh[1])] = 1

    g_binary = np.zeros_like(g_channel)
    g_binary[(r_channel > g_thresh[0]) & (r_channel <= g_thresh[1])] = 1

    b_binary = np.zeros_like(b_channel)
    b_binary[(r_channel > b_thresh[0]) & (r_channel <= b_thresh[1])] = 1

    combined = np.zeros_like(r_channel)
    combined[((r_binary == 1) & (g_binary == 1) & (b_binary == 1))] = 1
    return combined

# function for directional gradient threshold
def abs_sobel_thresh(image, orient='x', sobel_kernel=3, thresh=(0, 255)):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    if orient == 'x':
        abs_sobel = np.absolute(cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel))
    if orient == 'y':
        abs_sobel = np.absolute(cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel))

    scaled_sobel = np.uint8(255*abs_sobel/np.max(abs_sobel))
    grad_binary = np.zeros_like(scaled_sobel)
    grad_binary[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])] = 1
    return grad_binary

# function for sblv channels threshold
def color_thresh(image, s_thresh, l_thresh, b_thresh, v_thresh):
```

```
# function for sblv channels threshold
def color_thresh(image, s_thresh, l_thresh, b_thresh, v_thresh):
    luv= cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
    hsv = cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    lab=cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
    s_channel = hsv[:,:,1]
    b_channel=lab[:,:,2]
    l_channel = luv[:,:,0]
    v_channel= hsv[:,:,2]
    s_binary = np.zeros_like(s_channel)
    s_binary[(s_channel > s_thresh[0]) & (s_channel <= s_thresh[1])] = 1
    b_binary = np.zeros_like(b_channel)
    b_binary[(s_channel > b_thresh[0]) & (s_channel <= b_thresh[1])] = 1
    l_binary = np.zeros_like(l_channel)
    l_binary[(s_channel > l_thresh[0]) & (s_channel <= l_thresh[1])] = 1
    v_binary = np.zeros_like(v_channel)
    v_binary[(s_channel > v_thresh[0]) & (s_channel <= v_thresh[1])] = 1
    combined = np.zeros_like(s_channel)
    combined[((s_binary == 1) & (b_binary == 1) & (l_binary == 1) & (v_binary == 1))] = 1

    return combined

# function combining all thresholds
def color_gradient_threshold(image_undistorted):
    ksize = 15
    hsv = cv2.cvtColor(image_undistorted,cv2.COLOR_RGB2HSV)
    s_channel = hsv[:,:,1]

    gradx=abs_sobel_thresh(image_undistorted,orient='x',sobel_kernel=ksize,thresh=(50,80))
    grady=abs_sobel_thresh(image_undistorted,orient='y',sobel_kernel=ksize,thresh=(50,90))
    c_binary=color_thresh(image_undistorted,s_thresh=(70,100),l_thresh=(60,255),b_thresh=(50,255),v_thresh=(150,255))
    rgb_binary=rgb_select(image_undistorted,r_thresh=(225,255),g_thresh=(225,255),b_thresh=(0,255))
    combined_binary = np.zeros_like(s_channel)
    combined_binary[((gradx == 1) | (grady == 1) | (c_binary == 1) | (rgb_binary==1))] = 255
    color_binary = combined_binary
    return color_binary, combined_binary
```

```
# function for perspective transform
def perspective_transform(masked_image):
    top_left = [905, 884]
    top_right = [1215, 884]
    bottom_right = [1500, 1080]
    bottom_left = [500, 1080]
    top_left_dst = [500,0]
    top_right_dst = [1500,0]
    bottom_right_dst = [1500,1080]
    bottom_left_dst = [500,1080]

    img_size = (masked_image.shape[1], masked_image.shape[0])
    src = np.float32([top_left,top_right, bottom_right, bottom_left] )
    dst = np.float32([top_left_dst, top_right_dst, bottom_right_dst, bottom_left_dst])
    M = cv2.getPerspectiveTransform(src, dst)
    Minv = cv2.getPerspectiveTransform(dst, src)
    warped  = cv2.warpPerspective(masked_image, M, img_size)
    return warped, Minv, M

# function for lane detection
def finding_line(warped):
    histogram2 = np.sum(warped[warped.shape[0]//2:,:], axis=0)
    out_img = np.dstack((warped, warped, warped))*255
    midpoint = np.int(histogram2.shape[0]/2)

    leftx_base = np.argmax(histogram2[:midpoint])
    rightx_base = np.argmax(histogram2[midpoint:])+midpoint
    nwindows = 3
    window_height = np.int(warped.shape[0]/nwindows)
    nonzero = warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    leftx_current = leftx_base
    rightx_current = rightx_base
    margin = 100
    minpix = 40
    left_lane_inds = []
```

```
    left_lane_inds = []
    right_lane_inds = []

    for window in range(nwindows):
        win_y_low = warped.shape[0]-(window+1)*window_height
        win_y_high = warped.shape[0]-window*window_height
        win_xleft_low = leftx_current-margin
        win_xleft_high = leftx_current+margin
        win_xright_low = rightx_current - margin
        win_xright_high = rightx_current + margin
        cv2.rectangle(out_img,(win_xleft_low,win_y_low),(win_xleft_high,win_y_high),
        (0,255,0), 2)
        cv2.rectangle(out_img,(win_xright_low,win_y_low),(win_xright_high,win_y_high),
        (0,255,0), 2)
        good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
        (nonzerox >= win_xleft_low) &  (nonzerox < win_xleft_high)).nonzero()[0]
        good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
        (nonzerox >= win_xright_low) &  (nonzerox < win_xright_high)).nonzero()[0]
        left_lane_inds.append(good_left_inds)
        right_lane_inds.append(good_right_inds)


        if len(good_left_inds) > minpix:
            leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix:
            rightx_current = np.int(np.mean(nonzerox[good_right_inds]))

    left_lane_inds = np.concatenate(left_lane_inds)
    right_lane_inds = np.concatenate(right_lane_inds)

    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)
    ploty = np.linspace(0, warped.shape[0]-1, warped.shape[0] )
```

```
    ploty = np.linspace(0, warped.shape[0]-1, warped.shape[0] )
    left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]

    left_lane_inds = ((nonzerox > (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy +
    left_fit[2] - margin)) & (nonzerox < (left_fit[0]*(nonzeroy**2) +
    left_fit[1]*nonzeroy + left_fit[2] + margin)))
    right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[1]*nonzeroy +
    right_fit[2] - margin)) & (nonzerox < (right_fit[0]*(nonzeroy**2) +
    right_fit[1]*nonzeroy + right_fit[2] + margin)))
    print(left_lane_inds)
    print(right_lane_inds)

    return left_fitx, right_fitx,out_img, left_fit, right_fit,left_lane_inds,right_lane_inds,ploty

def sliding_window(binary_warped,left_fit,right_fit):
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    margin = 100
    left_lane_inds = ((nonzerox > (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy +
    left_fit[2] - margin)) & (nonzerox < (left_fit[0]*(nonzeroy**2) +
    left_fit[1]*nonzeroy + left_fit[2] + margin)))
    right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[1]*nonzeroy +
    right_fit[2] - margin)) & (nonzerox < (right_fit[0]*(nonzeroy**2) +
    right_fit[1]*nonzeroy + right_fit[2] + margin)))
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )

    left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
    right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
    window_img = np.zeros_like(out_img)
    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]
```

```
    #1 color and gradient detection
    color_binary, combined_binary = color_gradient_threshold(image)

    #3 interest region (with road lane in)
    masked = apply_region_of_interest_mask(color_binary)

    #4 perspective transformation
    warped_0, Minv, M = perspective_transform(masked)

    #5 sliding windows road lane detection
    left_fitx, right_fitx, out_img,left_fit, right_fit,left_lane_inds,right_lane_inds, ploty = finding_line(warped_0)

    #6 draw road lane on image
    warp_zero = np.zeros_like(warped_0).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))
    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
    pts = np.hstack((pts_left, pts_right))

    location = np.concatenate(location)
    location = np.array([location])
    location_warped = cv2.perspectiveTransform(location, M)
    count = 0
    for i in range(location_warped.shape[1]):
        if (int(location_warped[0][i][1]) > len(pts[0]) or (len(pts[0])-(int(location_warped[0][i][1])+1) > len(pts[0]))):
            continue
        elif (location_warped[0][i][0] > (pts[0][int(location_warped[0][i][1])][0] - 550)) and (location_warped[0][i][0] < (pts[0][len(pts[0])-(int(location_warped[0][i][1])+1)][0] + 550)):
            count += 1
        else:
            continue
    if count != 0:
        cv2.fillPoly(color_warp, np.int_([pts]), (0, 0, 255))
    else:
        cv2.fillPoly(color_warp, np.int_([pts]), (0, 255, 0))
    newwarp = cv2.warpPerspective(color_warp, Minv, (image.shape[1], image.shape[0]))
    result = cv2.addWeighted(image, 1, newwarp, 0.3, 0)
```

9. Results

As shown in section 7 detailed implementation, the result of the project is that the system is able to find the road lane and detect the vehicle by analyzing the image taken by the middle camera in front of the car. The testing video is taken at I-725 Ohio Highway in Dayton. The system successfully combine two algorithms in the end.

10. Conclusion

The detailed analysis of various lane detection and vehicle detection is discussed. Different methods and techniques presented by different authors for the lane detection and tracking during the last decades are presented in the paper. Various performance parameters required to detect the accuracy of the algorithm is discussed in these paper. The lane detection techniques play a significant role in Advance cruise control. The vision based approach is very easy and simple approach for detecting lanes. A lot of advancement has been done in the lane detecting and object tracking but still there is a scope of enhancement due to wide variability in the lane environments.