UDACITY

PROJECT

Advanced Lane Finding

A part of the Self Driving Car Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

Congratulations. This is one of the best submissions I have seen. Very thorough investigation along with great presentations for different stages of the pipeline. A really excellent report.

Good luck on the next project, and keep doing an awesome job.!

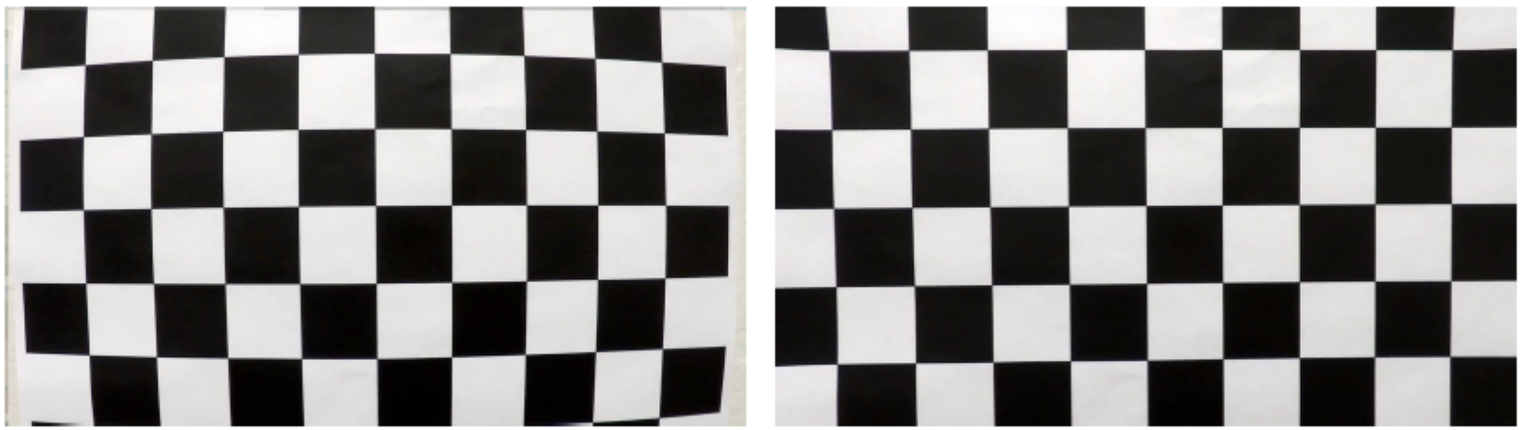
Writeup / README

	The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.
	Wow!

Camera Calibration

OpenCV functions or other methods were used to calculate the correct camera matrix and distortion coefficients using the calibration chessboard images provided in the repository (note these are 9x6 chessboard images, unlike the 8x6 images used in the lesson). The distortion matrix should be used to un-distort one of the calibration images provided as a demonstration that the calibration is correct. Example of undistorted calibration image is Included in the writeup (or saved to a folder).

Spot on.



Distorted calibration image (left) and corrected image (right).

Pipeline (test images)

Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.

Nicely done.

Distorted test image (left) and corrected image (right).

A method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.

Well done. Yellow color thresholds were applied to the image RGB channels for left lane line. White color thresholds were applied to the image RGB channels for right lane line. Comments indicate sorbel transform use coded up but being saved for more challenging videos.

By using color thresholding you are able to make the lane detection more robust by relying less on gradients for good results!

An idea for more robust extraction is to try color thresholding in the RGB, HSV and HSL channels for your yellows and whites! Other ideas are just thresholding in the R and V channels.

Here is some sample code to play around with, it should help with more tricky areas!

```
HSV = cv2.cvtColor(your_image, cv2.COLOR_RGB2HSV)

# For yellow
yellow = cv2.inRange(HSV, (20, 100, 100), (50, 255, 255))
```

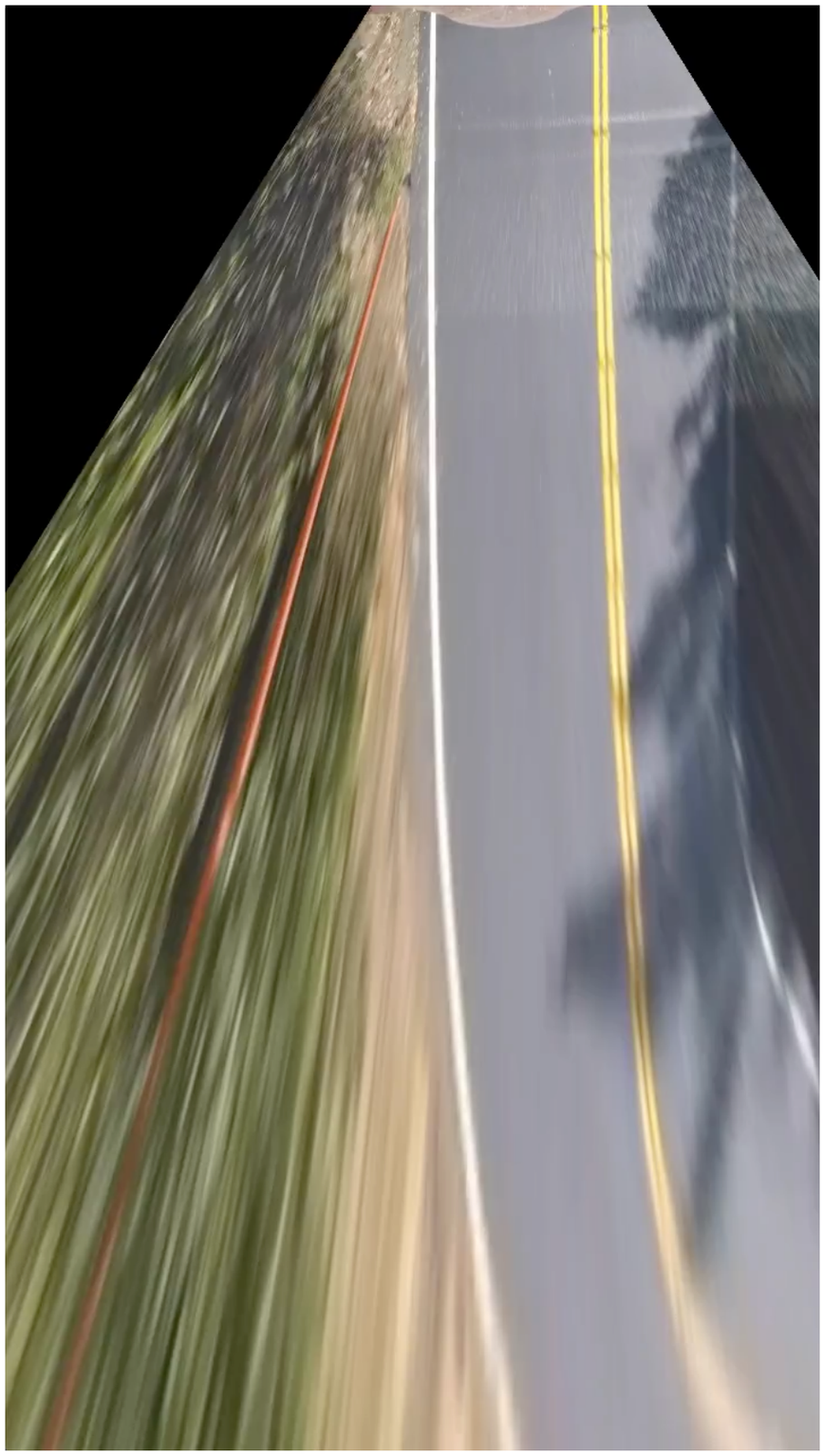
```
# For white
sensitivity_1 = 68
white = cv2.inRange(HSV, (0,0,255-sensitivity_1), (255,20,255))

sensitivity_2 = 60
HSL = cv2.cvtColor(your_image, cv2.COLOR_RGB2HLS)
white_2 = cv2.inRange(HSL, (0,255-sensitivity_2,0), (255,255,sensitivity_2))
white_3 = cv2.inRange(your_image, (200,200,200), (255,255,255))

bit_layer = your_bit_layer | yellow | white | white_2 | white_3
```

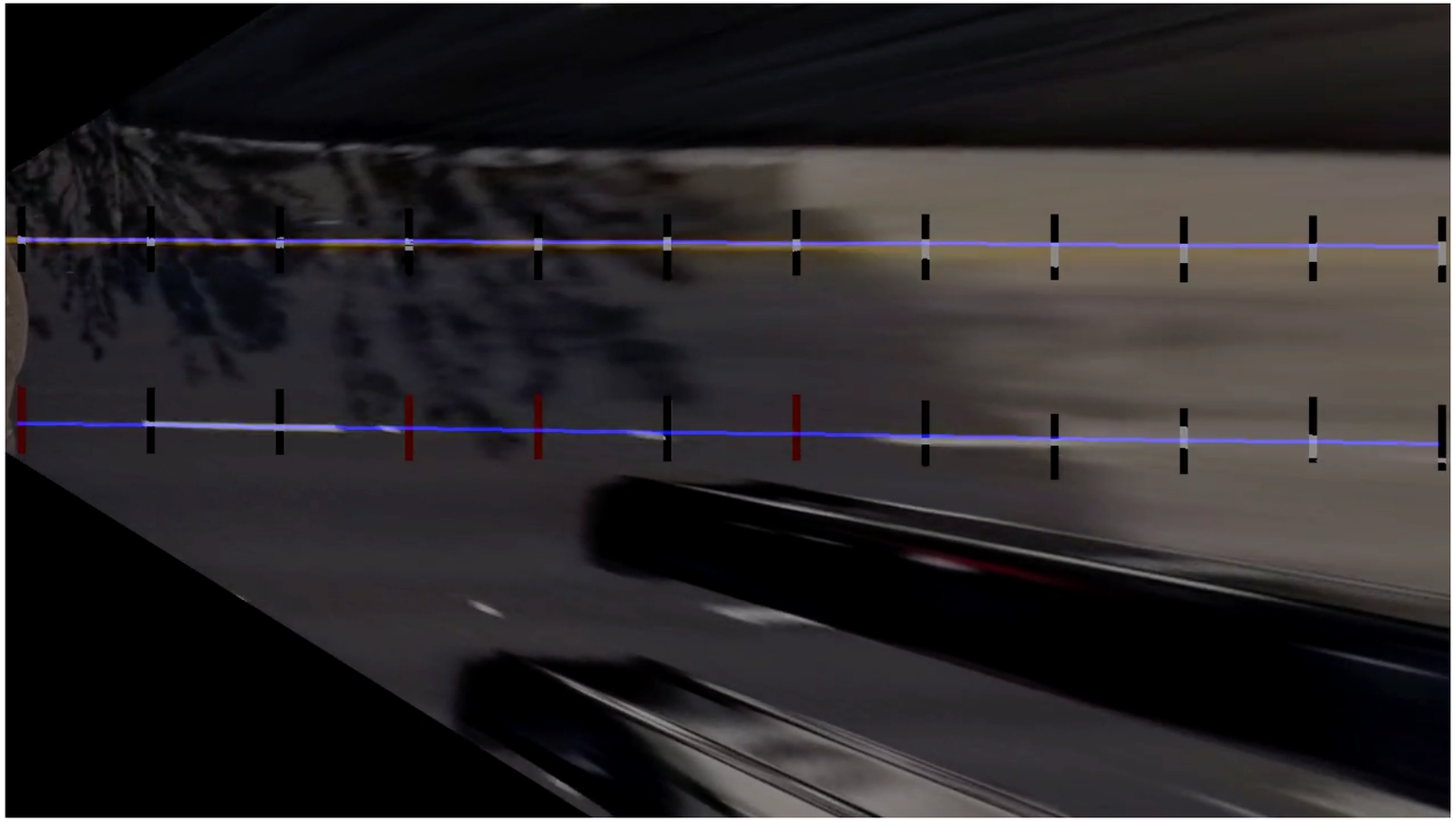
OpenCV function or other method has been used to correctly rectify each image to a "birds-eye view". Transformed images should be included in the writeup (or saved to a folder) and submitted with the project.

Truly remarkable work.!



Methods have been used to identify lane line pixels in the rectified binary image. The left and right line have been identified and fit with a curved functional form (e.g., spine or polynomial). Example images with line pixels identified and a fit overplotted should be included in the writeup (or saved to a folder) and submitted with the project.

Nice job implementing the histogram/sliding window search to identify lane pixels.



Here the idea is to take the measurements of where the lane lines are and estimate how much the road is curving and where the vehicle is located with respect to the center of the lane. The radius of curvature may be given in meters assuming the curve of the road follows a circle. For the position of the vehicle, you may assume the camera is mounted at the center of the car and the deviation of the midpoint of the lane from the center of the image is the offset you're looking for. As with the polynomial fitting, convert from pixels to meters.

Nice job calculating the radius of curvature and vehicle position and printing those values on the images. The calculated values in the video frames look very reasonable.

The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and

position from center should be included in the writeup (or saved to a folder) and submitted with the project.

The warping back results look terrific.!

Pipeline (video)

The image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and

not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.

Nitpick: There are a few instances in the video where the detections of one or both lanes is significantly off from the true position of the lane lines. The issues seem to be that the pipeline is somewhat slow to react, getting distracted by shadows and/or color changes on the road.

Suggestions for improvement:

- I am wondering if the sanity checks you have implemented are too aggressive because there are segments of the video where the polynomials appear to be frozen and do not update for several frames. I think this is a result of the sanity

checks deciding to use the last computation instead of the current frame. When the roads are curving, or if the car adjusts its position in the lane, and the polynomial from a previous frame is used, then the lines will naturally be off.
- Continue to experiment with different color spaces and thresholds to find a combination which is able to accurately detect the lane pixels without getting distracted by shadows and color/brightness changes. I recommend exploring the LUV and Lab color spaces as many students have had success with these.

This effect was most pronounced in the early part of the challenge video output. However in the basic video it is so minor that it is barely noticeable.

Discussion

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.

An excellent reflection on the pipeline, it's strengths, weaknesses and areas with potential for improvements.

DOWNLOAD PROJECT

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH