Advanced Lane Finding

Seymur Dadashov

August 2, 2020

Abstract

The purpose of the project is develop a pipeline capable of performing lane line identification and tracking through a series of images i.e. video. The output of the pipeline will be the original image overlayed with the region that has been identified to be the lane, marking it from the edge of the left lane to the edge of the right lane to which the ego vehicle currently belongs to. To achieve the output, a series of functions will be applied to the images involving lens distortion correction, gradient based filtering, perspective warping and curve fitting. The final output will also include a calculation of the radius of curvature R and offset of the vehicle from center of the lane.

1 Task 1 Camera Calibration

Camera calibration was performed using a series of images taken of a 9x6 chessboard at different angles and distances. The images are first converted to gray-scale and passed to the OpenCV function `cv2.findChessboard()` that can automatically return the pixel coordinates of the corners within a chessboard. The combination of this information and our object points, a matrix grid that corresponds to the location of each corner on the chessboard, is sufficient to allow OpenCV to run `cv2.calibrateCamera()`. The function returns specifically 2 important values for our use case, the Camera Matrix and Distortion Coefficients. The camera matrix is an intrinsic matrix that transforms 3D camera coordinates to 2D homogeneous image coordinates and can be decomposed into a sequence of shear, scaling, and translation transformations, corresponding to axis skew, focal length, and principal point offset. The latter output is a vector of distortion coefficients that capture the affect of the distortion within a few values and depending on the intensity of the distortion, more coefficients are used.

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

The camera matrix and distortion coefficients along with the original image are passed to `cv2.undistort()` in order to obtain the newly mapped and undistorted image as seen in Figure 1.

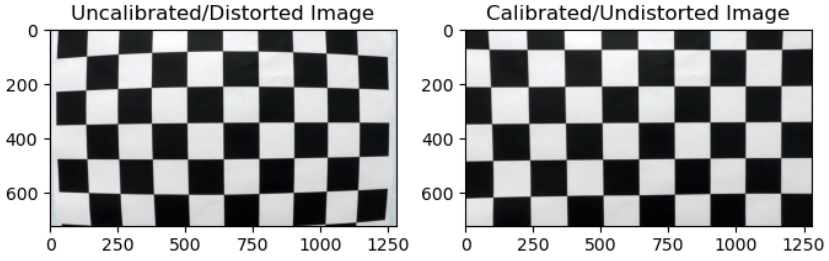

Figure 1: Image with and without calibration applied

2 Task 2 Pipeline

2.1 Calibration on Project Images

The following figure displays the same transformation that was applied in Figure 1, but now to real images from the provided data.

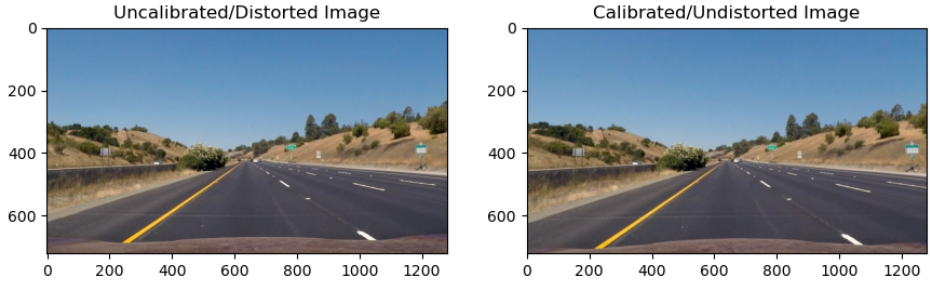

Figure 2: Project image with and without calibration

2.2 Binary Thresholding

In order to transform the original three channel image into a binary image that contained visibly distinct lane lines, a smaller pipeline had to be made strictly for this process.

The primary color space that was used to achieve the result was HLS or Hue-Lightness-Saturation. Through some experiments and observations made at the affects of each channel, it was interesting to conclude that Lightness and Saturation worked very well together. Lightness could see in areas that were poorly lit by picking up regions of the image that were poorly lit i.e. the light value of the pixel was lower. This is not to be confused with saturation as one could maybe associate the two, but saturation deals with how vibrant the color itself is, not necessarily how much light is reflected off that color and into the camera. Therefore, the saturation allows for detection of colors within a threshold that ranges from faded colors to vibrant colors. In Figure 3, it can be seen how Lightness can *"see"* sections of the image that Saturation cannot and vice versa. Combining the two binary images using `cv2.bitwise_or_()` we get the 1s from both imaged stacked on top of each other. The method naturally combines the noise from both images and in order to deal with that, a mask can be applied using a binary image that had $Sobel_x$ kernel applied to it. One can also argue that $Sobel_x$ by itself does a great job of producing the final image, but in my implementation a Low-Pass Filter (LPF) was used over the initial image just before the $Sobel_x$ kernel was applied. The LPF reduces the high frequency noise in the image as seen in Figure 4. The analogy is to this approach is similar to using a template for spray painting a design, where the $Sobel_x$ image with the LPF serves the purpose of a template and the points from the Lightness and Saturation channel are passed through that template. A closer look at the approach when compared to only using the $Sobel_x$ shows that there is less noise along the section that belongs to the hood of the car and less data points that correspond to the grass to the right of the right most lane. The extra steps show better results downstream in the pipeline when a histogram is used to initially detect that lanes and then use points within a certain margin to follow the lane's unique curves. Having excess outliers increases the inaccuracy of the implementation mentioned later in text.

The main function that performs these operations is titled `color_space_transform()`. The function is called in `grad_image()` that includes a small filter depending on the project video (i.e. project, challenge, and harder challenge) being inputted with some custom parameters for each scenario.

2.3 Image Rectification: *'Birds-Eye-View'*

The procedure for warping the image into a top-down view is simple and highly depends on the OpenCV function `cv2.warpPerspective()` that performs the desired command under a *flag* condition. In order to

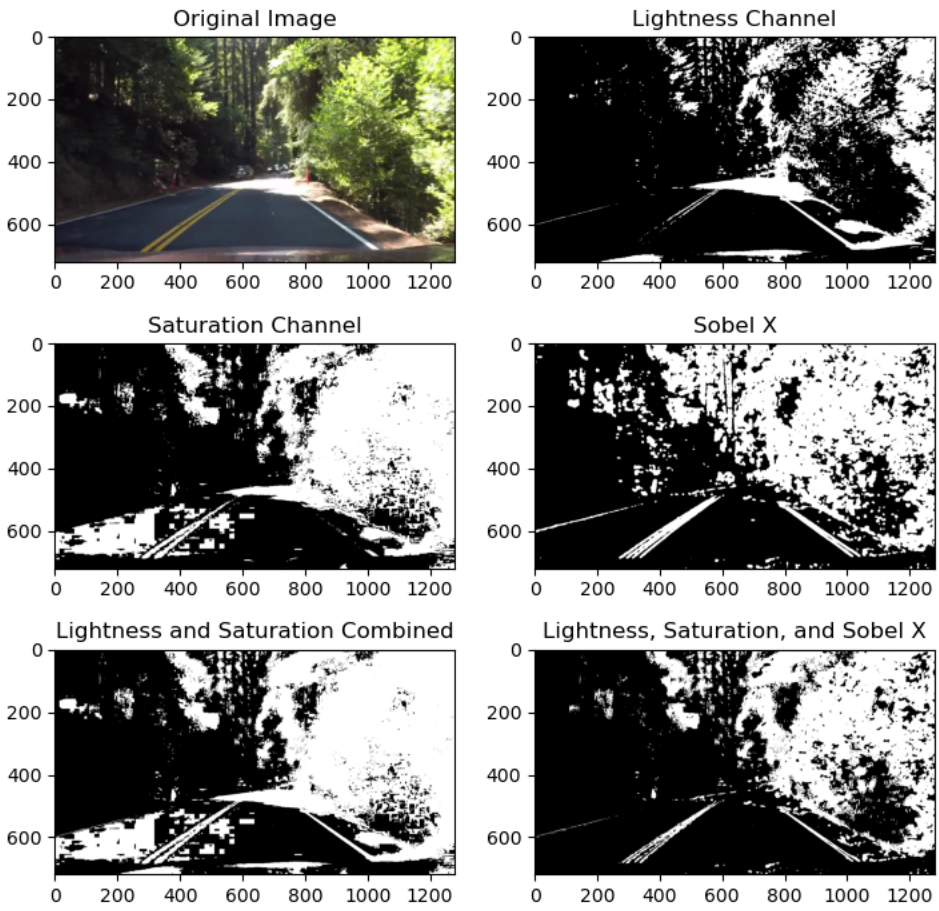

Figure 3: Different gradient and color transforms applied to each image in sequence

first work with the image, we undistort the effects due and apply the transform onto the binary image. Next, a warp matrix, M, must be computed by `cv2.getPerspectiveTransform()` that takes the desired points from our original image and transforms them into the destination space. For the source points, the vertices original set for the bounding of the lane lines is used as it directly captures our area of interest and the destination points are equal to the exact vertical bounds of the original image with slightly compressed horizontal margins in order to get a nice set of centered lane lines. Playing with the values of the source vertices usually occurs as the warped image would need to be tuned. The warp matrix is passed to the mentioned function and the direction of the warp is set using a flag `INTER_LINEAR` for going to top-down view, or `WARP_REVERSE_MAP` to get back from top-down view to our original image. The operation only takes several lines to successfully implement. Figure 5 show the expected result after implementation.

2.4 Fitting Curve to Lane Lines

Fitting a curve to a binary image of lane lines begins with using a histogram that bins the values along the y-axis into x-bins along the x-axis. This histogram allows to generally approximate where 2 bimodal

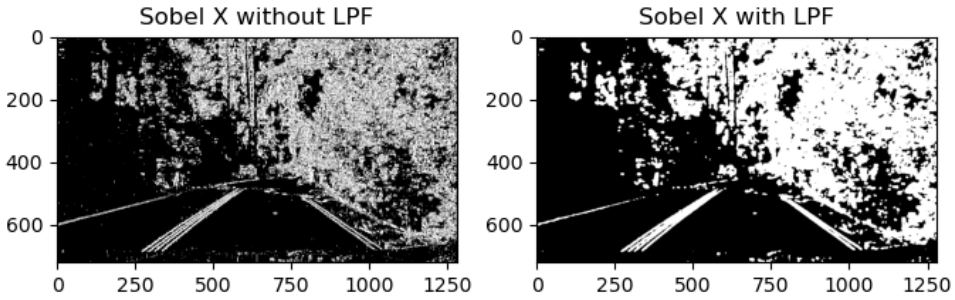

Figure 4: Comparison of $Sobel_x$ with and without Low-Pass Filter

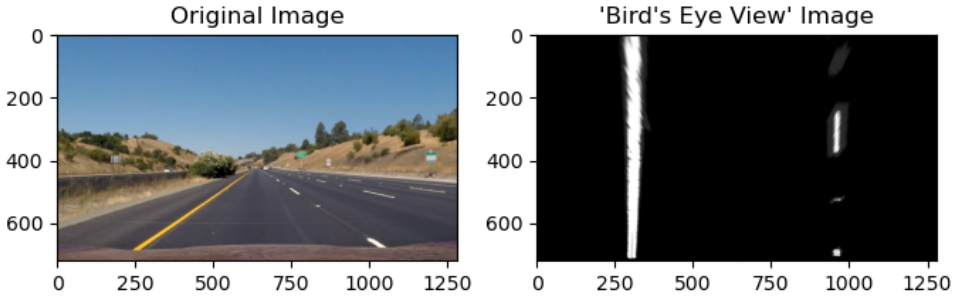

Figure 5: 'Birds-Eye-View' after applying OpenCV perspective transform

peaks occur and that will generally correspond to the lane lines. Depending the complexity of the image, the histogram can appear extremely noisy and not represent any relevant information with regards to the actual lanes; therefore, processing the actual histogram is very important. Several steps that were taken during the processing of the histogram was to crop the values appear generally too far left or too far right that almost never have lanes lines. In the code these values include ignoring the first 200 pixels and the last 280 pixels such that the domain of x belong to $200 < x < 1000$. The next step is floor any values that appear less than 15000 times in each of the x-bins as this experimentally was shown to be just noise collected in the image. The midpoint of the image was taken and then a maximum that belonged to the left half and the right half were evaluated. The approach generally works for simple lane lines; however, for more complex ones, a different metric was used to assess if clear lane lines had been established. The *Kolmogorov-Smirnov* test for goodness of fit was used to compare for the shape of the sample distribution. This performs a test of the distribution $F(x)$ of an observed random variable against a given distribution $G(x)$. Under the null hypothesis, the two distributions are identical, $F(x) = G(x)$. The alternative hypothesis can be either 'two-sided' (default), 'less' or 'greater'. For the metric used to evaluate the histogram in this case, the alternative hypothesis was used such that $F(x) >= G(x)$. Conveniently what the test returns is a value of 0 if the current histogram is non-bimodal and noisy or closer to a uniform distribution and returns a value between 0-0.5 depends on how clear the 2 peaks appear. In other words, the test although not traditionally meant for this purpose, can be used to show the confidence of filtered histogram and becomes crucial when trying to re-establish lane lines after the image has been blown out due to bright sun spots. The 'harder challenge' video used shows signs of the gradient picking up too much noise due to the image being blown out by bright

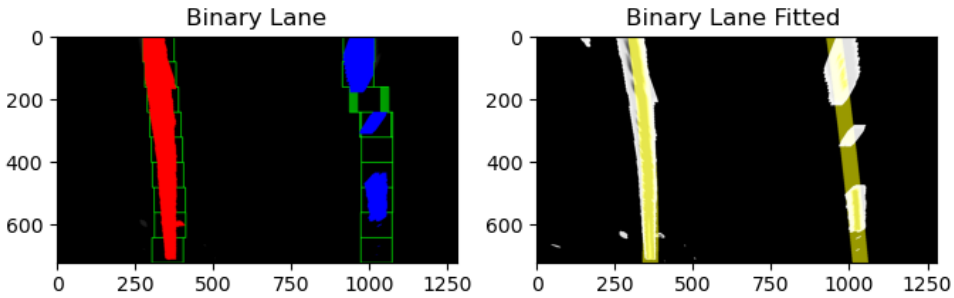

Figure 6: *Left:* Lane lines identified by segments for the initial image, *Right:* Quadratic curve fitted to lane lines for remaining frames in image sequence

spots and in order to maintain a robust lane search, a method had to be put into place that would only use the histogram approach when a confident reading came in i.e. the image is no longer blown out.

Having established the histogram method, we can choose to use it selectively when we lose complete sight of the lanes and need to start from scratch, but to avoid computationally expensive approach we can simply use the fitted curve from previous images to search for the curve in the new image. This was done by initially using the histogram on the first image to establish the initial base of the lanes and scan the image using a small box window that travelled vertically up the image and calculated the mean of the points within the bounding box that captured the lane. Figure 6 shows the implementation as these points are captured and then passed to a polyfit function. A quadratic curve is fitted to the points and used for the next sequence of frames.

In order to reuse the curve, we create another margin around the curve by shifting the curve to the left and right by a fixed amount and seeing what points lie within that margin. Additionally, the margin that corresponds to the interior of the lane was adjusted marginally to see if the lane was curving as the curvature of the lane would introduce points in the following frame to appear in between the left and right lane. Depending on the direction of the turn it would either be on the left lane interior or right lane interior. If points were detected, the entire margin would be shifted a little more until more points were captured using the old curve. Theses points were used to fit the new curve and essentially result in a more accurate representation of the the behaviour of the lane by conforming to the incoming curvature of the lane.

An additional check was added to monitor two aspects of these curved lines that included the distance between the curves and the A parameter in $f(x) = Ax^2 + Bx + C$. The distance was experimentally set to be checked to be no less than 400 pixels in the harder challenge video and 500 for the project video as anymore shrinkage typically was a result of errors or lane detection distorting such that the left lane becomes the right lane and vice versa. The A parameter was used to monitor the difference in the A value between both curves. If one curve started to become more narrow/wide, this usually indicated that it was approximating incorrect values and was observed to occur in areas of bright sunlight. When any of these conditions were seen, the previous curve was used as the new curve and in the current frame. This approach was seen to be successful as there is sufficient data within a single curve that can last several frames or even seconds as the road doesn't typically curve in an unrealistically rapid manner. Therefore, waiting for a good and stable frame to come in showed to be a better approach than trying to find a lane among a sea of noise.

2.5 Calculations: Radius of Curvature and Vehicle Offset

The implementation of the calculation of the radius of curvature was directly done using the equations provided in the classroom:

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

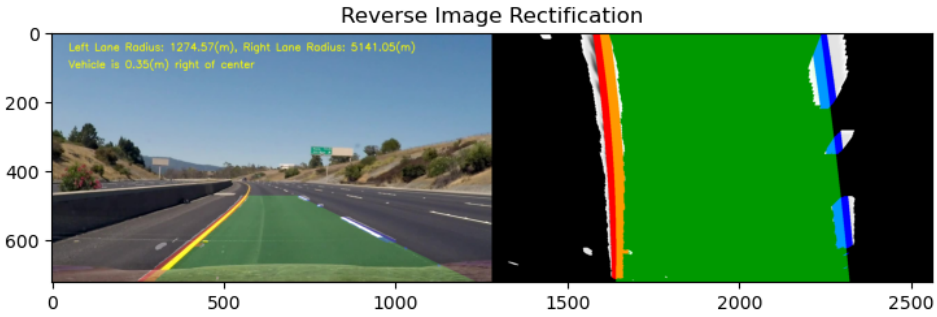

Figure 7: Plot the top-down view fitted curve back to the original image

The original curves were modified by scaling the coefficients in $f(x) = Ax^2 + Bx + C$ such that the conversion result in an output in the unit of meters:

$$x(y) = \frac{M_x}{M_y^2}Ay^2 + \frac{M_x}{M_y}By + C \tag{3}$$

where M_x is the conversion of x pixels to meters, and M_y is the conversion y pixels to meters.

The offset of the vehicle was calculated using the difference between the curves and comparing it to the midpoint of the x-axis. The general approach was $Offset = Midpoint - (Curve_{diff}/2 + LeftLane_x)$ and the sign of the offset value would indicate if the vehicle was to the left or right of center.

2.6 Reverse Image Rectification

Image rectification was done by simply altering the flag that is passed to `cv2.warpPerspective()` from `INTER_LINEAR` to `WARP_INVERSE_MAP`.

3 Discussion

During the implementation of this pipeline, I had faced many issues and the addition of certain checks and methods are seen throughout the functions that each tend to a certain problem. The outstanding issues however that were difficult to address include areas where the vehicle takes a sharp turn and loses complete visual of the a single lane that is underneath the corner of the front bumper when taking a left or right turn. The difference in this problem compared to something like a blown out image is that the curvature of the lane that disappears needs to be assumed and effectively appears out of frame. The video for the harder challenge shows how the current pipeline can just barely stay along such a tight corner, but would need a lot more work in order to an appropriate job of determining the lane lines. Another area where the vehicle lane detection can fail is if a blown out image is introduced for an extensive period of time where the road curvature can possibly change as the camera cannot detect through such a noisy image, but simply maintain its current curve until some less noisy image is captured. This could potentially be solved by attempting to manipulate values in a different color space and looking at creating different binary gradients for different scenarios, but another solution could be to look at hardware that is capable of capturing more data about the image even if excessive light is captured. The physical size of the pixel on the sensor could be increased allowing for more accurate light measurement to be taken on a per pixel basis. Overall, plenty of unique and distinct scenarios in the real world can throw a simple algorithm off track if certain conditions are not accounted for and then optimized such that the pipeline can parse these scenarios in real-time to determine the most accurate representation of a lane.