# Advanced Lane finding

The goals / steps of this project are the following:

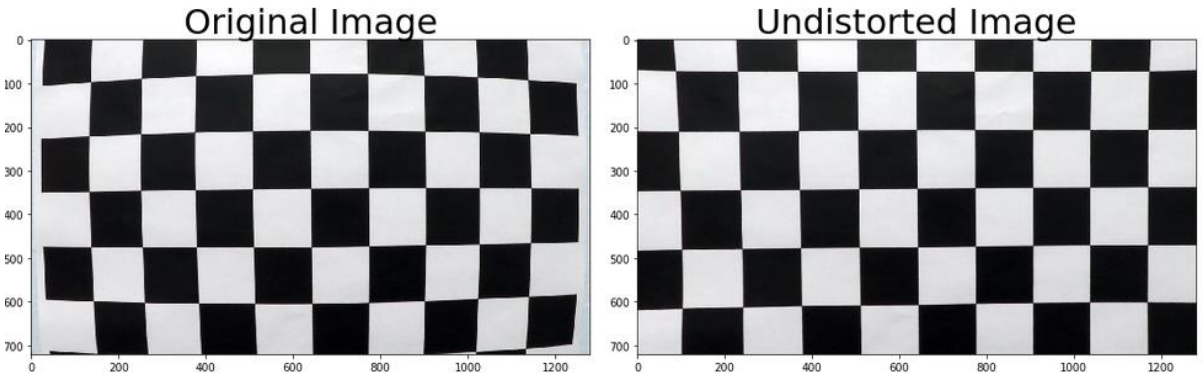
1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
2. Apply a distortion correction to raw images.
3. Use color transforms, gradients, etc., to create a thresholded binary image.
4. Apply a perspective transform to rectify binary image ("birds-eye view").
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

## 1. Camera Calibration

Calibration is necessary for determining the intrinsic and extrinsic parameters of a camera. The intrinsic camera coefficients will help us eliminate the radial and tangential distortions caused by the lens. For performing the calibration for this project I have used the Opencv calibrateCamera function. In the calibration section, the image coordinates of each chessboard image are computed using the cv2.findChessboardCorners function. We create two lists one with image points and one with the corresponding object points. The two lists are passed to the cv2.calibrateCamera function to obtain the camera calibration and distortion coefficients. The images are then undistorted using the cv2.undistort function.

**1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?**

In the image bellow we show the distorted and undistorted images of a calibration pattern.

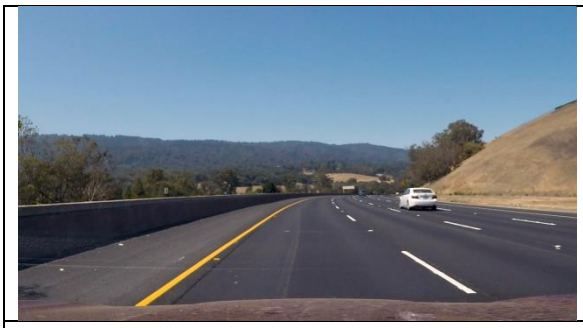
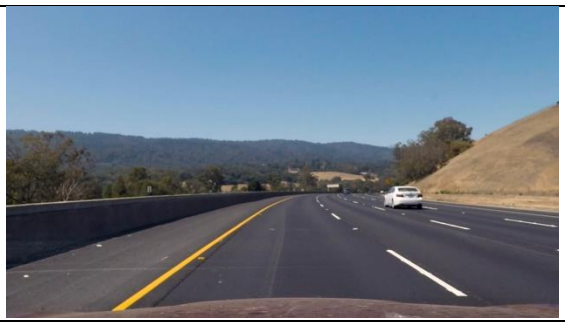



This calibration procedure appears in the jupyter notebook in section entitled 1.Calibration.

## 2. Pipeline (single images)

### 1. Has the distortion correction been correctly applied to each image?

In the image below we have an example of how the distortion correction is applied on one of the test images.

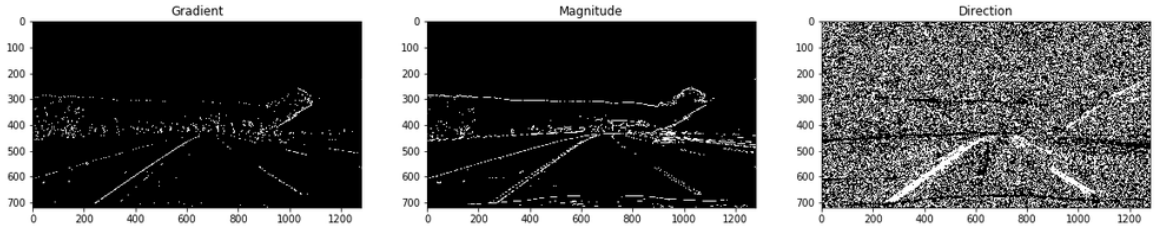
|  |  |
|---|---|
| Original Image | Undistorted Image |

### 2. Has a binary image been created using color transforms, gradients or other methods?

While testing several options for segmentation, I have seen that the detection works the best when combining the S and L channels from the HLS color space. The combination has been achieved using logic or between the two images which correspond to each channel. To remove false positives I have applied a threshold on S and L. (To get the edges I applied sobel on x direction on each image corresponding to the L and S channels before combining them).

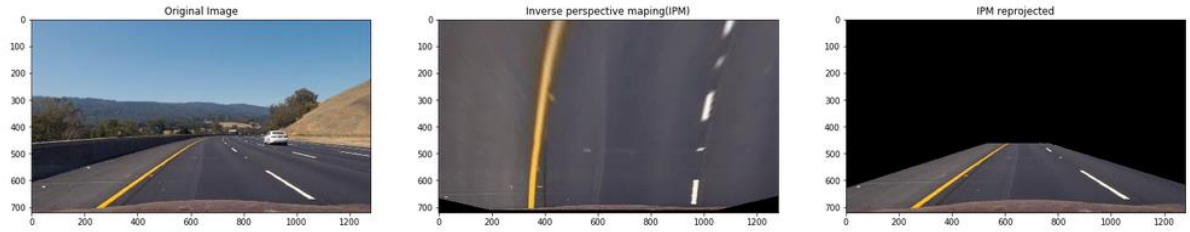
The code for this part appears in section 5 Convert to HLS and combine with Sobel on Grayscale in the function get_binary_image(img).

Other binarisation techniques have been presented in section 4.Image segmentation Functions.




### 3. Has a perspective transform been applied to rectify the image?

Perspective transform has been applied to the image. An example can be seen in section 2 called IPM. An example of the birds eye view on the test image is provided below.




src = np.float32(

[[(img_size[0] / 2) - 55, img_size[1] / 2 + 100],

[((img_size[0] / 6) - 10), img_size[1]],

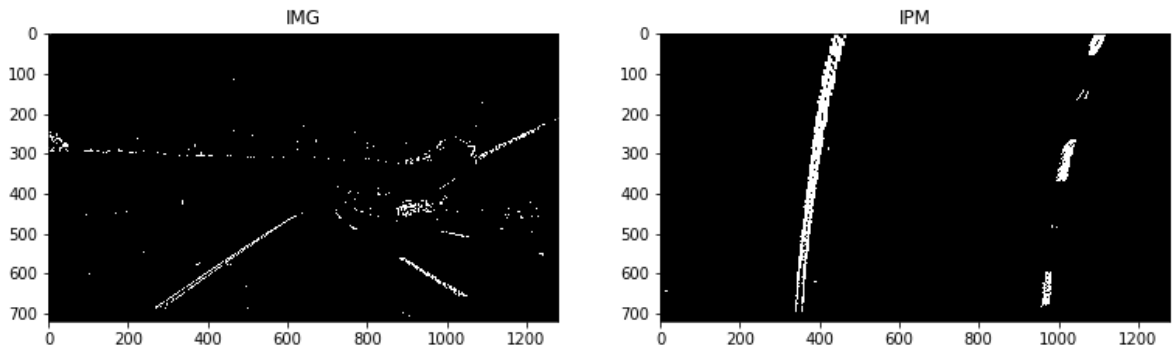
[(img_size[0] * 5 / 6) + 75, img_size[1]],

[(img_size[0] / 2 + 65), img_size[1] / 2 + 100]])

dst = np.float32([[300, 0], [330, 710], [960,710], [960, 0]])

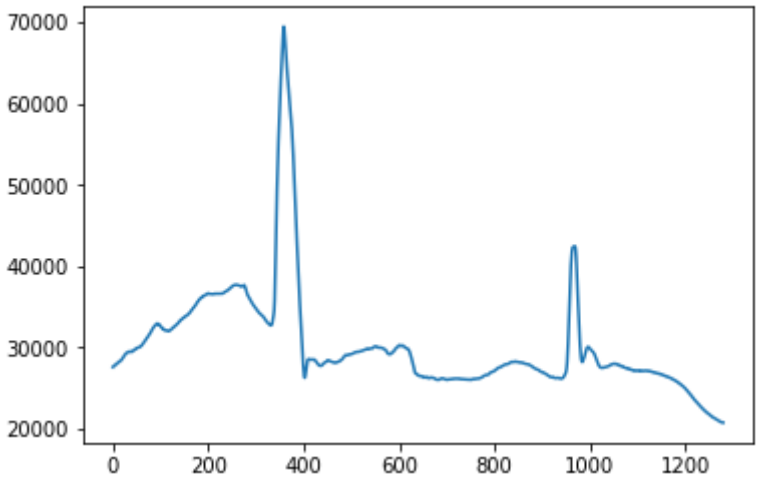
| Source | Destination |
|---|---|
| **585, 460** | 300, 0 |
| **203, 720** | 330, 710 |
| **1141, 720** | 960, 710 |
| **705, 460** | 960, 0 |

In short: I am first computing 2 matrices to project from perspective to birds eye view and from IPM back to perspective. This is done using cv2.getPerspectiveTransform function. I am warping the images back and forth using the cv2.warpPerspective.
Here is an example of the binary image projected to perspective.




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

To capture the lane curvature we are trying to fit a second order polynomial ($ax^2 + bx + c = y$) on the lane. We are using a histogram on the ipm image.

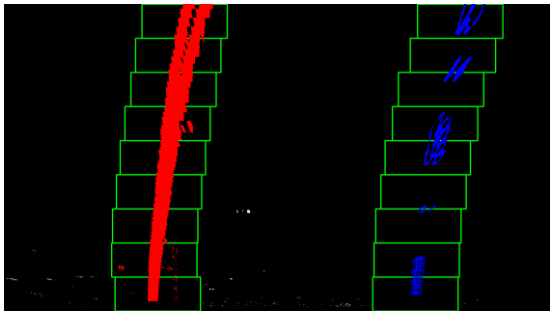



The above graphic can be found in section 3. Find lane points.

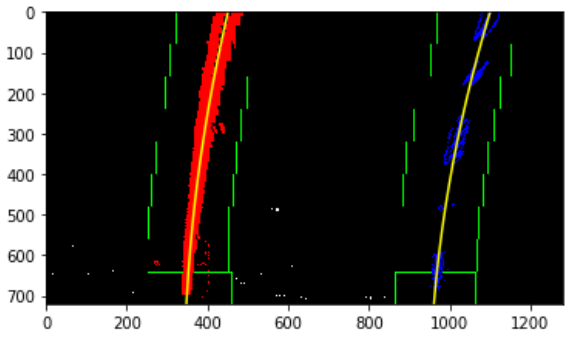
The places where the lanes are in the image will have the biggest spikes in the histogram. We perform this step just once.

Having the starting points we divide the image in 2 parts for the left and right lanes. For each left and right window we find the mean of it, recentering the window. The points from the windows are stored and we try to find the best second order polynomial by feeding the numpy polyfit function.

An example of image with the windows following the lanes can be seen below.



The fitted lanes identified can be seen below.




The code for this part can be seen in the 6. Fit First Time Lines and 7. Fit Lines Continuous after first frame section.

Please note that the windows are not seen very good(they are a bit fragmented) in the second image due to the resolution of the image.

**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 in some point is the radius of the circle that osculates it. The **radius of curvature** of the curve at a particular point is defined as the radius of the approximating circle. This radius changes as we move along the curve.(A reference has been given in the course regarding the radius of curvature http://www.intmath.com/applications-differentiation/8-radius-curvature.php)

The radius of curvature can be an indicative for the steering angle, and its formula is given bellow:

$$Rc = (\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}})/(\,|(d^2y)/(dx^2)|\,)$$

We will compute the radius of curvature for both lines at the base of the vehicle, which will be located at the bottom of the image. (To quote from the course notes : "The y values of your image increase from top to bottom, so if, for example, you wanted to measure the radius of curvature closest to your vehicle, you could evaluate the formula above at the y value corresponding to the bottom of your image, or in Python, at `yvalue = image.shape[0]`.")

Since the image is in pixels we have to estimate the real world dimensions from the photo. As described in the class instructions we can use:

ym_per_pix = 30/720 # meters per pixel in y dimension
xm_per_pix = 3.7/700 # meters per pixel in x dimension

To view the offset from the center of the lane, we assume the camera is mounted in the center of the car. To estimate the offset we consider the difference between the center of the image and the middle point of the begging of the lines( in pixels). To obtain the offset we just have to multiply the obtained value with the conversion factor.

The code for this part appears in section 8. Draw Line and Radius of curvature. To be more specific in the function:

get_curvature_and_distance_from_center(left_lane_coordinates, right_lane_coordinates, image_width, image_height).

The method uses the code presented in part 35 from the advanced line track:

```
def get_curvature(line_fit, y_eval):
    return ((1 + (2 * line_fit[0] * y_eval + line_fit[1]) ** 2) ** 1.5) \
                         / np.absolute(2 * line_fit[0])

def fit_line(line_x, line_y, order_polynomial = 2):
    fit, residual, _, _, _ = np.polyfit(line_y, line_x, order_polynomial, full = True)
    return fit, residual

ym_per_pix = 30 / image_height # meters per pixel in y dimension
xm_per_pix = 3.7 / lane_width_pix # meteres per pixel in x dimension

    # Find lanes fit in in meters
 left_fit_cr, _ = fit_line(left_lane_coordinates[1] * xm_per_pix, left_lane_coordinates[0] * ym_per_pix)
 right_fit_cr, _ = fit_line(right_lane_coordinates[1] * xm_per_pix, right_lane_coordinates[0] * ym_per_pix)

   # Calculate curvature of the lane at the bottom of the image, in meters
 left_curverad = get_curvature(left_fit_cr, image_height * ym_per_pix)
 right_curverad = get_curvature(right_fit_cr, image_height * ym_per_pix)# Now our radius of curvature is in
meters
```

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


The code that contains the overall pipeline of the process can be found in section 9.Process Lane. The image contains the lane drawn with green, the information regarding the curvature and offset for the lane are provided in the same image with white, I have also included a top view of the lanes(in the top right) and the original image (in the bottom right).

## Discussion

The pipeline for solving this project has been explained in the course notes. The project contains the following steps:

1. Calibrate (Just once to find the camera intrinsics)
2. Undistort input
3. Binarize
4. Generate binarized IPM
5. Find and fit lines on IPM
6. Display results (including center of curvature and offset)

Some limitations of the approach include the presence of shadows in the scene or, as I identified from one of the challenge videos, the presence of thickened tar on the road surface which may cause false lanes.

An improvement of the pipeline could consist in the tracking of the lanes using Kalman. Another possible improvement would be the fusion of the color information with some cheap 3D sensor for example a sparse LIDAR. We could get better information about the lane markings by fusing the color information with the reflectivity information from the LIDAR. Another Idea could be to take advantage of the fact that the lines usually consist of dark white dark transitions. We could therefore implement a shadow removal approach similar to the one presented in the paper bellow: http://ieeexplore.ieee.org/document/477027/