HC Writeup

Advanced Computer Vision - Lane Detection

Advanced Lane Finding Project

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.

- Calib_img and result

- Lane_img and result

- COLOR

- COMBINED

- RESULT

[]:

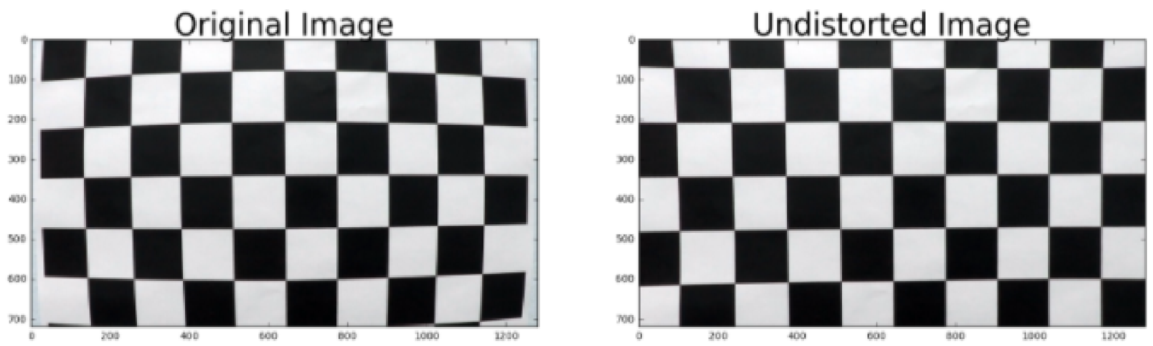
Writeup / README

The advanced computer vision techniques conducted with two main parts, Camera Calibration and Advanced Lane Finding.

Camera Calibration

1. Image distortion occur when a camera looks at 3D objects in the real world and transforms them into a 2D image. It causes object's appearance which can appear closer or farther away than actual. Distortion changes what the shape and the size of 3D objects appear to be. Due to image distortion.

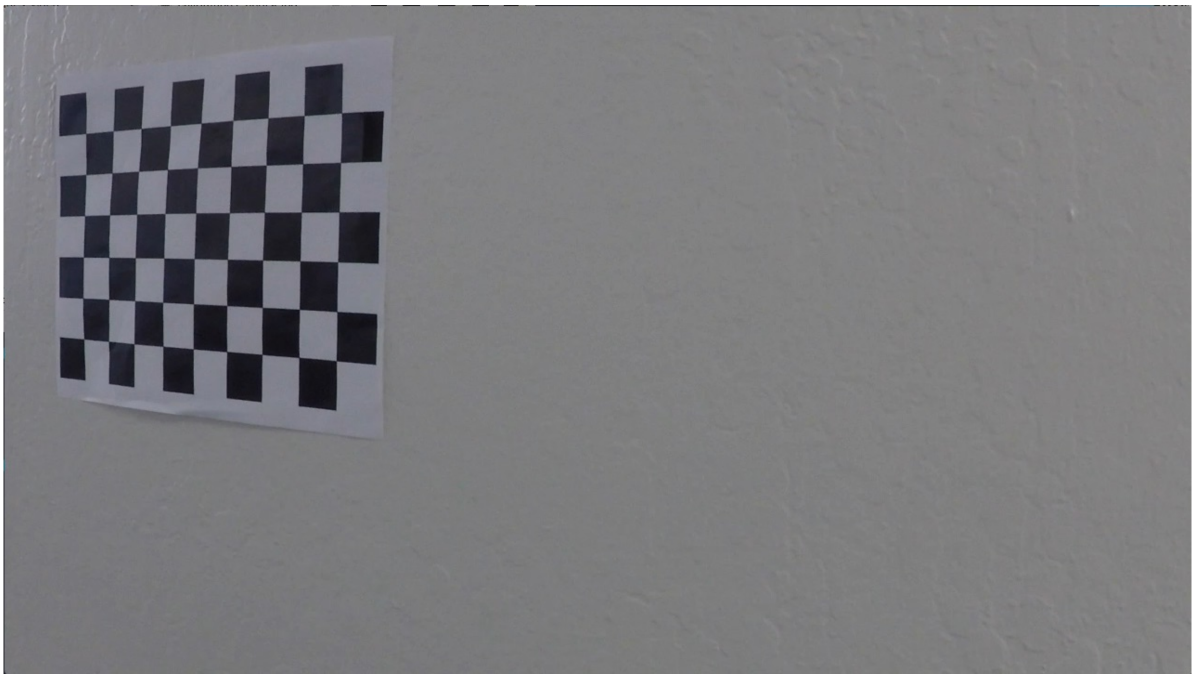
Goal of this step is to undistort images



To undo this distortion:

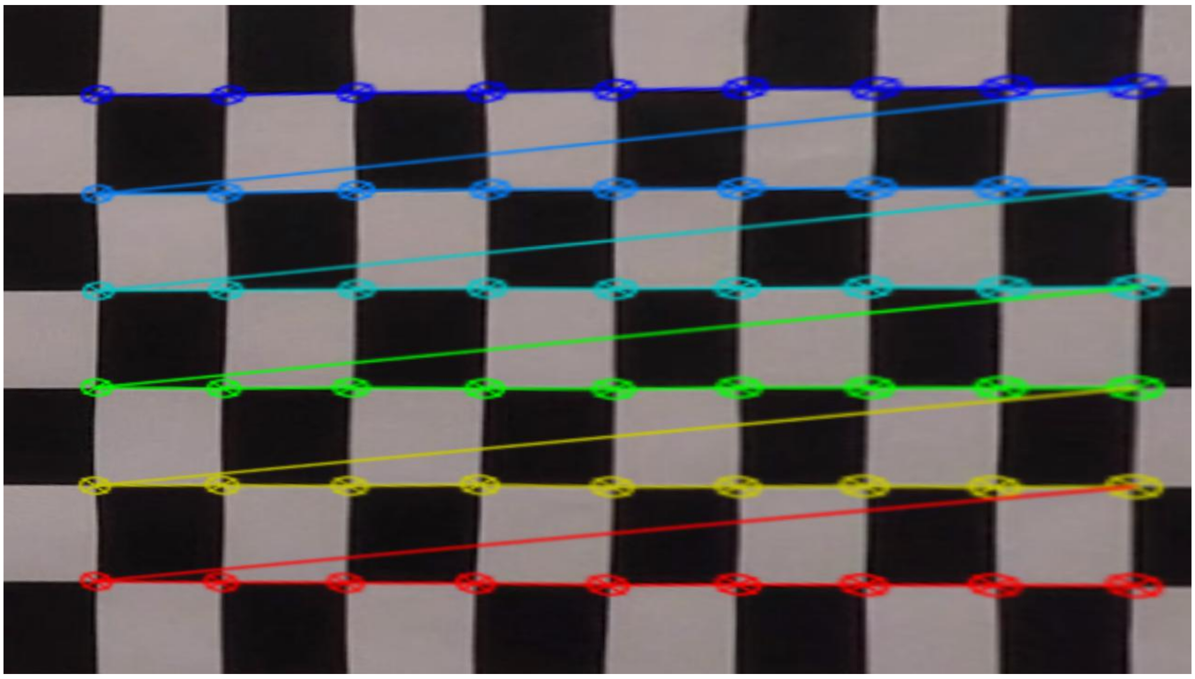
The code for this step is contained in the first code cell of the IPython notebook located in "./examples/video_Final_HC.ipynb".

In given image: calibration5



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` (3D point in real world space) will be appended with a copy of it every time I successfully detect all chessboard corners in a test image where using the `cv2.findChessboardCorners`. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output `objpoints` and `imgpoints` (2D points in image plane) to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function which returns the camera matrix, distortion coefficients, rotation and translation vectors. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



Rest of undistorted images are located in `/output_images/HC_camera_cal_output`.

Pipeline (single images)

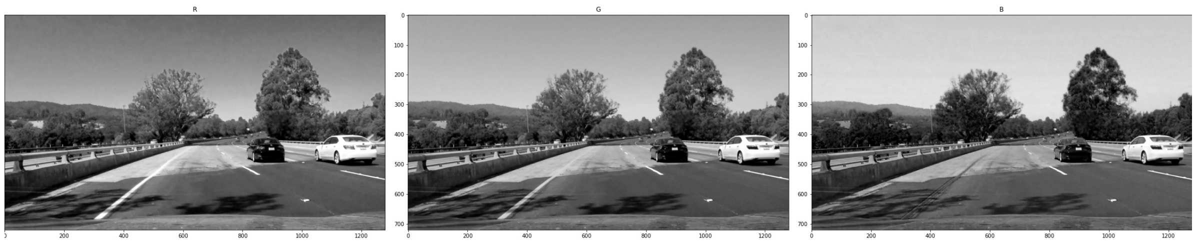
1. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

2. Color Space helps to picking up lane lines under different conditions such as light reflection, shade area and mostly eliminate noise. Not only color space but also applying Gradient does a cleaner job of picking u the lane lines.

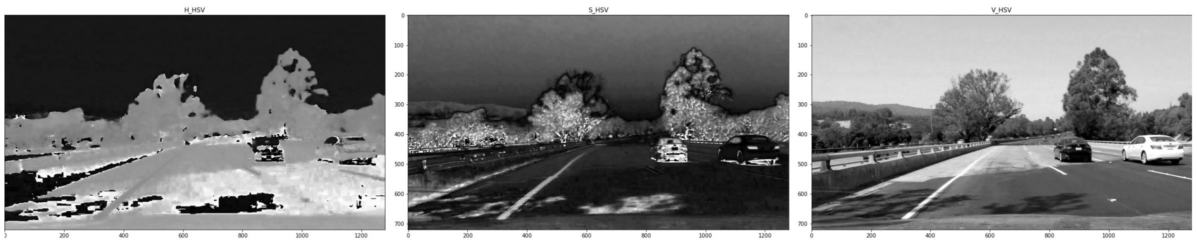
In Color Space, lane lines are either yellow or white. Therefore, we have to ascertain the values for the white and yellow lane colour. I used `cv2.cvtColor` to converts images from one color space to another

RGB



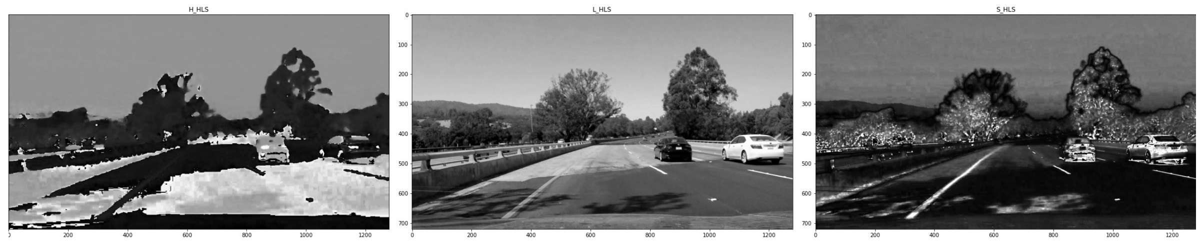
Due to RGB figures above, R color in RGB can picking up both white and yellow lane lines.

HSV



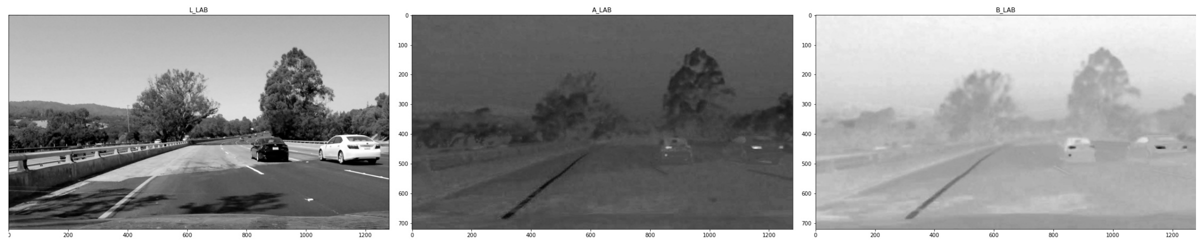
HSV color space is one of many color spaces that separate color from intensity such as YCbCr, LAB. HSV is analogous to shining a white light on a colored object such as shining a bright white light. Elements "Value" (V) refers to the lightness or darkness of a color. Therefore, I used V for value color. Unlike this example, in challenge video, there are lots of noise due to light reflection and shade area.

HSL



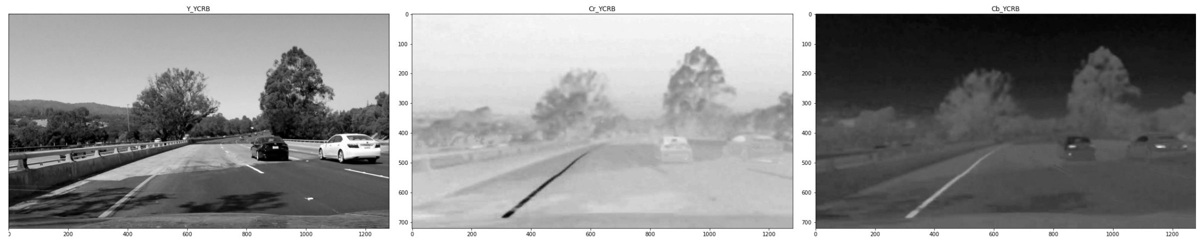
HLS color with maximum lightness in HLS is pure white. Change the hue to take a trip around the color wheel. Change the saturation to get deeper or more muted colors. Change the lightness to essentially mix in black or white. Saturation visually shows how shade parts are eliminated.

LAB



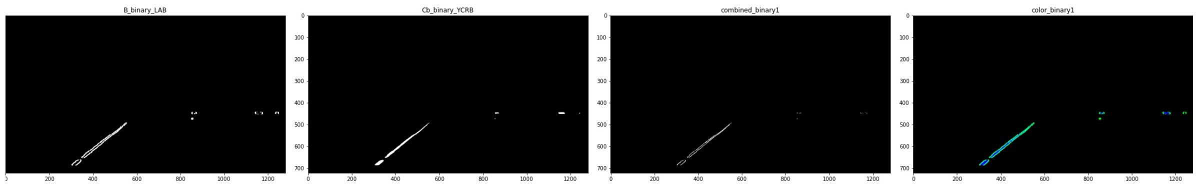
LAB color space corresponds with three components, like RGB does. L represent lightness, a is for red/green value, b is for blue/yellow value. As shown above in LAB figure, both a and b can picking up yellow lane lines clearly.

YCbCr



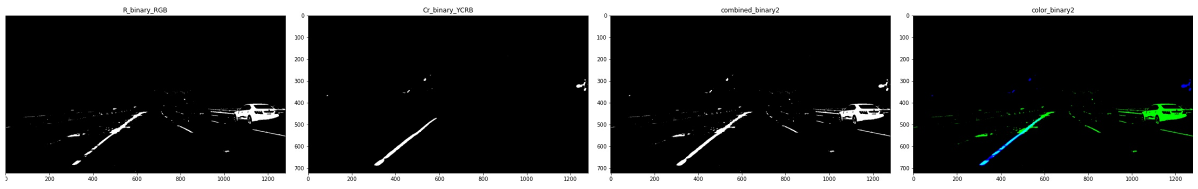
YCbCr is like HSV, but in different coordinates. YCbCr color space is used to detect and remove shadows from images. The approach based on YCbCr color space is proposed for detecting shadows.

Combined1



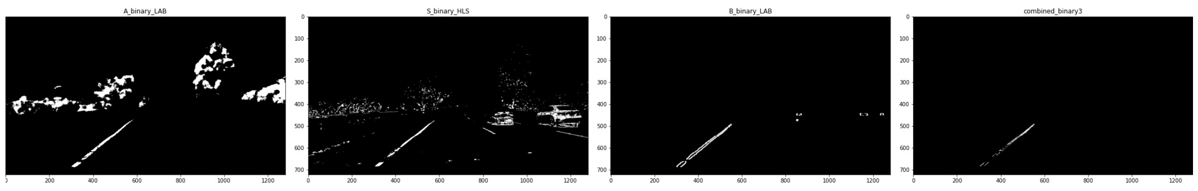
This figure in combined1, it is composed with Brighness of LAB and Cb of YCrCb. This combine process, used 'and' logic. Therefore, I can only collect right lane line only. Therefore, as shown in third picture, we only collected left white lane.

Combined2



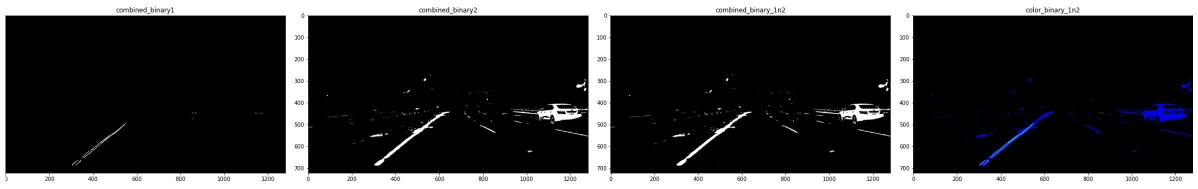
In combined2, it is composed with R of RGB and Cr of YCrCb. Since both of the color spaces that picked can picking up line clearly, I used 'or' logic to combine both. As shown in third and fourth picture, binary images shows better line image.

combined3



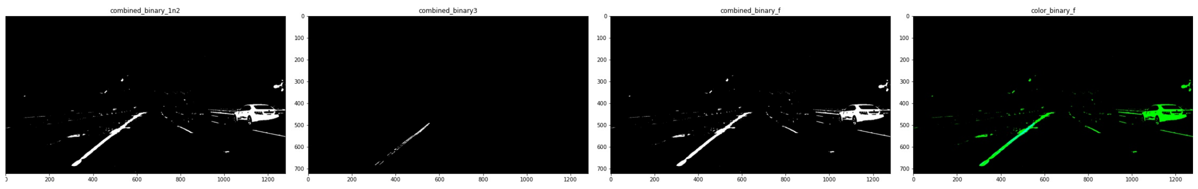
Combined3 combined with a and b of LAB and s of HLS. Under different circumstance such as shade area, or bight area, there are more noise in this combinations. In challenge problem, this step can picking up better lines.

Combined_1n2



Since combined binary 2 already have enough lane line, it already have done better job to picking up lane line

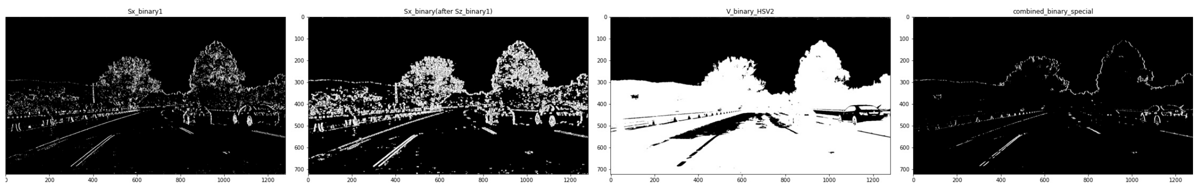
Combined_1n2 and combined 3



As well as combined with combined binary 3 and combined_1n2, since they are picked up enough lane line to keep necessary lane line it already have better lane line.

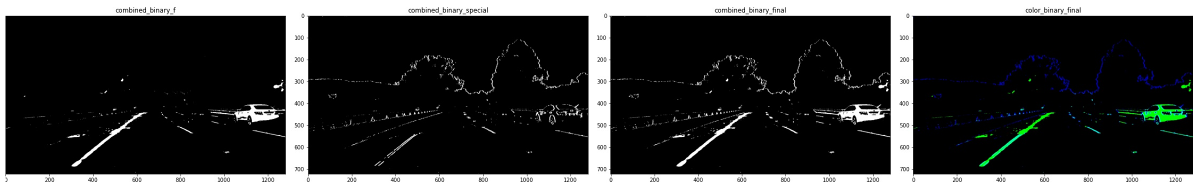
---

combined special (image_combined_Sx1_Sx_V_HSV2)

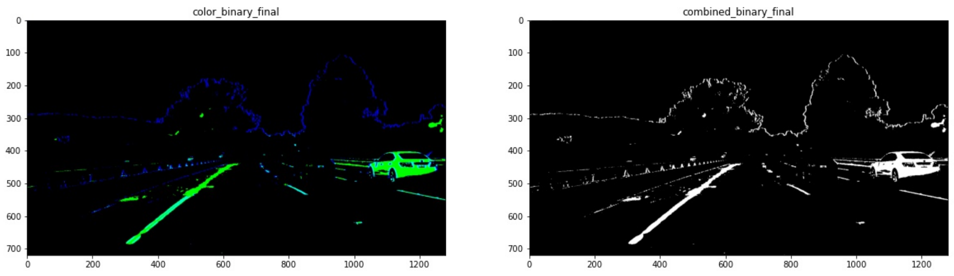


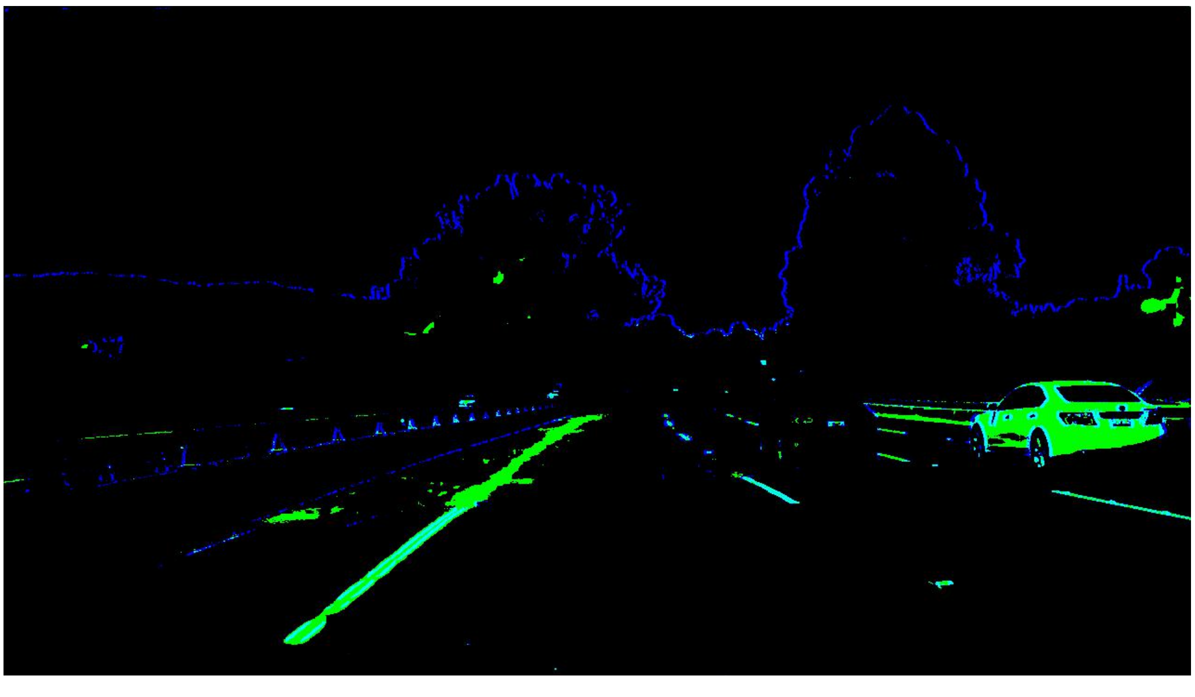
In this special combined with Gaussian on x axis on Sx_binary1 and then redo the Gaussian on Sx_binary1 which is Sx_binary. Sx_binary shown in second image in figure, it generate clear lane line. However, there are noise. To eliminate noise, used 'and' logic to not combined with V of HSV. This process can reinforce to pick up lane and eliminate noises.

Last combined



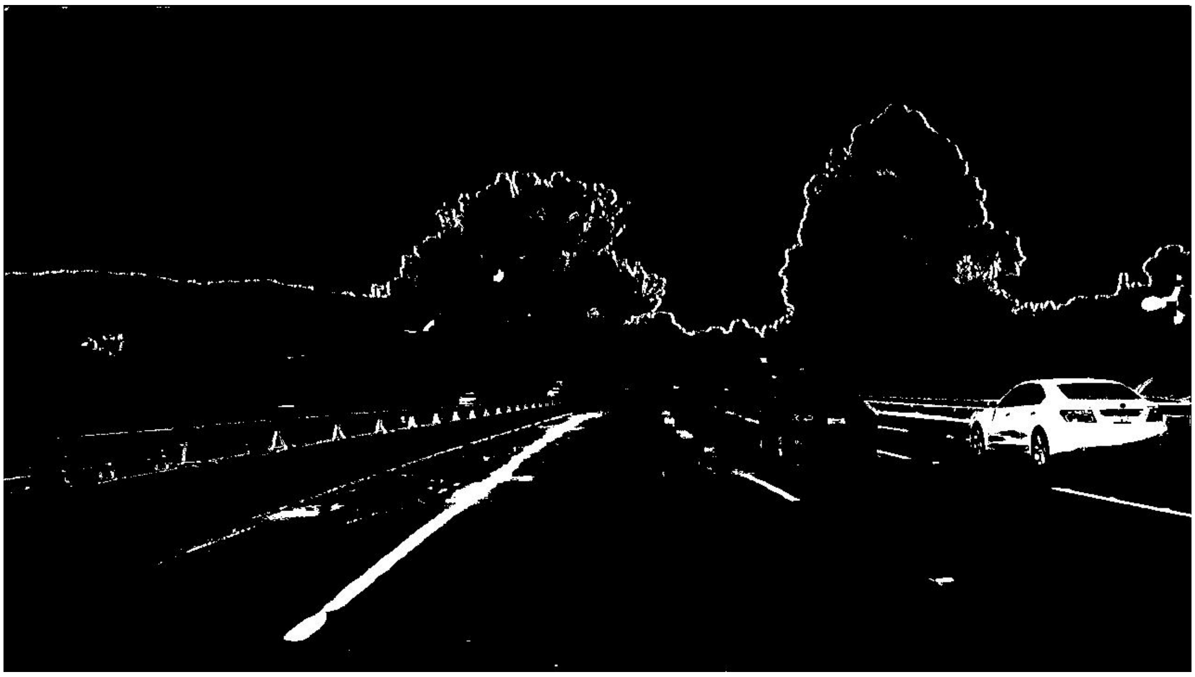
Final result for binary combination





Combination of color and gradient thresholds to generate a binary image indicates in this figure. Here in this example of my output for this step. This condition are necessary to collect two main lane lines to create curve lines which will be explain following step.

combined_binary_final



3. To performed a perspective transform and provide an example of a transformed image.

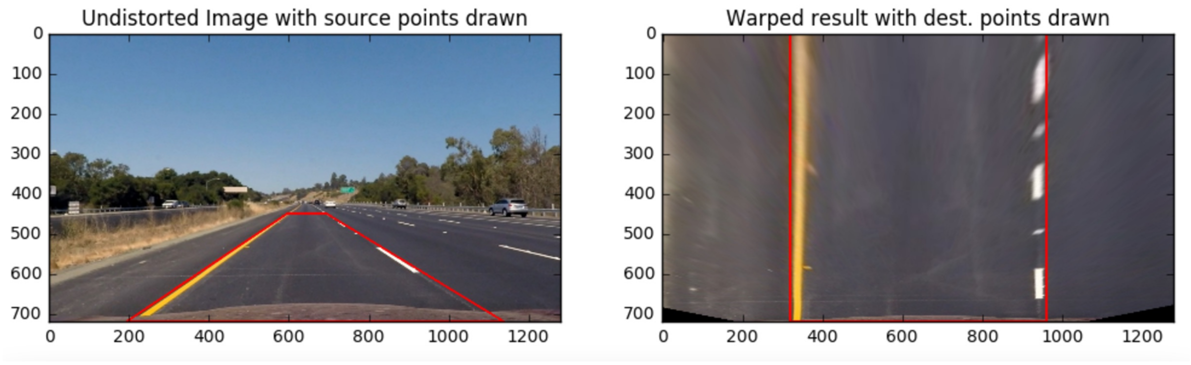
The code for my perspective transform includes in the function called `process_image()`, which appears in lines 604 through 614 in the file `video_final_HC.ipynb`. The `warped` variable and `newwarp` variable used for warp perspective. it takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

```
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) + 60, img_size[1]],
    [(img_size[0] / 2 + 55), img_size[1] / 2 + 100]])
dst = np.float32(
    [[(img_size[0] / 4), 0],
    [(img_size[0] / 4), img_size[1]],
    [(img_size[0] * 3 / 4), img_size[1]],
    [(img_size[0] * 3 / 4), 0]])
```

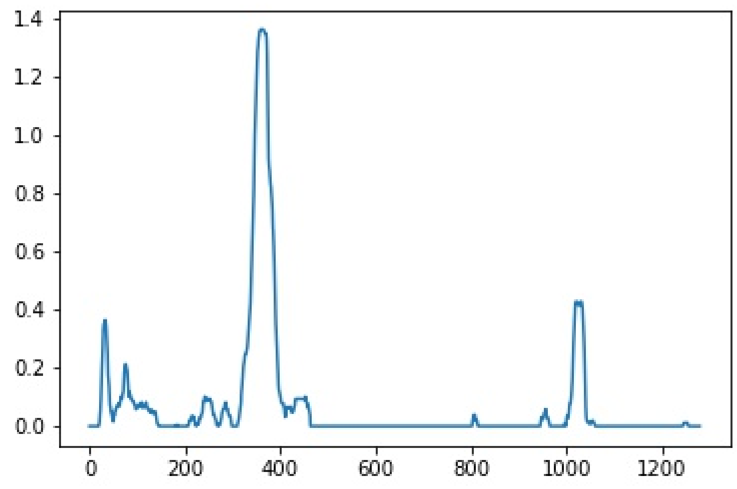
This resulted in the following source and destination points:

Source	Destination
585, 460	320, 0
203, 720	320, 720
1127, 720	960, 720
695, 460	960, 0

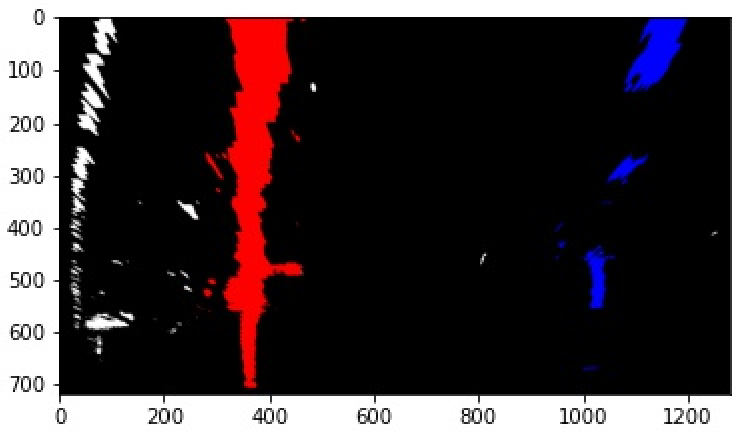
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

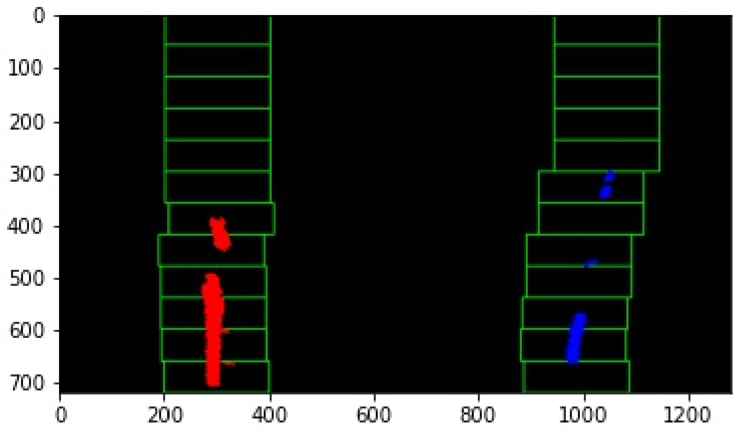

Undistorted Image with source points drawn
Warped result with dest. points drawn

4. To identified lane-line pixels and fit their positions with a polynomial.

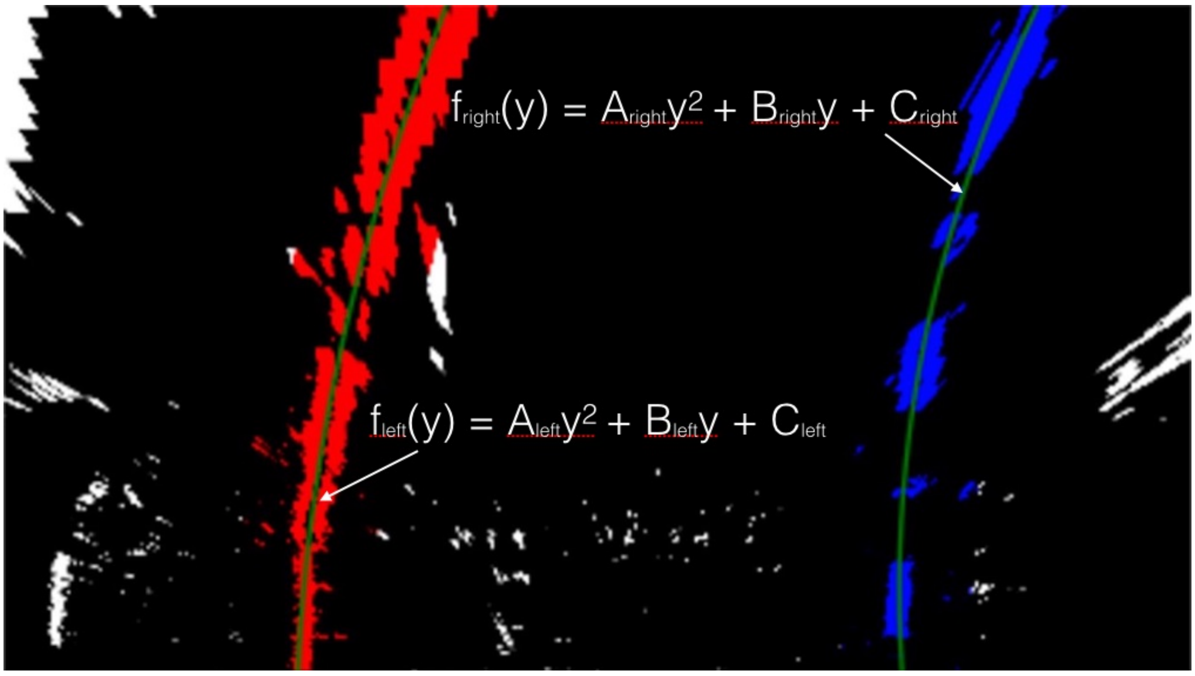


Histogram visually shows where beginning lane line have higher pixel values for both left and right base lines. Based on this base lines allow to calculate the offset for car position and center of the lane.





As shown sliding windows in figure above, each windows identify boundaries that have pixel value then extracted left and right line pixel positions. Function `find_lane_pixels()`


$f_{right}(y) = A_{right}y^2 + B_{right}y + C_{right}$
$f_{left}(y) = A_{left}y^2 + B_{left}y + C_{left}$

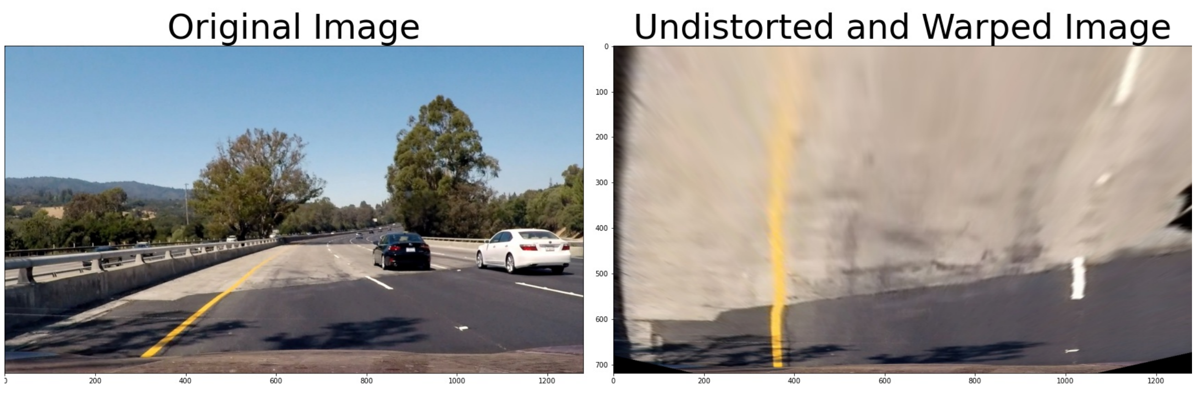
Then I did `numpy.polyfit()` to fit a second order polynomial to each which for left lane line and right lane line. Therefore, it will be located the lane line pixels, used their x and y pixel positions to fit a second order polynomial curve.

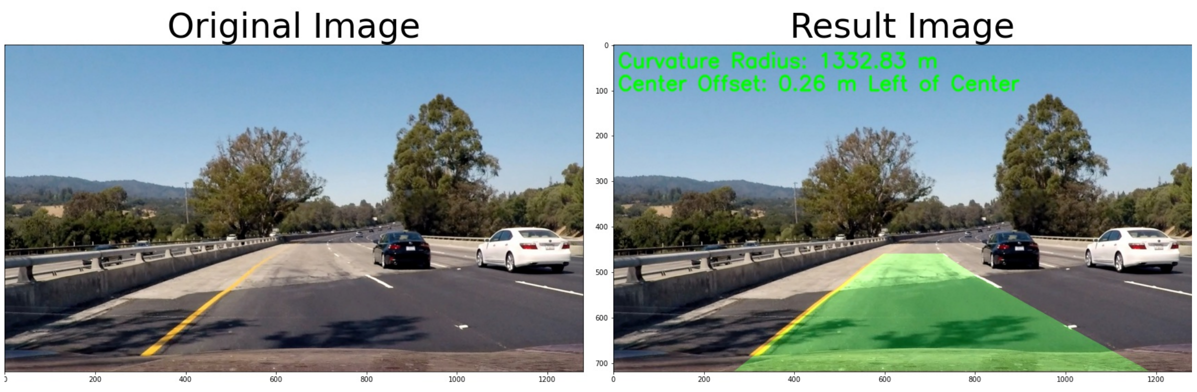
5. To calculated the radius of curvature of the lane and the position of the vehicle with respect to center, used `measure_curvature_real()` function. [Support Web Site for finding Radius of Curvature]

Since calculating the radius of curvature based on pixel values, so the radius we are reporting is in pixel space, which is not the same as real world space. Therefore, apply conversions in x and y from pixels space to meters.

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

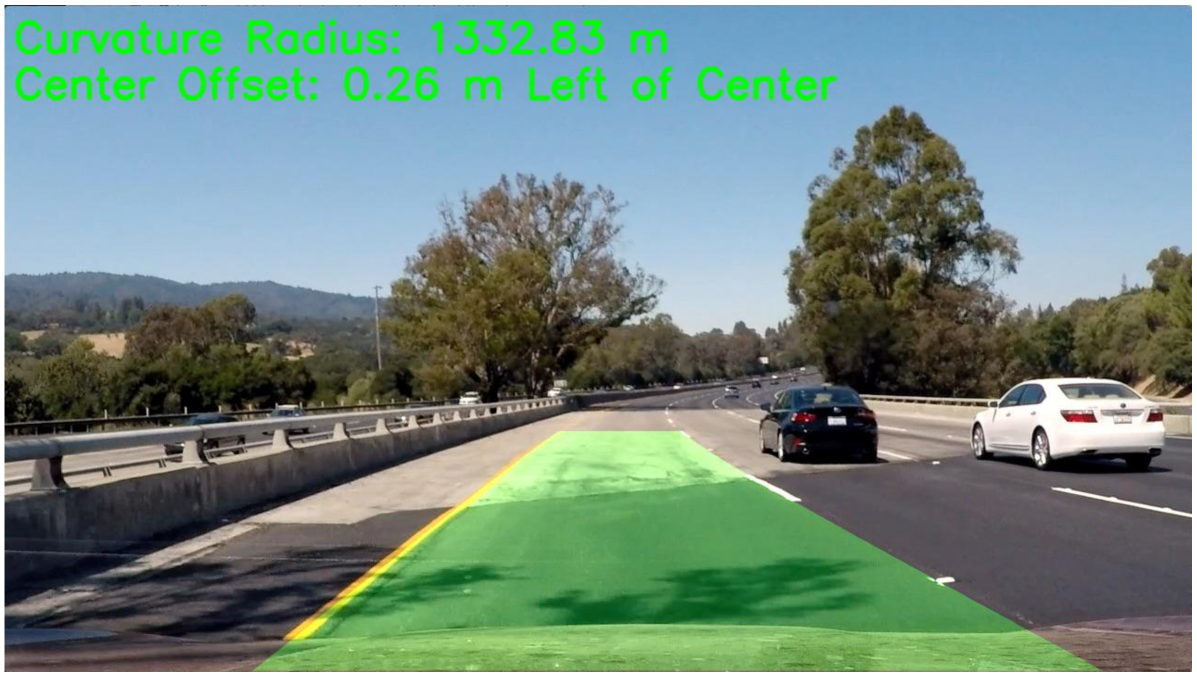
To approach real world space: Parabola is `x= a*(y**2) +b*y+c`. Let `mx` and `my` are the scale for the x and y axis, repectively (in meters/pixels) then the caled parabola is `x= mx / (my ** 2) *a*(y**2)+(mx/my)*b*y+c`.




Original Image
Result Image
Curvature Radius: 1332.83 m
Center Offset: 0.26 m Left of Center

6. Final image for the project

To combined warped image, radius and offset it merged in the function `process_image_end()`.


Curvature Radius: 1332.83 m
Center Offset: 0.26 m Left of Center

Pipeline (video)

1. Link below proves final video output

Here's a [link to my video result]

Discussion

1. Through this pipeline, `project_video.mp4` have no error appeared for collecting lane lines. However, in both `challenge_video.mp4` and `harder_challenge_video.mp4` through error which unable to append lane line. To improve those two challenge video, I had to try different color space to reinforce picking up lane line. Also selecting boundaries of the windows can improve collecting left and right lines.