P2 Writeup: advanced lane finding

reference rubric is here: **https://review.udacity.com/#!/rubrics/1966/view**

The goal of this project is write a piece of code that finds road lanes from a video feed. The video feed is taken with a single camera installed in front of a car approximately at the centre of the vehicle.

Since a video is a sequence of images, the steps of this project are given per each image, as follows:

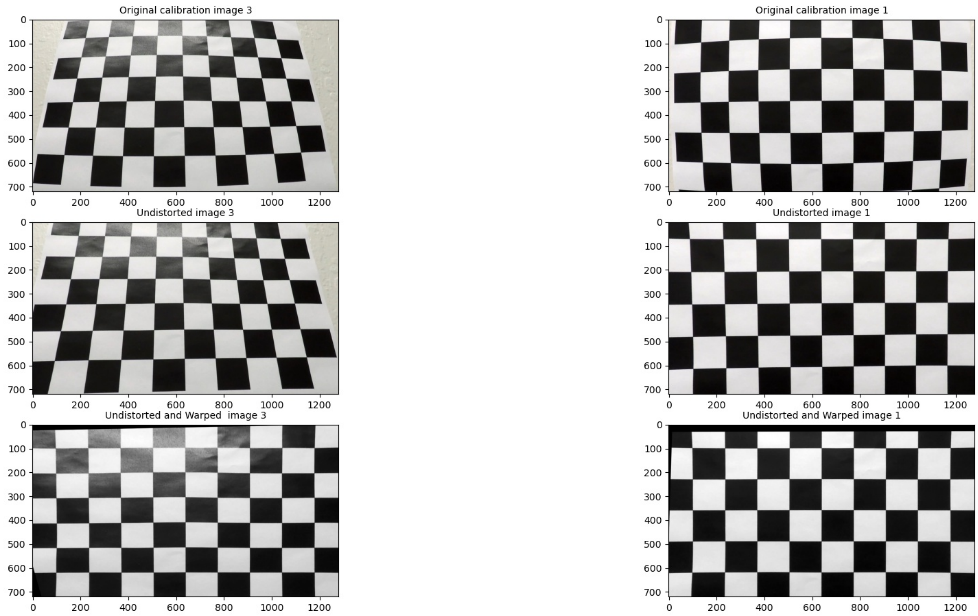
1. Camera calibration
2. Distortion correction
3. Apply a perspective transform to rectify binary image ("birds-eye view")
4. Use color transforms, gradients, etc., to create a thresholded binary image.
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

1. Camera Calibration

All images and videos are taken using the same camera. So camera calibration is done only once, using a set of chessboard images (given as reference). The camera calibration process provides key coefficients and parameters used later to correct other images. The figure below shows the result of the calibration and correction of two of the given images (on the left and on the right). The top row shows the original undistorted images. The second row shows the same images after applying the correction distortion. The images on the bottom show the result of the perspective transformation (birds-eye-view) applied to the images using the edges of the board as source and destination points.

2. Distortion correction

The distortion correction uses the camera matrix and the distortion coefficients to undistorted images. If a given image shows optical distortions, such as in the two top



chessboard images below, the undistorted image is an image where all optical distortion are removed. With images of chessboards, it is easy to see the effects of distortion correction because the lines that seem rounded because of the radial distortion (top images) are now straight (middle images). The effects of a distortion correction might be less evident in test images below where the same process is applied.

3. Apply a perspective transform to rectify binary image ("birds-eye view")

For the purpose of this project is convenient to focus the lane finding algorithms on a a very specific area within the image because we know where the lane lines will be with respect to the camera view. The top layer of the picture below shows the same undistorted images as before with highlighted the trapezius that I have chosen as region of interest. The position of the edges of the trapezius was done “manually”. I observed the effect of the choice of the trapezius edges in the original (undistorted) image space on the birds-eye view image (second row in the image below). My goal was to make sure that in the birds-eye view perspective, the road lanes seem to be straight. I have added some vertical lines to help my observations. After the manual inspection, I fixed the source and destination points of the perspective transformation and used them for the rest of the project.

4. Use color transforms, gradients, etc., to create a thresholded binary image.

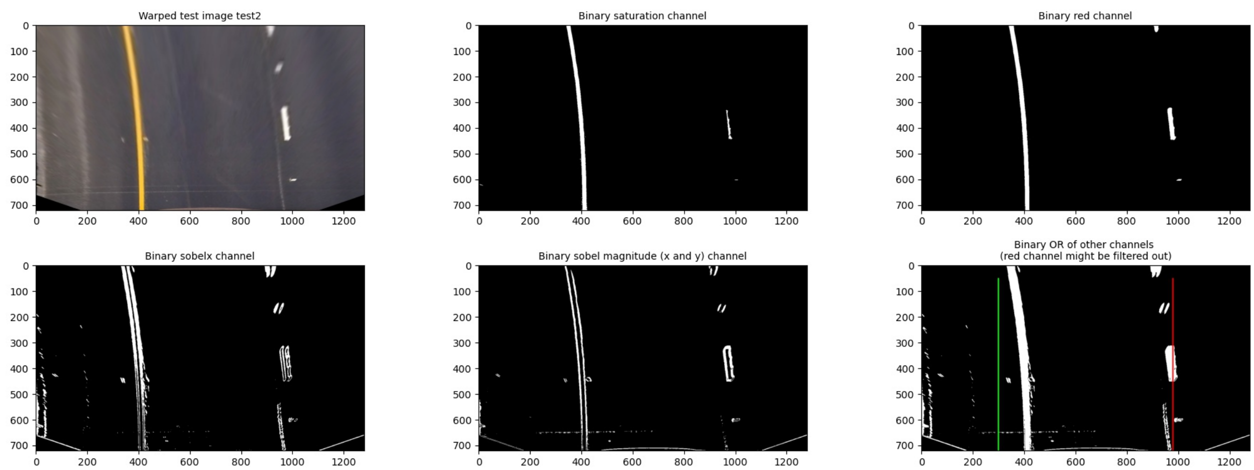
Since the project videos show many different situations (shaded areas, change of pavement colors etc.) I decided to observe the individual effect of color filters and gradients and then combine the best filters.
The figure below shows the result for one of the test images. The first row shows (from left to right)

1. the birds-eye-view of the test image
2. the binary image of the saturation channel only, filtered with some thresholds
3. the binary image of the red channel only, filtered with some thresholds

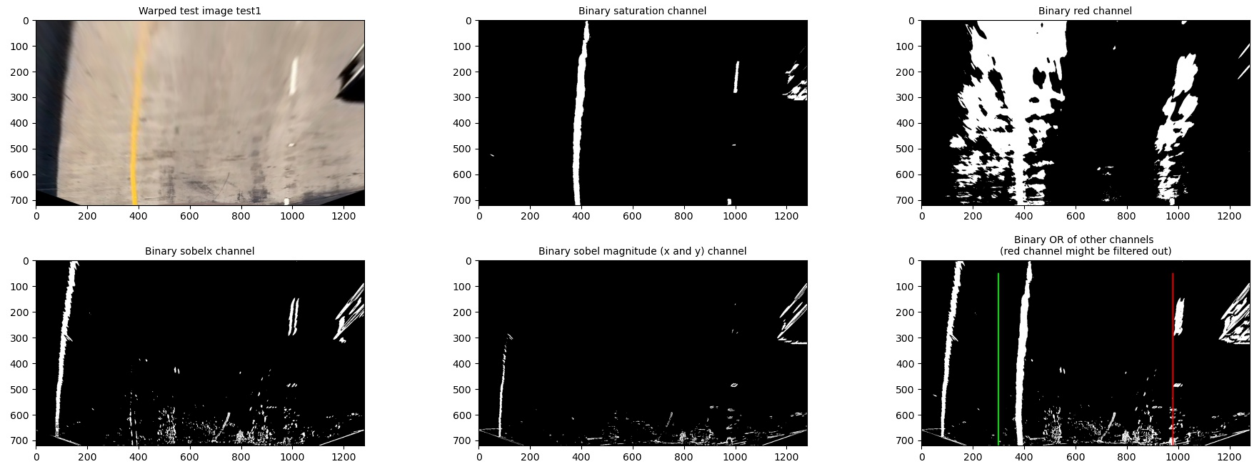
the second row (from left to right)

4. the binary image after applying a Sobel operator on the x axis, filtered with some thresholds
5. the binary image after applying a Sobel operator on both axes and taking their magnitude, filtered with some thresholds
6. the combination of all the previous filters as logical OR

The result seem to work well for this image, but the next image shows some drawbacks of this simple approach.



The image below shows how for the test image 1, the red channel filter is too sensitive (top right image). The other filters seem to be still useful. To overcome the effects of the red channel, I have decided to set a threshold to the amount of activated pixels in the "red" image. I observed some images and set the threshold to "count_light_in_red_ch > 150000". The bottom right figure of the image below shows the result of the logical OR of all other filters without the red channel.



7. Detect lane pixels and fit to find the lane boundary.

The lane detection pipeline (function called "process_frame(img)") uses two different lane finding strategies, one based on the histogram and vertical sliding windows and one based on the search around prior poly line. My goal is to use the search around prior poly method as much as possible because it is less computational intensive than the histogram based approach.

At the beginning of the video I use the histogram approach to find the first few lines, every time some lines are found, they are saved into a buffer of lines until the buffer is full of lines. I use a ring buffer with size 5 (it seems to work well enough) to store the lines. When the buffer is full, I average all the lines in the buffer (left lines and right lines are treated separately). From the moment when the buffer is full, ideally I will only search for lines using the second method, unless there is some issue with the new lines.
If the detected lines with the prior search method are "good enough", I store them in the buffer, I use them for the rest of the pipeline, and move to the next frame.
My definition of "good enough" is as follows:

- The lane with is between 3.3m and 4.2m
- The car offset is less than (in absolute value) 45cm
- The difference between the poly fit coefficient of the new line and the averaged poly fit coefficients in the buffer is less than 30 (which is a value I chose looking at some observations)

If the lines are not good enough, then I clear the buffer and try the histogram approach once again until the buffer is full.

My current implementation of the pipeline works well (it is not perfect) only for the project video. Thanks to the smoothing process and the red filtering, I can discard the effects of shadows and the effect of changing color of driving surface.

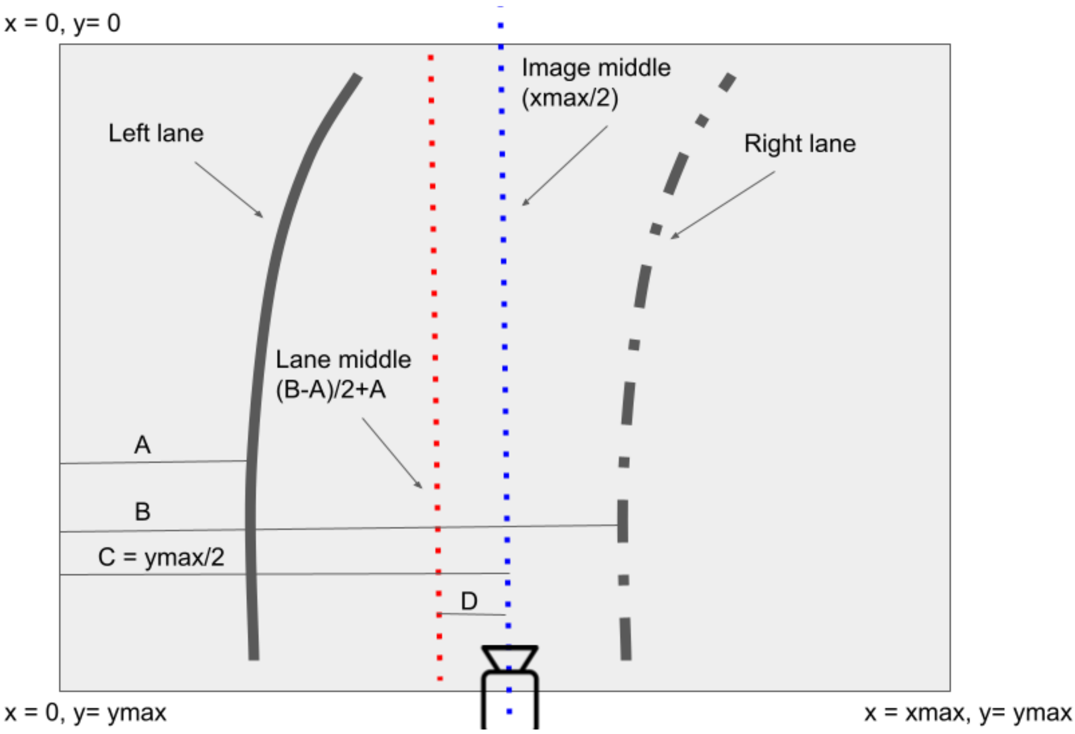
Nevertheless, the pipeline does not work reliably with the challenge video and the harder challenge video.

The challenge video is characterised by many black lines that overlap the road lanes for some time and then diverge from them. My current definition of good enough lines is not robust to discard such lines. Possible improvements could be checking the curvature radius of the new lines and discard them if they diverge too much. Another improvement could be to explore additional color and gradient filters to remove the black lines from the binary image.

The harder challenge video shows very windy curves and my current implementation of the pipeline works vey bad. First improvement I can think of is to modify the histogram detection approach to detect if one of the sliding windows “touches” the side of the image. This probably happens very often because of the very small curvature radius of the curves in the video. The shining light in front of the camera is also a challenge for the detection, therefore a smarter color and gradients filter should be used.

8. Determine the curvature of the lane and vehicle position with respect to center.

The drawing below shows how I calculated the car offset with respect to the lane middle (the length of the segment D in the drawing). A and B are the x coordinate of the left and right detected lines (the drawing shows A and B calculated for different y values, but I used the same value of y, that is ymax). The offset is calculated as D= [(A+B)/2+A] - xmax/2 that is D=lane middle-Image middle . If D>0 then the is on the left of the lane center.



Conversion from pixels to meters
Following the same thinking as described in the class, I converted the curvature radius and the car offset into meters. The conversion rate I used is based on the visual observation of the different images and on the fact that the legal lane width in the us is 3.7m. Since I observed that in the pixel space two lanes (in the warped/birds-eye-view space) have a distance of about 600 pixels, the conversion rate for the x axis is 3.7/600. For the y axis I based my observation on the fact that dashed lines have a nominal distance of 3m. So the conversion rate I used is 15/720 where 720 is the entire height of the image.

9. Warp the detected lane boundaries back onto the original image.
The figure below shows one frame that was edited with the entire pipeline. The image shows the detected lane boundaries and the corresponding "driving space" between them. The lane detection and driving space colouring was done in the birds-eye-view space and warped back into the image space using the inverse perspective transformation matrix.

10. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.
The image below shows the pipeline applied to the first frame of the project video. The original picture is shown on the top left corner. On the top right corner you can see the result of the pipeline including the curvature radius (in the expected range) and the car offset with respect to the lane center. The figures on the bottom layer show the birds-eye-view of the region of interest and the detected polynomial lines with the vertical sliding windows method.


Original image 2
Tagged image
Average curvr:324.06[m]
Car offset:0.40[m] to the left
Warped result test image 2
Detected lanes with
corresponding polynomial drawn