Advanced Lane Finding Project

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. Display lane boundaries and numerical estimation of lane curvature and vehicle position.
9. Run pipeline in a video.

Camera Calibration

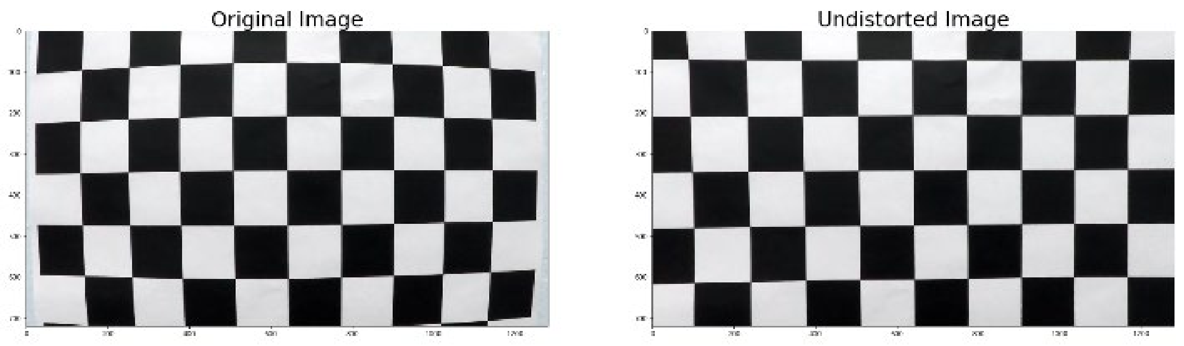
I started with the camera calibration. I have used some images of chessboard taken with a same camera and from different angles.

I called for each image the functions *cv2.findChessboardCorners()* in order to find corners in chessboard image. I used these corners with the imagepoints to calibrate the camera using *cv2.calibrateCamera()* with this function I received also the distortion coefficients.

Distortion Correction

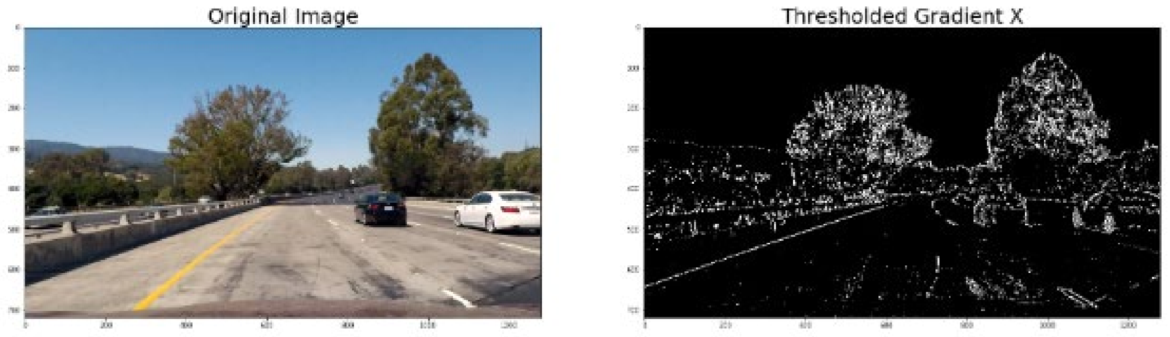
I called *cv2.undistort* , used this method and these distortion coefficients to undistort the images.

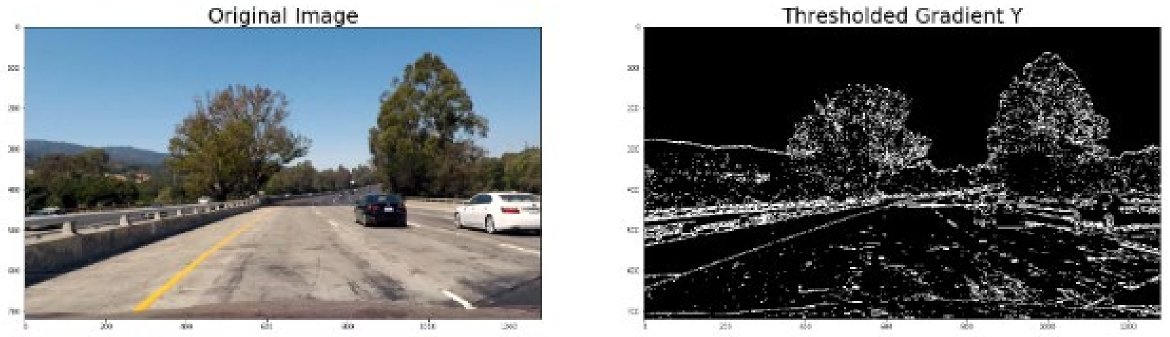
- Below is the examples of the images before and after distortion correction.

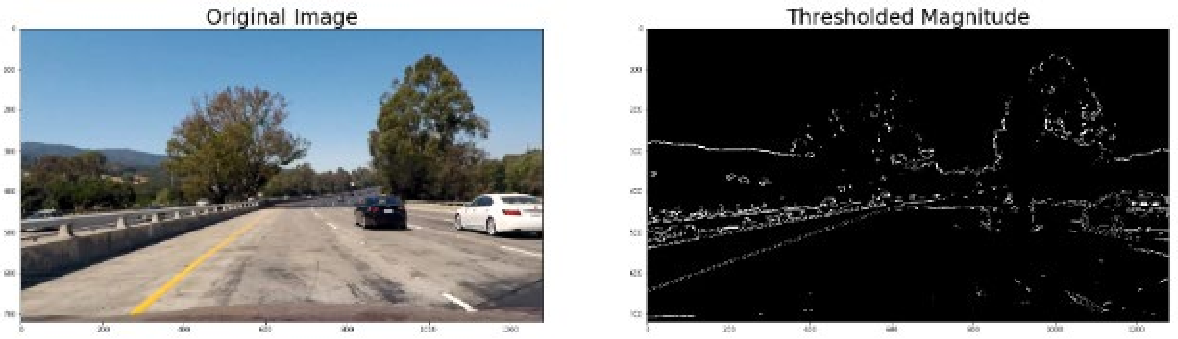


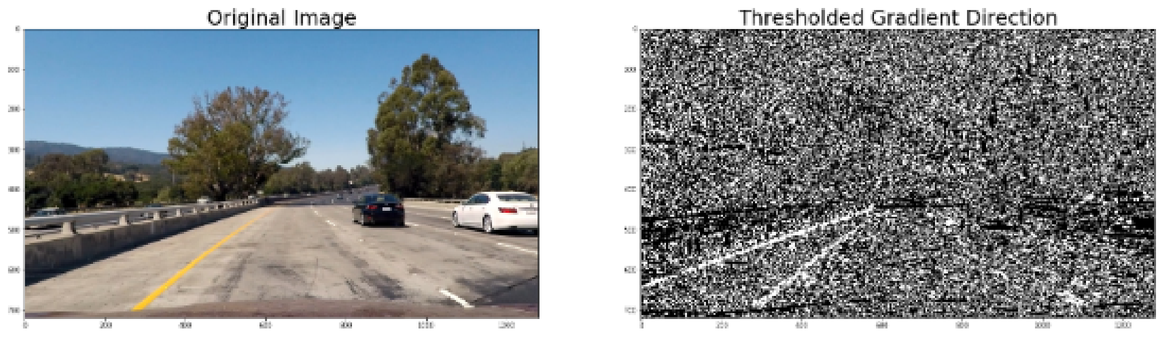
Color transforms, gradients and other methods to create a thresholded binary image.

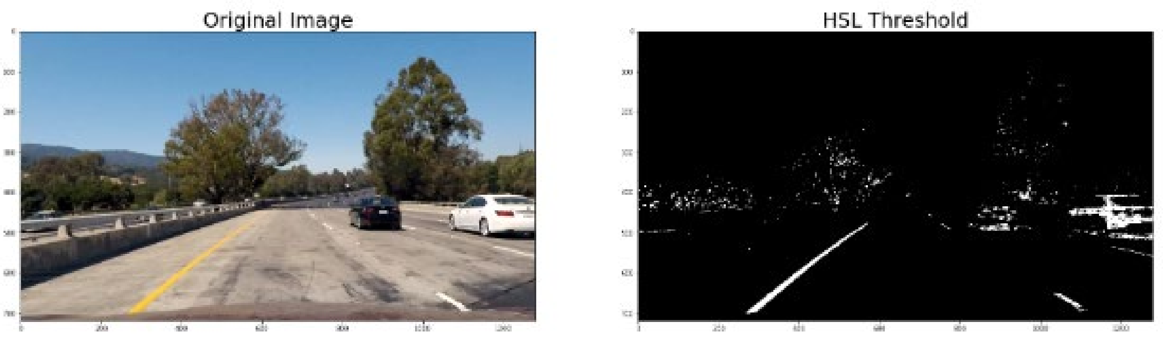
I used different functions and method to obtain a binary image.
I defined four functions to calculate several gradient x, y, magnitude, direction and HLS colorspace.



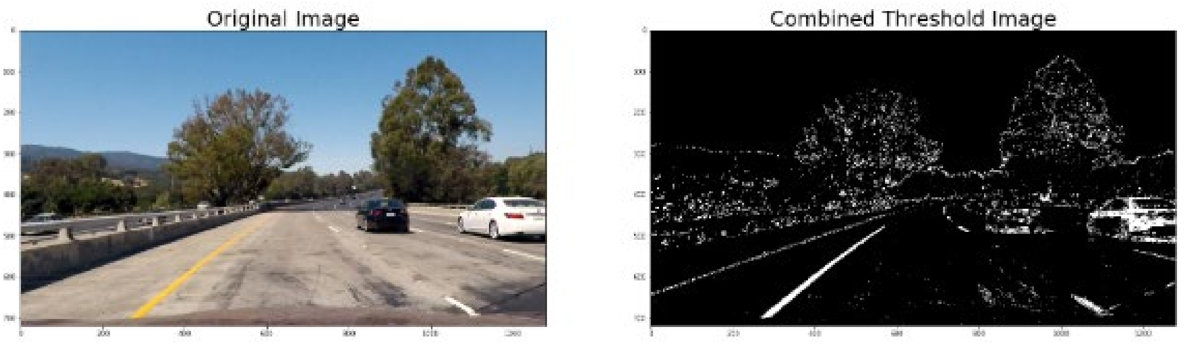








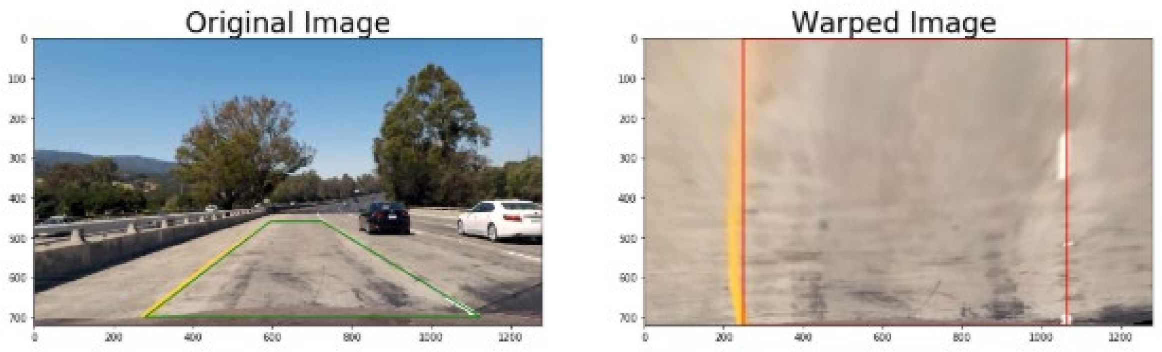
- Below is the example of an image before and after with threshold applied.



Perspective Transformation and birds-eye view.

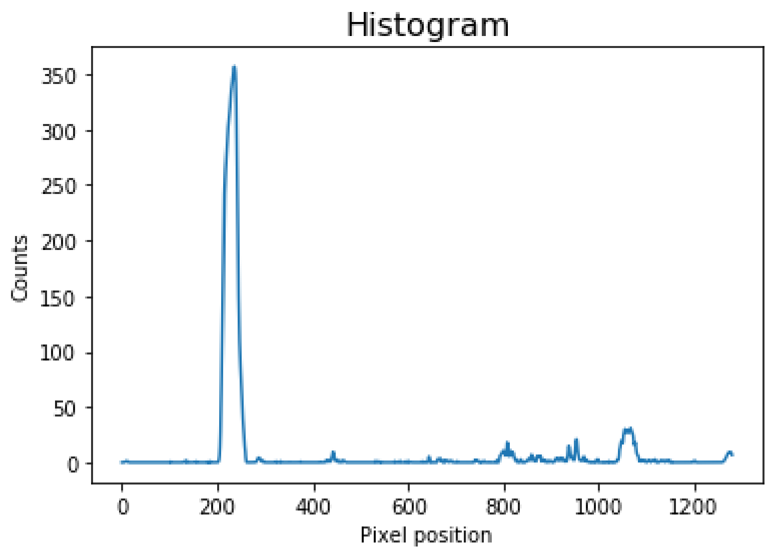
I transformed the image from the normal point of view to the bird eye view.
I selected the coordinates corresponded to a trapezoid in the original image and calculated the parameters to transform the image.
I used getPerspectiveTransform() function to calculate the perspective transform matrix, and then I used cv2.warpPerspective() function to correct the images to a "birds-eye view"

- Below is an example of these lane lines image before and after perspective transformation.



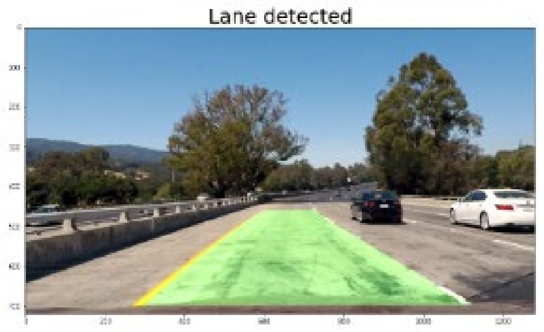
Lane pixels and find the lane boundary.

To detect the lane pixels from the warped image I used a function Histogram. With this histogram I added the pixel values along each column in the image. After that I used a technique known as Sliding Window in order to identify the most likely coordinates of the lane lines in a windows.



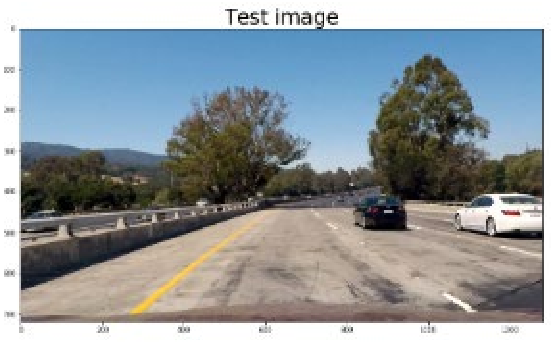
The left and right line have been identified and fit with a polynomial curved functional form.

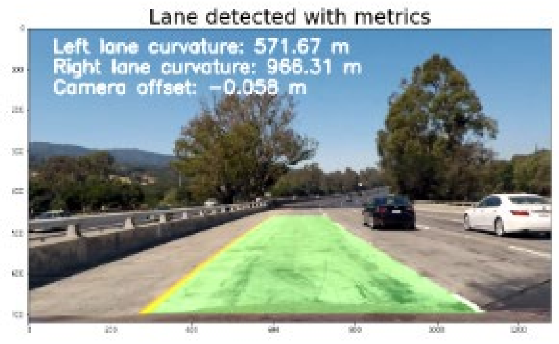
- Below is an example with the result.



Curvature and offset to the image

The information about the measurements like, where the lane lines are, estimate how much the road is curved and where the vehicle is, are very important. The radius of curvature is given in meters assuming the curve of the road follows a circle.

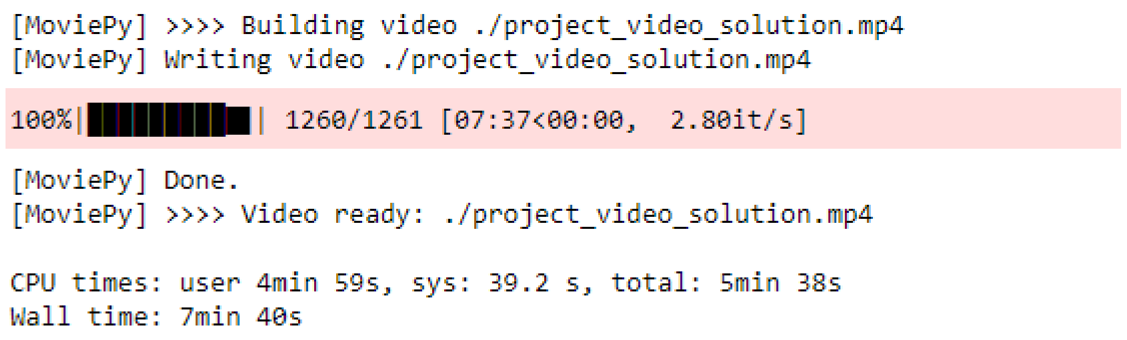




```
Left lane curvature: 571.67 m
Right lane curvature: 966.31 m
Camera offset: -0.058 m
```

Run pipeline in a video.

I used all the previous steps to create a pipeline that can be used on a video. In order to create a video and use all functions I created a class.



```
[MoviePy] >>>> Building video ./project_video_solution.mp4
[MoviePy] Writing video ./project_video_solution.mp4

100%|██████████| 1260/1261 [07:37<00:00,  2.80it/s]

[MoviePy] Done.
[MoviePy] >>>> Video ready: ./project_video_solution.mp4

CPU times: user 4min 59s, sys: 39.2 s, total: 5min 38s
Wall time: 7min 40s
```

For me this this project war very challenged but I made this project and I´m very Happy about it.

p.s. I´m in delay with the project ☺