Writeup Template

1. Camera Calibration

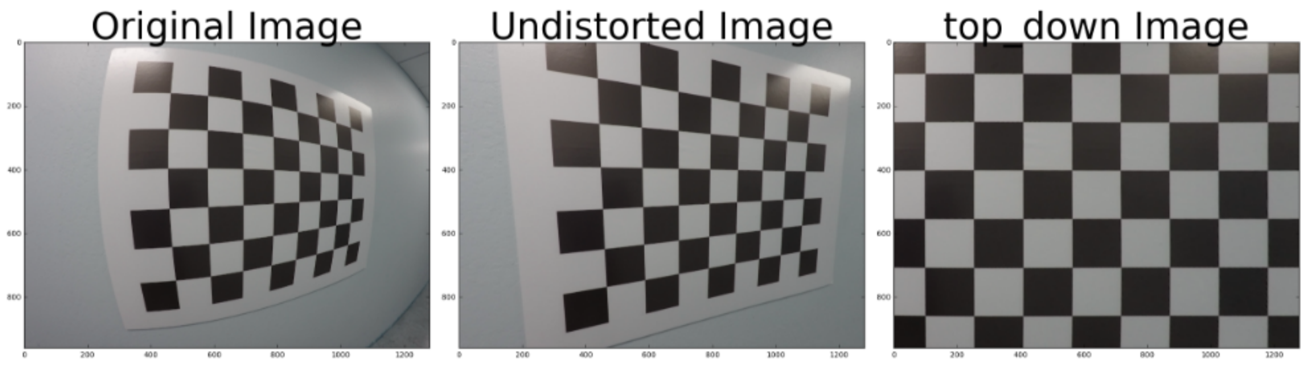
Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first step of the IPython notebook located in Project_advanced_lalne_fingding.ipynb

(1) I use the cv2.findChessboardCorners() function to find the chessboard corners.

(2) I then use the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I apply this distortion correction to the test image using the cv2.undistort() function and obtain the second picture.

(3) At last, I use cv2.getPerspectiveTransform and cv2.warpPerspective to obtain the third picture.



2. Pipeline

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

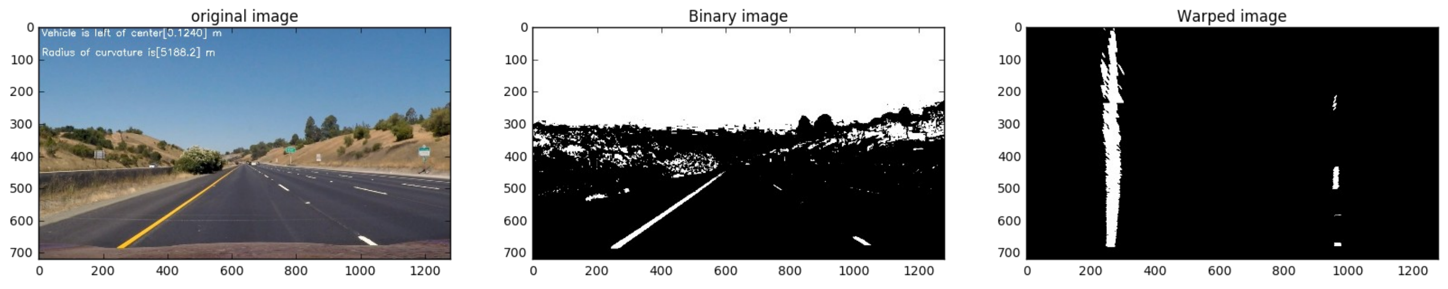
Using undistortion matrix obtained at the calibration step, I execute cv2.undistort function to undo the distortion. You can see result below:

(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.

I've observed that detection works better on S channel of HLS on some set of images. So I used OR combination to get both. I also noticed that some false positives can be masked by threshold on the S of HLS, so I applied that step as well.

```
# Define a function that thresholds the S-channel of HLS

def hls_select(img, thresh=(0, 255)):

        hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)

        s_channel = hls[:,:,2]

        binary_output = np.zeros_like(s_channel)

        binary_output[(s_channel > thresh[0]) & (s_channel <= thresh[1])] = 1

        return binary_output
```

Here is what result looks like:

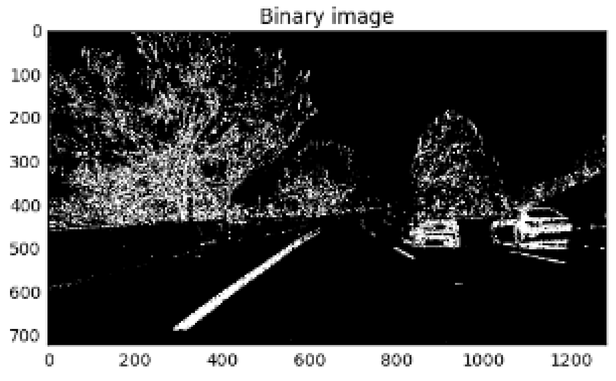

original image
Vehicle is left of center[0.1240] m
Radius of curvature is[5188.2] m
Binary image
Warped image

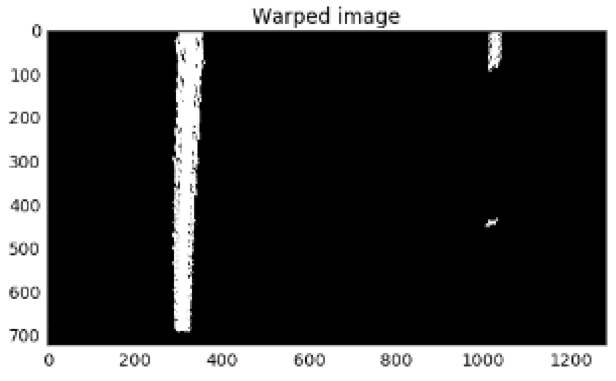
(4) Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for this step is contained in the three part located in Project_advanced_lalne_fingding.ipynb

Here is how source and destination points are defined:

Source	Destination
250,680	250,680
1050,680	970,680
700,450	1070,0
600,450	300,0





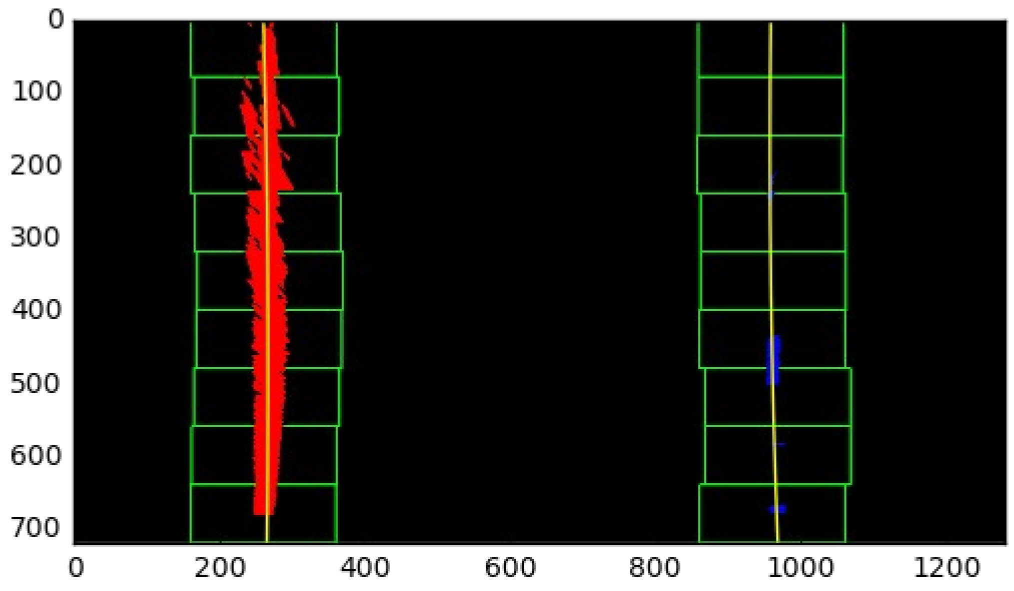
(5) Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

To identify lane line pixels in the perspective transformed mask, I initially use hist_search() at line 434 of utils.py. hist_search() uses a sliding window histogram of the pixel values to select pixels around the two peaks (one for each line).

- Then I create empty lists to receive left and right lane pixel indices
- Step through the windows one by one and concatenate the arrays of indices
- I extract left and right line pixel positions, then fit a second order polynomial to each
- Generate x and y values for plotting

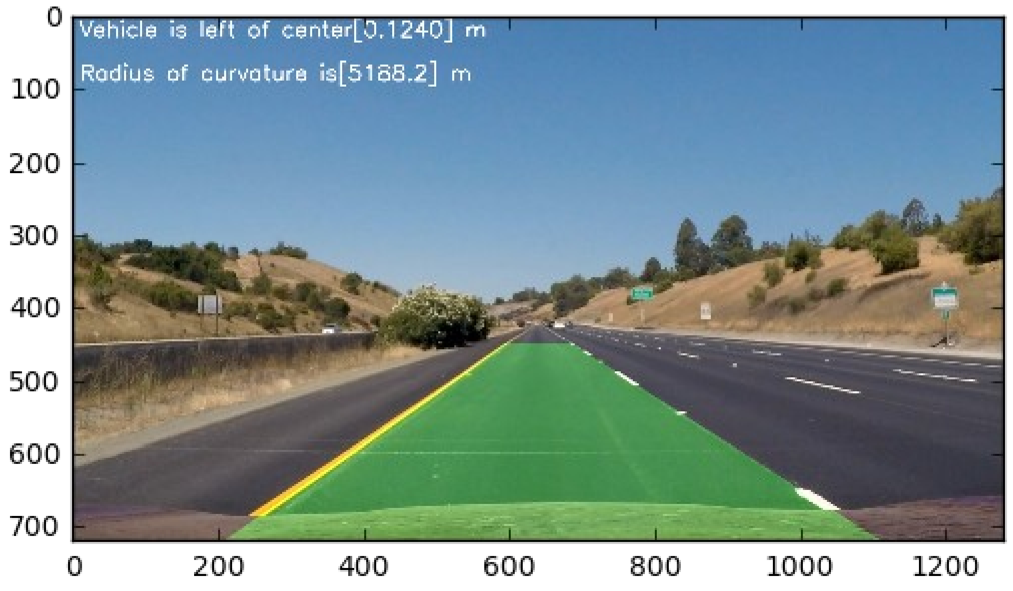
(5) Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I calculate this by measuring the distance from the center of the frame to the bottom of the left and right lines and taking the difference of those. In my code, negative distance represents that the car is to the left of center, while positive distance represents to the right.



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

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


Vehicle is left of center[0.1240] m
Radius of curvature is[5188.2] m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

3. Pipeline(video)

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!).

4. Discussion

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?

What I have learned from this project is that it is relatively easy to finetune a software pipeline to work well for consistent road and weather conditions, but what is challenging is finding a single combination which produces the same quality result in any condition. My model is easily changed by varying lighting and weather conditions, road quality, faded lane lines.