Advanced Lane Finding

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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

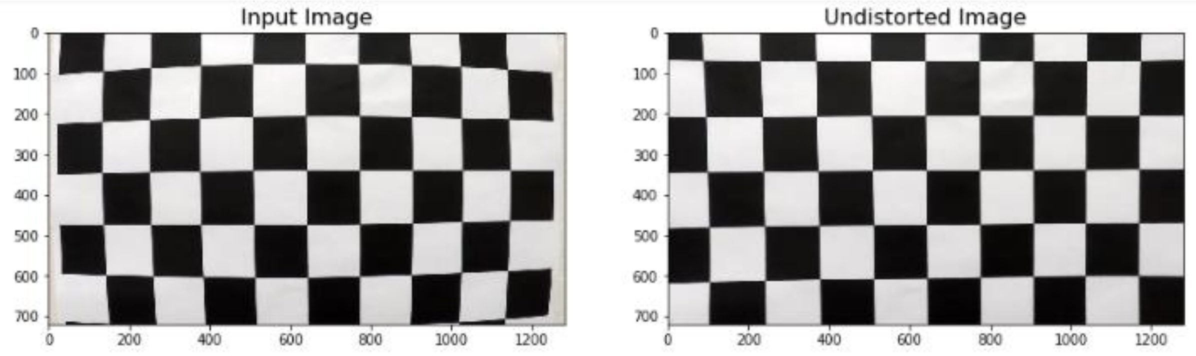
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

You're reading it!

Camera Calibration

1. **Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image. (advanced-lane-finding.ipynb - Camera calibration using chessboard imgs)**

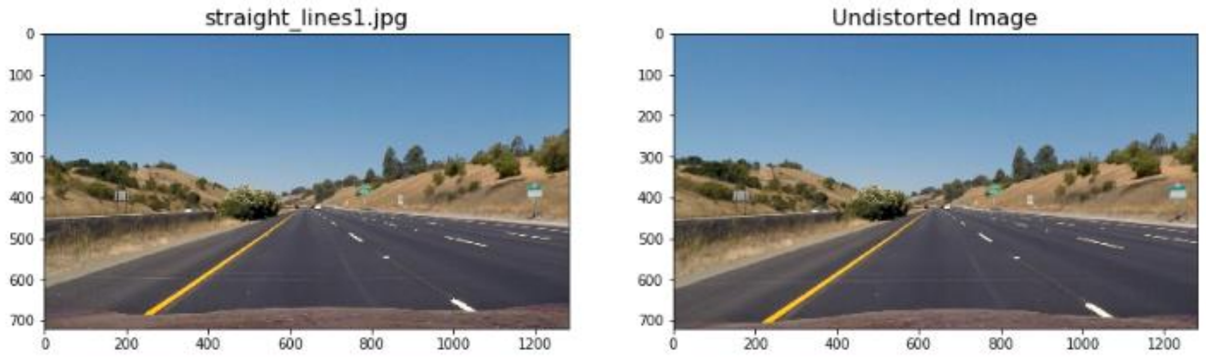
    - Used the OpenCV functions findChessboardCorners() and drawChessboardCorners() to automatically find and draw corners in the image of a chessboard pattern.
    - Used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. Applied this distortion correction to the test image using the `cv2.undistort()` function



Pipeline (single images)

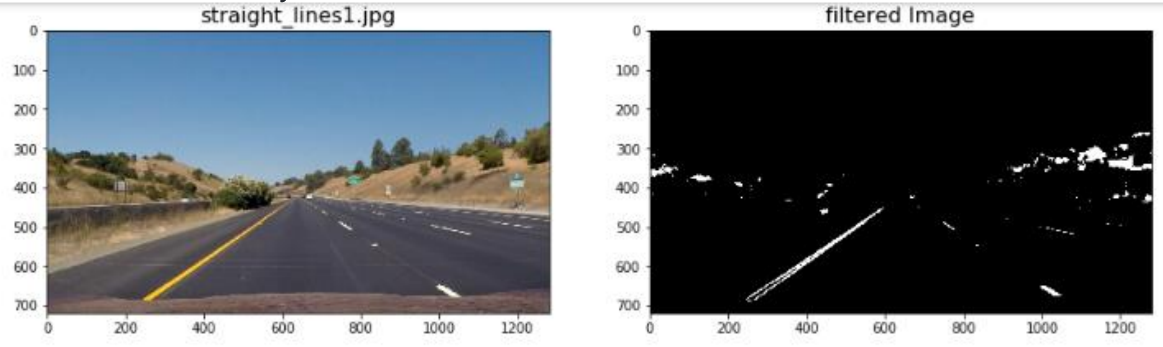
1. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

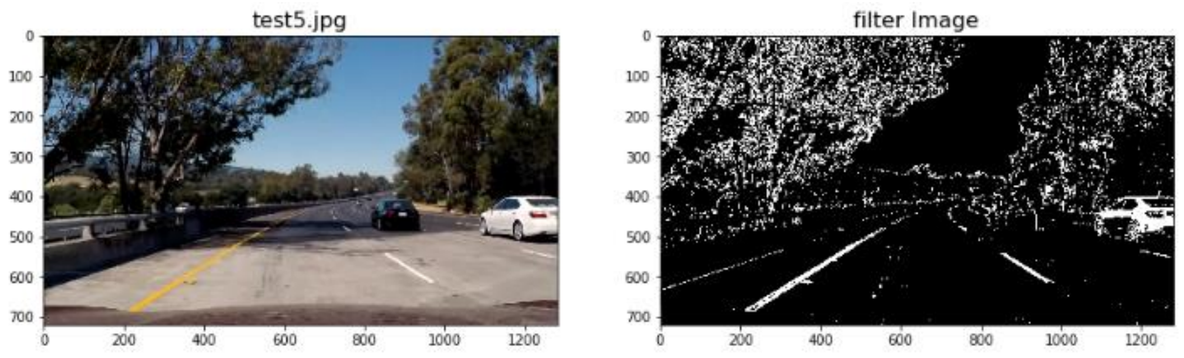


2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a threshold binary image. Provide an example of a binary image result. (advanced-lane-finding.ipynb - Warping and Color filter)

- Converted the image to different color spaces and created binary threshold images which highlight only lane lines and ignore everything else
- Following color channels were used:
  - 'S' Channel from HLS color space, with a minimum threshold = 180 & max threshold = 255
    - Pro: Identifies the white and yellow lane lines,
    - Cons: Did not pick up 100% of the pixels and has the tendency to get distracted by shadows on the road.
  - 'L' Channel from LUV color space, with a min threshold = 225 & max threshold = 255,
    - Pro: Picks up almost all the white lane lines.
    - Cons: completely ignores the yellow lines.
  - 'B' channel from the LAB color space, with a min threshold = 155 & max threshold = 200,
    - Pro: Identifies the yellow lines much better than S channel
    - Cons: Completely ignores the white lines.
  - Created a combined binary threshold based on the above three mentioned binary thresholds.



  - Gradient threshold: Used Sobel X taking derivative in the X direction and combining it with the resulted image from the color threshold.

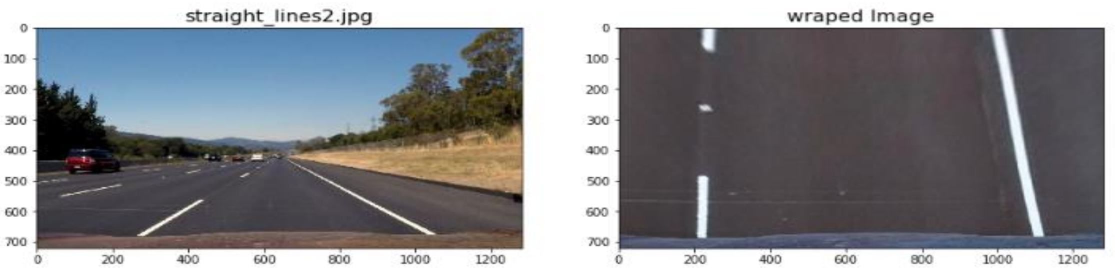


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image. (advanced-lane-finding.ipynb - Warping and Color filter)

- The code for perspective transform includes a function called birds_eye_view()
- The function takes as inputs an image, as well as source and destination points. I chose to hardcode the source and destination points :

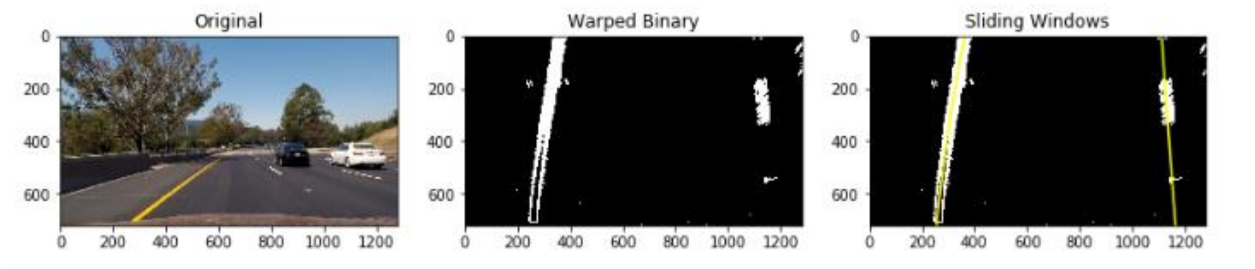
This resulted in the following source and destination points:

Source	Destination
490, 482	0, 0
810, 482	1280, 0
1250, 720	1250, 720
40, 720	40, 720



4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial? (advanced-lane-finding.ipynb - Sliding window search technique)

- Used combined binary image from the previous step to isolate lane line pixels and fit a polynomial to each of the lane lines.
- Taken a histogram of the bottom half of the image and found the peak of the left and right halves of the histogram These will be the starting point for the left and right lines
- Used sliding windows and the non-zero pixels are identified in each window and these indices are appended to a list and concatenated.
- Second order polynomial is fit to each thus we get the lane lines as shown below.

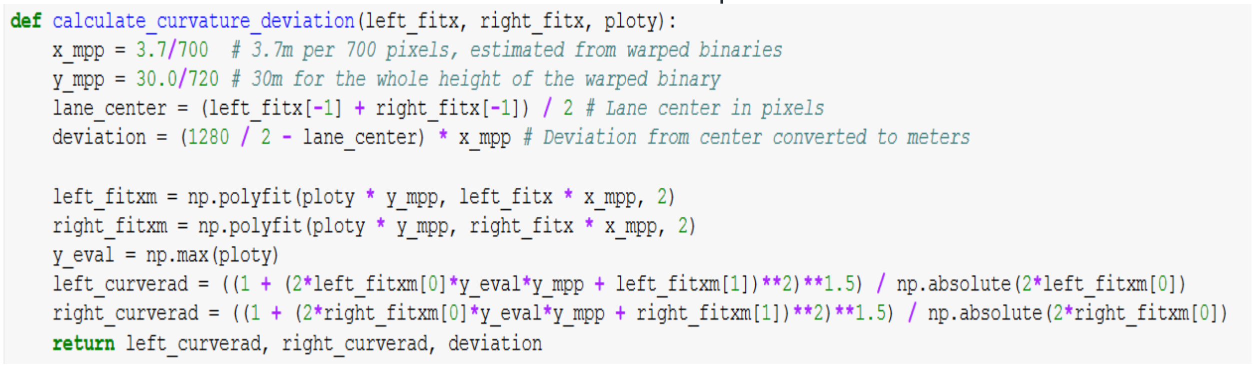


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center. (advanced-lane-finding.ipynb - Curvature and Drawing)

- Here the polynomials are used to calculate the curvature of the lanes using the formula.

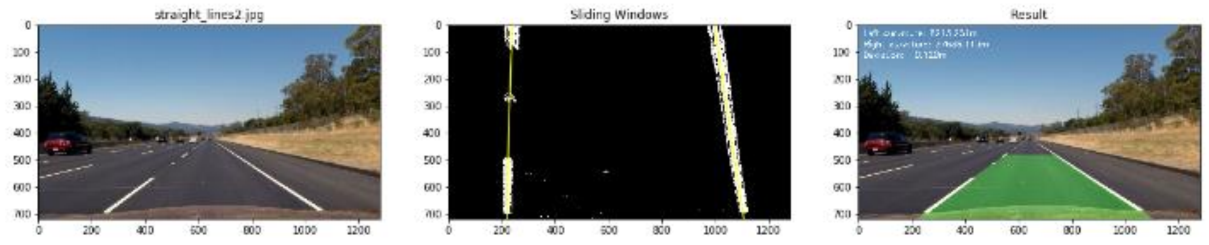
$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

- Deviation from the center is also calculated and implemented as follows.



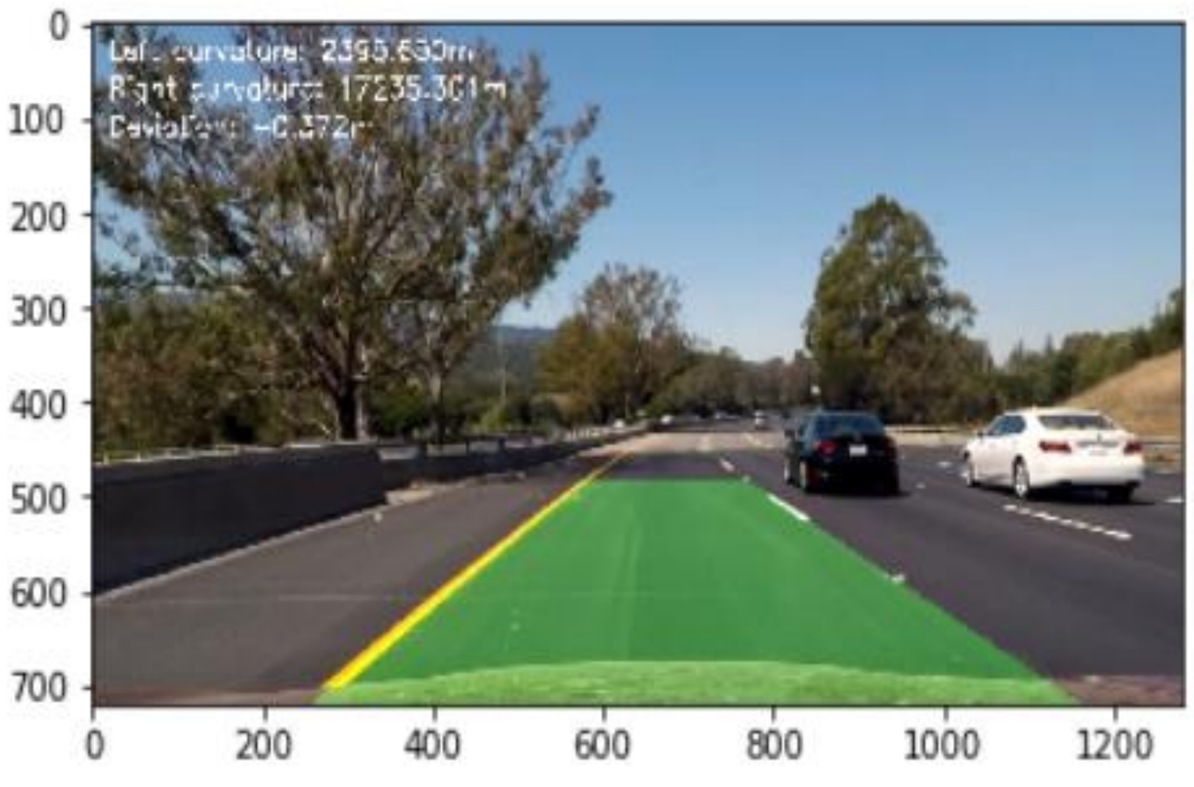
```python
def calculate_curvature_deviation(left_fitx, right_fitx, ploty):
    x_mpp = 3.7/700  # 3.7m per 700 pixels, estimated from warped binaries
    y_mpp = 30.0/720 # 30m for the whole height of the warped binary
    lane_center = (left_fitx[-1] + right_fitx[-1]) / 2 # Lane center in pixels
    deviation = (1280 / 2 - lane_center) * x_mpp # Deviation from center converted to meters

    left_fitxm = np.polyfit(ploty * y_mpp, left_fitx * x_mpp, 2)
    right_fitxm = np.polyfit(ploty * y_mpp, right_fitx * x_mpp, 2)
    y_eval = np.max(ploty)
    left_curverad = ((1 + (2*left_fitxm[0]*y_eval*y_mpp + left_fitxm[1])**2)**1.5) / np.absolute(2*left_fitxm[0])
    right_curverad = ((1 + (2*right_fitxm[0]*y_eval*y_mpp + right_fitxm[1])**2)**1.5) / np.absolute(2*right_fitxm[0])
    return left_curverad, right_curverad, deviation
```

- Next, the lane positions and a polygon are drawn to the original image.



6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly. (advanced-lane-finding.ipynb - Curvature and Drawing)

Here is an example of my result on a test image:



Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Now taking everything together, a pipeline is built to process the project video:

1. Perform camera distortion correction

2. Apply the color filter to build the binary image where the lane pixels dominate

3. Apply birds-eye-view transformation

4. Use the sliding search technique (due to better results than the convolution technique) to find the lane pixels positions

5. Perform a polynomial fit of both lane pixels, drawing a polygon and lane lines on warped space

6. Apply the reverse warp to the drawings

7. Merge the drawings and undistorted image

8. Measure and annotate curvature, deviation

9. Cache the results on each frame

10. Use a weighted average between frames to smooth out outliers

11. Use the cached lanes for sanity checking, refusing the ones found if they differ too much

12. Subsample by skipping every other frame to improve performance

The Video is attached with the submission folder.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

There were many issues addressed to produce the desired pipeline output, few of these being

- The various parameters used for perspective transform, thresholding both in color and gradient etc. required a lot of tuning and tweaking.

- Still the challenge video is not working properly and require working on these parameters.

Further improvements

1. Improve my computer vision pipeline. Presently, it works with project video, but if more experimentation is done on the color and threshold gradient it can work properly for other videos as well. Other parameters also require tuning.
2. I would like to explore machine learning approaches suitable to address lane finding problem.