March 27, 2020

Advance Lane Finding Project

Goal:

Write a software pipeline to detect a highway lane in a video captured by a car front camera.

This software should also calculate the **Lane curvature and lateral position of the vehicle** on the lane as part of the autonomous vehicle data feeds.

An overview on lane detection principles was already introduced in my previous paper *Lane Detection*.

Advance Lane Detection general pipeline:

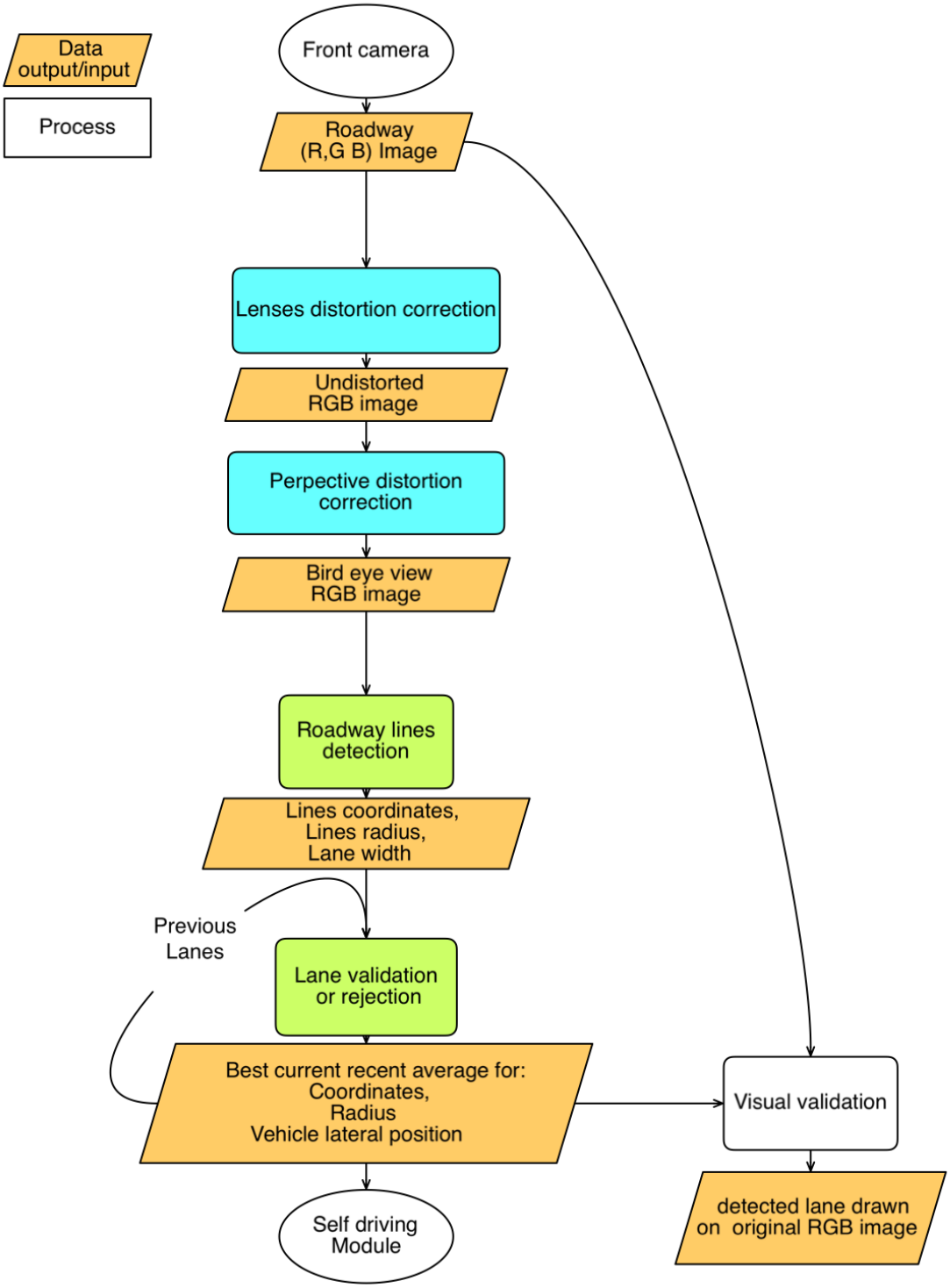
This project is one step forward in the lane detection process as it addresses the detection of curved lanes. It also also outputs the current radius of the lane and the lateral position of the vehicle.

The project is built on 3 mains processing blocs:

- **Optical corrections** addresses lenses induced distortions (barrel, cushion…) and perspective distortion (parallax effect…) .
- **Roadway Lines detection** operates on each image (i.e frame from the video feed) combining edge detection algorithms and single channel thresholding .
- **Line vetting** discards lanes with irrelevant width as well as lanes irrationally deviating from the current flow. (validation of rejection)

Data outputted by each process is outlined in the pipeline chart below :

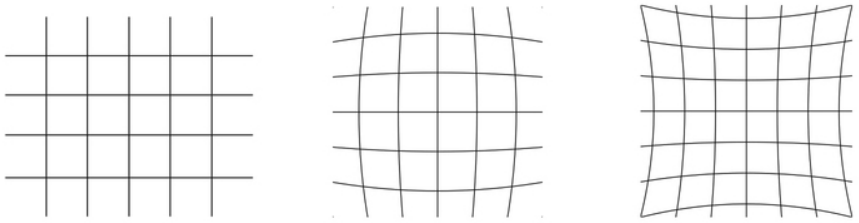
Pipeline Overview:



Optical distortion corrections :

Camera Distortion Correction:

Optical lenses induce two types distortions : barrel and pincushion



The degree of these distortions varies with the distance of the objects to the lenses .

See Appendix 3: Camera distortion, chessboard correction

Camera Distortion correction algorithm principle:

A chess board is photographed at different angles and distances from the camera. Pictures and chess board dimensions are then compiled to create a rectification matrix.

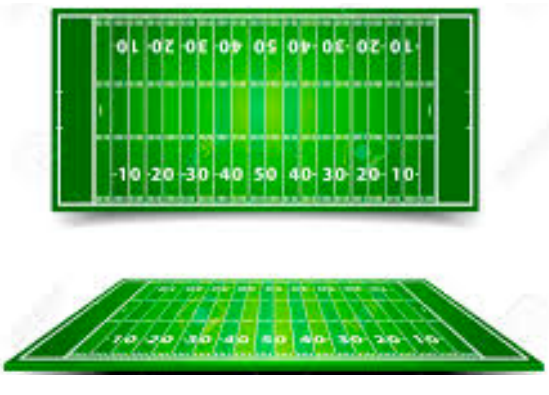
This matrix will be then applied to each frame of the video stream.

Implementation:

Running this matrix calculation through a range of detectable chess squares combinations (6 to 10 lines) by (5 to 7 columns) is time consuming . Consequently once calculated, the matrix is stored for later use. *(pickle.dump([mtx, dist], f) (mtx_pickle, dist_pickle = pickle.load(f))*

Perspective Correction:

A picture is a 2D representation of a 3D object. Objects closer to an observation point (camera or human eye) appear larger while farther objects are visually dwarfed . Consequently a straight section of highway which in fact is similar to a football field appears as a trapezoid .



Perspective Distortion correction algorithm principle:

Four points corresponding to a rectangular section in real dimension are picked on the perspective picture. A perspective correction matrix is calculated from the position of the points within the picture and the real space . This matrix will be then applied to each frame to correct the perspective distortion.

Implementation:

Determining the right points on the source picture and their corresponding location on the corrected picture (destination) is a bit tricky : this tool measurement tool available of the web made it easier :

https://yangcha.github.io/iview/iview.html (courtesy of *Yangcha*)

The source and destination points we picked are:

src = np.float32([[941,720],[339,720],[600,457],[679,457]])

dst = np.float32([[896,720],[384,720],[384,0],[896,0]])

See Appendix 1 : bird eye views

Roadway lines detection:

Specific edges within the picture are detected and eventually qualified as roadway lines .

Edge Detection :

Sobel edge detection:

Edges in an images are tied to the high variance in values within the pixels neighboring pixels. (as in Canny edge detection in LaneDetection project).

Sobel Edge Detection can be refined by direction:

- either detecting vertical edges (X axis: Sobel_X)
- or horizontal edges (Y axis: Sobel_Y)) .

- even by angle : ratio (Sobel_X , Sobel_Y)

Channel Detection:

We'll convert the original (R,G,B) image in an HLS image (Hue, Luminosity, Saturation) and will only use the S channel which offers more contrast (higher variance around edges) .

Edge Detection pipeline:

- the image is converted from RGB to HLS
- within the HLS image only the S channel is retained
- because of the average position of the lines (mostly vertical) we discard all low horizontal (Sobel_X) gradient pixels (hence "denoising" the picture) in this S picture
- then we discard all edges not angled within a +/- 45 degree of the vertical axis
- we take the untreated S picture and remove all pixels under a threshold
- we then combined these 2 last pictures

See Appendix 2 : Sobel & S Channel

Lines detection :

The roadway line detection techniques exploit the general direction of the lines : stretching forward with a possible moderate curvature. Consequently on the bird eye view, lines appear mostly vertical or possibly slightly curved .
An histogram running in the horizontal direction of the bird eye picture will first detect the peaks of pixel density indicating the rough lateral position of each line.

Then a rectangular filter will convolute from the bottom of the image to the top of the image to collect the position of the relevant pixel . This filter will drift transversally to follow the curvature of the line as it progresses upward step by step (*Find_Lane*) starting from the histogram peak.

Based on all these findings a second degree polynomial function outputs the coordinates of the corresponding line

The next pictures will collect the pixels within the vicinity (*Search_Around*) of the line detected in the previous picture in one single pass. It will then also outputs a corresponding

polynomial line . This quicker search is carried on each following picture . The histogram search is only launched again when no relevant lines are found!

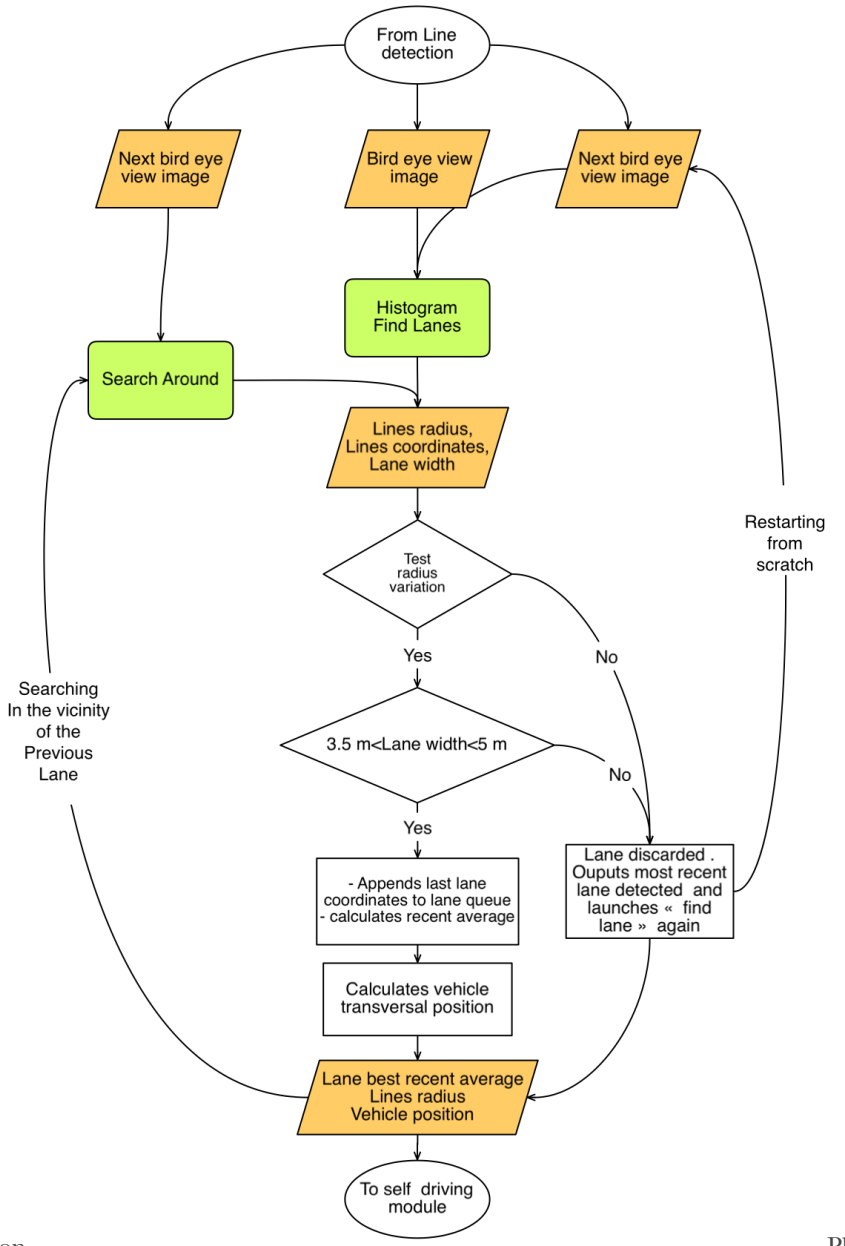
Implementation:

Find_Lane and *Search_Around* are off the shelf algorithms (implementation can be found on line or within this project workbook) .

Lane Validation (or rejection):

Visual details can be wrongly picked up by the edge detection algorithm. Consequently lines outputted by the line detection algorithm have to be verified to pass the following requirements:

- lane width (3 to 6 m)
- be compatible with the recently previously outputted lanes (radius values should be in the same range than previous ones)



Only qualifying lanes are added to a queue of recent lanes.

The average of this queue is then outputted. This averaging process avoid the flickering the lines .

For validation purposes and visualization the outputted lines are then superimposed on the original video . A 5 seconds preview is viewable here (Apple Pages documents required) .

Conclusion:

The quality of the output on the given video sample is satisfying . But this pipeline would not scale flawlessly in more challenging conditions as:

- lighting conditions making the roadway markings hardly visible:
  - saturated sun light (fading)
  - wet roadway and lights reflections
  - landscape and other vehicle shades
  - shades (from landscape and other vehicles)
- other challenges linked to the traffic :
  - covering of the lines by an other vehicle
  - erratic detection of and other vehicle (histogram)

Hence extra steps should taken to make this pipeline more robust given the above challenges

- enhancement of the edge detection (shape base enhancement image preprocessing)
- recognition of rectangular shapes (other vehicles) for dismissal

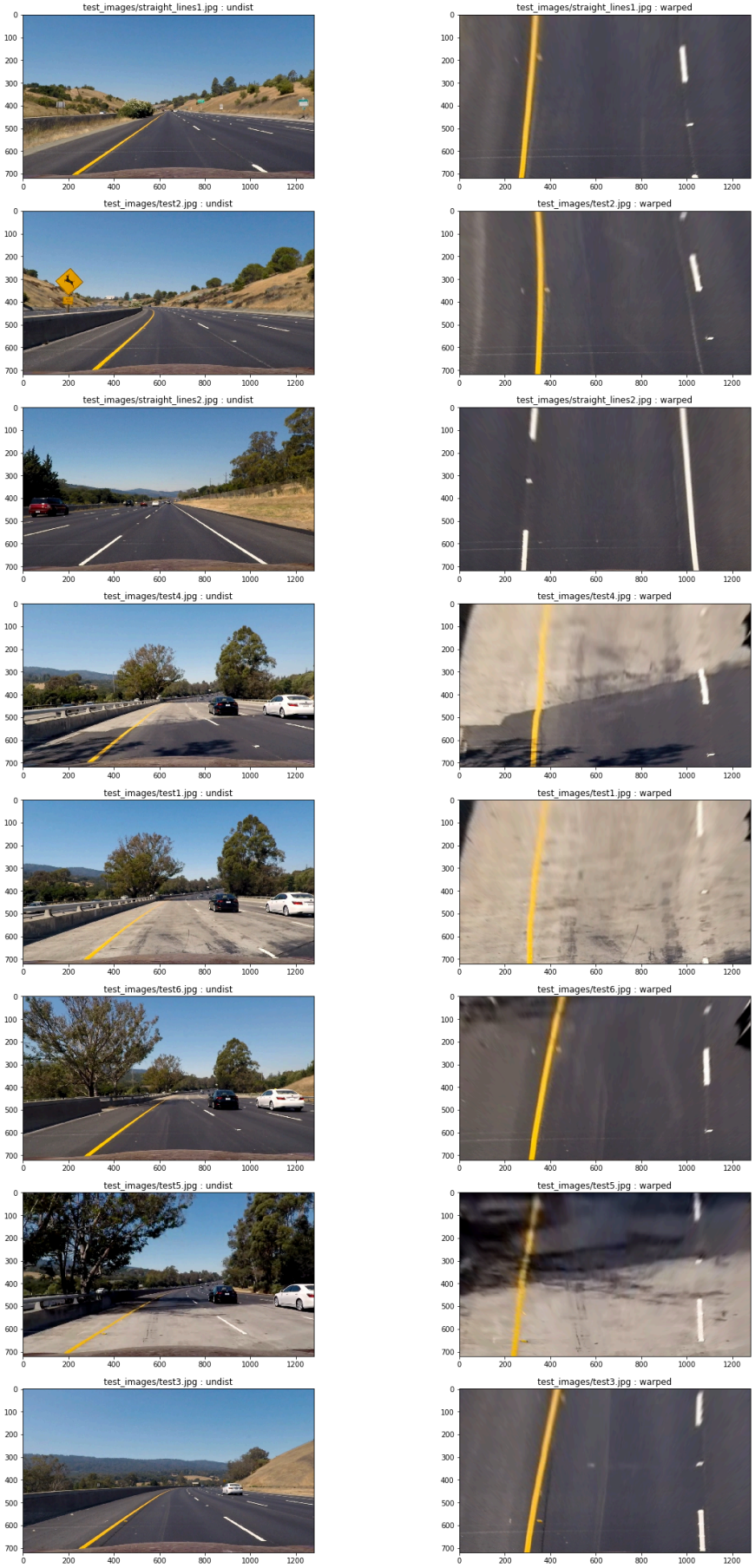
Processing time might also have to be optimized .

This project gives a first glimpse on some of the Lane Recognition basic techniques . As it reveals their limitations it lead us to understand how AI techniques mixing image recognition and sequence to sequence could be used to bring the recognition to the next level.

NB: appendices section next pages

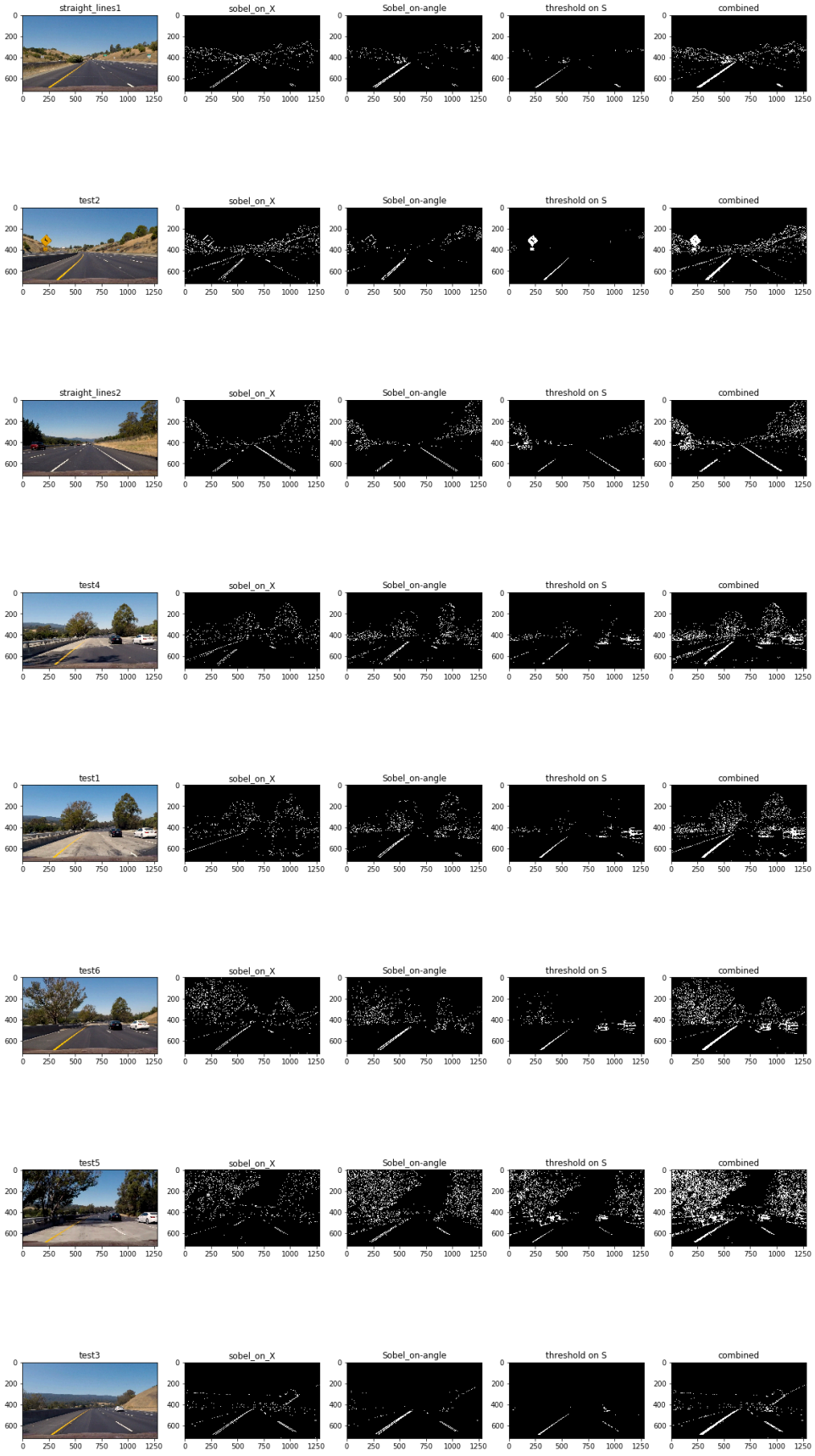
Appendix 1:

Bird eye view



Appendix 2:

Sobel & S Channel



Appendix 3:

Camera distortion: chessboard correction