Project 2 Advanced Lane Finding

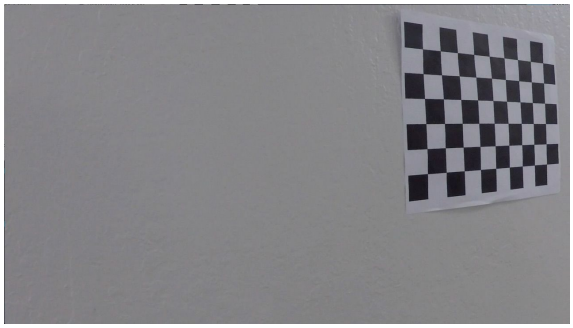
Camera Calibration

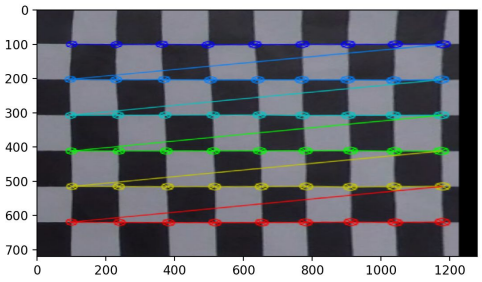
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

Two steps for this item

1. Use the calibrate_camera function in camera_calibration.py to calibrate the camera. In this function, we can get the camera_calibration.mtx, camera_calibration.dist. Which can be used to undistort the images. I load all the images from calibrate folder to get the above values. And use perspective transform function from cv2 to process the input image

2. Undistort the images, use cv2.undistort with the values we get from step 1 to undistort any given image





Pipeline (single images)

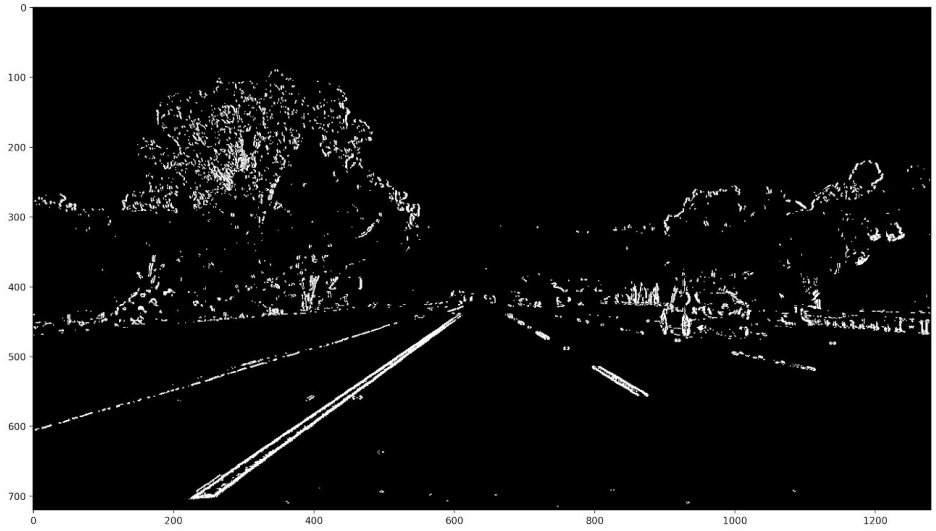
1. Provide an example of a distortion-corrected image.

After we get the parameters of the camera, we use the smae undisort and perspective transform function to convert live captured images

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 used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines # through # in another_file.py). Here's an example of my output for this step. (note: this is not actually from one of the test images)

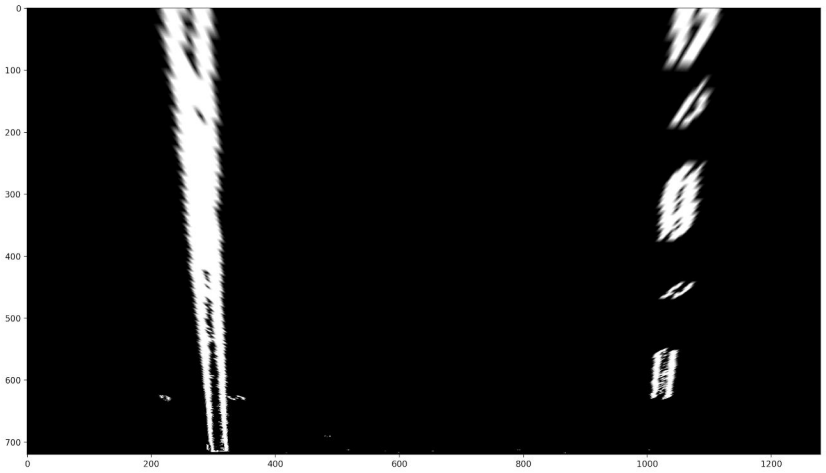
The code of this step is in gradint_thresh_combo. (detailed parameter value can be found in this function)

1. Calculate the absolute value of sobel_threhold in x direction, and tune min and max
2. Calculate the magnitude of the gradient tune kernel number and threshold
3. Calculate the director of the gradient
4. Use S channel of HSL
5. Use the thresholds above to filter out points on the original image



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

1. tune/ calculate the src and dst points. The src area is a Trapezoid included the two lines tightly and the dst area is a rectangle
2. Use the perspective transform function and the values we get from camera calibrate



4. Describe how (and identify where in your code) you identified lane-line pixels and fit

their positions with a polynomial?

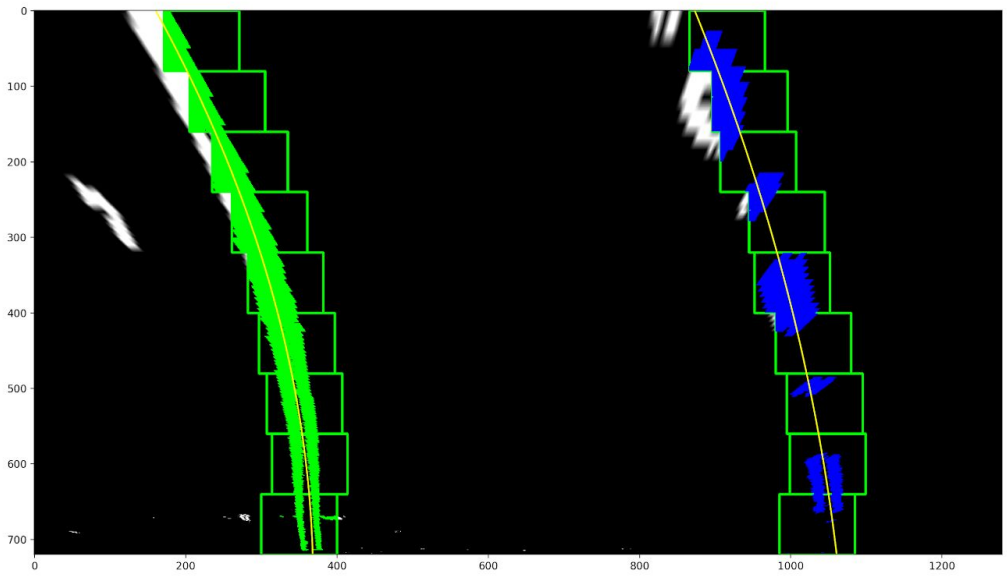
Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:
The main routine is in poly_lane_fit.fit_polynomial
There are two subroutines in this step:
find_line_init

1. Use the histogram to find out two peaks in the chart as the start point
2. From start point search around to find points in a rectangle
3. Use the previous rectangle middle point's x and the width to find out the next rectangle, mark more points until we find all the points in the picture from bottom to top
4. Polyfit the left and right set of points

Search_around_poly

1. After we get the parameter the previous 2-degree function. We can search in a range.
2. Select points in the previous' range and do the polyfit to these points to get new polyfit function
3. If there are too less points we found using previous' value. We redo the above method

For the right line and left line record the last 10 polyfit result as the previous result.



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.

Function get_car_offset is used to get offset

1. Find a y value at the very bottom of the view
2. Calculate the x_leftmost and x_rightmost values using y and the two 2-degree poly fit function for left and right lines
3. Pixel_offset = (x_rightmost - x_leftmost) - img_center
4. Convert from pixel to meter

Function measure_curvature_real is to get the radius of the lane. Mainly is use the bottom y value to calculate the curvature via formula we get from class

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.
This demo image is generated after processing image. In main.py 132-135

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

https://www.youtube.com/watch?v=kfHGw_mg9NQ

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?

There are several parts can be improved:
1. It is hard to filter out some point from the image. Maybe we can use convolution to help to filter out noisy point
2. Sometimes there is only one line can be detected in reality such as when we turn sharply. If we can’t find the two lines, we need to follow the remaining lines.