

Dear Reviewer:

My video does not cleanly fill in the lane region, and I would appreciate it if you can help me figure out why. The code works perfectly on images. I even extracted images from the video and it works perfectly if it is run in sequence. However, it does not run on the video and I have been stuck on this portion for past 10+ days. I have opened a topic on forum, but no one has responded since I posted it three days ago (<https://discussions.udacity.com/t/code-works-fine-on-images-extracted-from-video-but-not-on-video/657710>). I asked my mentor and he said he can take a look when he has time, but suggested I ask the reviewers. So here I am, asking for your assistance. Thank you in advance!

Advanced Lane Finding Project

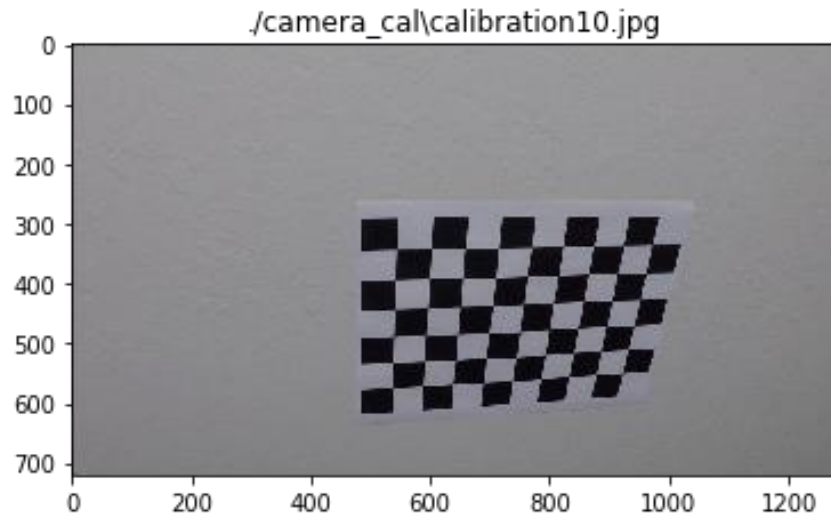
The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration

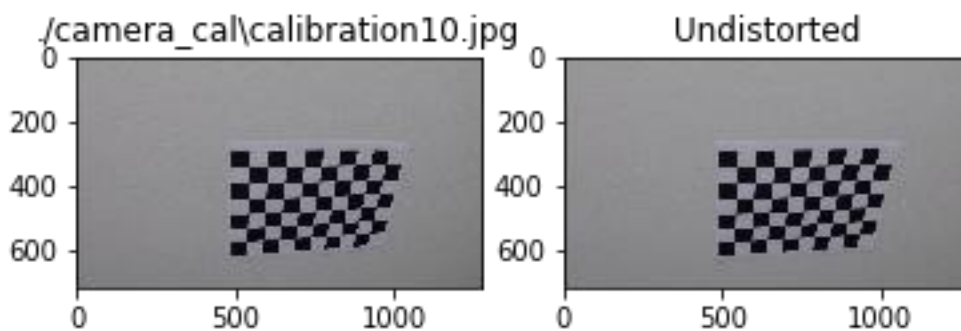
For camera calibration, I used the chessboard images provided by the course. Eighteen of twenty had a shape of 720 by 1280, while the other two had a shape of 721 by 1281. Chessboards are of various sizes are located in places of the image. Here is an example of a chessboard image.

Figure 1: Chessboard image



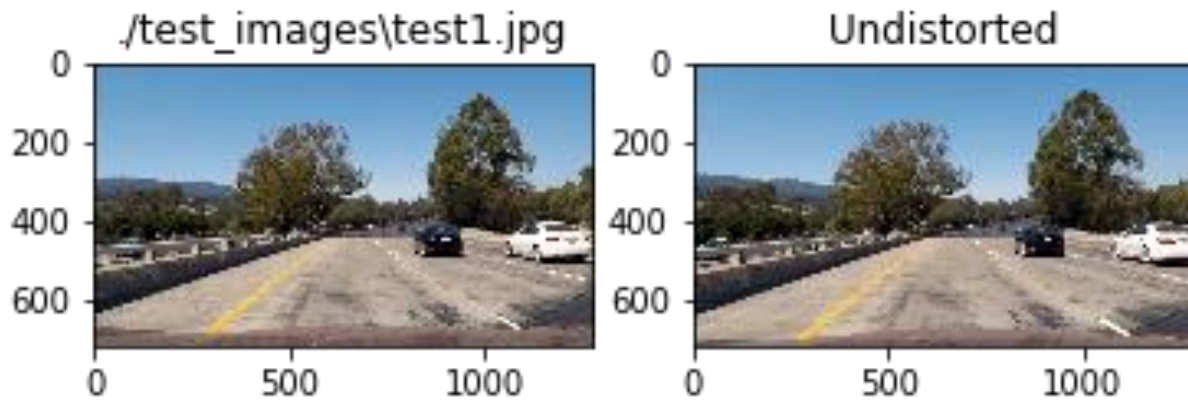
To calibrate the camera, I used the `cv2.drawChessboardCorners`, which uses corners of chessboards. There were 9 points across columns and 6 points up and down rows. Three of the images (images 1, 4, and 5) had at least one chessboard corners that were outside the images, thus corners were not drawn for these images. Figure 2 shows the calibrated chessboard images.

Figure 2: Calibrated chessboard images



I used the same calibration in six test images of the road. Figure 3 shows original and undistorted images of the one test image.

Figure 3: Original and undistorted image of a road



Gradient and color transforms

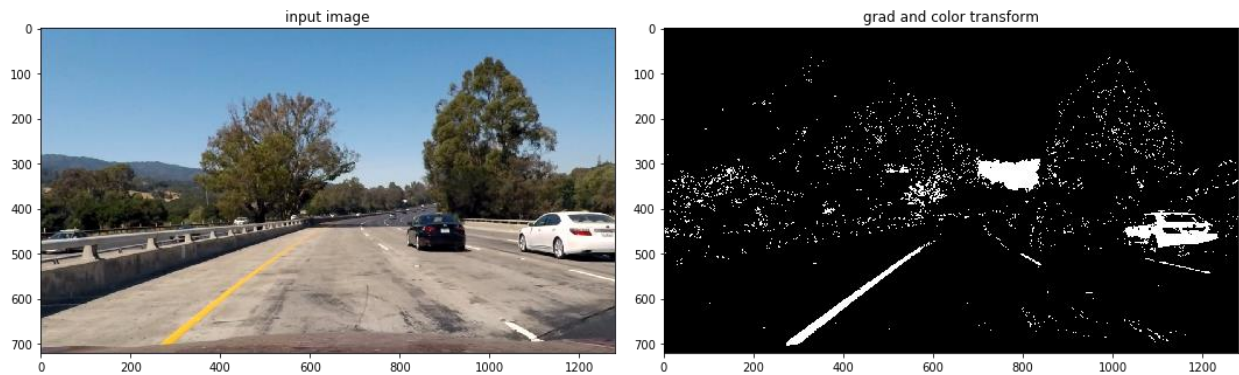
To better capture the lanes from images, I applied gradient and color transforms on the undistorted images.

I had several transformation functions that I tried out using different threshold parameters including following:

- Sobel transformation along x or y axis
- Magnitude of sobel x and y
- Direction of the gradient on sobel x and y
- Transformation on RGB colors
- Transformation on HLS color channel

After iterating through many different combinations, I decided to use the HLS transformation to detect yellow and white lanes and combine with sobel gradient transformation in X direction and direction gradient transformations. Figure 4 shows the image before and after the transformation.

Figure 4: Image before and after transformation



Perspective transform

To better detect lanes, I then applied perspective transformation into bird-eye view. I used the following source and destination points.

Source	Destination
590, 450	200, 50
720, 450	1080, 50
1150, 720	200, 720
250, 720	1080, 720

Figure 5 shows the images before and after perspective transformation.

Figure 5: Image before and after perspective transformation

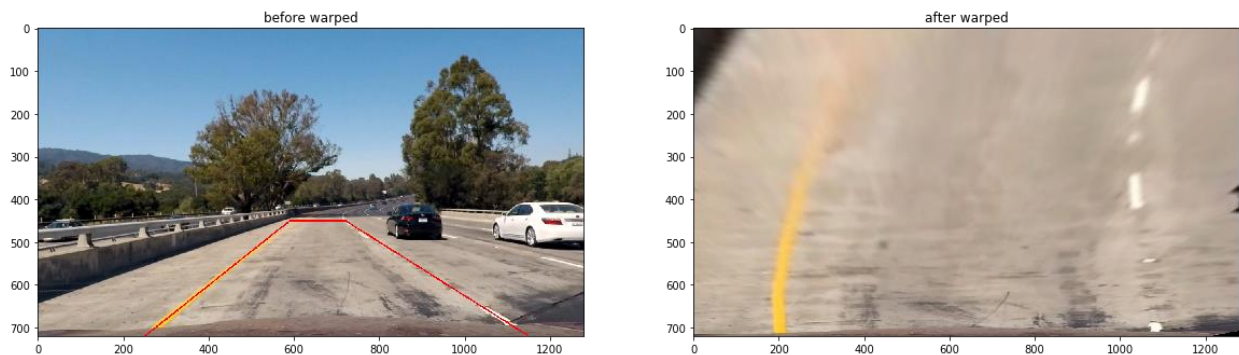
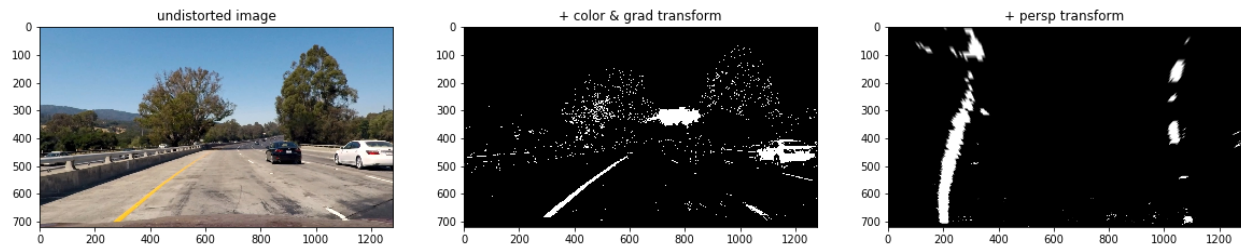


Figure 6 shows an undistorted image after color and gradient transformation followed by a perspective transformation.

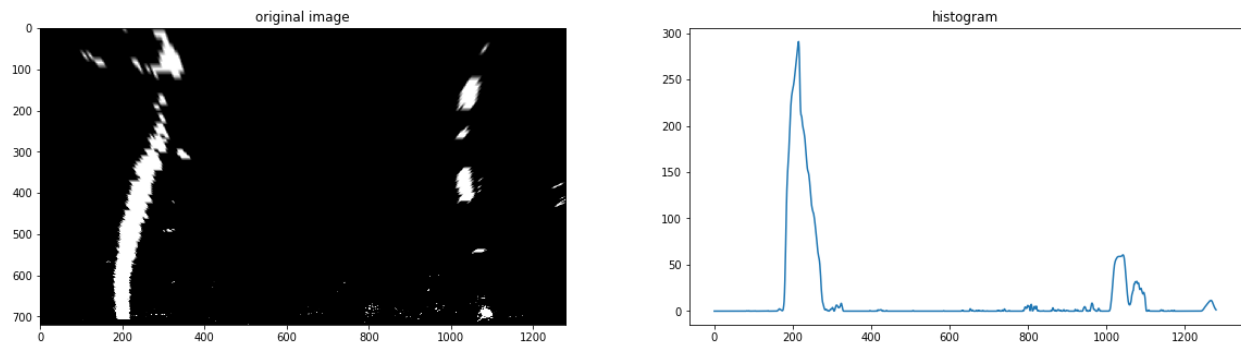
Figure 6: Combined transformation



Finding lanes from the image

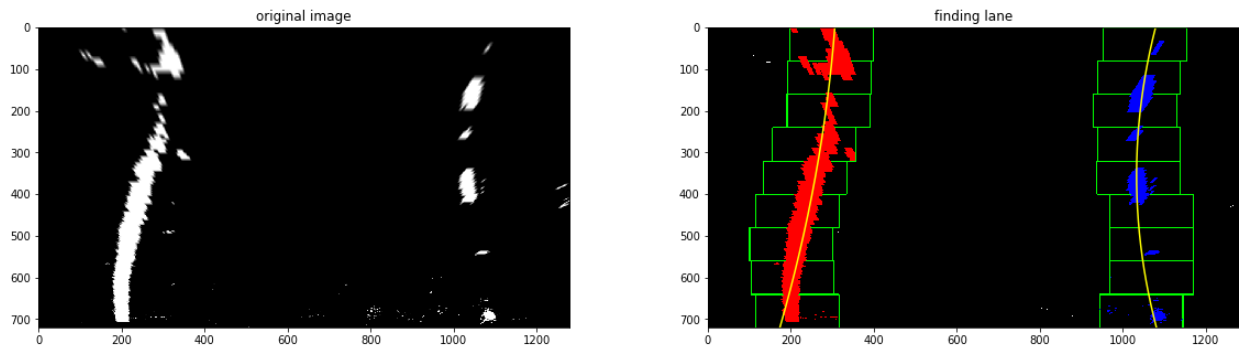
At this point, left and right lanes will be identified from the transformed images. First, we will use the histogram along the columns in the lower half of the image to identify where the lane might be. Figure 7 shows how histogram can be used shows where the lines are.

Figure 7: Histogram to identify lane location



Next we will add sliding windows to lane locations derived from histogram and fit a polynomial line as seen in figure 8. I used the 2nd order polynomial. Additionally, we can use the previous fit to estimate the new fit to save computation time.

Figure 8: Using sliding window to locate lanes



Here is an example of an image that goes through above transformations and identifications, and with the lane region highlighted.

Figure 9: Lane region plotted



Using in the video.

As I mentioned in the beginning, lane region is not clearly identified in the video as you can see in the `project_video_output.mp4`. If I extract the images from the video and run them sequentially, they work fine as seen in “This is running with images extracted from video. It works fine” section of the jupyter notebook codes. Here, I only included six seconds of image clips to save memory.

One thing I noticed was if I add `time.sleep(num_seconds)` line to my video pipeline code, my video output will change. I don’t understand why pausing a bit would change my output.

Also, if there is anything that I am missing on this write-up, please let me know as well.

Once again, I really appreciate the help!