Johan Fanas Rojas – Advanced Lane Lines

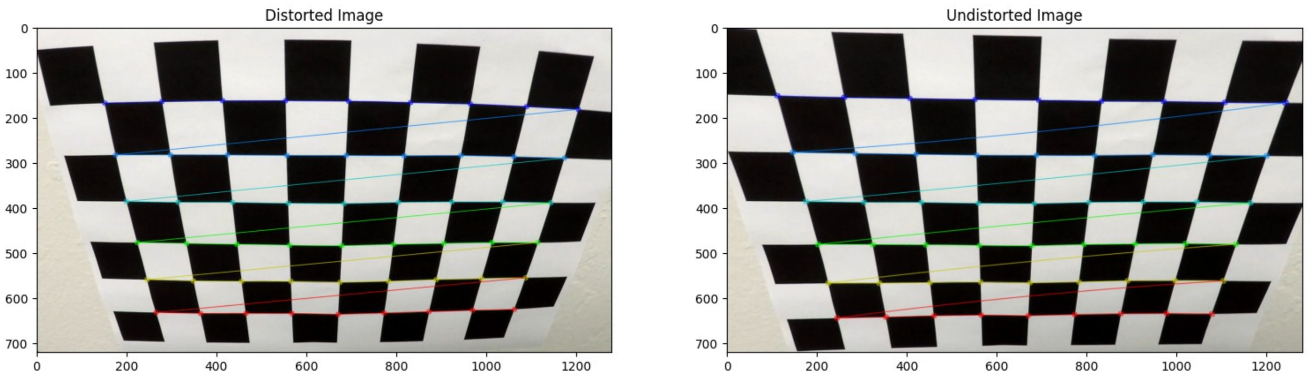
Advanced Lane Lines

Camera Calibration

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

The camera calibration segment can be found between line 60-79 of the pipeline. The code is called Advanced_Lane_Detection.ipynb.

First of all, I created a class called AdvancedLaneLines with all the necessary functions to successfully detect lane lines using the methods learned in the course. I created a function called cameraCalibration that I initialize in order to calibrate the camera before doing any further calculations. I start by inputting all the calibration images and storing them in a variable called images. Then, I declare and initialize two empty arrays called objpoints and imgpoints. These arrays are going to store x,y and z coordinates of the chessboard (which z = 0 in our case). The array imgpoints is going to contain the detected corners of the chessboard in our calibration images and objpoints is going to contain these coordinates of the corners of the undistorted image. Using the commands cv2.findChessboardCorners and cv2.drawChessboardCorners, we detect and draw the corners of the chessboard given the size of the board (in this case 9x6). Then I use cv2.calibrateCamera to obtain the mtx and dist coefficients and a store them by making them an attribute of the class so I can access them later. Since I have to do the same with all calibration images, I iterate through all of them and apply the same steps but I only want the function to retrieve the mtx and dist coefficients if ret is equals True because some images are taken in an angle that not all the corners can be seen. The output of one of the calibration images is shown below.



Johan Fanas Rojas – Advanced Lane Lines

Pipeline (single images)

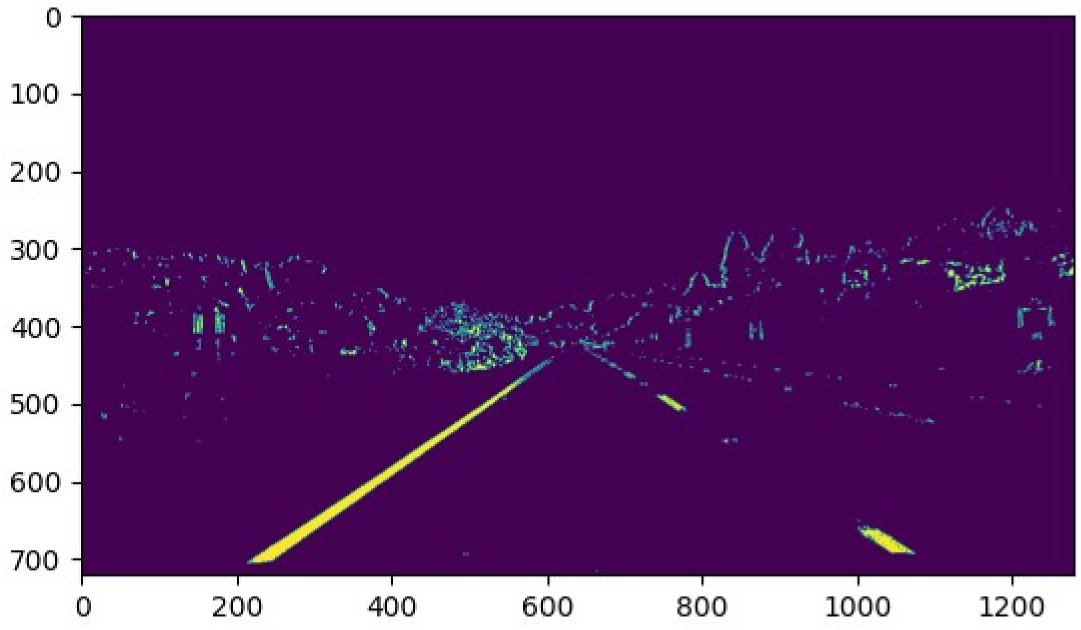
1. Provide an example of a distortion-corrected image.

After the mtx and dist coefficient were stored in variables, they were used to undistort the test images using cv2.undistort. An undistorted test image is shown below.

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.

The color and gradient transforms are both done in a function called Gradient. This function can be found between lines 81-120.

This function takes as an input the read image (or frame), the mtx and dist coefficients. In this function the image gets undistorted, gets converted into gray scale and we apply a gradient in the X direction. We then get the absolute value of the binary image and apply the thresholds. Also, we separate the S channel from the inputted image in order to apply a threshold to better capture the lane lines. Below is an image of the output of this function.

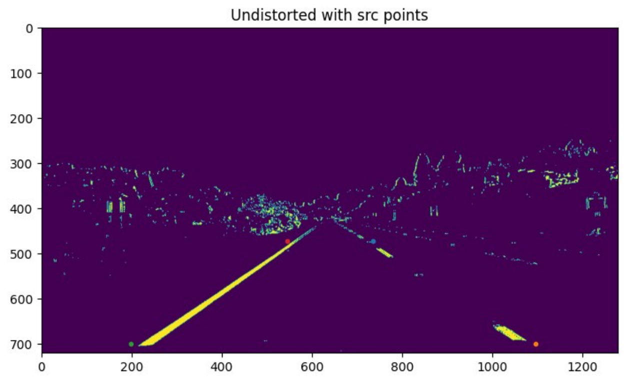


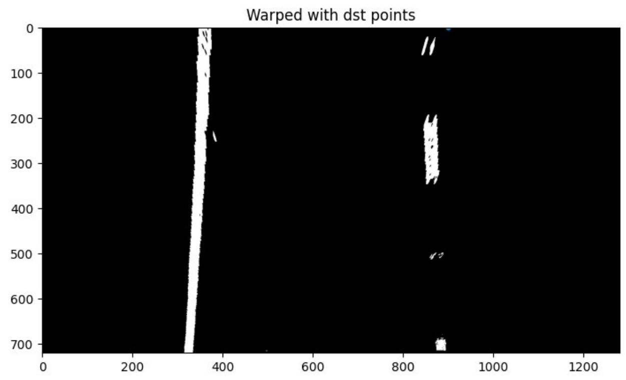
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The perspective transform of the image was performed in a function called Perspective (can be found between lines 126-143). This function takes the output image from the Gradient function as an input. The src points of the region to be warped were selected by manually. The dst points are the location of the corners where we want our region to be warped to. The src and dst points are shown in the following table:

Source	Destination
735, 472	900, 0
1096, 700	900, img.shape[0]
198, 700	300, img.shape[0]
545, 472	300, 0

I consider my warped image works very good and it can be seen in the image below. I plotted points on both images to indicate where the src and dst points are but you can’t see the points in the warped image very well because I chose them near the axis limits.

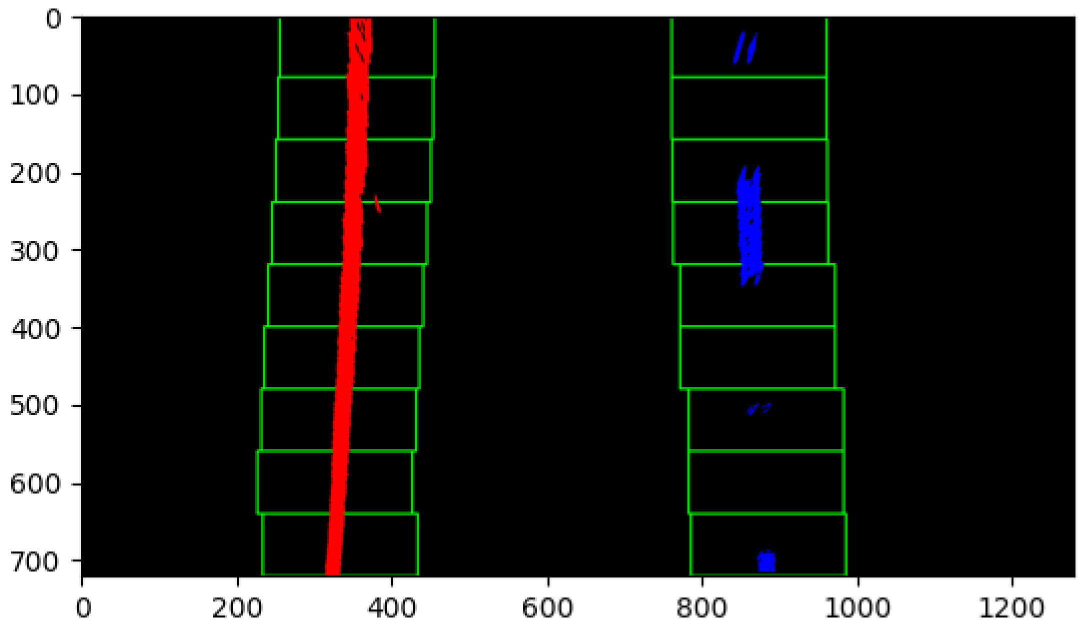




4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The segment for this code can be viewed between lines 149-263.

In our LanePixels function, we get the histogram of the lower part of the warped image to determine where the lane lines are. We use this position to enclose that area with a rectangle (sliding window) and identify non zero pixels to store them in two variables right_lane and left_lane. These points are used used in our FitPolynomial function to fit them into a second order polynomial using cv2.polyfit. The output should be something similar to the image below. Here we can see the displayed sliding window and the detected lane lines within the enclosed area.



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 by using the following equation:

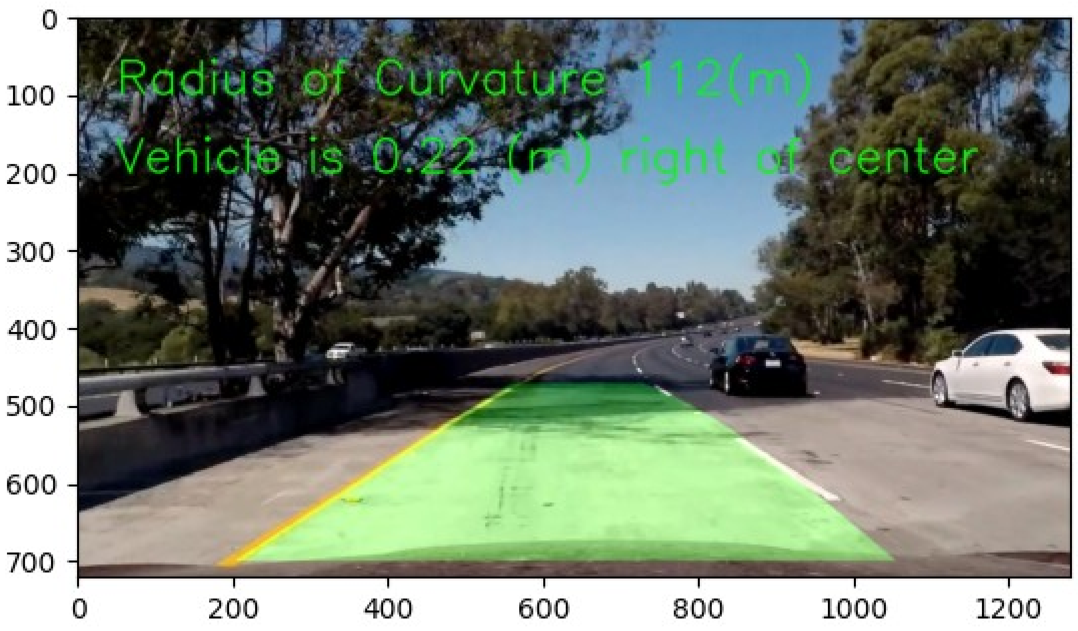
$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

Where A and B are coefficients from our second order polynomial of the left and right lane. Y is the max value corresponding to the bottom of the image. This has to be translated into real world dimensions; therefore, for the pixel to meters conversion we used 5.5 meters per img.shape[0] in the Y direction and 3.7 meters per 600 (since our warped image has a length of 600 in the X direction) in the X direction. For the Y direction we used 5.5 metes because each dashed line is approximately 2 feet and the space between them is 10 feet; therefore, we estimated our region should have 18 feet.

The segment for this code can be found between lines 323-338.

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 345 through 384 in the function Pixel2Real. Here is an example of my result on a test image:


Radius of Curvature 112(m)
Vehicle is 0.22 (m) right of center
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Johan Fanas Rojas – Advanced Lane Lines

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!).

I uploaded the outputted video of my code in the output images folder. The video is called Output Video. I you want to see the code working just open the Advanced_Lane_Detection.ipynb file and run it. I left it set up for the video to run. I should work on your computer for the video and images because I tried to make it generic.

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?

I consider my code works very good but it has some issues when it goes through shadows. My code will definitely fail with a video similar to the harder_challenge.mp4 because of the tight turns and you can barely see the right lane. To solve this issue I would try to make two separate pipelines and let the code choose which one to choose depending of the situation or if one becomes unstable to use the other one and keep switching back and forth.