Advanced Lane Finding Project
Mon 13-Jul-2020
John Hilbun

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:

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

Specification

Camera calibration was accomplished using OpenCV functions. Each image in ./camera_cal/calibration*.jpg was converted to grayscale using cv2.cvtColor(), the corners were determined using cv2.findChessboardCorners() and the results stored in a datastructure. The calibration then occurs at the end of notebook cell 1 using cv2.calibrateCamera() with the previously obtained data structures.

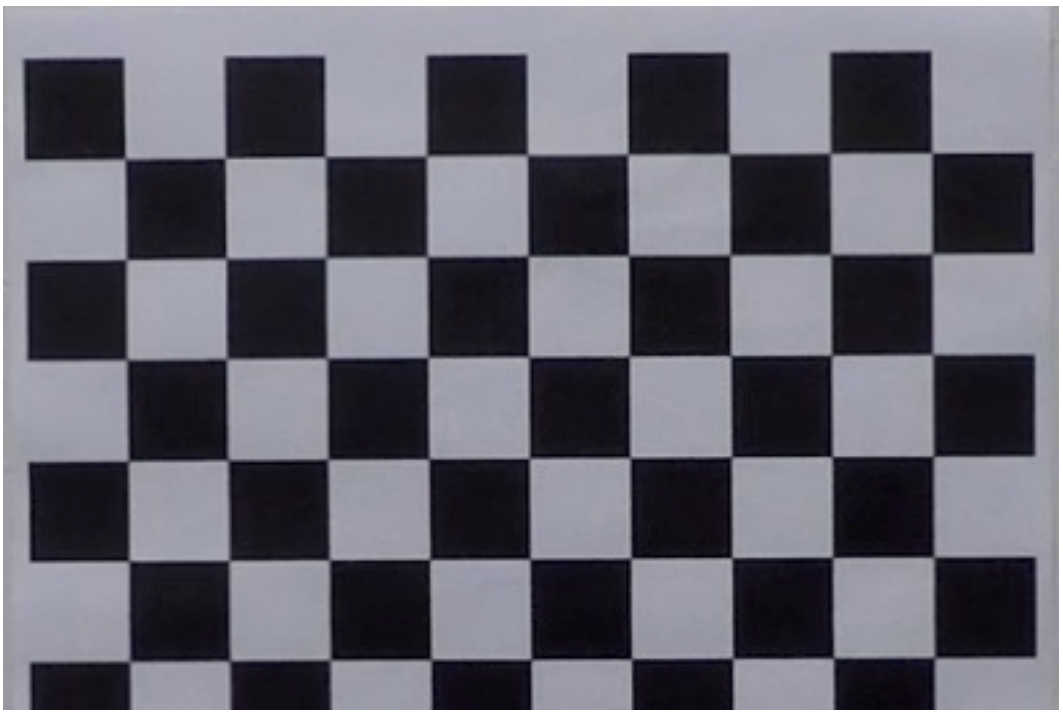

Figure 1: Calibrated Chessboard image

Pipeline (test images)
Provide an example of a distortion-correction image.

Specification

Undistorted image shown in Figure 2.

Figure 2: Undistorted image

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.

Specification

In notebook cell 5 and 6, these threshold functions are defined:

abs_sobel_thresh(x/y orientation is an argument)
- color transformation to grayscale.
- basically a cv2.Sobel() wrapper function.

mag_thresh()
- color tranformation to grayscale.
- square the sobelx/sobely values and take the square root.

dir_thresh()
- color transformation to grayscale.
- use Numpy arctan2 function.

hls_binary()
- color transformation to HLS.
- look at S and L values

The output of these functions is a binary image (Figure 3).

Figure 3: Binary image

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

Specification

In pipeline() (notebook cell 20) after creating a binary image and then masking for a trapezoid where the lane lines are likely to be (function region_of_interest() in notebook cell 4), a perspective transformation was performed using Udacity 'knowledge' answer values for destination (in function pTransform() in notebook cell 7).

Source:
[(img_size[0] / 2) - 55, img_size[1] / 2 + 100],
[((img_size[0] / 6) - 10), img_size[1]],
[(img_size[0] * 5/6) + 10, img_size[1]],
[(img_size[0] / 2 + 55), img_size[1] / 2 + 100]

Destination:
[1000,0], # upper right
[1000,719], # lower right
[280,719], # lower left
[280,0] # upper left

OpenCV cv2.warpPerspective() was used for the perspective transformation in notebook cell 7.

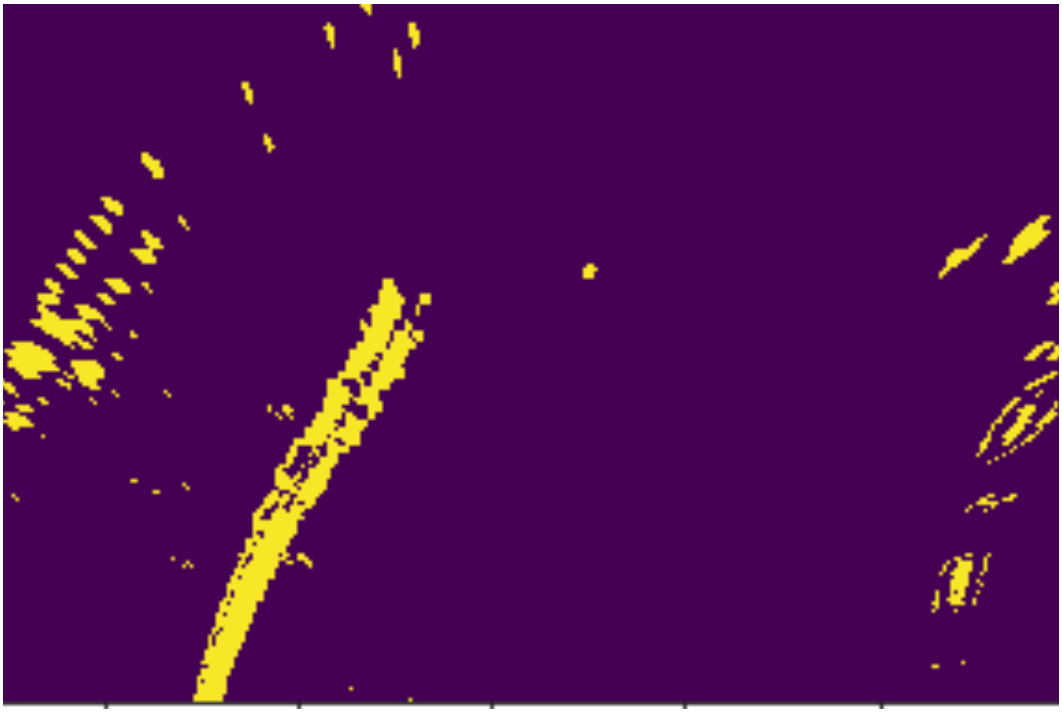

Figure 4 : Perspective Transform

Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial.

Specification

To determine the location of lane lines, the binary image previously created was passed to function find_lane_pixels(). Here the image was searched on the left half and the right half using a histogram on the vertical axis for each side. The highest value of each histogram was used to determine the location of the lane line. Once 50 pixels were counted, the next window was checked. These calculations take place in notebook cell 8. An example of the process is shown in Figure 5.

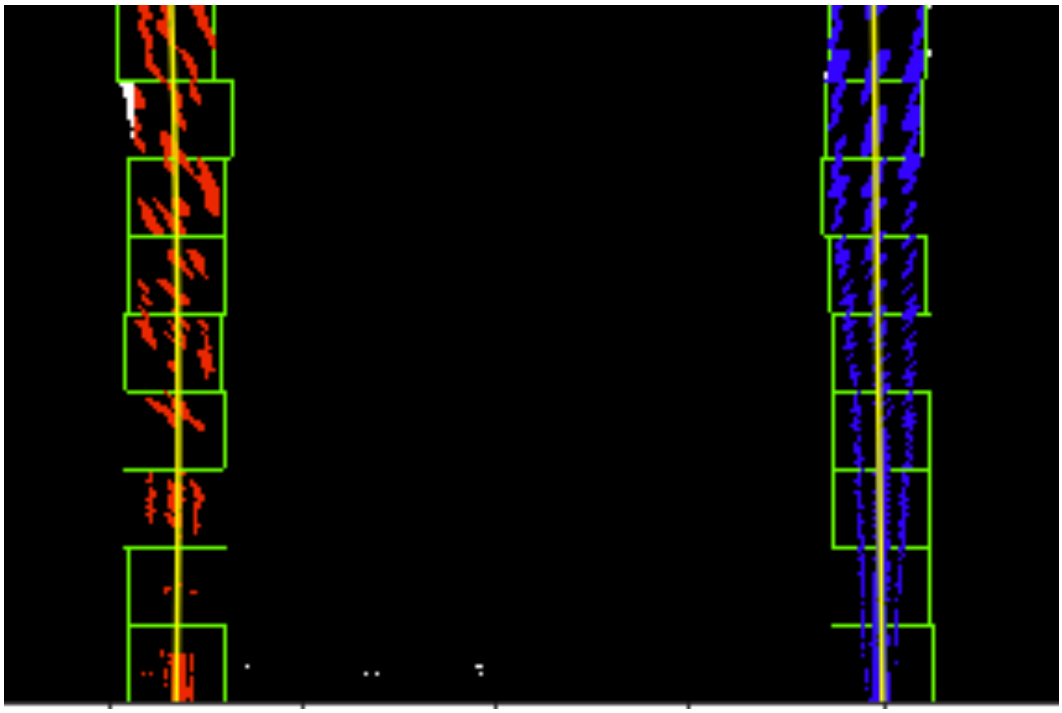

Figure 5: Detected lane lines

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

Specifications

The pipeline() function calls curve_and_position() with arguments that include the polynomial coefficients for the lane lines in meter values and the x values for the detected left and right lanes. This is in notebook cell 14. curve_and_position() then calls measure_curvature_real() to determine the left lane and right lane curve radius (based on meters, not pixels) at the bottom of the image (location closest to the car).

Provide an example image of your result plotted back down on the road such that the lane area is identified clearly.

Specifications

The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This demonstrates that the lane boundaries were correctly identified. An example image with lanes, curvature and position from center is included in Figure 6.

Pipeline (video)

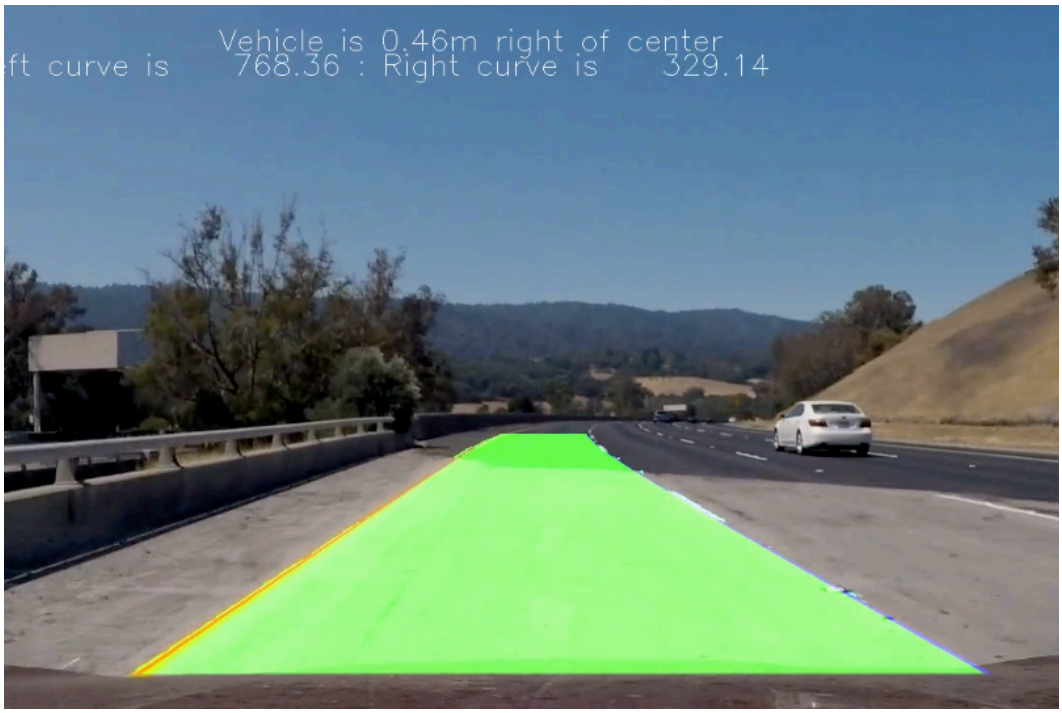

Figure 6: Detected lane lines, curve radius and vehicle center

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!).

Specifications
The image processing pipeline that was established to find the lane lines in images successfully processes the video. The output is a new video where the lanes are identified in every frame and outputs are generated regarding the radius of curvature and does not fail when shadows or pavement color changes are present.

https://www.dropbox.com/s/2nvobnfxne8xwtq/project_results.mp4?dl=0

Discussion

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?

Specifications
Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline and what hypothetical cases would cause their pipeline to fail.

Problems / Issues:

Technical issues around Jupyter setup initially. Had “%matplotlib qt” in a cell instead of “%matplotlib inline”. Turns out this was causing a kernel exception when using cv2.imshow. So had real concerns over platform stability until figured this one out.

Ran into problems with smoothing and np.delete(array, location). This does not delete the entry from the array (it’s not inline type functionality). Once this was recognized, moved to .pop(location) and smoothing was back on track.

Finding a good combination of values to have a really solid binary image was difficult. I was looking for perfect and ended up realizing it needs to be good, not perfect or I’ll never get past that step.

Where the pipeline will likely fail:

If traveling on a curving mountain road, may have difficulty if the lanes move off camera to the left or right before they reach the top of the screen.

Under snow, heavy rain, blowing dust conditions, I would expect the computer vision algorithm to have difficulty detecting lanes.

What would make it more robust?

Better sanity checks:
- Have shell code in there but the if statement is set to ‘True’ for now (notebook cell 11).
- Did not check the distance between the fit polynomials.
- Did not compare the values for the left and right polynomials.

More intelligent averaging where the most recent detections carried more weight than older values. Currently an equal weighting of the last 24 values is used (1 second of video at 24 Frames Per Second).

Do not search blindly for the lane at the start of each frame. Could take advantage of the information gained in the previous frame.