Eduardo Ribeiro de Campos - Udacity / SDCND - Apr./2019.

Self-Driving Car Engineer Nanodegree

Project2: Advanced Lane Lines Finding on the Road

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.

All 8 item above was covered (step by step) in the Helper code (./Helper.ipynb). All the images presented in this document were created there.

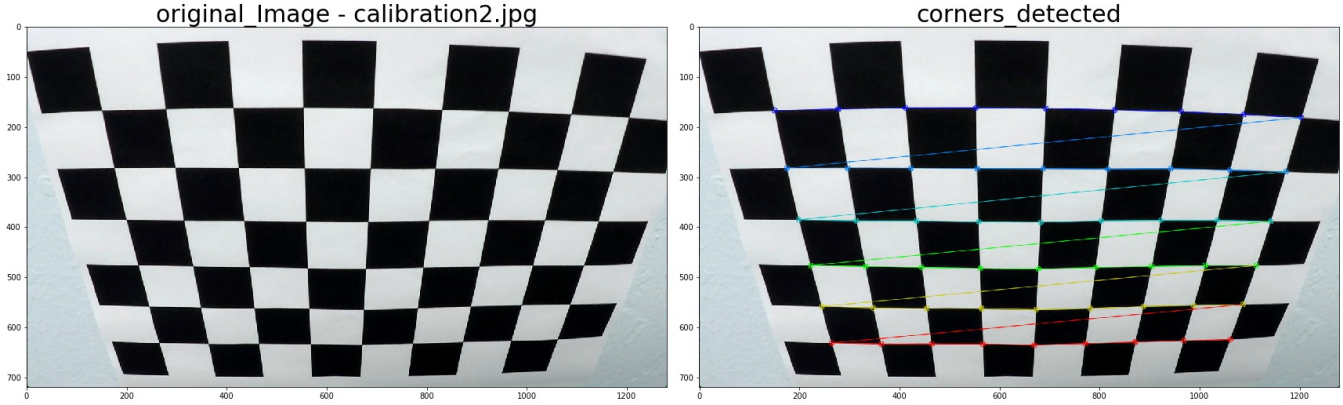
In the main code "Advanced_Lane_Finding.ipynb", In the second cell we import a python dictionary with some number constants generated in the Helper code (./Helper.ipynb). For example , the camera calibration to undistortion the images , the matrix to the perspective transform and lane lines width calculated in pixels.

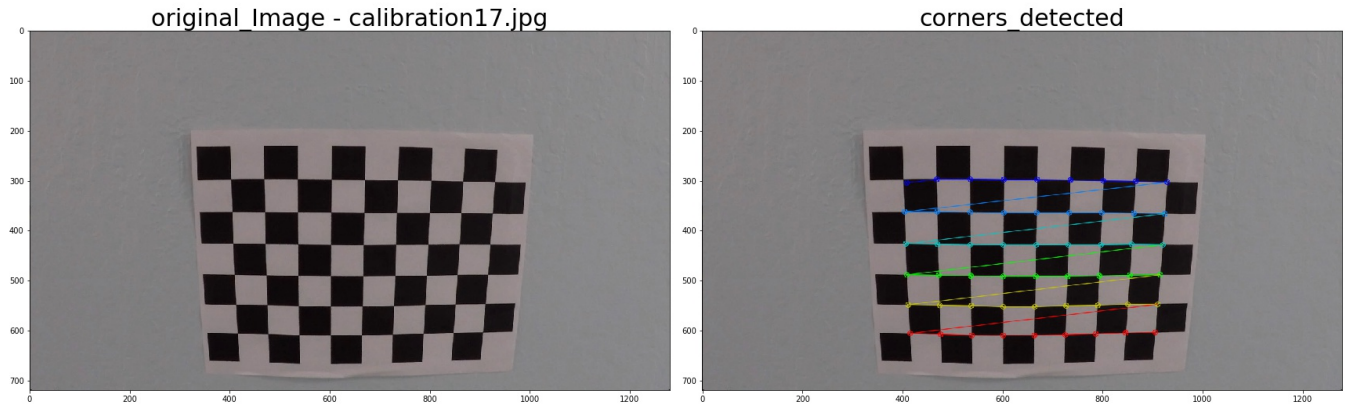
---

1 - Calibration Camera.

1.1 - Chessboard corners.

in this step was created lists to append the coordinate points of the chessboard corners in the real world and the coordinate of this points in pixels.The function used was `cv2.findChessboardCorners()`.The pictures below shows the results.

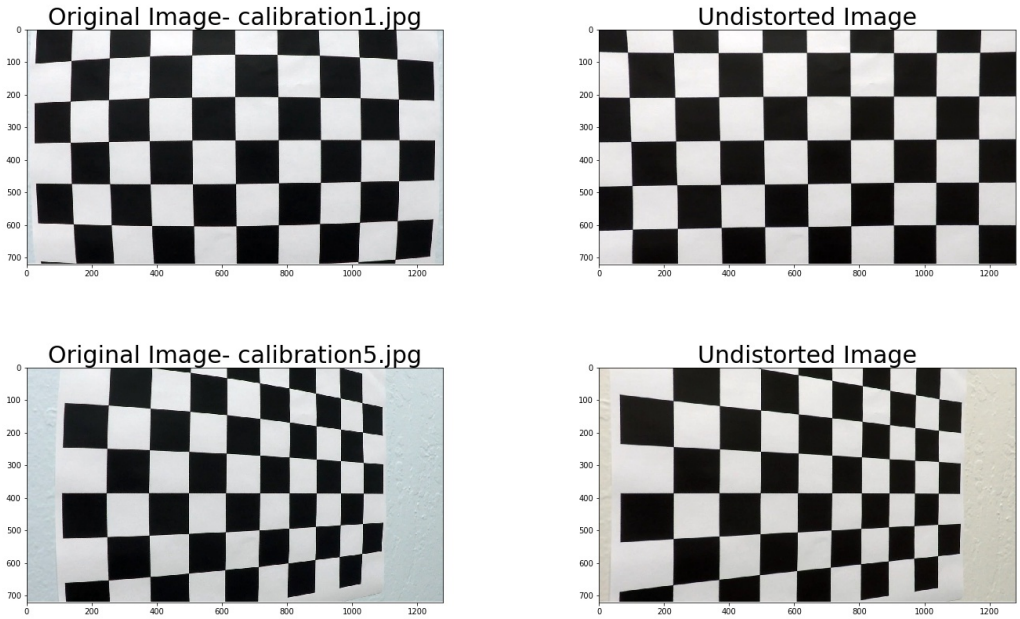




Here's a link to more results (./camera_cal/ChessboardCorners) for chessboard find corners.

1.2 - Undistortion Chessboard .

With the lists of the previous step, It was computed the distortion coefficients using the function `cv2.calibrateCamera()`. And after was applied the function `cv2.undistort()` to check the procedure.The pictures below shows the results.

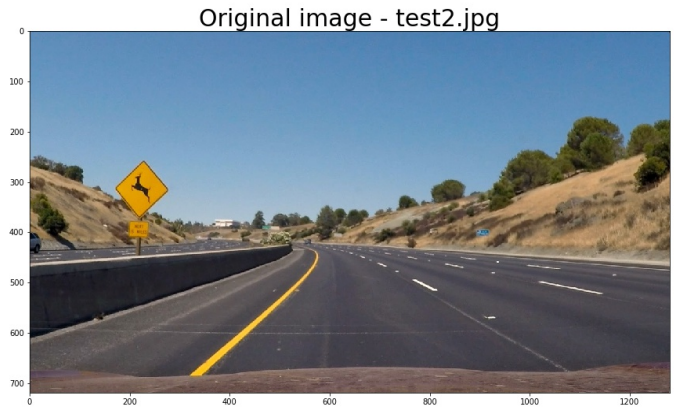


Here's a link to more results (./camera_cal/Chessboard_Undistortion) for chessboard undistorsion.

---

2 - Distortion correction .

With the coefficients generated in the step 1, it was applied the function `cv2.undistort()` in the real images available in the test_images folder (./test_images) and the results could be saw in the next 2 pictures. The relation and position of the other car and the traficc sign are different compared to the original image.So it is possible make a statement that the results were satisfaied.



Here's a link to more results (./output_images/Undistorted_images) for undistorsion in real images.

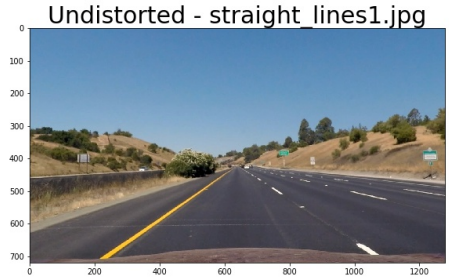
---

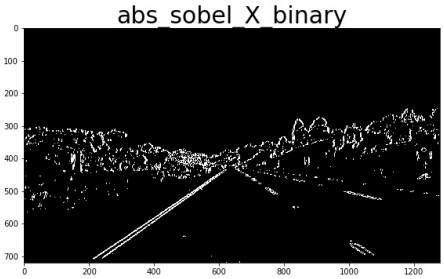
3 - Gradient thresholds and color transforms.

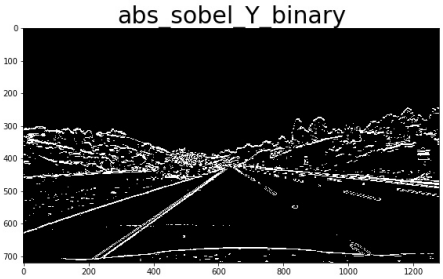
In order to generated a clean binary image to classify the lane lines, it was analysed different gradients and color spaces channels.

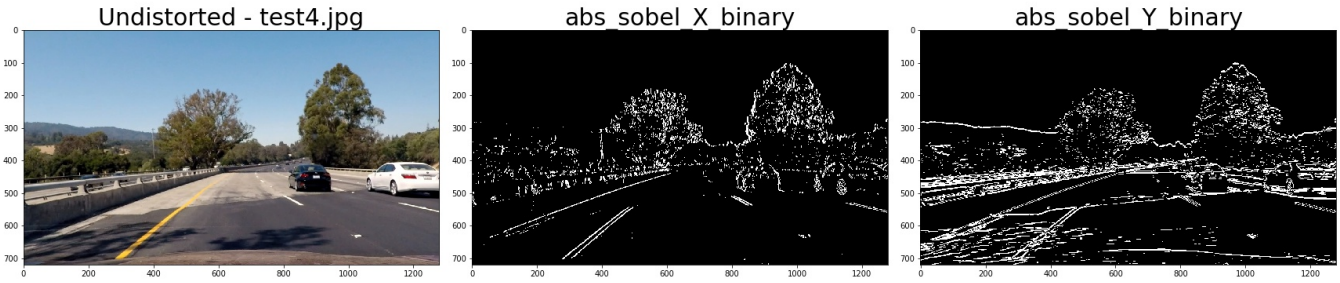
3.1 - Absolute Sobel.

In the Sobel gradient was calculated the absolute value of the x and y derivative using the functiom `cv2.Sobel()`, the range of the threshold to select pixels was 30 as minimun and 120 as maximum. The result for the x derivative were better than the y derivative, regard lane lines detection. Below 2 examples of the results.





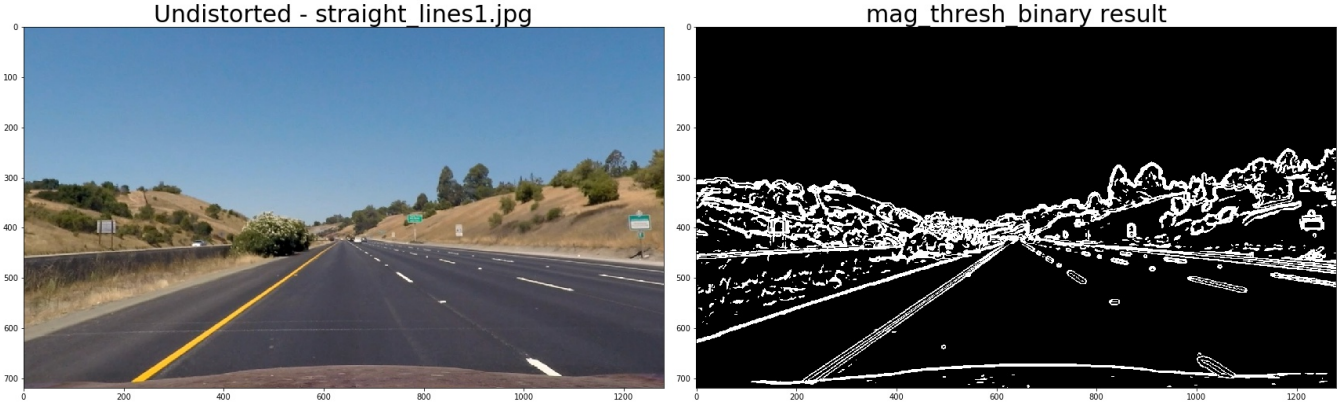


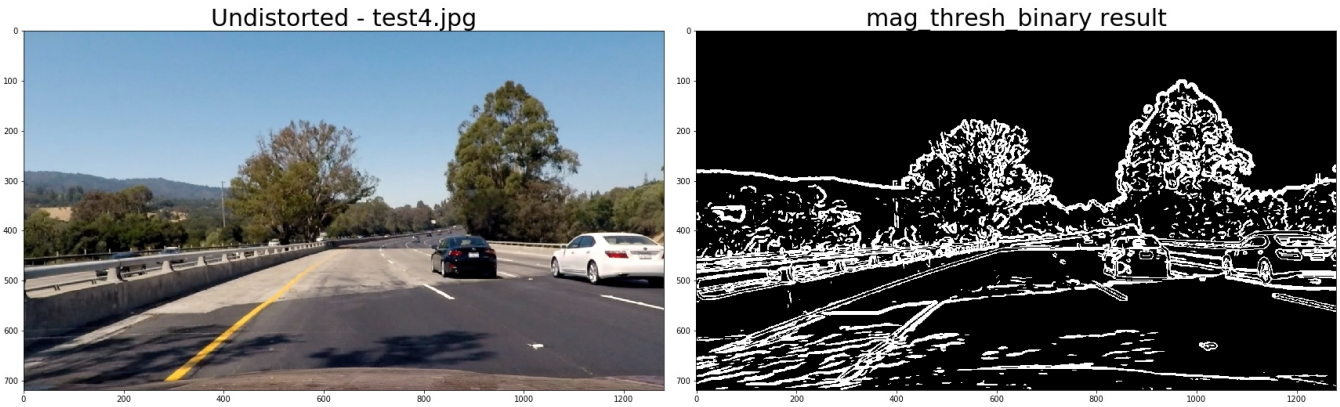


Here's a link to more results (./output_images/Abs_Sobel) for Absolute Sobel gradient threshold.

3.2 - Magnitude of the gradiente.

Here was applied a threshold to the overall magnitude of the gradient with the function `cv2.Sobel()` in both x and y. and after calculated the Magnitude with the formula $mag = \sqrt{sobelx^2 + sobely^2+}$. The threshold to select pixels was 30 as minimun and 120 as maximum (to keep comparison base with the others gradients), the results it is very clean, but in some cases it is not possible identify the lane lines. Below 2 examples of the results.

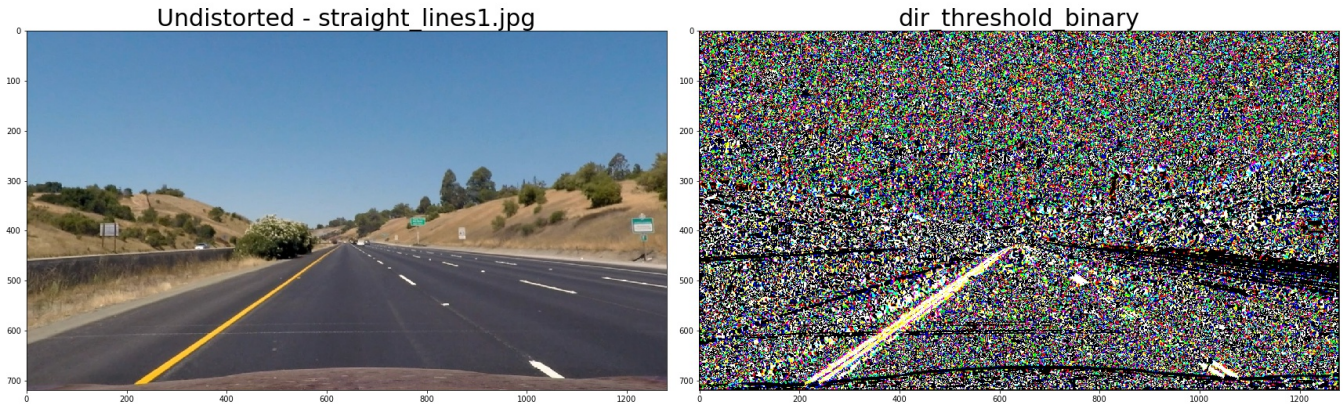


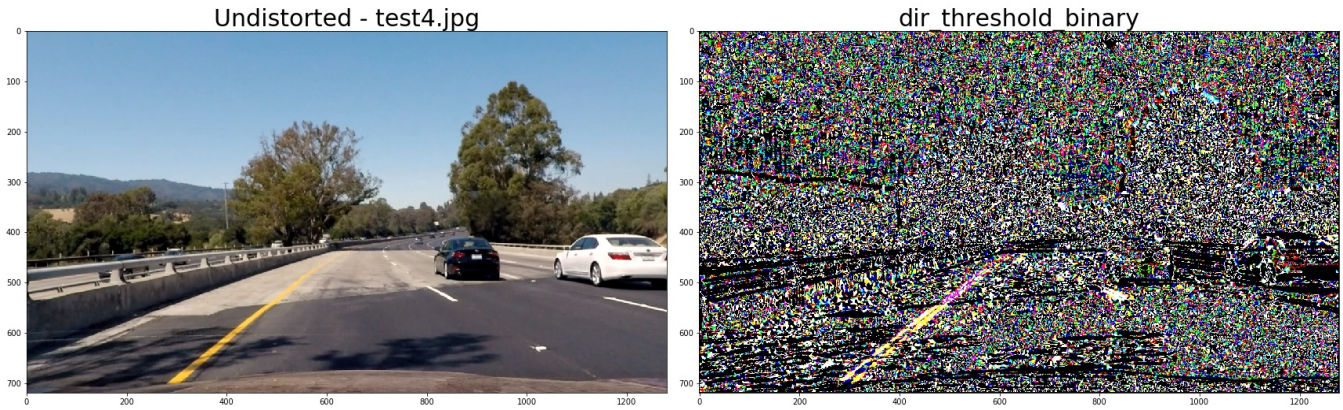


Here's a link to more results (./output_images/magthresh) for magmitude of the gradient.

3.3 - Direction of the gradiente.

Here it was calculated the direction of the gradient, Each pixel of the resulting image contains a value for the angle of the gradient away from horizontal in units of radians The direction of the gradient is simply the inverse tangent (arctangent) of the y gradient divided by the x gradient, the range used here was (0.6, 1.2).Below 2 examples of the results.

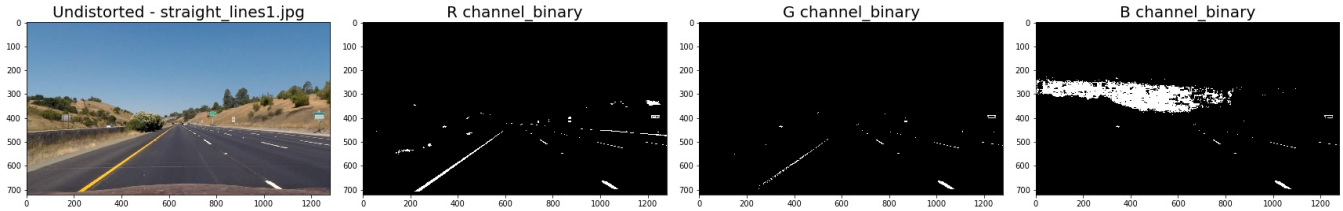


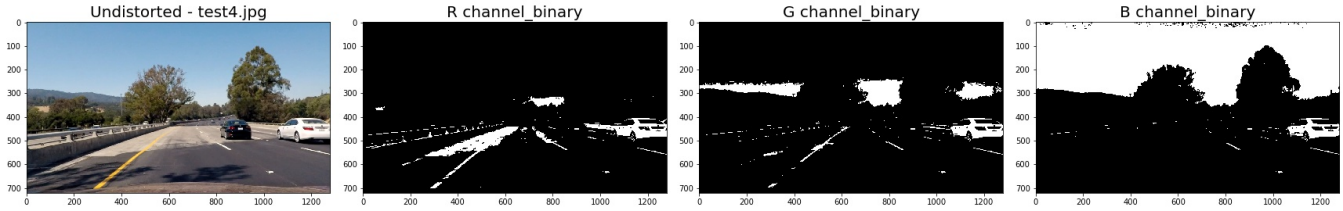


Here's a link to more results (./output_images/dir_threshold) for direction of the gradient.

3.4 - Color spaces RGB channels.

Here it was isolated each channel (RGB) and converted to binary inside the threshold range of (200,255), It is possible identify best results in te R channel, regard lane lines roads detection. Below 2 examples of the results.

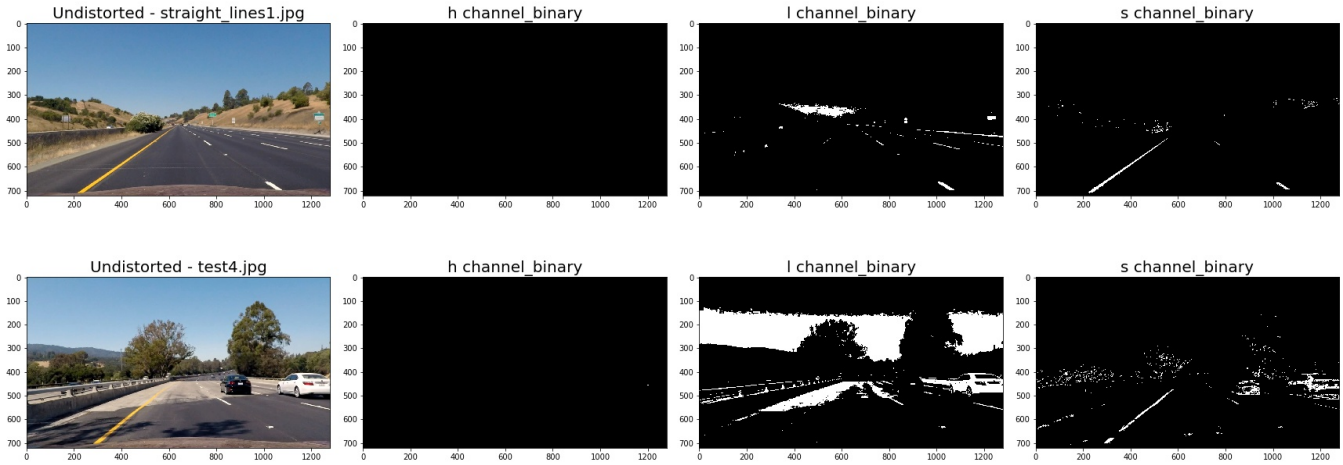




Here's a link to more results (./output_images/rgb_binary) for RGB Channels.

3.5 - Color spaces HLS channels.

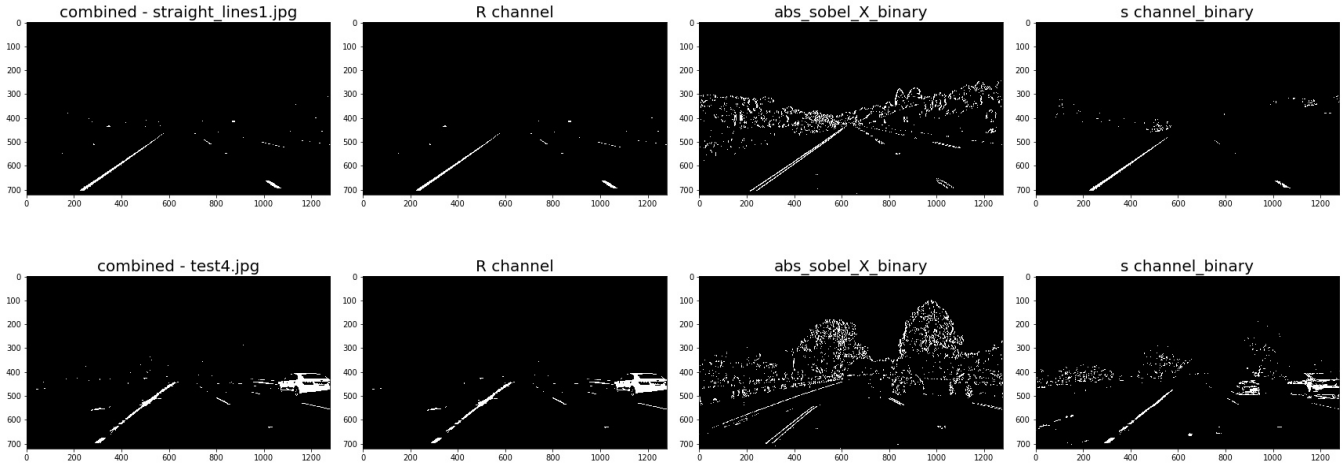
Here it was coverted the image from RGB to HLS color space and isolated each channel (HLS) and converted to binary inside the threshold range of (180,255), It is possible identify best results in te S channel, regard lane lines roads detection. Below 2 examples of the results.



Here's a link to more results (./output_images/hls_binary) for HLS Channels.

3.4 - Combined thresholded binary image.

After analyse different gradients and color spaces, proving a lot of different thresholds ranges , it was considered 3 good classifiers and combined in order to get a final binary image to identify the lane lines.



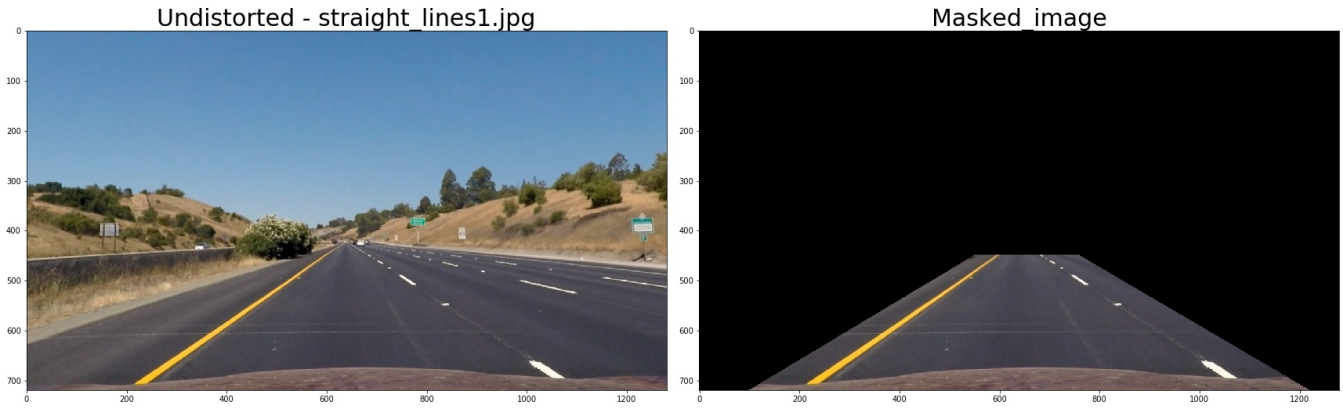
Here's a link to more results (./output_images/combined_binary) for combined binary images.

---

4 - Perspective transform ("birds-eye view").

4.1 - Mask image.

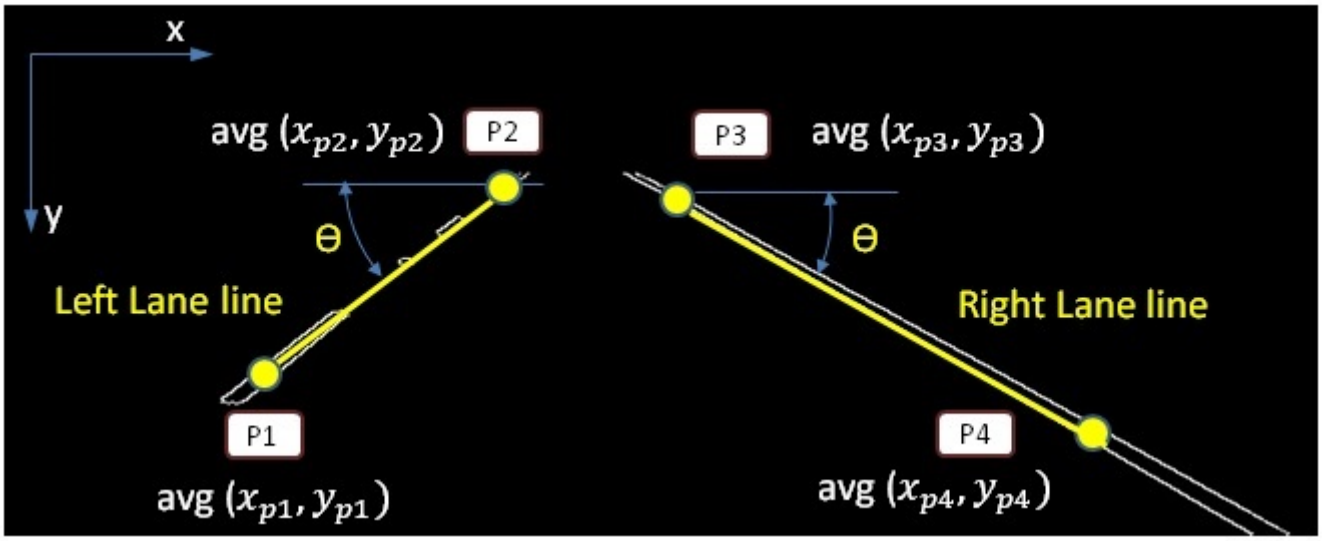
In order to identify coordinate points to define the left and right lane lines, it was created a mask to make easy the classification. The trapezoidal figure was created manually and ploted to see the results. the values fromn the vertices are P1(100,720), P2(550,450), P3(760,450) , P4 (1220,720). The result could see in the picture below.



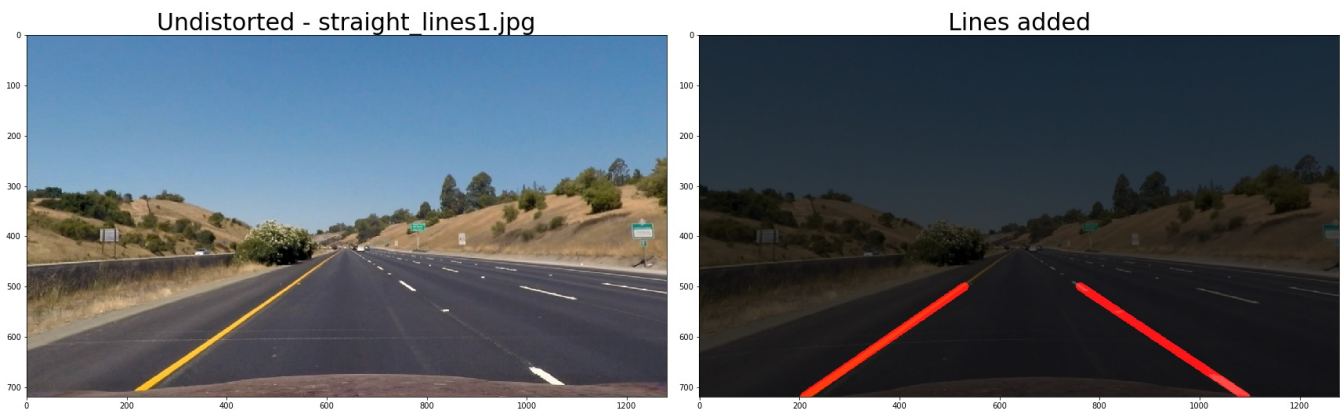
Here's a link to more results (./output_images/masked) for masked images.

4.2 - Generating extended straight lines

With the function`cv2.HoughLinesP()`, it was possible indetify and created lines setting some parameters as the minimum lenght and maximum gap. As second step, a for loop code was used to store the coordinate points and after found the average of the values, it was possible plot 2 individual lines for each lane line.



With the 4 coordinate points, it was defined the$Y_{upper} = 500pixels$ .with the derivative Δy/Δx , it was found the X coordinates for upper , lower for both lines. The picture below show the results.



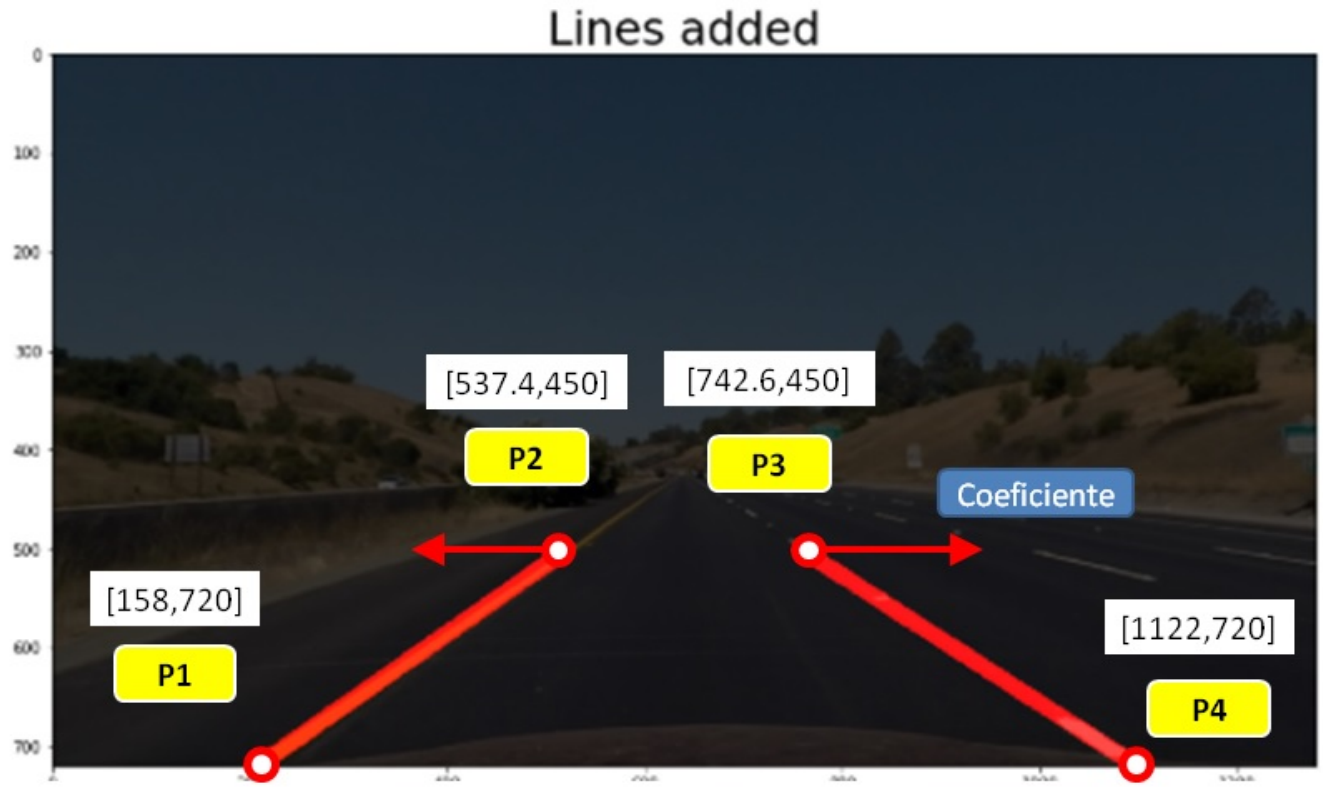
Here's a link to more results (./output_images/get_coordinate_lines) with extended straight lines applied.

4.3 - Defining coordinates to warp image

With the coordinate points identified using the Straight lines image example, It was calculate the tangent for each lines , and the values was the same, showing that in the straight lines image, the lane lines are simetricals.

In the next step, a for loop code have stored the minimum value X cooordinate for P1 (lower left point) and the maximum X coordinate for P4 (lower right point). the values are [158, 1122] , showing a second condiction with simmetry and does not matter if we work with the left or right line.

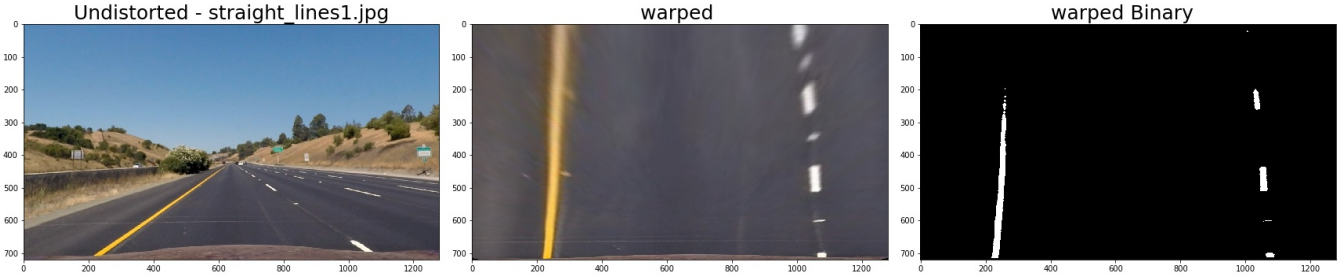
to tune the wrap results, it was added a coefficient to translate the upper points (P2 and P3). In the next schematic picture are shown the coordinates and where the coefficient was adopted.

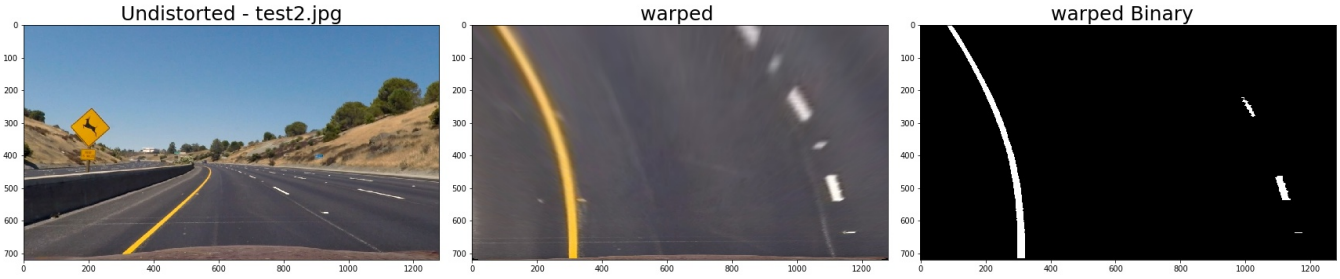


To create the transformatio matrix 'M', the source points (src) already explained in the previous step was used with the dst points (desired coordinates). In the dst coordinates was used the image size values with a margim = 200 pixels.

A function named "warp_images" was created to first apply the undistonsion with the function `cv2.undistort()` and after the perspective transforme with the function `cv2.warpPerspective()`.

The next images show the lane lines with the "Bird-eye-view" applied. It is possible identify in some cases the parallel condiction between the lane lines.





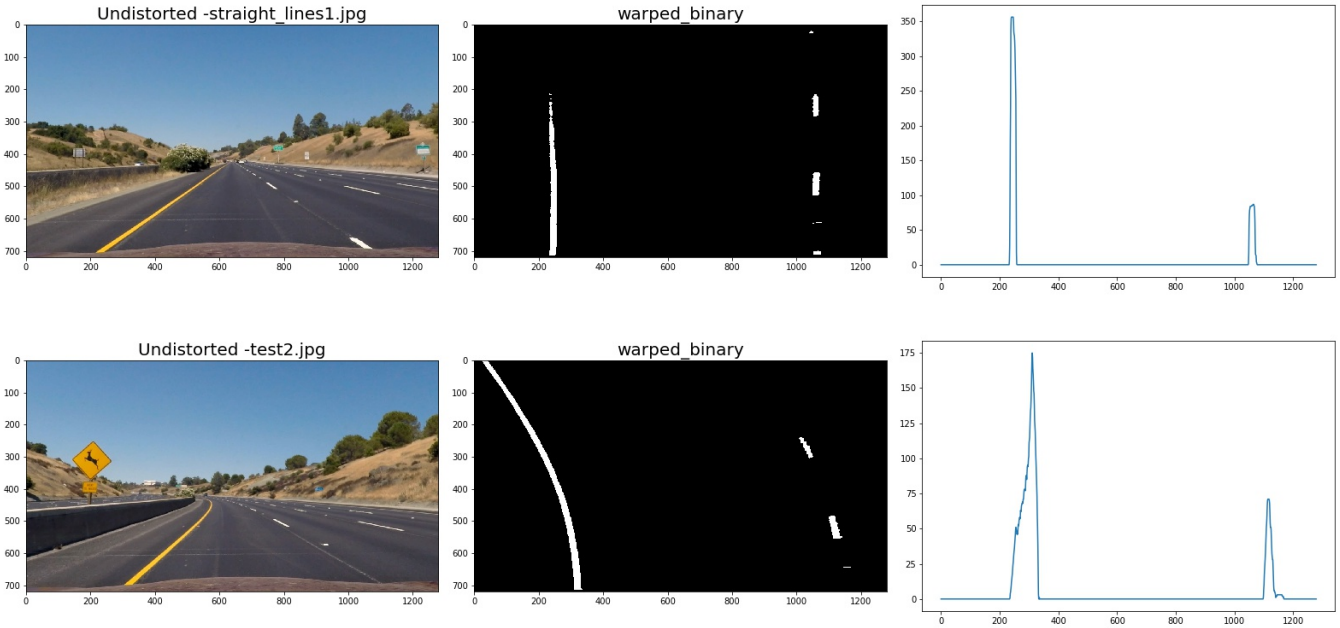
Here's a link to more results (./output_images/warp_images) for warped binary images.

---

5 - Detect lane pixels and fit to find the lane boundary

5.1 - histogram analysis

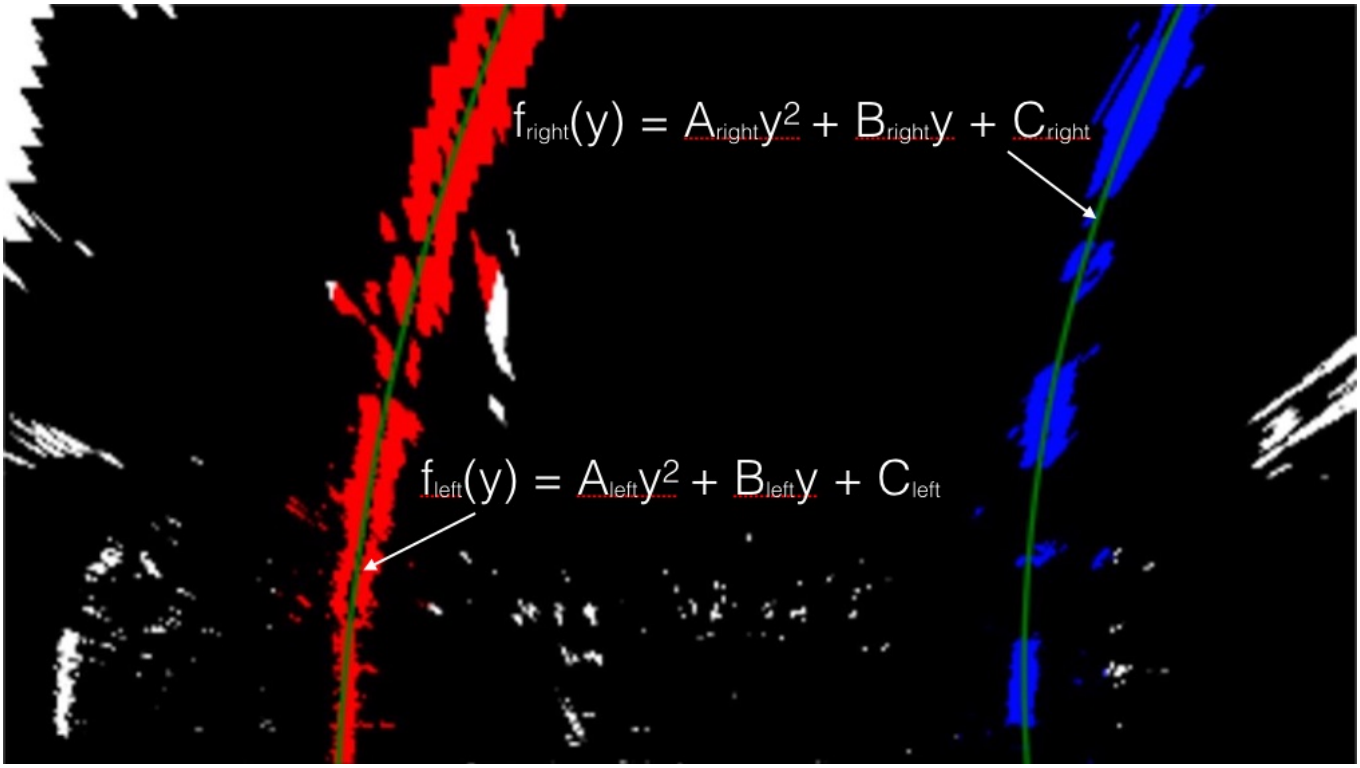
After figured out the bird eye view and convert the images to binary, it is possible plot a histogram and identify if the lane lines was found using the procedures. In the next 2 images is possible check that gradients and color spaces functions applied reach good results.



Here's a link to more results (./output_images/histogram) for histogram analysis.

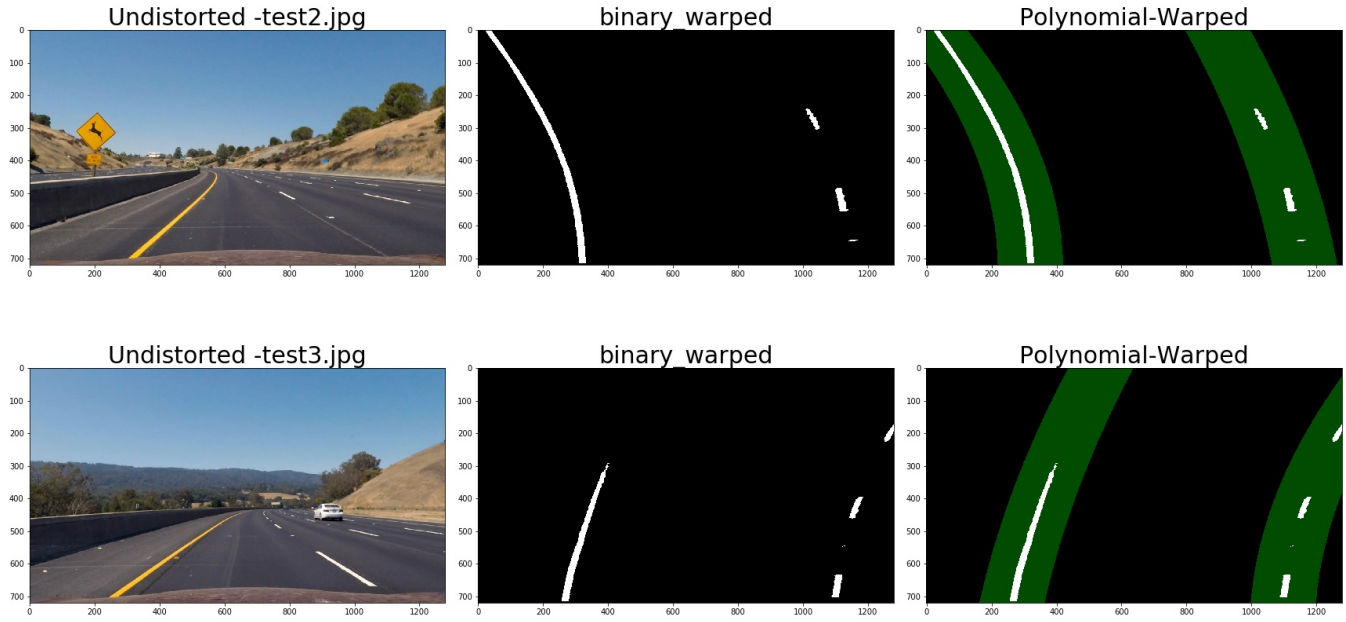
5.2 - Applying Polinomial curves

Using the histogram , it is possible identify the x position of the white pixels for each row of the image. by this criteria is possible identify the values [A,B,C] to fit a polynomial curve.



2 methods to identify the polynomial curves could be used. the first is finding the white pixels across the rows or range of rows "slide windows" , or use a previous polynomial coefficients to search around inside a margim predefined.

The images below show the polynomial curves identified by the method (slide windows) and the curves were filled using the function `cv2.fillpoly()`



Here's a link to more results (./output_images/polynomial) for polynomial curves applied.

---

6. Determine the curvature of the lane and vehicle position with respect to center.

Here the main trick is convert the pixels in meters. With the lower coordinate points (P1 and P4), and the information from lane line width equal 3.7m was defined the relatio in X Direction , the Y direction was assumed the UDACITY entry:

6.1 - Measuring the curvature.

Using the coefficients [A, B, C] generated in the step to identify the polynomial curves, it is possible to calculate the radius of the curvature using the formula:

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

If the left curvature is greather than the right means that the vehicle turn radius is to right, the opposite means the turn radius is to left.If the radius is greater than 1500m the vehicle is going straight

---

6.1 - Measuring the Offset.

With the coordinate points detected in the row 720 in the pixels detection (polynomial function) is possible indentify the middle of the lane, considering that the camera was installed in the middle of the vehicle, the diference between the center of the image and the center of the lane lines will indicate the offset distance of the vehicle in the lane lines. The sign of this distance will indicate if the vehicle is on the left or on the right compared to the center of the lane.

---

7. Warp the detected lane boundaries back onto the original image.

Using the src and dst coordinates used to create the Matrix $[M]$ `cv2.getPerspectiveTransform(src,dst)`, Now wil be used the same function used before , but with the values inverted to generate the a new Matrix M_{inv} `cv2.getPerspectiveTransform(dst,src)`. and after with the polynomial filled in the bird eye view, we use the function `cv2.warpPerspective()` to warp the information and merge to a undistorted image using the function `cv2.addWeighted(undist, 1, newwarp, 0.3, 0)`

Here's a link to more results (./output_images/warp_back) for lanes detected transformed to real images.

8. Output visual display of the lane and numericals estimation.

Using the funtion `cv2.putText()` it is possible add information in the image with variable information. Below 2 images showing the final result of the pipeline.

Here's a link to more results (./output_images/final_images) for final result images.

Pipeline (video)

To simulate the code with video, was created the file Advanced_Lane_Finding (./Advanced_Lane_Finding.ipynb).

All the function tested in the Helper (./Helper.ipynb) code was used, with only a difference in the polynomial detection curves. we start the video using the slide windows to detect pixel and after 3 frames, the code will use the search around function to take a count previous curves detected inside a margim of 100 pixels.

Below a quick overview regard the pipeline:

before run the `cell 11` from the jupyter notebook to generate the video, is necessarie execute the `cell 10` to reset the pickle file previous_info (./output_images)

1. Undistorsion function
2. Apply the perpective transform with
3. Convert to Binary
4. import a external dictionary to acess the length of the variable "in list" (numbers of times that the frame have run). If the number is less than 3, the function used will be slid windows. else the function must be search around.
5. Append the values from polynomial information.
6. Calculate the offset and vehicle position at the lane line.
7. Calculate vehicle turn radius and direction.
8. Draw the lines in the undistorted image.
9. Add texts with the information calculateds
10. Export currect values from polynomial detecteds and export as pickle file to be used in the next frame.
11. # Return image (frame)

Here's a link to my video result (./project_video_Edu.mp4)

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?

It was possible identify that the code does not calculate the radius correctly due some issues to detect the lane lines in some frames. I've tried work with a set of information and calculate the curves using a average