Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

Rubric Points

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

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Camera Calibration

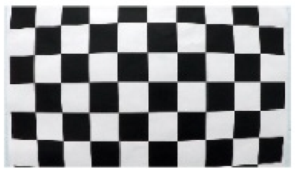
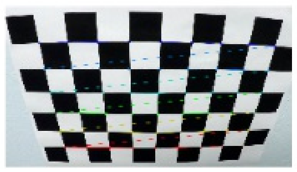
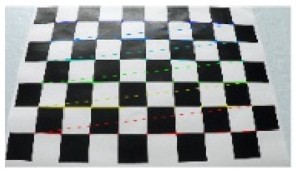
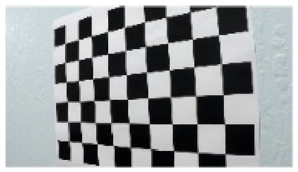
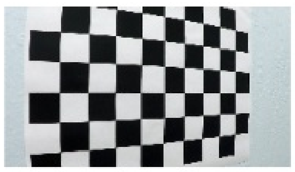
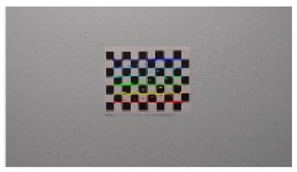
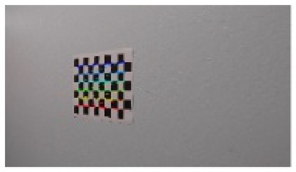
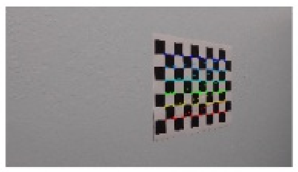
1. This section is briefly state how I computed the camera matrix and distortion coefficients. （Code in cell-#1)

The code for this step is contained in the cell-#1 of the IPython notebook located in "advanced*lane*pipeline.ipynb" .

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 successful chessboard detection.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function.

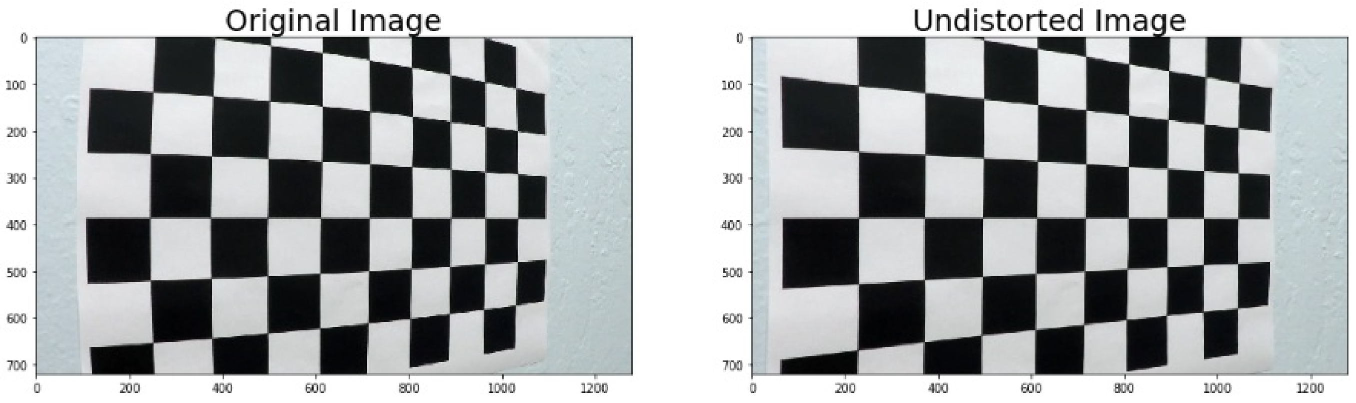
Camera calibration

Pipeline (single images)

1. Provide an example of a distortion-corrected image. (Code in Cell-#2)

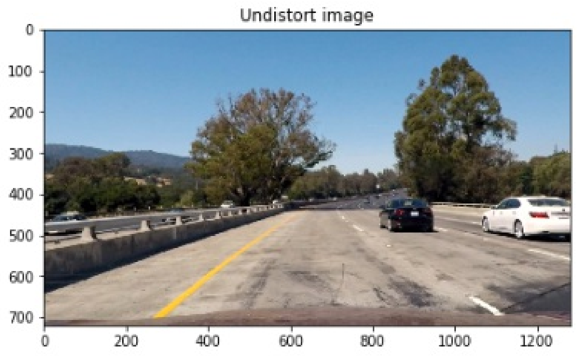
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one: (Because the real road image is difficult to show the distortion of image, so I have to use the distorted chessboard image here to show the function works!)

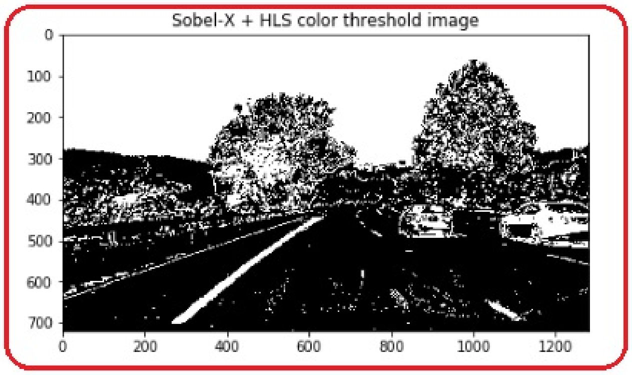


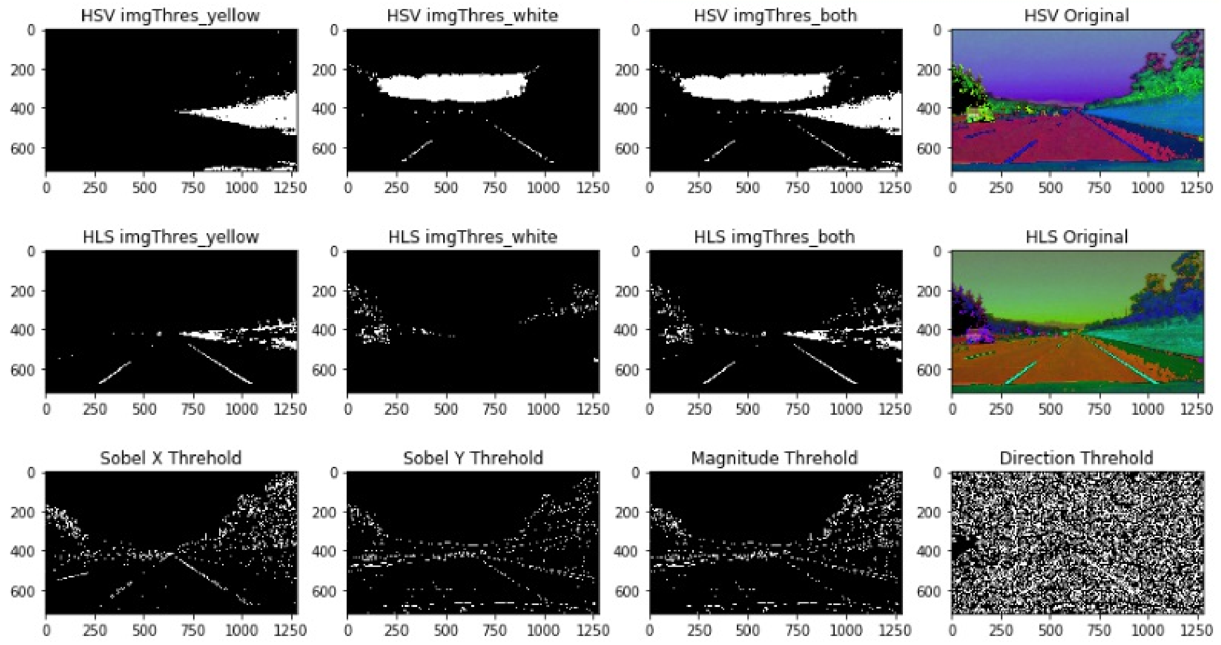
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result. (Code in Cell-5)

I used a combination of color and gradient thresholds to generate a binary image . After compared with the Sobel-X/Y, magnitude, direction in HSV and HSL color space, we can found that the "Sobel-X + HSL" will track the lane lines more better.

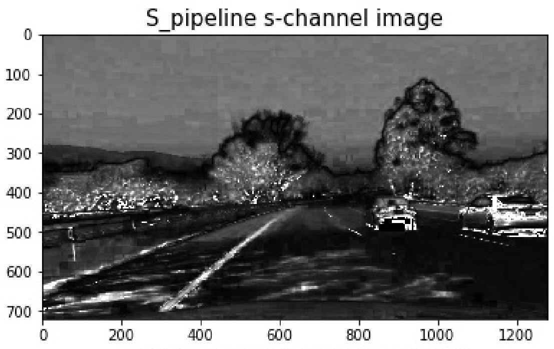
Color Gradient Threshold Comparison

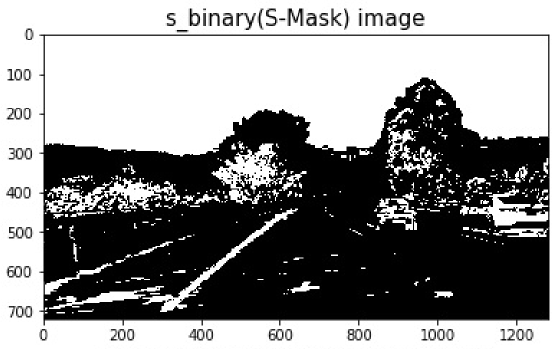


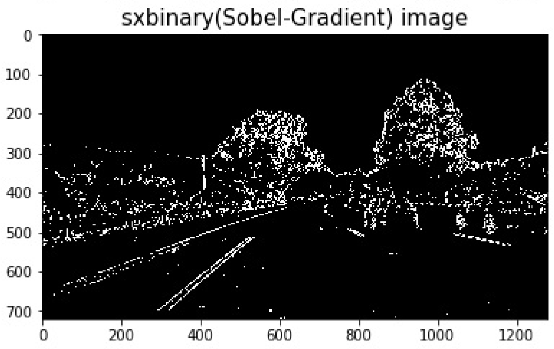


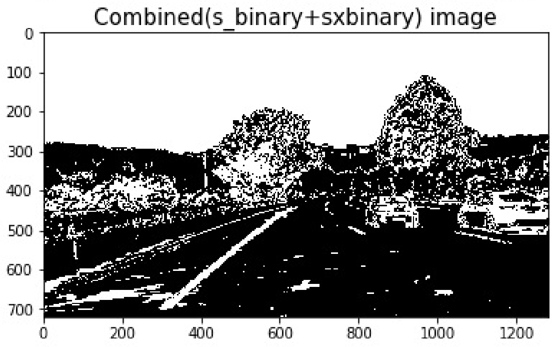


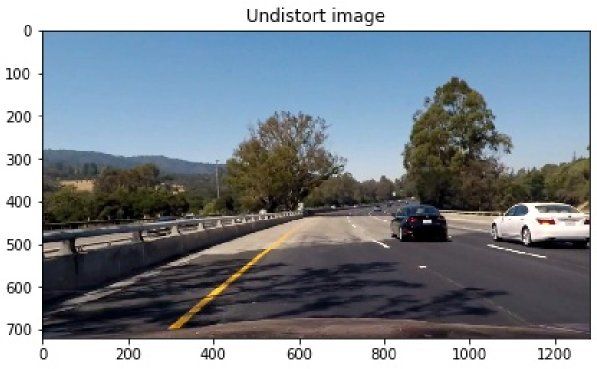
But for some lanes with shadow, the above method will failed to filter these noise like below: HLS+SobelX fail to filter shadow nosize

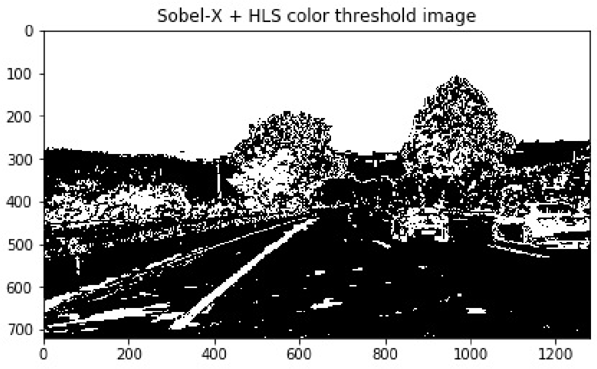




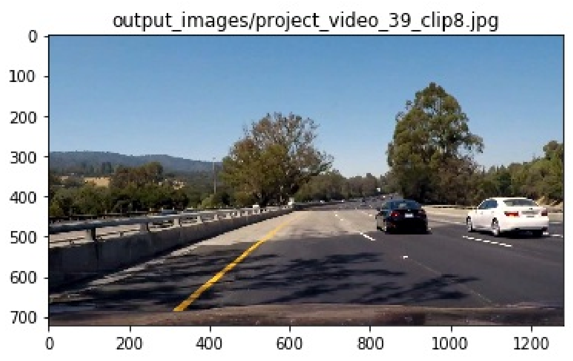


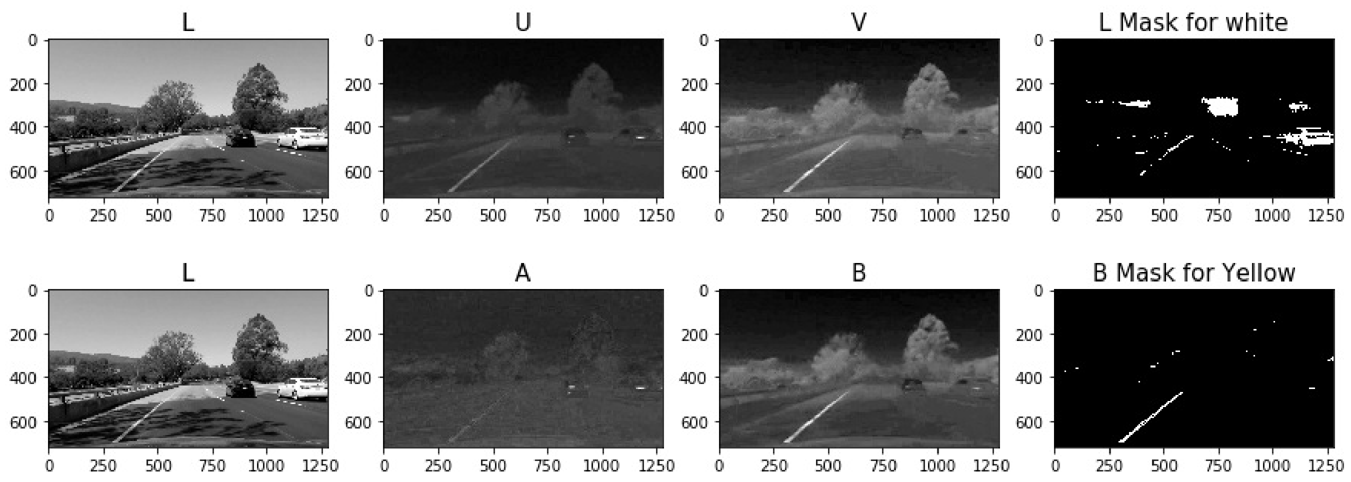


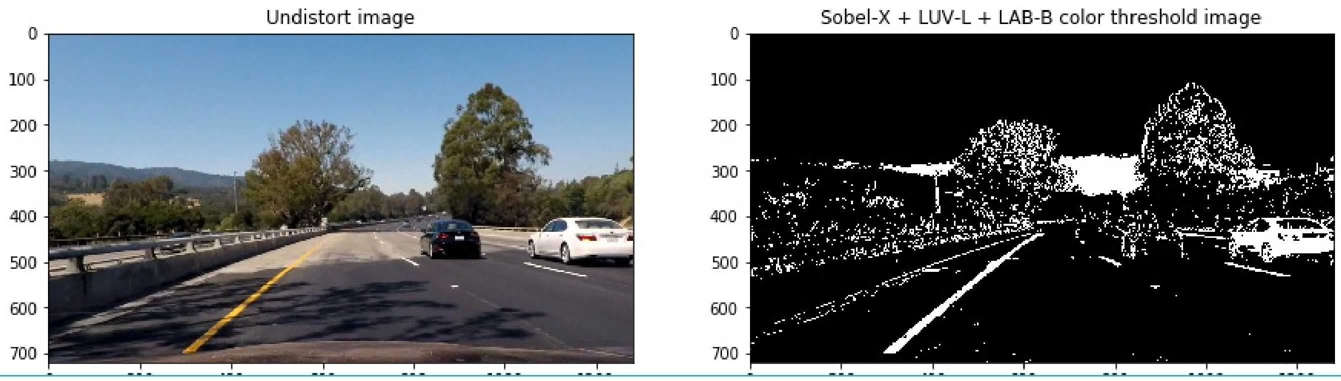




L channel of LUV for white lane lines, and b channel of Lab for yellow lane lines. (Code in L_pipeline() in cell-#5)







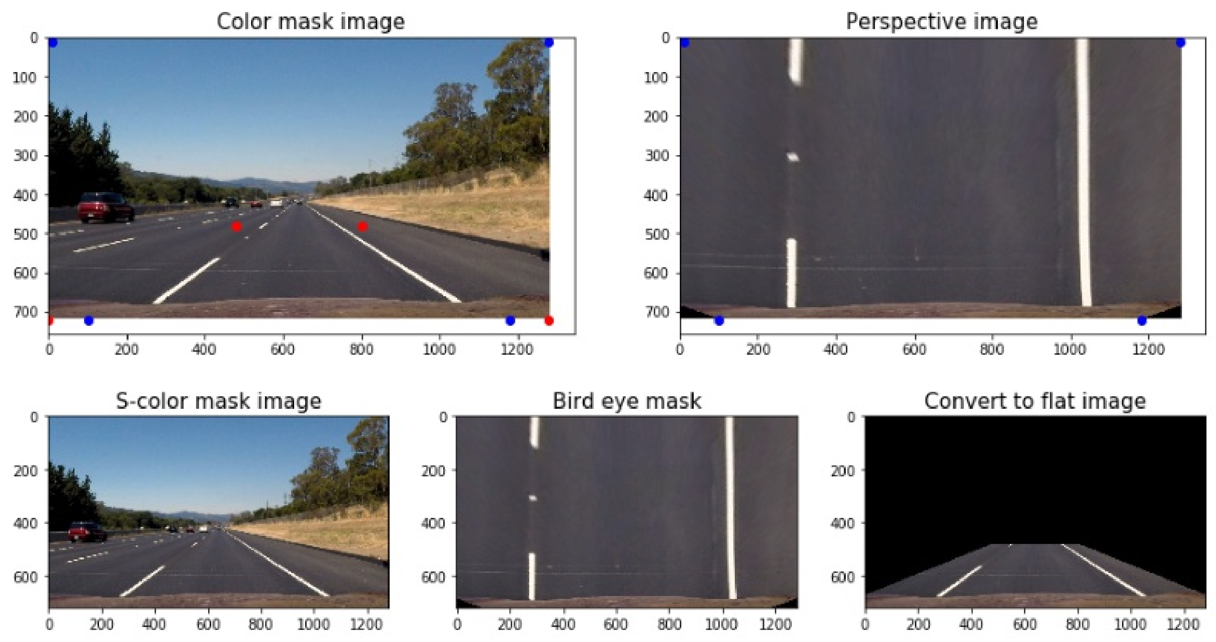
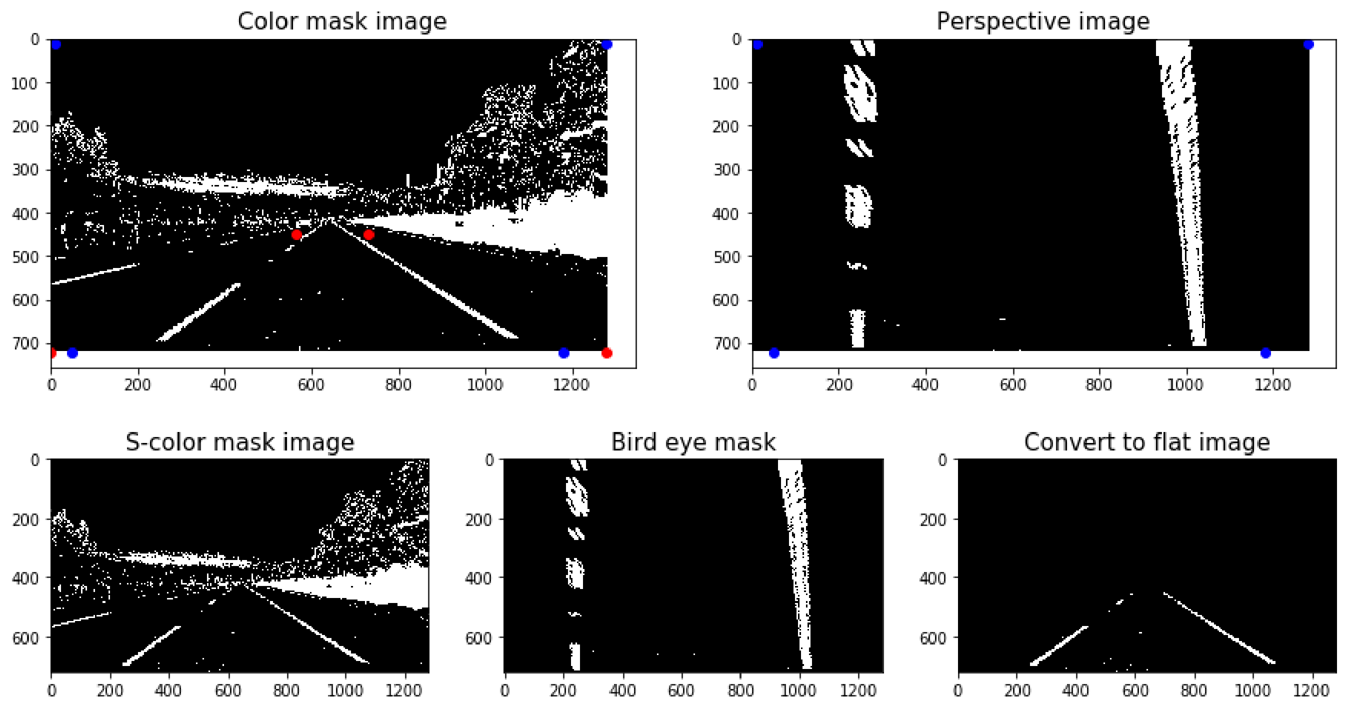
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image. (Code in Cell-#7)

The code for my perspective transform includes a function called `calc_perspective_transform()`(in cell-#?). This function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points which will be calculated in it.. I chose the hardcode the source and destination points in the following manner:

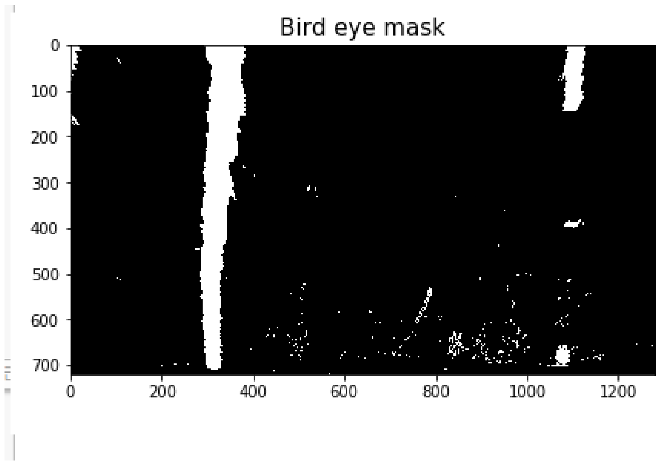
Below code calculate the perspective transform

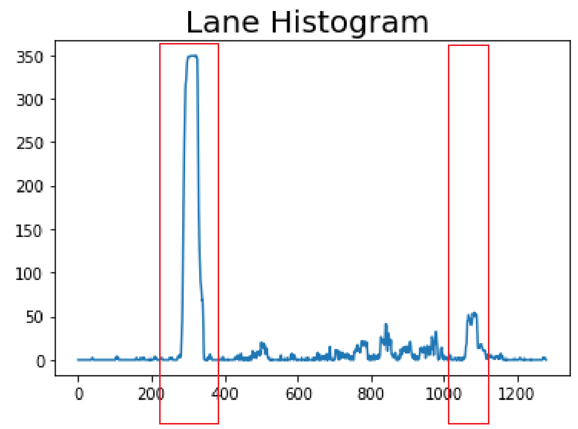
``` img*size = img.shape[1::-1]*
*ht*window = np.uint(img*size[1]/1.6) # 720/1.6=450 hb*window = np.uint(img*size[1]) c*window = np.uint(img_size[0]/2) # 1280/2 = 640

```
ctl_window = c_window - .12*np.uint(img_size[0]/2) # Adjust the trapezium to match the curved lane more.
ctr_window = c_window + .14*np.uint(img_size[0]/2)
cbl_window = c_window - 1*np.uint(img_size[0]/2)
cbr_window = c_window + 1*np.uint(img_size[0]/2)

#[bottom_left, bottom_right, top_right, top_left]
src = np.float32([[cbl_window, hb_window], [cbr_window, hb_window], [ctr_window, ht_window], [ctl_window,ht_window]]) # ok for hoo
dst = np.float32([[0+50,img_size[1]],[img_size[0]-100,img_size[1]],[img_size[0],10], [10,10]])
```
```

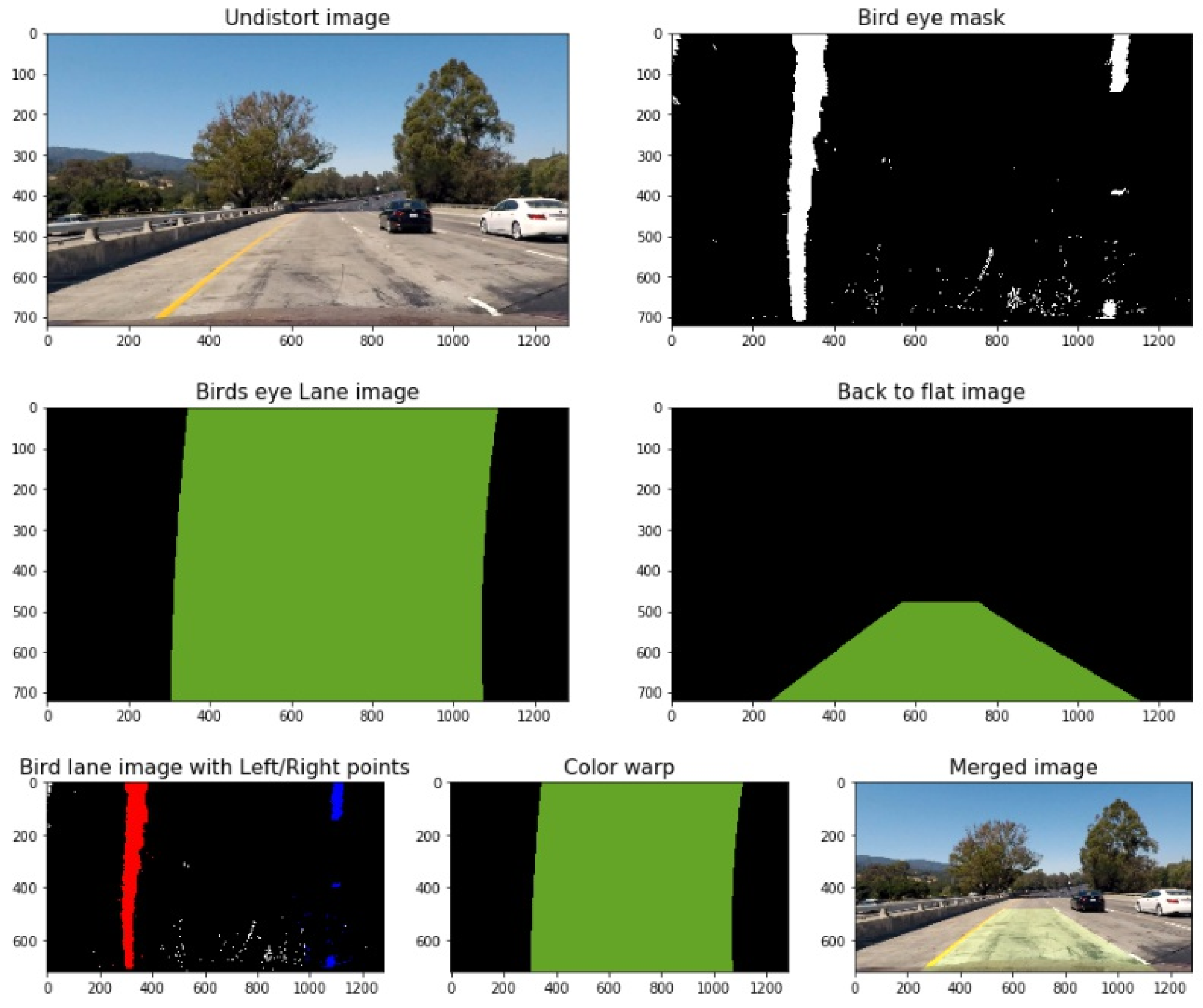
```

This resulted in the following source and destination points:

| Source | Destination |
|:-------------:|:-------------:|
| 0, 720 | 50, 720 |
| 1280, 720 | 1180, 720 |
| 730, 450 | 1280, 10 |
| 560, 450 | 10, 10 |

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.

Perspective Transform Image (Red point is the 'src' points, blue points is the 'dst')



---

Perspective Transform Mask



**4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial? (Code in Cell-#8)**

Sum across image pixels vertically
```

In the previous combined mask, we can see many white points which some of the lane lines, but others are not. we can compute the sum across image pixels vertically to decide explicitly which pixels are part of the lines and which belong to the left line and which belong to the right line.



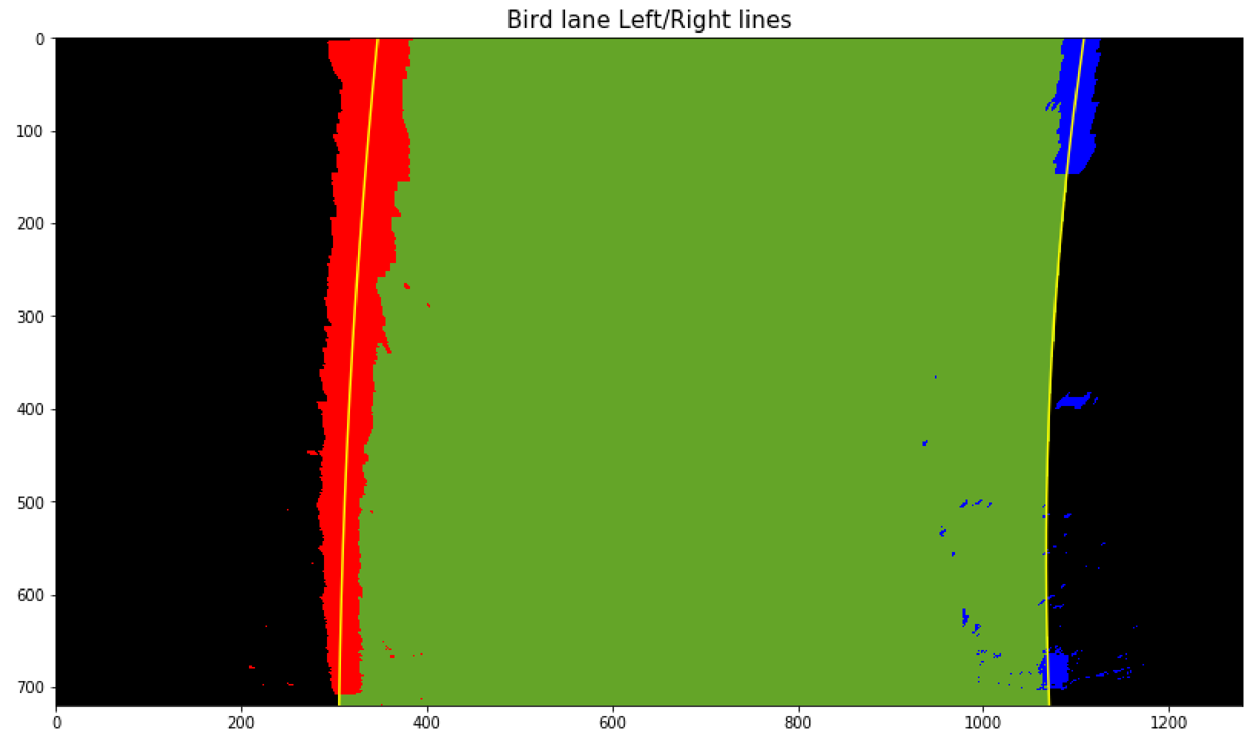


Then I did some other stuff and fit my lane lines with a two order polynomial kinda like this:



5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center. (Code in Cell-#11)

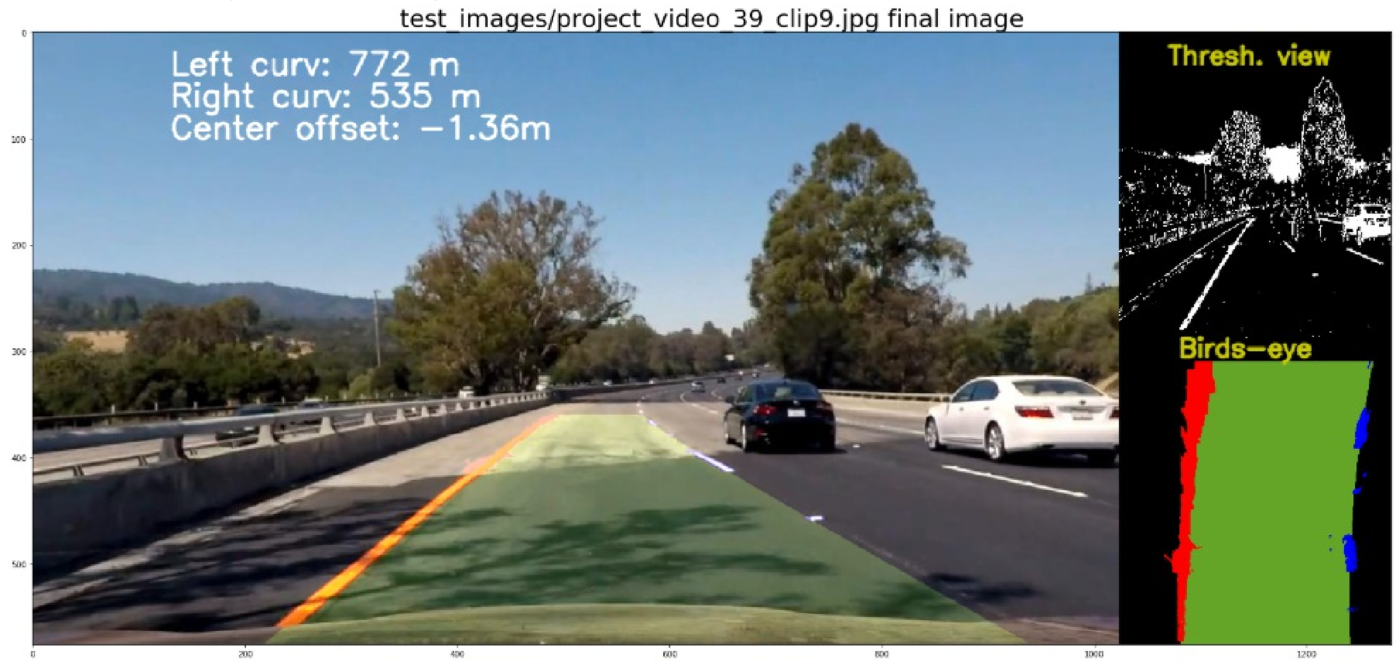
I did this in measure*curvature*real() (Cell-#11)



6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in process_image() in Cell-#12.

Here is an example of my result on a test image:



Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Here's a link to my video result

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?

1. Many thanks for the tutor to given the suggestions such as in different color threshold, and the point out the problem when I calculate the radius of the curvature of the lane lines.

2. I have one question about the perspective transform. Now I have to hardcoded the source points and dest points, if I choose one group of points which suit for straight lines, but it's has some deviation for curved lines, and vice versa. Is there any general way to calculate the src and dst points?