Advanced Lane Finding Project

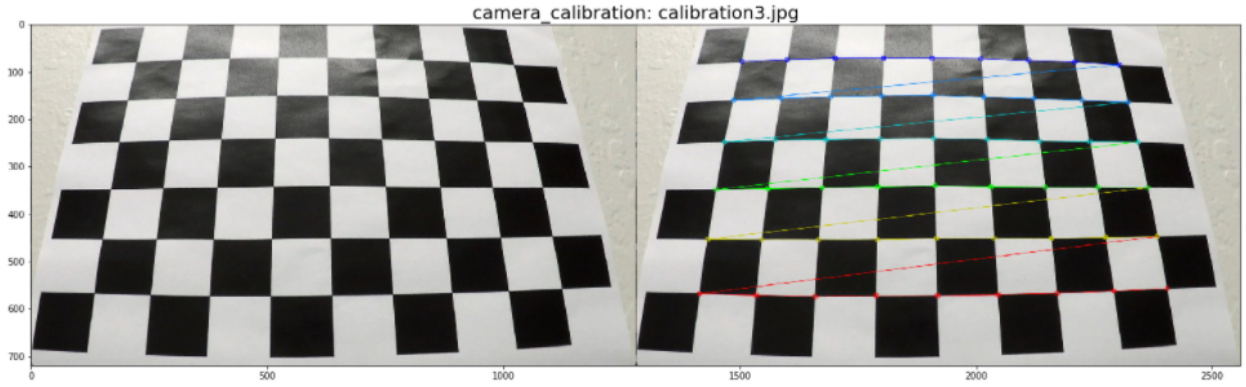
The goals / steps of this project are the following:

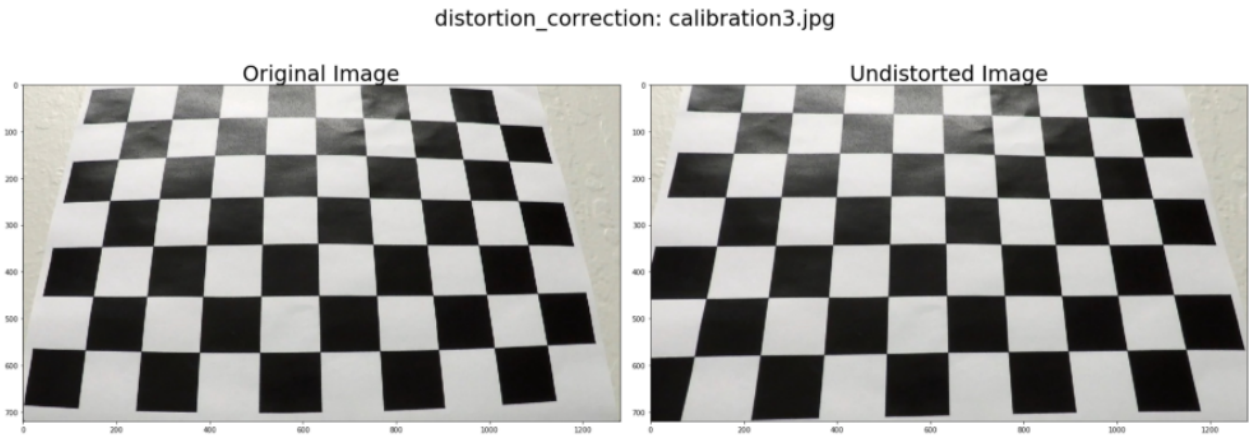
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").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

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

- We created a function “camera_calibration” that takes a group of images for a chessboard and uses it for the calibration parameters of the camera.
- This function is called only one time and we will use the camera calibration matrix and distortion coefficients through the rest of the program.
- 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.





Pipeline (test images)

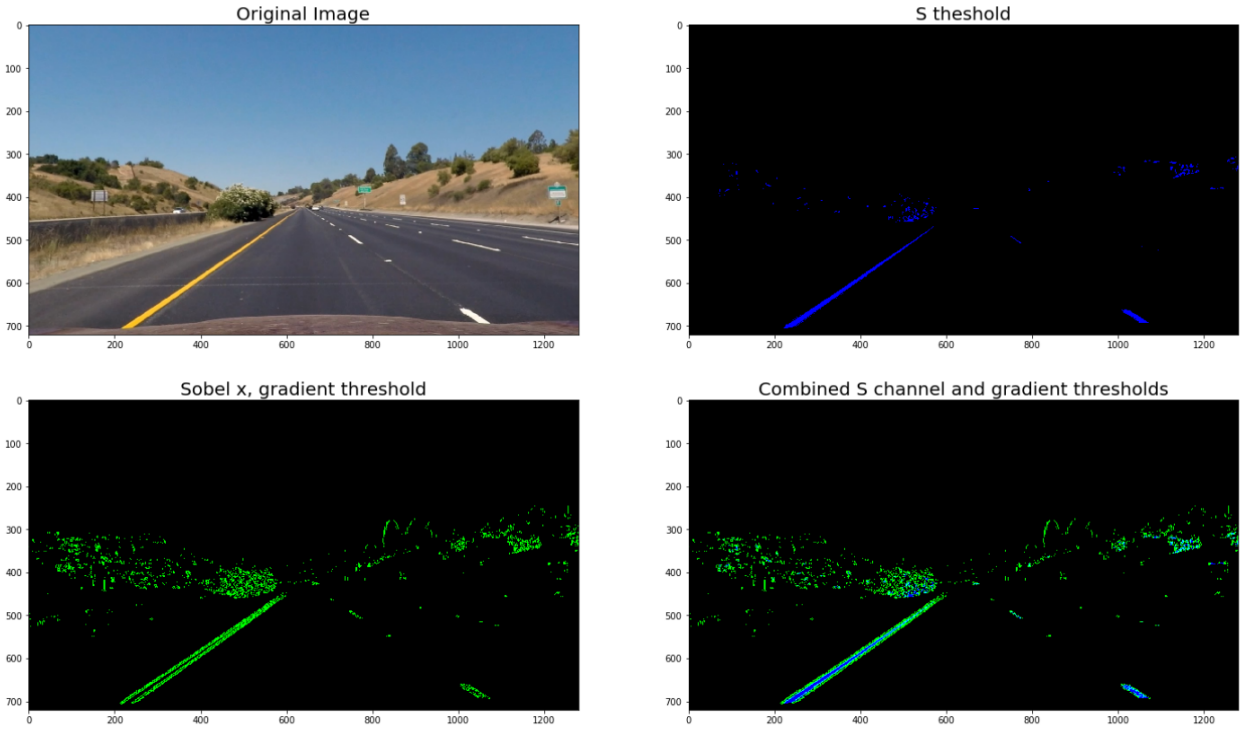
Rubric: “Provide an example of a distortion-corrected image”

1. The function used to undistort an image in the future is “distortion_correction” and it is using the output from the “camera_calibration” function.
2. The core of this function is “cv2.undistort”

Rubric: “Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.”

1. I used a combination of color and gradient thresholds to generate a binary image
2. The function is “color_gradient_threshold”.

color_gradient_threshold: straight_lines1.jpg



Rubric: “Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.”

- The code for my perspective transform includes a function called perspective_transform(),
- The function takes as inputs an image (img). I chose the hardcode the source and destination points in the following manner:

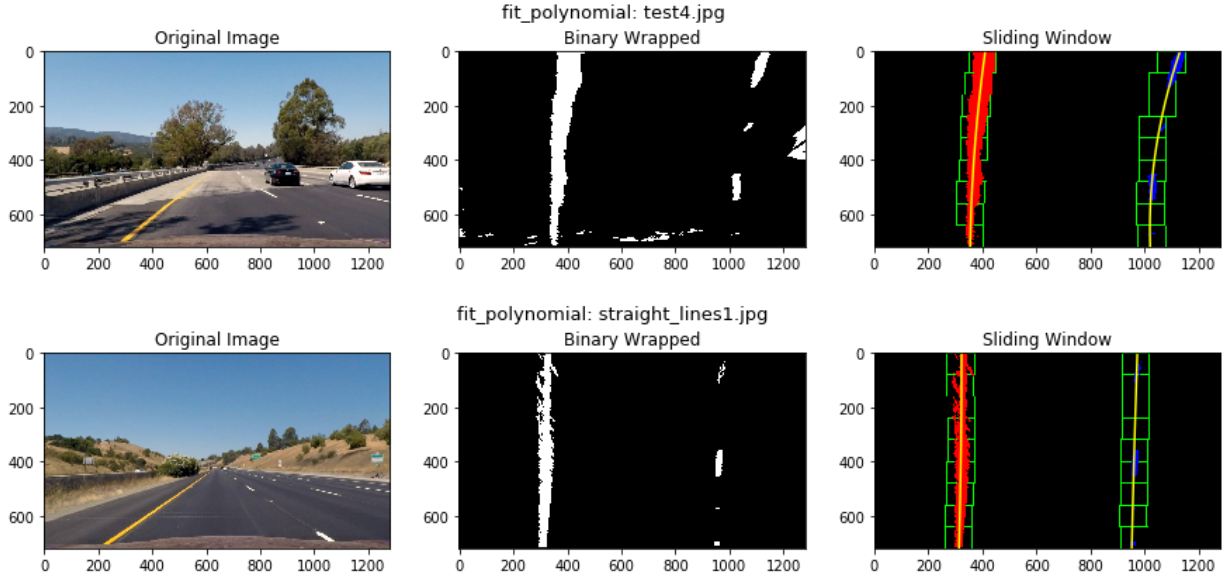
```
src_bottom_left  = (220,  720)
src_upper_left   = (580,  460)
src_upper_right  = (700,  460)
src_bottom_right = (1115, 720)

dst_bottom_left  = (320, 720)
dst_upper_left   = (320, 0  )
dst_upper_right  = (970, 0  )
dst_bottom_right = (960, 720)
```

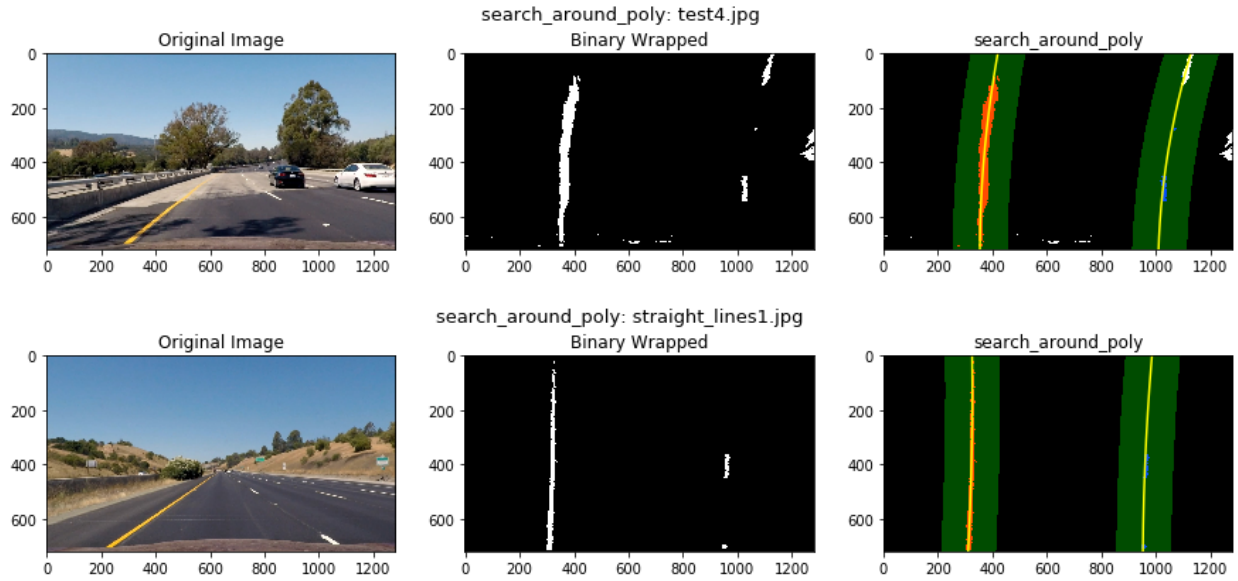
- I verified that my perspective transform was working as expected by drawing the src and dst points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

Rubric: “Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?”

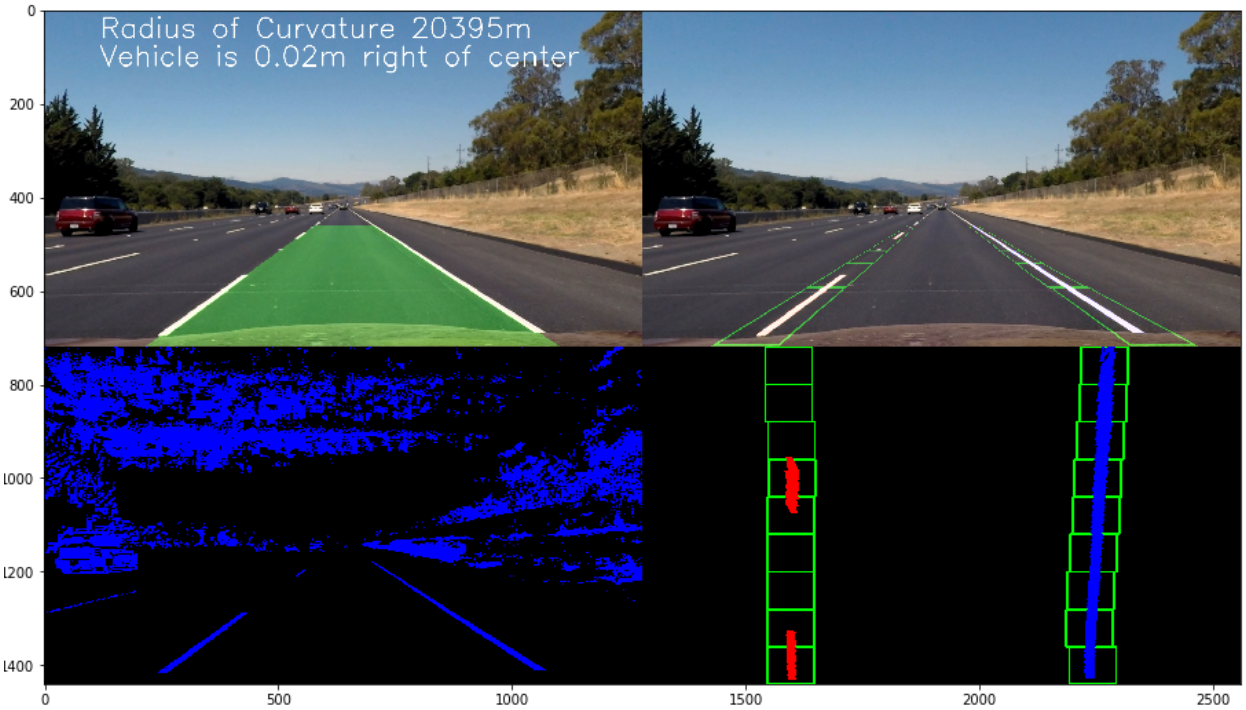
- At the start, I used the sliding window to detect the lane line. I used the function “fit_polynomial”

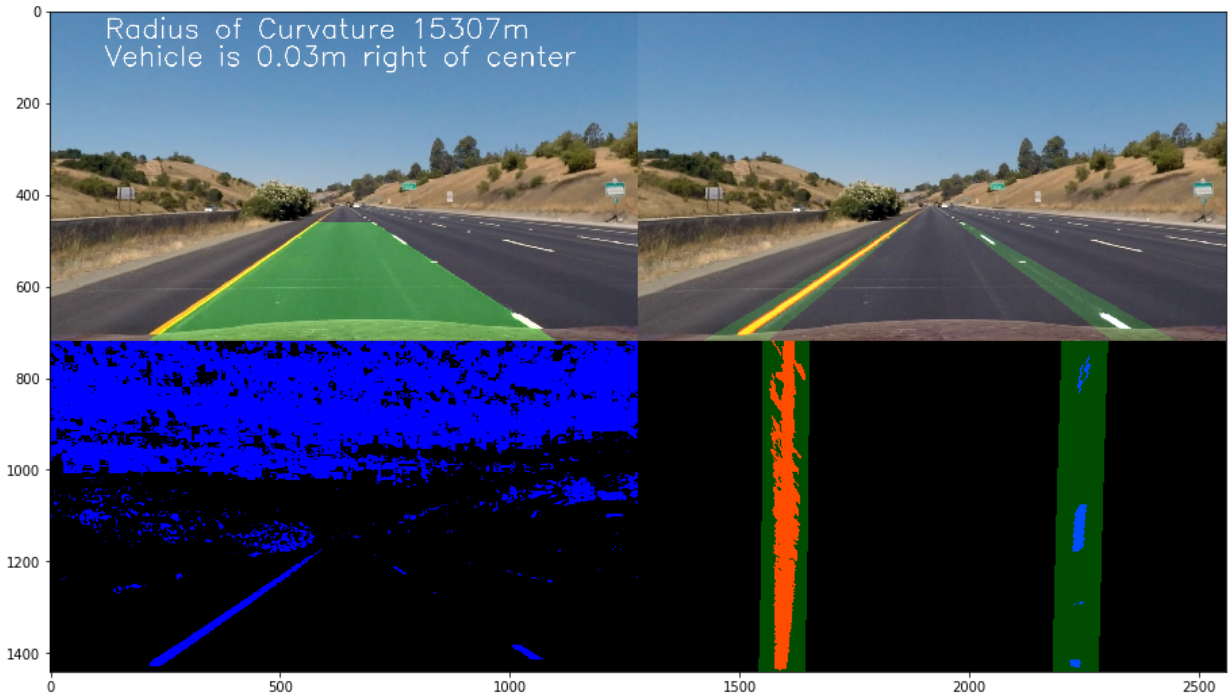


- After that, I used the function “search_around_poly”. It used the previous calculated values of the polynomial function to determine the region of interest for finding the lane.



- If the function “search_around_poly” fails to detect the lane line we can go back to the sliding window approach.



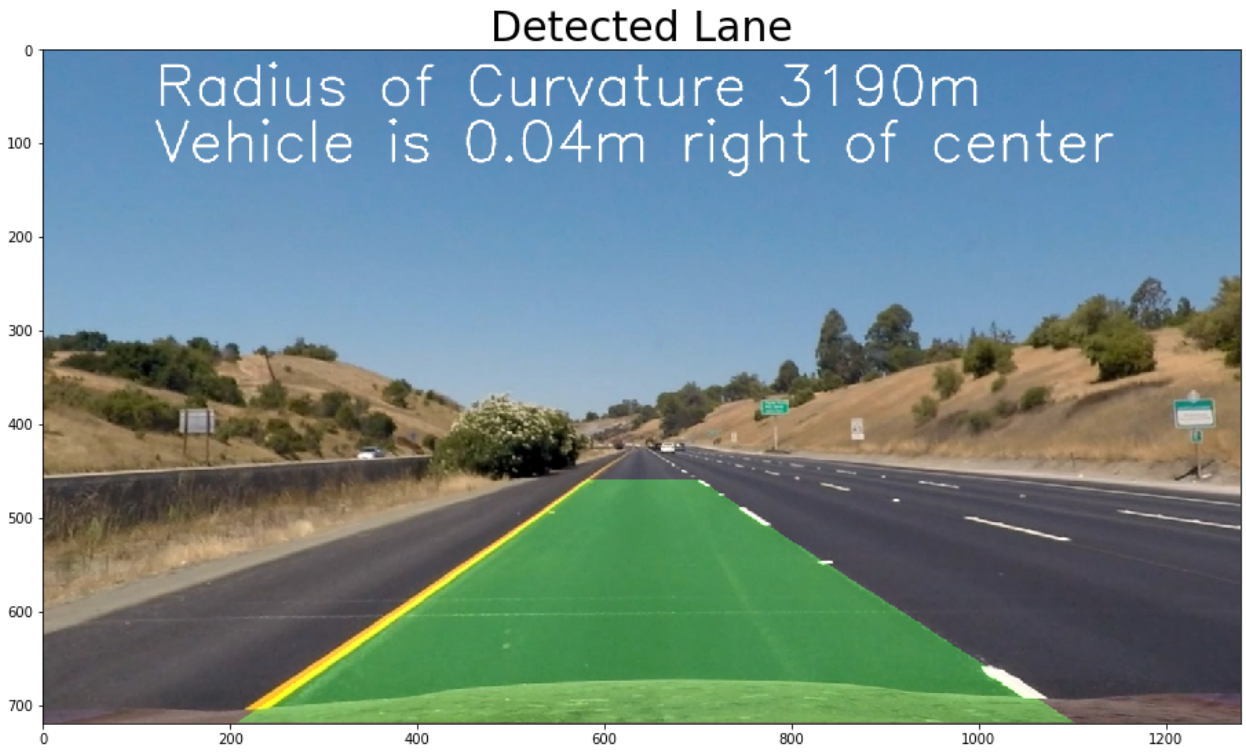


Rubric: “Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.”

- Radius of curvature “measure_curvature_real()”
  - I collected pixel points of the right and left lanes.
  - Converted the to meters instead of pixels
  - Fit a second polynomial for each lane
  - Calculate the left and right curvature.
  - Getting the average lane curvature
- Position of the Vehicle with respect to the center “measure_car_offset()”
  - We assumed that the camera is mounted in the middle of the front windshield
  - We subtracted the averaged the lane locations from the middle of the screen.
  - By knowing the sign of the difference; we get to know if the car is

Rubric: “Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.”

- The pipeline is implemented in the function “lane_line_pipeline()”



Pipeline (video)

Rubric: “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!)”

- The video is present in the main folder with name “Output_project_video.mp4”

Discussion

Rubric: 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?

- I had some issues with color and gradient threshold so that I identified the lane correctly. So I changes multiple time the thresholds to get the best results
- I needed to take into consideration if the search around polynomial fails and we need to return back to the sliding window algorithm
- I had an issue when the algorithm detected the right and the left lanes with the same polynomial and solved this by checking if the coefficients of both lanes are the same. If they are same, I return back to the sliding window algorithm