Advanced Lane Finding

This project is about finding lane sections on straight and curvy road. Following are the steps to create an advanced lane finding algorithm.

A. Camera Calibration and Un-Distorting Images

Due to usage of lenses in modern camera the images captured is distorted. Two of the main types of distortion are Radial distortion and Tangential distortion. It is possible to measure the coefficients of radial and tangential distortion for any camera if we have knowledge about the actual image without distortion and the distorted images. Using these coefficients, we can undistort and image.

OpenCV has a good method to obtain these coefficients for camera calibration and performing image un-distortion. It works in a chessboard with predefined rows and columns. For example, we need to print image of any chessboard with predefined rows and columns and take images of it on a plain surface from different angles and distances from the camera that needs to be calibrated.

In our project, we are already provided with the sample images taken using the same camera used for capturing lane images and video. This sample images can be used to calibrate the camera and undistort lane images.

Find below few examples of original and un-distorted images.

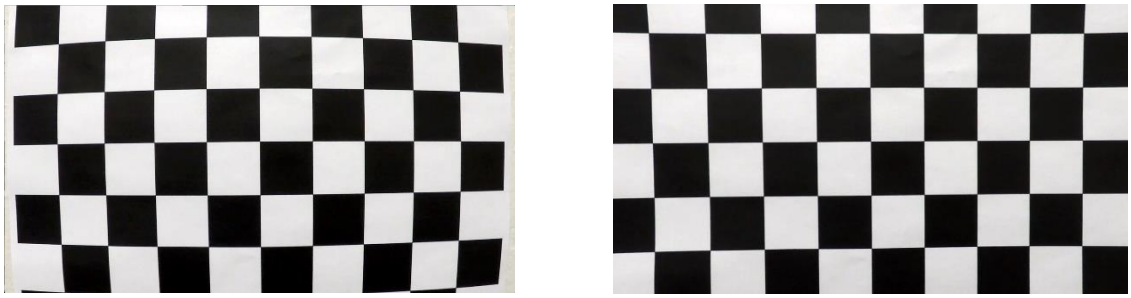

Figure 1:- Original and Un-distorted Chessboard Images

Figure 2:- Original and Un-distorted Lane Images

Code that performs camera calibration and un-distortion is present in the camera_calibration.py module

B. Gradient and Colour Threshold

Second step in the pipeline is to try and retain only the information corresponding to Lane lines. We use combination of 3 approaches to perform this step

1. **Absolute Sobel Thresholding**

   Sobel operator is a way of taking derivative of the image in x or y direction.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Figure 3:- Sobel Filter of Kernel size 3

   SobelX would try and retain any vertical lines in the image, whereas SobelY will try and retain all the horizontal lines

   Following set of images show the result of performing Sobel Transformations on example image.

Figure 4:- Original, SobelX and SobelY transformed Image

   As evident form above images SobelX and SobelY retains vertical and horizontal lines respectively. We obtain a binary mask that can be combined with other filters to create a unified thresholding filter.

2. **Sobel Magnitude and Direction Thresholding**
   This is like above approach, except for the fact that instead of considering the individual values of SobelX and SobelY we consider their magnitude and direction. We apply thresholds on magnitude and direction to obtain the binary mask that can be combined with other thresholding approaches mentioned here.

Magnitude of a SobelX and SobelY is given by $\sqrt{(sobel_x)^2 + (sobel_y)^2}$

And direction as $arctan(sobel_y / sobel_x)$ where a value of 0 denotes vertical line and +/- π indicates horizontal line

Following set of images show the result of performing Sobel Magnitude and Direction Thresholding on example image.

Figure 5:- Original, Sobel Magnitude and Sobel Direction Threshold Transforms

3. Colour Thresholding

Final approach in thresholding is the colour-based thresholding. A white or yellow lane can be separated our using RGB colour thresholding. However, this would fail if lighting conditions are considered. To solve this problem, we use the HLS colour space which is represented by Hue-Lighting-Saturation as shown below.

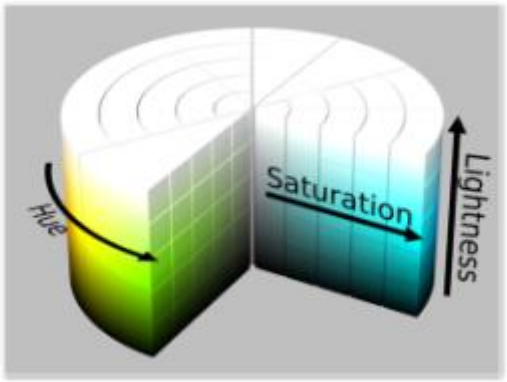

Figure 6:- HLS Colour Space

Using the Saturation component allows us to overcome the lighting effects.

Following set of images show the result of performing HSL Thresholding on example image.

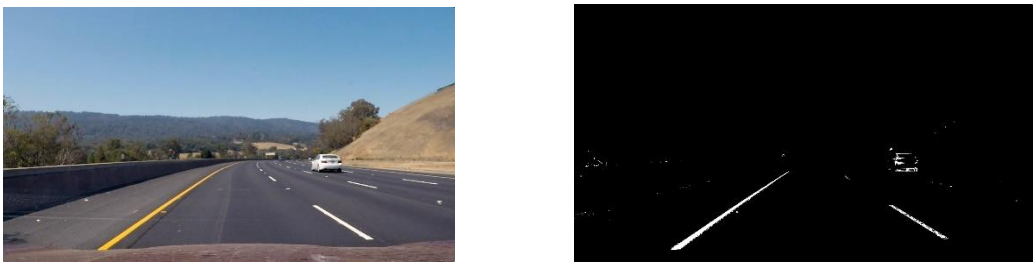

Figure 7:- Original and Saturation Threshold Image

Once we obtain output from all the 3 steps, we combine them to create a final image.

Following set of images show the result of performing HSL Thresholding on example image.

Figure 8:- Original and Combined Thresholding output

All the code for Gradient and Colour thresholding can be found in color_transforms.py module. Also output images can be found in “output_images/color_and_gradient_thresholds” folder

C. Perspective Transformations

A perspective transform maps the points in each image to different, desired, image points with a new perspective. The transformation we are interested in is the Bird’s eye view. This view will help use understand or view the lanes lines and curvature more appropriately compared to straight camera image.

For performing a perspective transform we need to select a set of 4 point in the source image and corresponding set of point in the destination image to which it needs to be mapped to. We can do this by visualizing the image in an editor which shows the pixel co-ordinates and try to pick the points that form a polygon enclosing the lane section with some margin. For the destination points we can choose any rectangular area.

The code for perspective transformation is given in the perspective_transforms.py module. Also, examples can be found in “output_images/perspective transforms” folder.

Find below and example showing perspective transform.

Figure 9:- Original, Gradient Threshold and warped Image

D. Finding Lane Lines

Once we have the warped image, we can apply the following set of heuristics to find and draw lanes.

1. **Find Lane pixels**
   We use a sliding window approach to find the left and right lane line. The approach is described below.

   - ➔ **Find Histogram**: - Create a column wise histogram of bottom half of the warped image.
   - ➔ **Find Left and Right lane Starting bottom points**: - Once we have the histogram, we can identify 2 points where the histogram peaks. We identify one point on each side of the vertically split image. These points will act as the starting points for left and right lane.
   - ➔ **Sliding Window**: - We then choose a margin and height for a rectangular window and centre it at both the left and right starting points. We slide the window towards image top. While sliding in y direction towards top, we calculate the mean centre of lane pixels within the window and we use this to re-centre the window across x axis.
   - ➔ **Fitting a Polynomial**: - Using the coordinates obtained by sliding window across the image we fit two polynomials – one each for left and right lane. This polynomial defines our lane pixel.

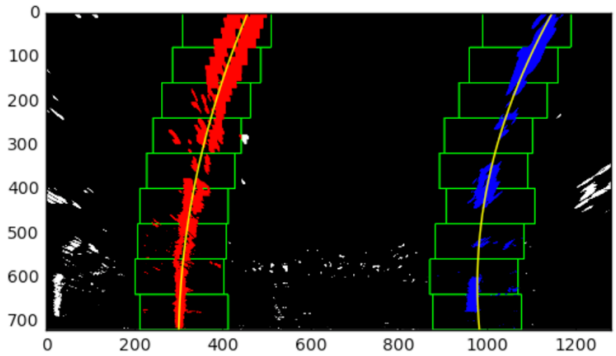

Figure 10:- Sliding window approach on Warped Image

2. **Searching around the polynomial**

Though the above approach works, it is inefficient. In the next frame of video, we need not do a blind search again, but instead we can just search in a margin around the previous line position, like in the above image. The green shaded area, in below image, shows where we searched for the lines this time. So, once we know where the lines are in one frame of video, we can do a highly targeted search for them in the next frame.

This is equivalent to using a customized region of interest for each frame of video and should help us track the lanes through sharp curves and tricky conditions. If we lose track of the lines, we can go back to our sliding windows search or other method to rediscover them.

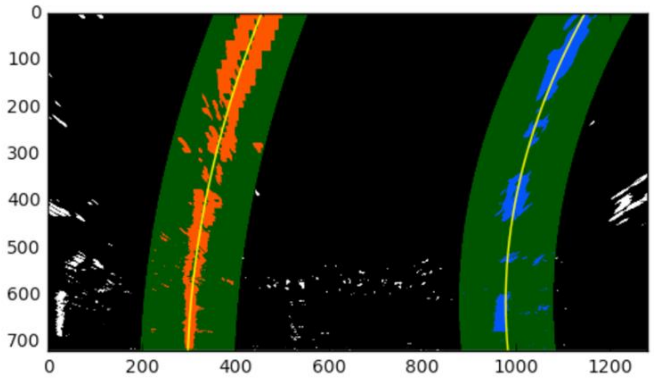

Figure 11:- Searching Around the Polynomial

3. **Finding Curvatures and Centre**

Using above approaches, we have found the lane lines in the image. Now the formula to find the curvature is given by

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

In the case of the second order polynomial above, the first and second derivatives are:

$$f'(y) = \frac{dx}{dy} = 2Ay + B$$

$$f''(y) = \frac{d^2x}{dy^2} = 2A$$

So, our equation for radius of curvature becomes:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

4. **Drawing Lane Section**

   Using the above information, we can draw the polygon to denote the lane section. Also, if camera location is at the centre of the car, we can determine the car position with respect to lane lines.

   One additional point to be noted while quoting the distance and other numbers is the conversion from pixels to real world metrics.

   All code for lane finding, finding curvature and drawing lane section is written in the module find_lanes.py. Also, example images are present in the folder "output_images/lane_lines".

   Find below some example lane marking images.

Figure 12:- Original and Lane Marked Images

E. Pipelines

1. **Image Pipeline**

   Image pipeline takes single image and passes to through following steps:

   Image → un-distort image → apply (sobelx + sobley + sobel magnitude + sobel direction + HLS) Thresholding → Find Lane Pixels → Find Curvature and Centre → Annotate Original Image with lane and curvature information

   Module main.py provides a way to consume this pipeline as shown below: -

   python main.py -i <input_folder_path> -o <output_folder_path>

2. **Video Pipeline**

   Video pipeline is like the above-mentioned image pipeline, except for the fact that it loops the above pipeline for every image in the video and also makes use of the optimized technique of searching around the polynomial obtained from previous image.

   Current implementation uses a python library name "moviepy" to extract images from video, process images from video and stitch the processed images back to video.

   Module main.py provides a way to consume this pipeline as shown below: -

   python main.py -i <input_folder_path> -o <output_folder_path>

   A sample output video is present at "output_video" folder.

F. Conclusion

This set of algorithms implanted allows us to find lane lines in a descent manner. However, there are few areas of improvement like: -

1. Sliding window algorithm assumes that in the Birds eye view lanes lines would appear as near vertical with slight curvature. Thus, it would fail at lanes making right angles or more
2. Improvements in speed for real time use.