Advanced Lane Lines – term 1 – project 4

Writeup Template

By ~Karanj

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

My project includes the following files:

- camera_calibration.ipynb: file containing Chessboard calibration and produces distortion matrix
- Advanced_Lane_find.ipynb: Final Project File
- project_video_output.mp4: Final Video output file.
- writeup_report_Advanced_Lane_Lines.pdf summarizing the results

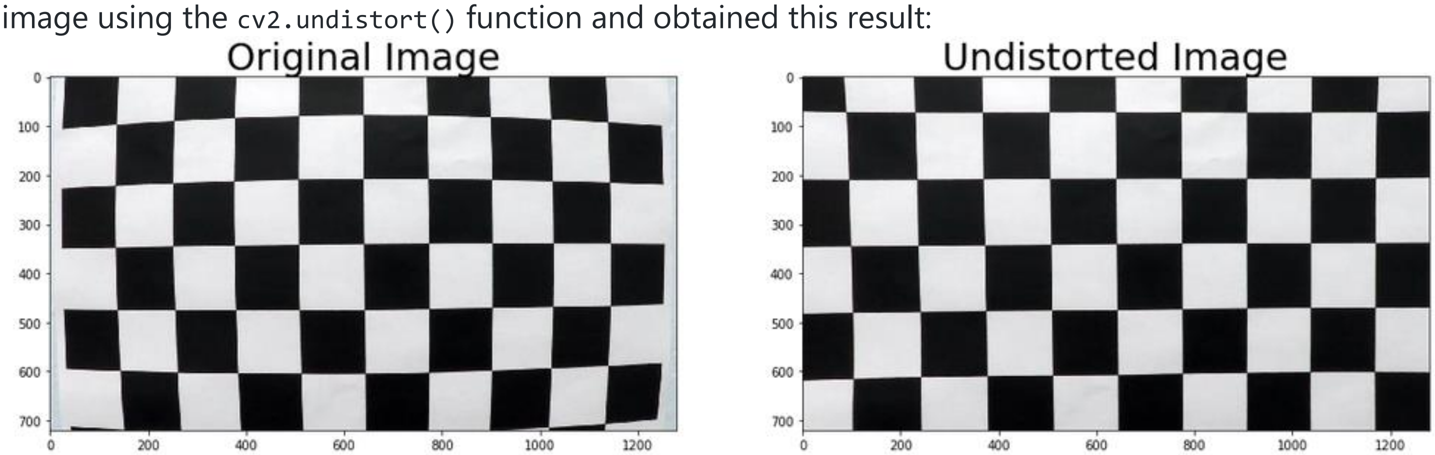
You're reading it! and here is a link to my [project code (ipynb)].

- Important Note :
  - Cells marked with *** (i.e. Cell12 and Cel 13...) are not called in final pipeline for video rather they are directly implemented in Cell 21
  - Untitled-Copy3.ipynb gives better view of distinctive functions and implementation just for single image (not video)

Camera Calibration

1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

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:



Also, I saved the camera calibration result for later use ("camera_cal/output/cal_dist_pickle.p"). All other test images (Input: “camera_cal/”) output are saved at “camera_cal/output/”.

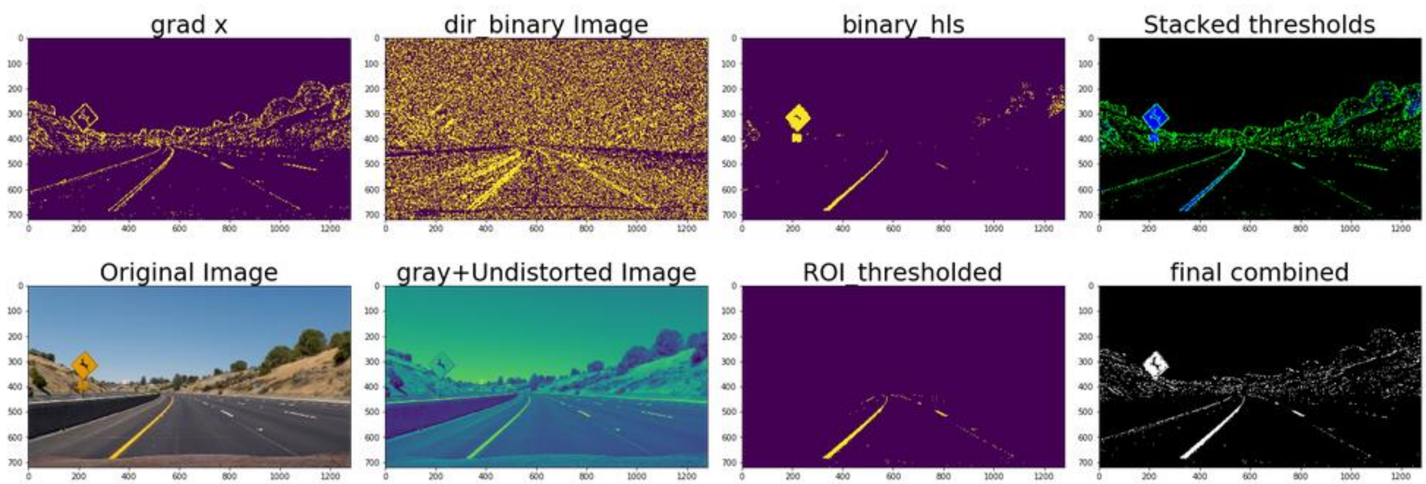
Pipeline (single image)

1. Processing each image (Cell 2 to Cell 10)

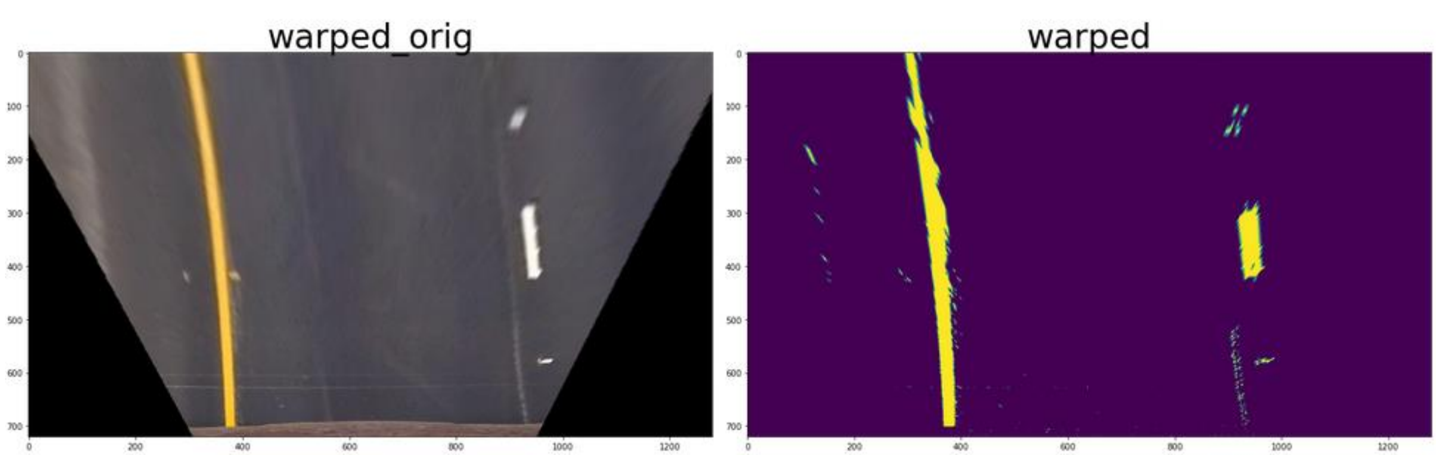
- Processing done in these cells are :
    - Load saved distortion matrix
    - Function to undistort images
    - Grayscale conversion
    - Absolute sobel threshold x and y direction (we used x direction only)
    - Rgb threshold
    - Directional threshold
    - Hls threshold
    - Mask Region of Interest
    - Perspective transform to get bird eye view

2. Thresholding and perspective view (Cell 11 to Cell 13)

- Calling functions from pre-processing above, to apply filters to images. Mask_ROI has 'imshow', which produces ROI image (Cell 11)
- Cell 12: Visualization:

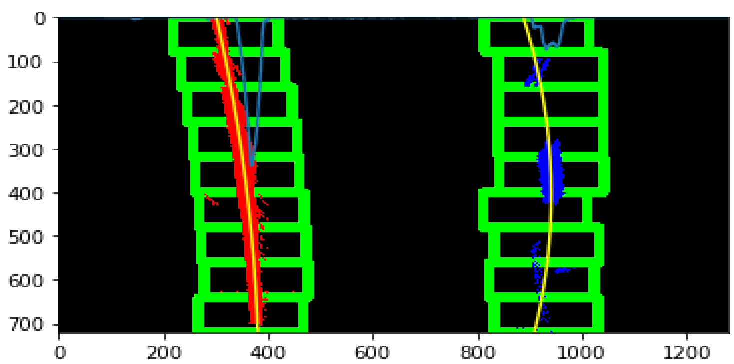


- Cell 13: Warped images (original and grayscale):

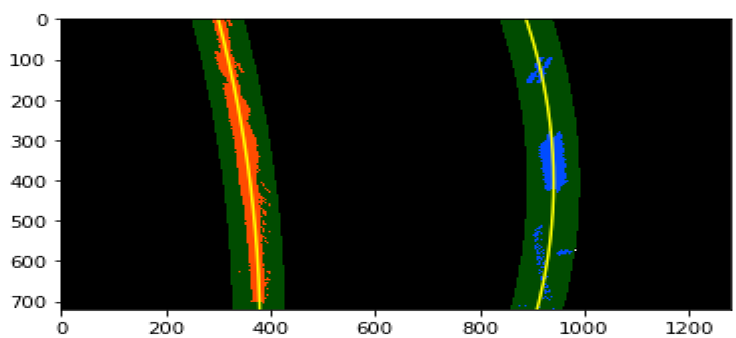


3. Histogram, polynomials and Lanes (Cell 14 to 17)

- Cell 14: Shows Histogram with two peaks superposed on the grayscale image
- Cell 15: Find lanes:
    - Create non-zero array from images
    - Define small windows and its height and width
    - Identify the nonzero pixels in x and y within the window
    - If you found > minpix pixels, recenter next window on their mean position
    - Extract left and right line pixel positions within those windows (leftx, lefty, rightx, righty)
    - Extract left and right line pixel positions within those windows
    - Fit a second order polynomial to fit curve and generate prediction of curved line (left_fitx, right_fitx)
    - Show output:

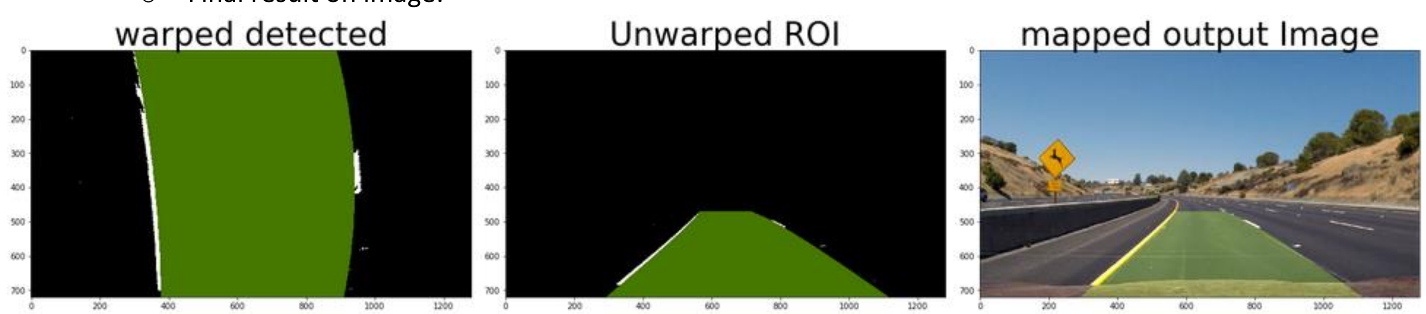


- Cell 16: Find next lines for next frames of videos
- Cell 17: Draw lanes:



4. Radius, offset and mapped output image (Cell 18 to 20)

- Cell 18: radius in pixels
- Cell 19: averaged radius of both lanes and offset of car from lane center
- Cell 20: Inverse perspective transform of detected region between lanes and superpose on original image.
    - Final result on image:



Pipeline (Video)

1. Sanity Check, Soothing and Reset(brute) (Cell 21 and Cell 22)

- Function “get_line_predictions”: Given coordinates of non-zeros pixels and coordinates of non-zeros pixels within the sliding windows, this function generates a prediction for the lane line.
- Function “brute_search”: This function searches for lane lines from scratch.Thresholding & performing a sliding window search.
- Function “get_averaged_line”: This function computes an averaged lane line by averaging over previous good frames.
- Function “get_mean_distance_between_lines”: Returns running weighted average of simple difference between left and right lines
- Function “pipeline_final”: Final pipeleine which uses all of the above functions and” perspactive_transform” and “get_thresholded_image” from Cell 10 and 11. In addition to that it also calculates Radius and offset similar to the pipeline (single image).
- Cell 22: Testing above pipeline with one of the image in ‘test_images/’ folder:

2. Video Processing (Cell 23)

- Output file is saved as: **“project_video_output.mp4”**

Project Discussion:

- Initially, it was complicated to decide which thresholding fuctions to use out of sobelx, sobel,y directional sobel, absolute sobel, magnitude threshold, rgb threshold, hls threshold. Also which of them to combine and how.
    - Sobel in y direction was not very useful as lanes are vertical thru video. Also magnitude thresholding was not helpful for this project. So final combination derived was:
        - combined[((gradx == 1) & (dir_binary == 1)) | ((hls_binary == 1) & (rgb_binary == 1))] = 1
- gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) doesn’t visualize pure grayscale image (it was showing violate and yellow instead of balck and white.)
    - solution : plt.imshow(combined**, cmap='gray'**) can be used to visualize in pure grayscale.
- While measuring radius of curvature (xm_per_pix = 3.7/700 # meters per pixel in x dimension), number ’700’ should be approx. distance between lanes and not width of image frame.
- For harder and challenging videos which may have stronger curves, this pipeline will fail.
    - Averaging previous lane lines won’t work, because position of pixel may vary vastly.