# Advanced Lane Lines Finding

# Project Writeup

## Goals :

The goals / steps of this project is to create an advanced lane lines finding pipeline following the steps bellow:

- The goals / steps of this project are the following:
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

## Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

## Camera Calibration

1. The code for this step is contained in the first code cell of the IPython notebook located in AdvanceLaneLinesFinding.ipynb. The camera class represent an object family of camera offering properties and methods to calibrate, undistort images.

**Class Camera :**

**Properties :**

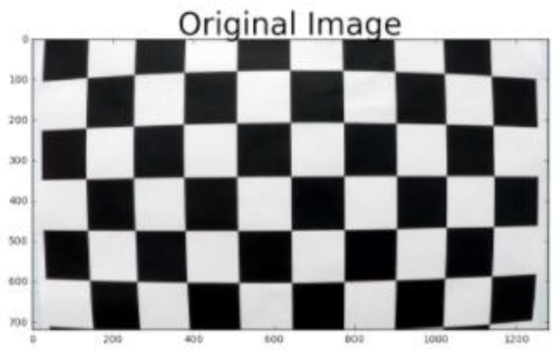
- calibration_images : set of calibration images directory
- corners_x : horizontal corners number
- corners_y : Vertical corners number
- reuse_mtx=True : indicating to class if we want to generate a new calibration file or reuse the existing. This property shall be positioned to False at the begining

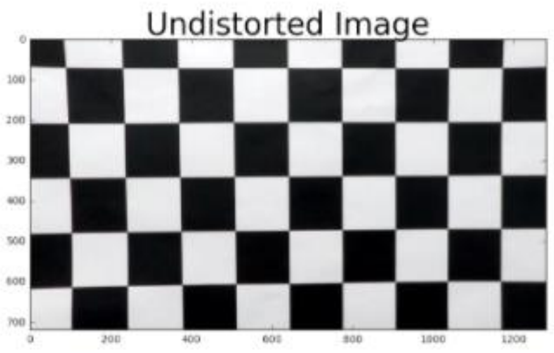
**Methods :**

- **_calibrate** : called internaly by the class itself to calibrate the camera and generate the calibration file. It is executed if the reuse_mtx is False.
- **undistort(self,image, path='',show=False,rgb=False,save=False)** : called by the user to undistort raw image.
    - Path : is empty when there is no need to save image to disk
    - Show : if positioned to True the function will display the raw and unidistorted image
    - Rgb : if true, the displayed image with rgb component
    - Save : if true, the undistorted image is saved to disk
- **batch_undistort(self, dirpath,show=False,rgb=False,save=True)** : Called to by the user to undistort a bunch of images located in input directory.
    - Dirpath : input directory
    - Show : True to display undistorted and raw images
    - Rgb : True to display colored images
    - Save : True to save undistorted images to disk
- **get_instance()** : static method to automatically provide  the user a calibrated camera object
- **run(calimages,input,output)** : static method to automatically undistort raw images located at an input directory and save them to undistorted output directory.
    - Calimages : calibration images because it calls the get_instance() method which in turn need to instantiate a camera image
    - Input : input raw images directory
    - Output : output undistroted images directory

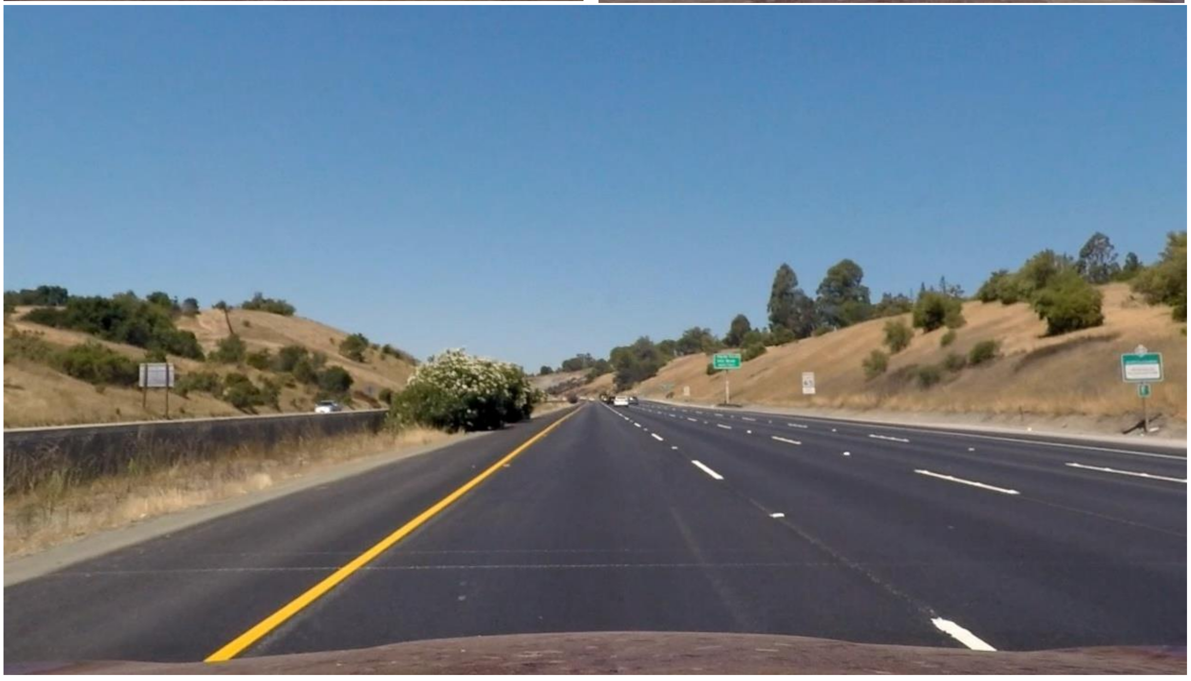
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successfull chessboard detection.

I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:





Real images :



# Perspective Transformation

## Class Perspective :

### Properties :

- src_points: source points of the perspective transform on the undistorted image
- dest_points: destination points of the perspective transform on the warped image
- M : Transform matrix
- M_inverse : inverse transform matrix

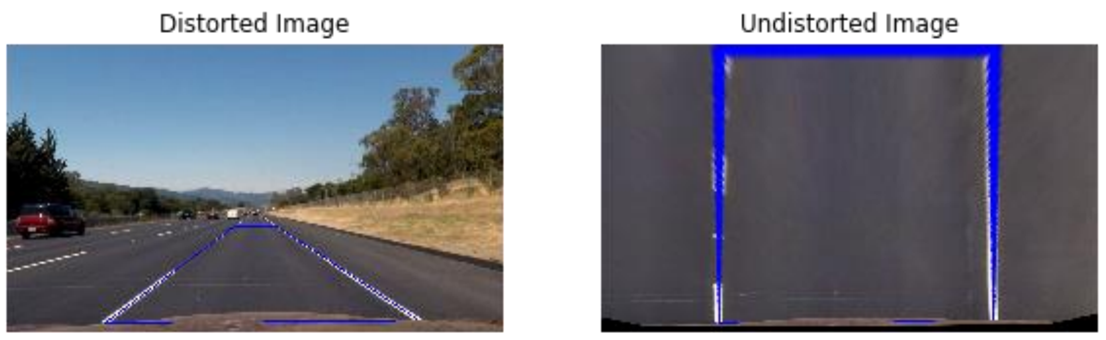
### Methods :

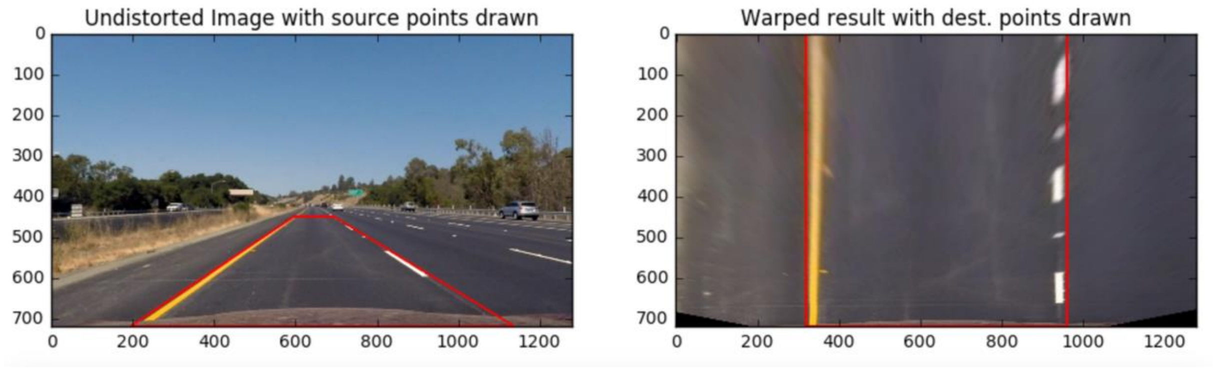
- **transform(self, image,path='', show=False,rgb=False,save=False)**: called to transform the undistorted image to a warped image using the
  - **Image** : the undistorted image to be transformed
  - **Path** : directory location to which it saves the warped image
  - **Show** : if true, the warped and undistorted images are showed
  - **Rgb** : if true, colored images are showed
- inverse_transform(self, src_image) : called to to transform back from warped image to undistorted image. Usefull to visualize the lane on the video
- get_instance() : static method. It is called to instantiate automatically a perspective object
- drawlines(path,btleft,topleft,topright,btright) : static method. it draws the corners box on a warped image
- batch_transform(input,output,perspective,btleft,topleft,topright,btright) : a static method to transform a bunch of undistorted images located in input directory and save them to an output directory. It also show all the input and output images
  - input : input directory images
  - outpout : output directory
  - perspective : a perspective object instance
  - (,btleft,topleft,topright,btright) : corners
- run(corners,input,output) : it gets a perspective instance from the get_instance(), draws corners box, transforms and displays the demo image and transforms a bunch of input images using batch_transform method

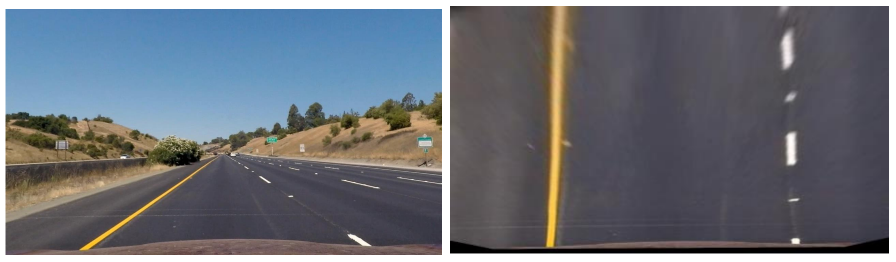
The perspective object is initialized with the choosen source and destination corners :

| SOURCE | DESTINATION |
|---|---|
| (253, 697) | (303, 697) |
| (585, 456) | (303, 0) |
| (700, 456) | (1011, 0) |
| (1061, 690) | (1011, 690) |

The code for my perspective transform (see above) is applied to input undistorted images and the result is the follwing :




Undistorted Image with source points drawn
Warped result with dest. points drawn
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



## Image Binarization

**Class Binarizer :**

**Properties :**

- output: output directory to which binarized images are saved

**Methods :**

- **binarize(self,image, path='', gray_thresh=(20, 255), s_thresh=(170, 255), **

  **l_thresh=(30, 255), sobel_kernel=3,show=False,rgb=False,save=False)**: called to binarize the warped image to a binarized image using the

  - **Image** : the undistorted image to be transformed
  - **Path** : directory location to which it saves the warped image
  - **Gray_thresh** : gray threshold to apply for sobel transform
  - **L_thresh** : l channel threshold to apply for lightning transform
  - **S_thresh** : s channel threshold to apply for saturation transform
  - **Sobel_kernel** : kernel to apply sobel x transform
  - **Show** : if true, the warped and undistorted images are showed
  - **Rgb** : if true, colored images are showed
- **run(input,output, gray_thresh=(20, 255), s_thresh=(170, 255), **

  **l_thresh=(30, 255), sobel_kernel=3,show=True,rgb=False,save=True) :** gets a binarizer instance and binarizes a bunch of images from an input location. It saves the resulting binarized images to an output directory and displays them

- get_instance() : gets an instance of a Binarizer object

The binarization process is applied to undistroted image to get a simple binary image of the lane lines. To do that, i use filtering technics seen in the course :

- Sobel operation in X direction
- Color thresholding in S component of the HLS color space.
- Color thresholding in L component of the HLS color space.

The result of the binarization process is the following :

## Advanced Lane Lines finding and Curvature Calculation

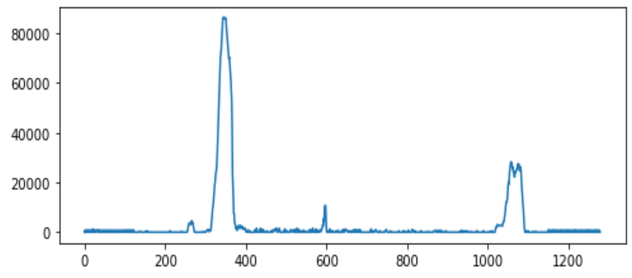
**Class Line :**

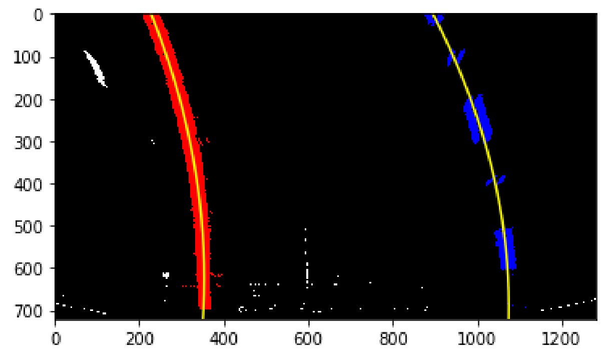
**Properties :**

- left_fit: pixels that are belonging to left line
- right_fit : pixels that belonging to right line
- perspective : perspective object
- camera : camera object

**Methods :**

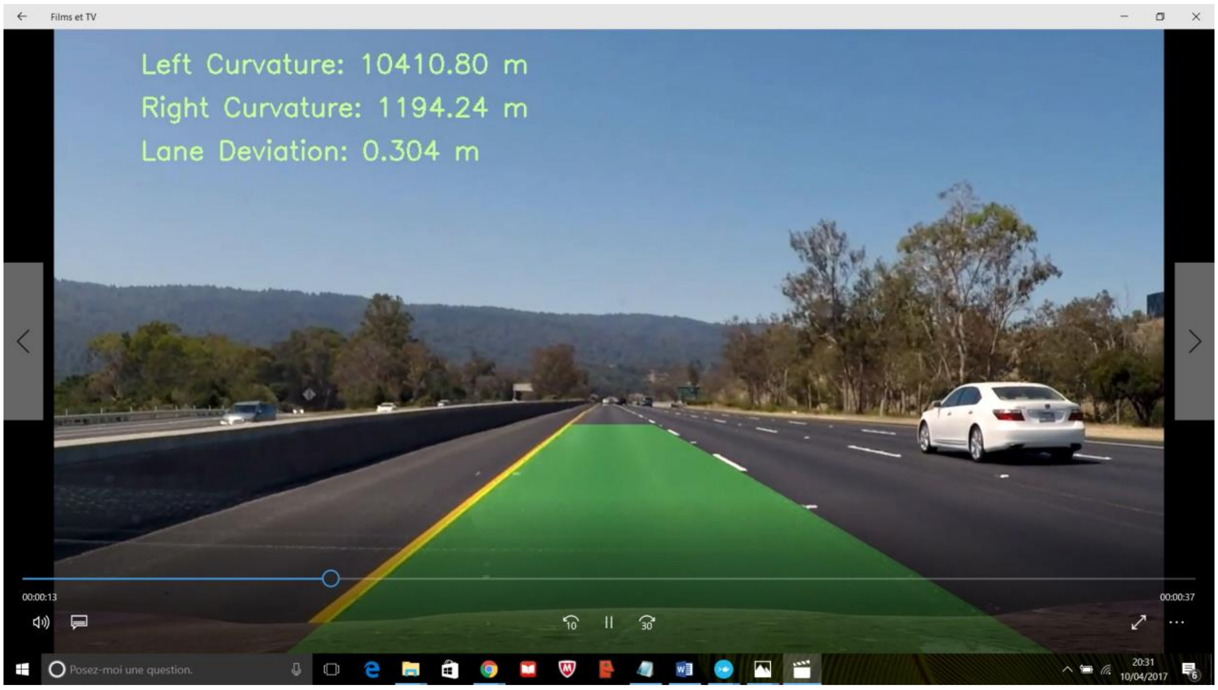
- **firstime_extractor(self, binary_warped)**: called to extract lane lines from the warped image provided in input. It uses a histogram of the pixels intensity of the lower half of the image. The peak of the histogram show the x position of each peack.

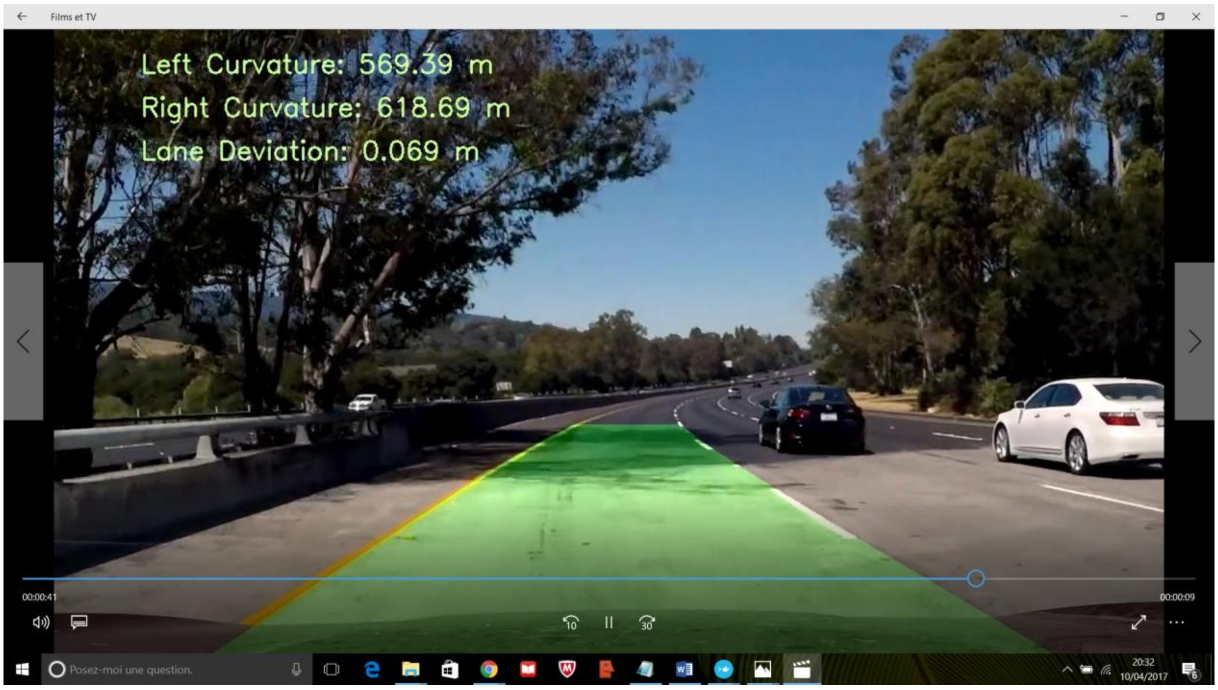




  - **binary_warped**: the binary warped  image to be processed
- **lazy_extractor :** gets the curvature and lines pixels positions for the next images without starting the process from the begining (calculate the histogram to deduce peaks ....)
- **lane_properties(self, image_size, left_x, right_x) :** calculate the curavatures radius and the lane deviation.
- **fill_lane(image, fit_left_x, fit_right_x) :** static method to fill the space between the lines of the lane. The filled space is colored in green
- **fuse(self, binary_img, src_image) :** merge the input image and the binary image after lines highlighting
- **run(self, image) :** processes all the line object pipeline in the following order :
  - image undistortion
  - image perspective transformation
  - image binarization
  - firstime or lazy extraction
  - curvature and lane properties calculation
  - drawing lane space and lane properties on the binary image
  - merge the binary image with the undistorted image

The run pipeline method produces the result bellow.





- After applying the run method to a provided video : project_video.mp4, it results in a output video file in which the lane is highlighted in green and its properties are displayed in the lef top corner. You can find it here

## Discussion

**Drawbacks**

1. The lane lines finding algorithm is very complicated and not trivial to tune. One of the futur enhacement is propably to simplify it and make it more object oriented to abstract the complexity.
2. I tested my program with the challenge video but it fails when calculating the lane propreties because it computes a negative lane width. I conclude that it don 't generalize well to other videos.

**Future Works**

1. The lane lines finding algorithm should be tuned at the curavature calculation level to make the program generalize well
2. Apply deep learning or convolution network to line finding