Advanced Lane Lines

Advanced Lane Finding Project

The goals / steps of this project are the following:

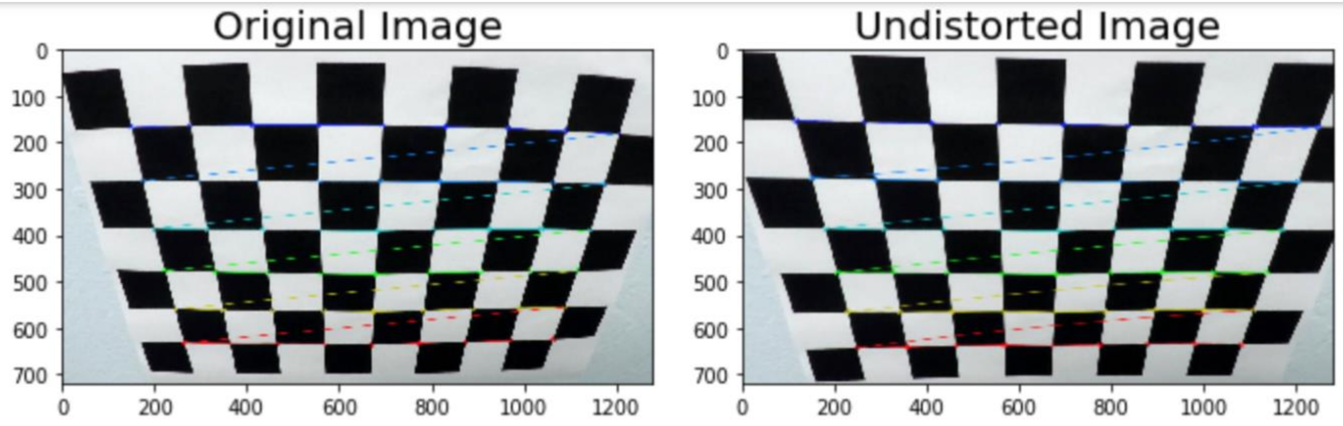
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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

1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

The code for this step is contained in the second code cell of the IPython notebook. I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

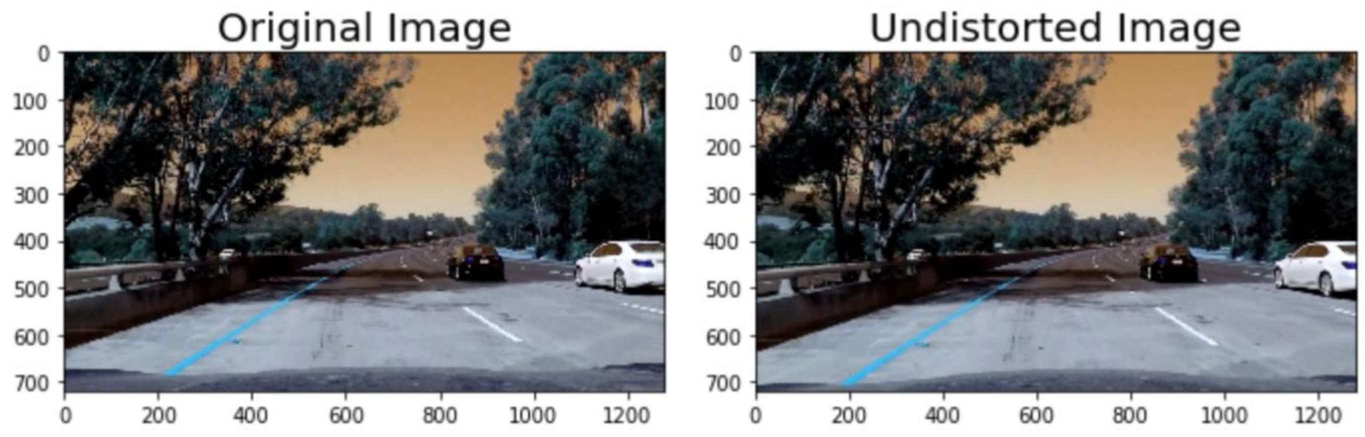
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:



Pipeline (single images)

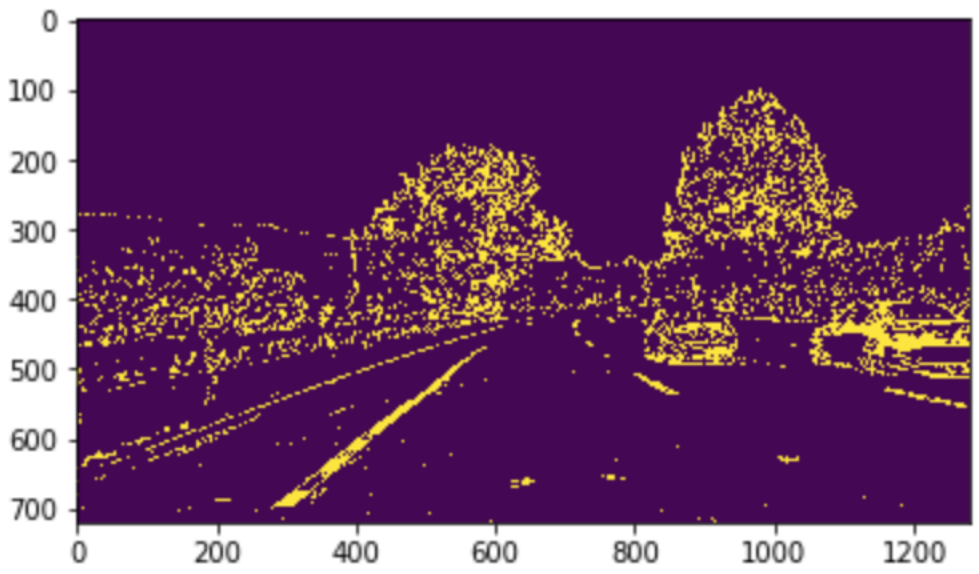
1. Has the distortion correction been correctly applied to each image?

I applied the distortion correction to the test raw image using the cv2.undistort() function and obtained this result:

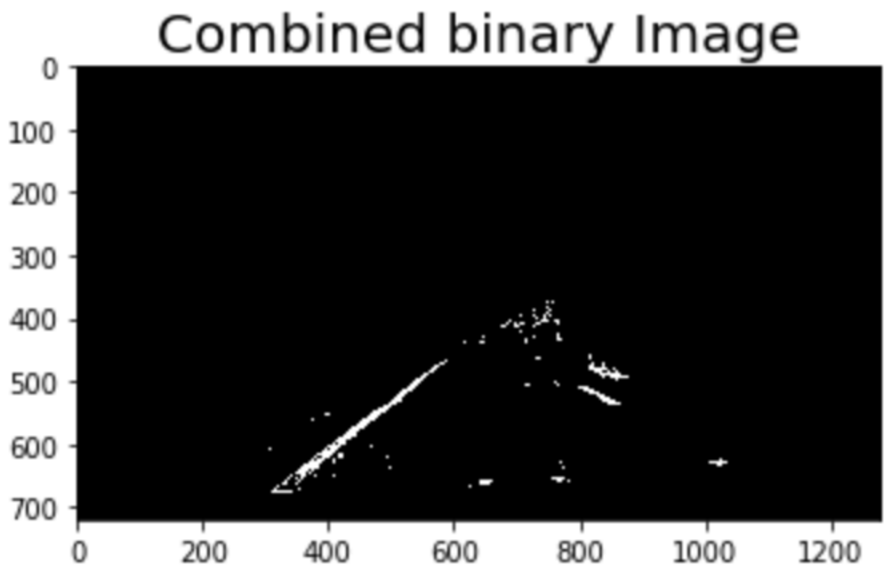


2. Has a binary image been created using color transforms, gradients or other methods?

I experimented with multiple color channels and found s channel from hls color space to be most useful. I combined the information from s color channel , b channel from lab color space(for shading and brightness changes) and l channel from luv(for white lines) and r channel from rbg(for yellow lines) along with scaled sobel mask from derivative on x dimension to create a thresholded image.



To further reduce the noise I also used a region of interest mask. I got the below result using the mask:



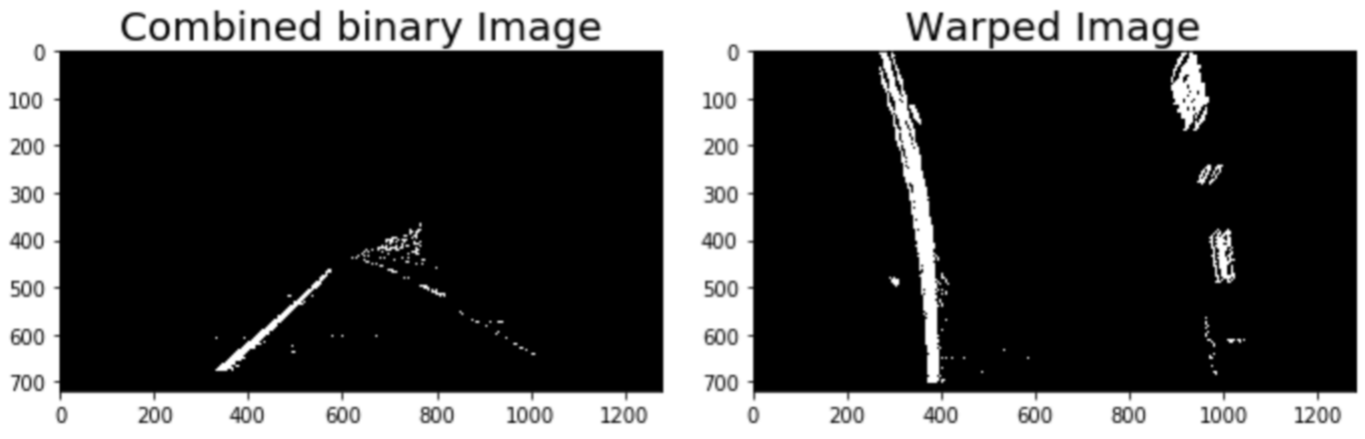
3. Has a perspective transform been applied to rectify the image?

The code for my perspective transform includes a function called perspective () , which appears in 6th cell in the IPYTHON notebook. The perspective () function takes as inputs an image (img), as well as source (src) and destination (dst) points. I chose the hardcode the source and destination points in the following manner:

src = np.float32([(203,720),(1099,720),(707,463),(580,463)])

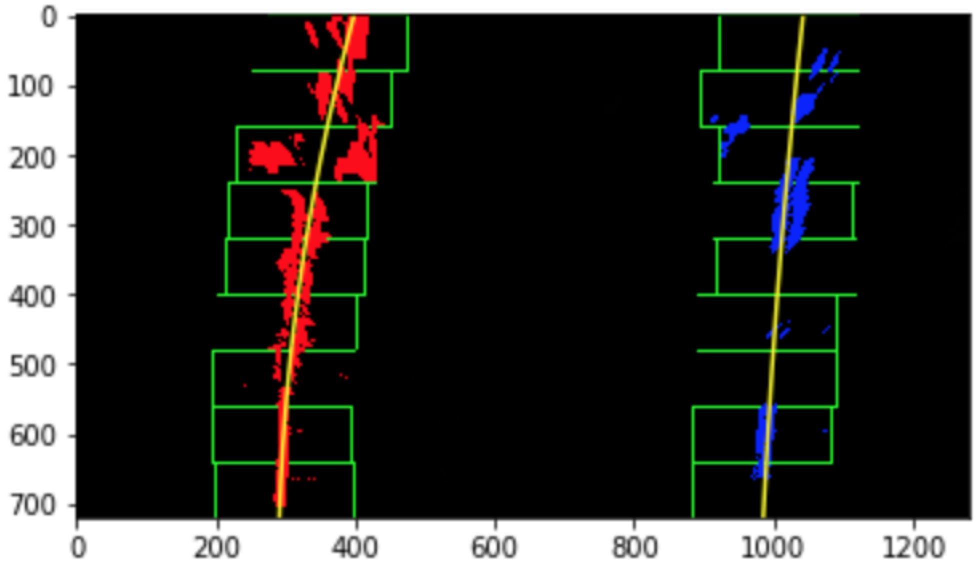
dst = np.float32([(203,720),(1099,720),(1099,0),(203,0)])

I got the below result with perspective transform applied :

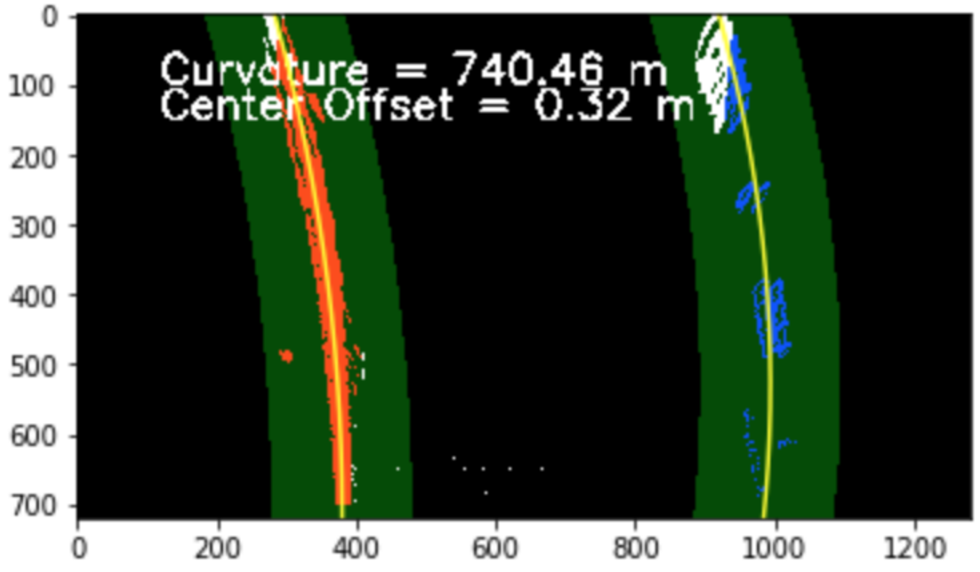


4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

To identify the lane lines in the warped image I took the histogram of the bottom half of the image and used the peaks in the histogram graph as starting points for lane lines. Then I used sliding window technique to slide a window to the top of the image from starting points to identify possible lane lines. Then we feed these points to polyfit() function to fit a curve to the lane lines and identify them as left and right lanes. I got the below result:



I also implemented a search from prior sliding window function which does a highly targeted search around lane lines found in previous frames to find them in the next frame. This allows for a more efficient way to identify lane lines.



5. Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

We located the lane line pixels, used their x and y pixel positions to fit a second order polynomial curve:

f(y) = Ay^2 + By + C*f*(*y*)=*Ay*2+*By*+*C*

The radius of curvature at any point xx of the function x = f(y)x=f(y) is given as follows:

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

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:

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

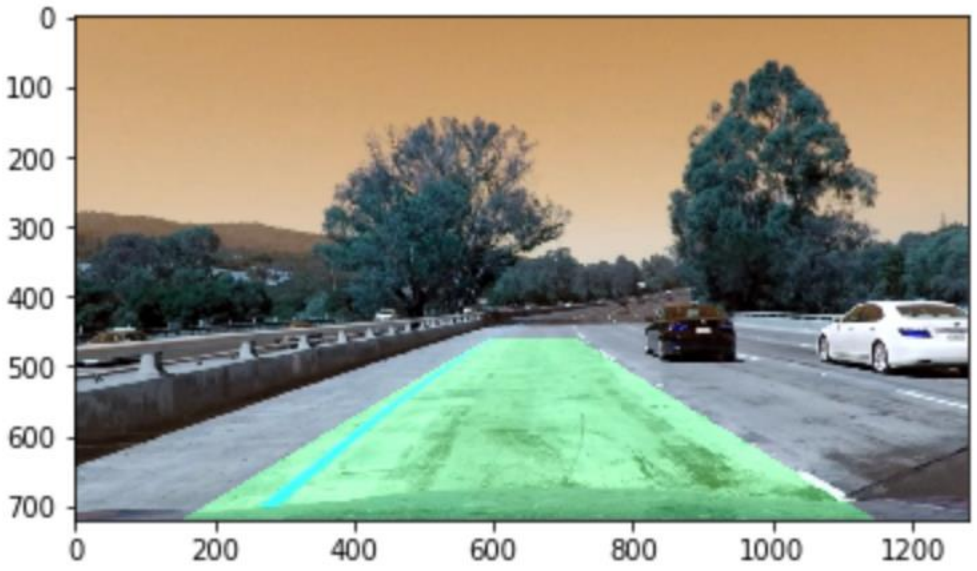
We use the above formula to calculate the curvature of left and right lanes and then average them to get the curvature of the road. For this project, we can assume the lane is about 30 meters long and 3.7 meters wide. So we use the below constants to translate measurements from pixel space to meters space:

```
# Define conversions in x and y from pixels space to meters
```

```
ym_per_pix = 30/720 # meters per pixel in y dimension
xm_per_pix = 3.7/896 # meters per pixel in x dimension
```

To calculate the center offset of the vehicle I first calculated lane center using distance between left and right fitted lines. Then I calculated car center assuming car camera is at the center of image.To get the center offset we subtract lane center from car center and then multiply it with x_m_per_pix .

We the warp the detected lane boundaries back onto the original image using cv2.warpPerspective() function. Results from the same can be seen below:



Pipeline (video)

1. Does the pipeline established with the test images work to process the video?

Yes it does. I have included the processed video file at test_videos/project_video_output.mp4

Discussion:

While the pipeline did pretty well on the project video it still has some limitations. I didn’t fully utilise the Line class to track important parameters and did not use them in deciding how to efficiently detect lane lines due to shortage of time. Experimenting with different color channels and using them to create the thresholded image will certainly improve the robustness of lane detection. I will also explore on the ways to tackle sharp turns and flares on the camera to make the pipeline work for challenge and harder challenge videos.