Vinh Nghiem

Advanced Lane Finding Project

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 can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

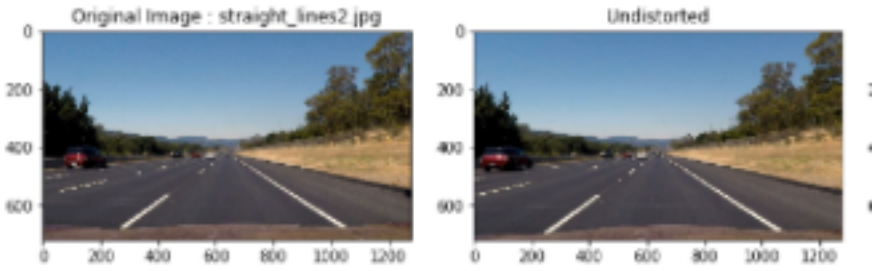
Camera Calibration

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

I used the cv2.calibrateCamera() openCV method, which maps inputs of image points and object points. Image points are 2-D corners of supplied checkerboard images found in the supplied camera_cal folder. The code iterates through all 20 calibration images in that folder and maps inside corners of each checkboard to evenly-spaced object points in 3-D space. The method to calibrate the camera returns the camera matrix and distortion coefficients. The code for this is in the calibrate_camera(…) method in lines 11-40 of i.ipynb.

Pipeline (single images)

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



I applied distortion correction, as shown in the example image above, by calling the cv2.undistort(…) method on the original color image (in RGB format), along with the camera matrix and distortion coefficients returned from the cv2.calibrateCamera(…) method.

The code for distortion corrections is in line 42-43 in P4.ipynb.

2. 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.

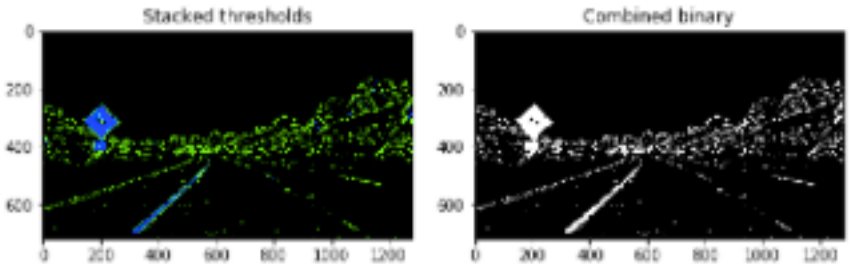
I used a combination of a Sobel operator in the x direction and a color transform on the saturation channel.

To use the Sobel operator, a color RGB image is converted to grayscale, then the cv2.Sobel() operator is applied to the grayscale image, followed by taking the absolute value of each Sobel operator value (to capture magnitude only and not direction).

For the color transform, an HLS transform is applied on the color RGB image. An image is then taken only from the S (saturation) channel.

A binary image is then formed from whether each pixel meets respective threshholds for both of the operations above.

An example of a combined thresholded binary image, where the Sobel operator is in green and the thresholded S-channel is in blue, along with an image with both combined transformations in white, is shown below:



The code for the threshholded binary result is in lines : 45-80 of P4.ipynb.

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

I performed the perspective transform by using the openCV method warpPerspective(…) , which requires mapping between source points from the unwarped image and destination points from the resulting warped image. I hardcoded the source and destination points in the following manner:

```
src = np.float32([[568,468],[717,468],[1077,717],[217, 717]])
offset = 300
img_size = (undistorted.shape[1], undistorted.shape[0])
dest = np.float32([[offset, 0], [img_size[0]-offset, 0],
                                    [img_size[0]-offset, img_size[1]],
                                    [offset, img_size[1]]])
```

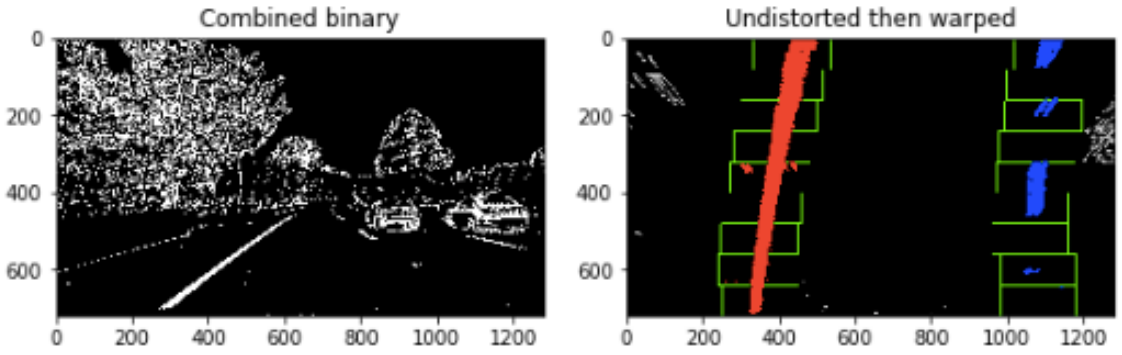
This resulted in the following source and destination points:

The code to perform the perspective transform is in line 237 of P4.ipynb.

Source	Destination
568, 468	300, 0
717, 468	980, 0
1077, 717	980, 720

Source	Destination
217, 717	300, 720

An example of an unwarped image compared to a warped image is shown :



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

The location for each of the left and right lane lines is approximated first by constructing a histogram of image intensity values along each x-column. Starting from the bottom of the image, windows with fixed y-coordinates are bound around areas representing the highest likelihood of lane detection. All pixels within those boxes are marked to fit with a second-degree polynomial (parabola), using the openCV polyfit(…) method.

The code to identify lane pixels and fit their positions with a polynomial is in method mark_lanes(binary_warped) in lines: 82-163 of P4.ipynb

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.

The radius of curvature is calculated using the radius of curvature equation involving the parabola coefficients fit through the lane points, given in the lesson. I did this in method get_rad_curve(leftx,rightx, ploty, img_size) in lines: 273-288 in P4.ipynb.

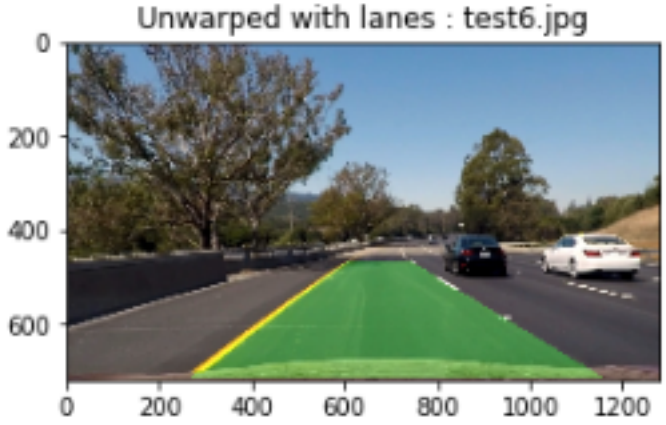
The radius of curvature is presented here in meters, so all pixel distances are converted to distances in meters, when substituting into the radius of curvature equation. The relationship between pixel distance and meters is calibrated using the reference that lane lines are separated by 3.7 m , while dashed lines have a standard length of 10ft (3.048 meters). Lane width and dashed line length were then measured in pixels on the unwarped image of the lane, in order to determine the number of horizontal pixels in one meter, and the number of vertical pixels in one meter, respectively.

The position of the vehicle with respect to center is calculated in the unwarped binary image with gradient and color thresholds after fitting the second degree polynomial through the points constituting the lane lines. The position of the vehicle with respect to center is determined by finding the distance between the midpoint of the fitted line points at the bottom-most part of the image, and the center of the image itself.

The part of the code calculating radius of curvature and position of the vehicle with respect to center is function get_rad_curve(…) in lines 273-288.

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

I implemented this step in lines 165-188 P4.ipynb in the function `unwarp_with_lanes(…)`. 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!).

My video file, project_video.mp4, is included.

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?

My pipeline used was : calibrate camera, undistort image, unwarp image, convert to binary image based on gradient and color thresholds, fit second-degree polynomial through lane points, warp image back with lane detected.

Specifically, I used openCV methods to calibrate the camera, apply the Sobel gradient operator, undistort the image, and warp and unwarp the image. The second-degree polynomial was fit using the numpy polyfit(…) method.

The points to fit the polynomial were selected by histogram generation of the pixel intensity across all x values of the image and stacked windows up the image were used to pinpoint the range of pixels that could constitute the lane boundaries.

After the polynomial was fitted for each frame, a sanity test was performed to gauge whether the lines were parallel (if the top distance between lines and bottom distance between lines differed by less than a tolerance) and whether the car moved too much from the last frame (if the distance from center in this frame was less than the last acceptable distance from center by a certain tolerance). If the sanity test passed, the fitted lines for the lane is used; otherwise, the last accepted fitted lines are used. Though the standard video had heavy shadows from the trees, which lit up as possible lane points, the sanity test technique was successful, though it took some trial and error tweak the tolerance values.

The current pipeline does not work on the challenge video. The overly twisting road caused the sanity test to fail continually, since the car's position from center

would change by large values from one frame to the next. Because the sanity test failed often, the same old marked lanes were used for successive frames. Also, the objects within the same lane as the car, such as the motorcyclist, tricked the algorithm into detecting these objects within the lanes as the lane boundaries themselves.

The challenge video is an interesting challenge indeed but I had to postpone working on it in the interest of time. Being able to recognize objects in the lane as the appropriate class of objects would be helpful in detecting the boundaries of the actual lanes.