Akshay Khole

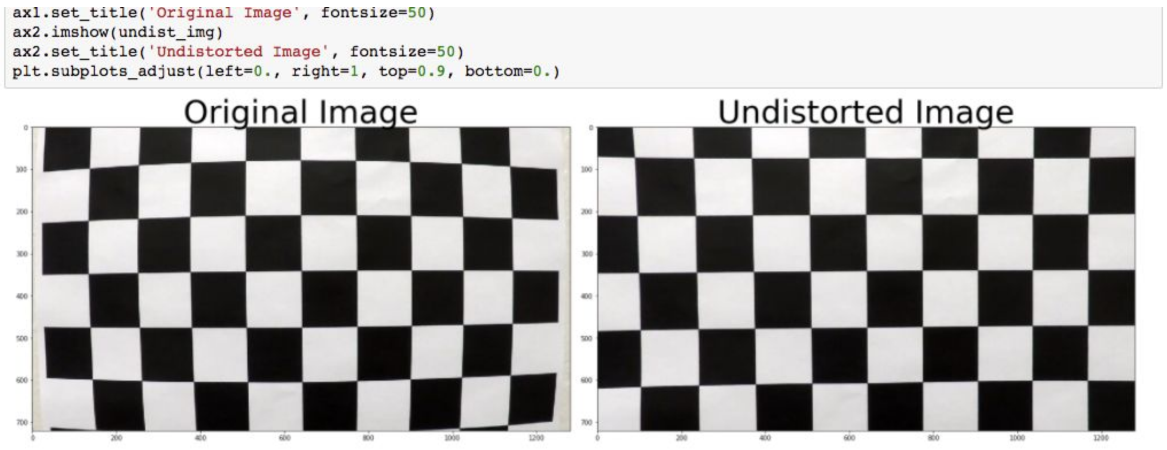
SDCND Project 4 - Advanced Lane detection

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

The process of calculating the camera matrix and distortion coefficients is as follows:

- We use images of a known pattern like a chessboard to calibrate our camera. We use images taken of the chessboard from different angles to ensure we calibrate correctly.
- We feed the images to OpenCV and get the “Object Points” and “Image Points” of all images. Points in 3D space are called object points and their mapping to 2D space for images are called image points
- We use OpenCV’s findChessboardCorners to find all interior corners of the chessboard use them to feed the calibrateCamera function to get the camera matrix and distortion coefficients.

Provide an example of a distortion-corrected image.



```
ax1.set_title('Original Image', fontsize=50)
ax2.imshow(undist_img)
ax2.set_title('Undistorted Image', fontsize=50)
plt.subplots_adjust(left=0., right=1, top=0.9, bottom=0.)
```

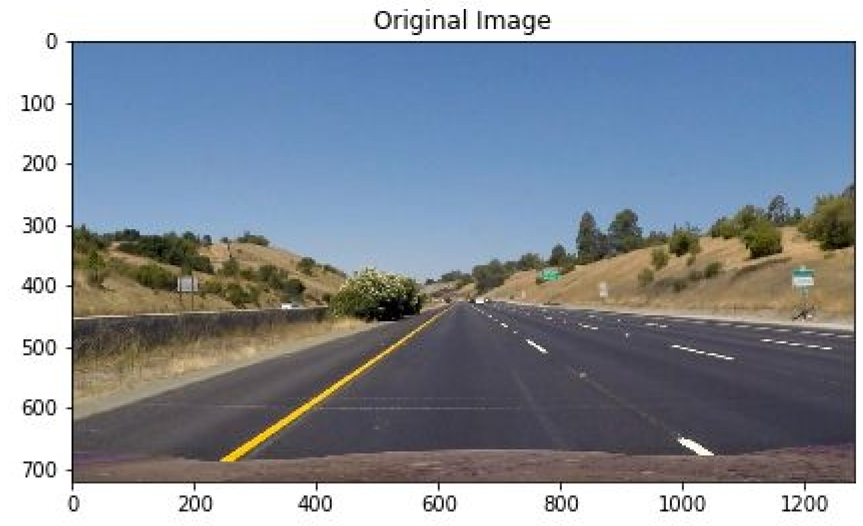
The above images are example of undistorting the image. This is also present in the https://github.com/chmod600/carnd1-project4/blob/master/test_image_pipeline.html output file.

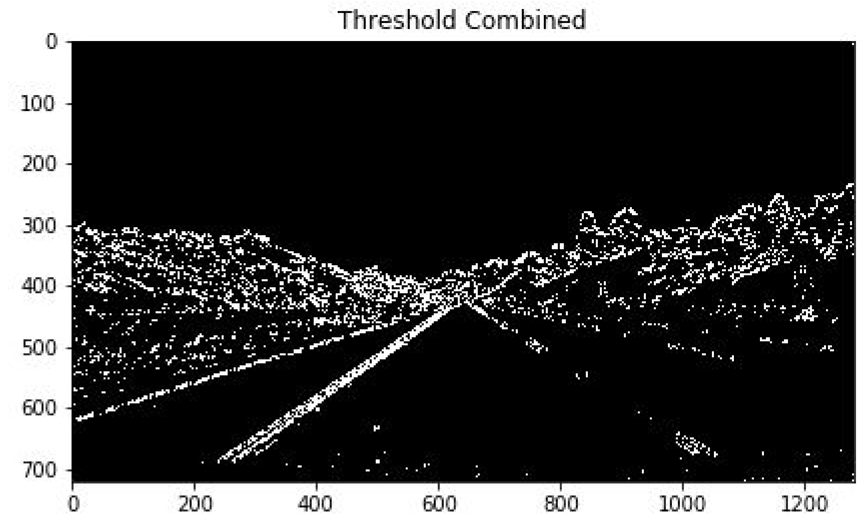
Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

To get a binary thresholded image the following steps were followed:

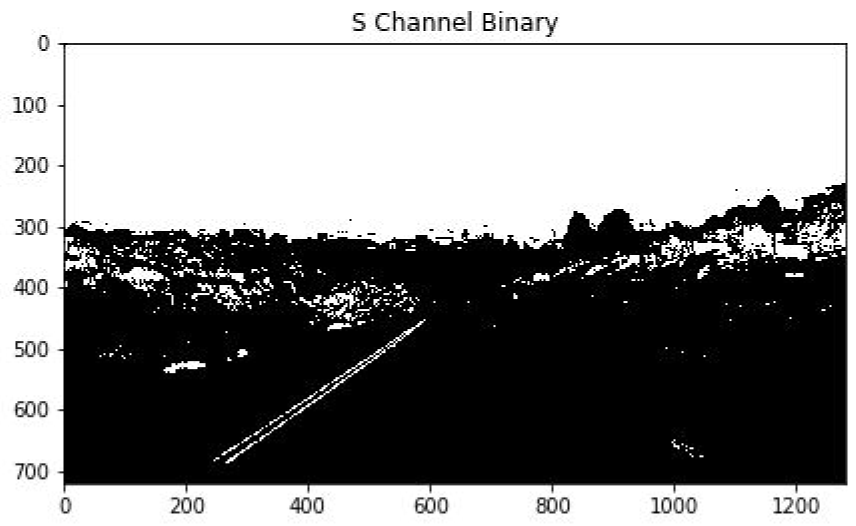
- Apply magnitude thresholding of gradients along X and Y axes on input image.
- We get the directional binary of the image along the X and Y axes of input image.
- We combine magnitude and directional binaries.
- Get the S Channel binary from the HLS space of the input image
- Combine magnitude, directional and S channel binaries to generate a final thresholded image that we can use. We use these two because they provide us with the most relevant information without too much noise.

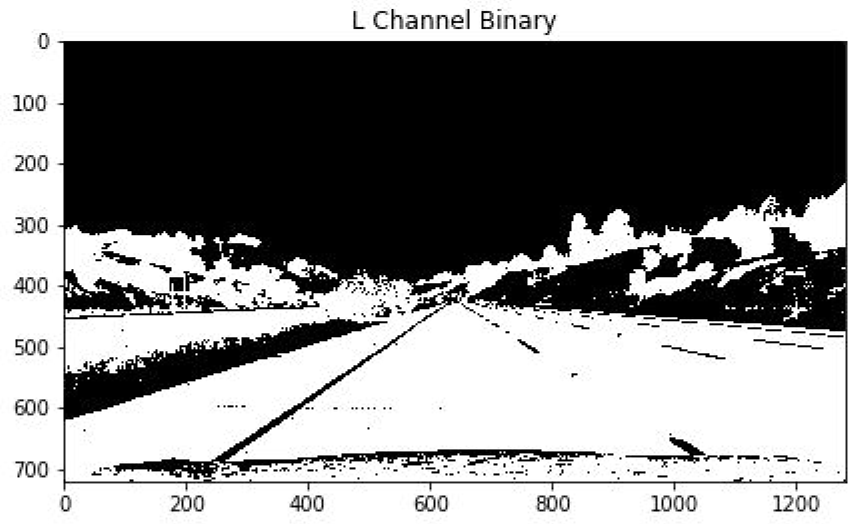
- I have also visualized the L channel. It can also be used to further filter out noise and make lane markings more distinct but for purposes of this project, it was not necessary.





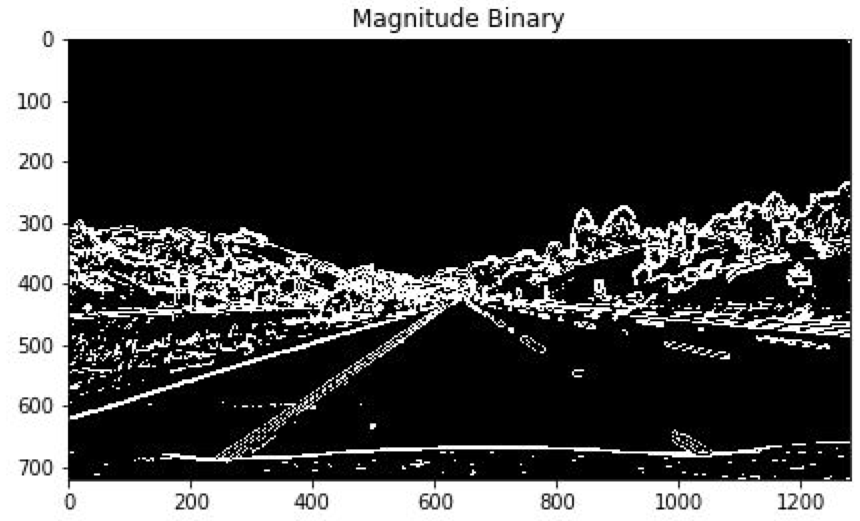
In the above images, we are visualizing the original image and the output binary thresholded image.

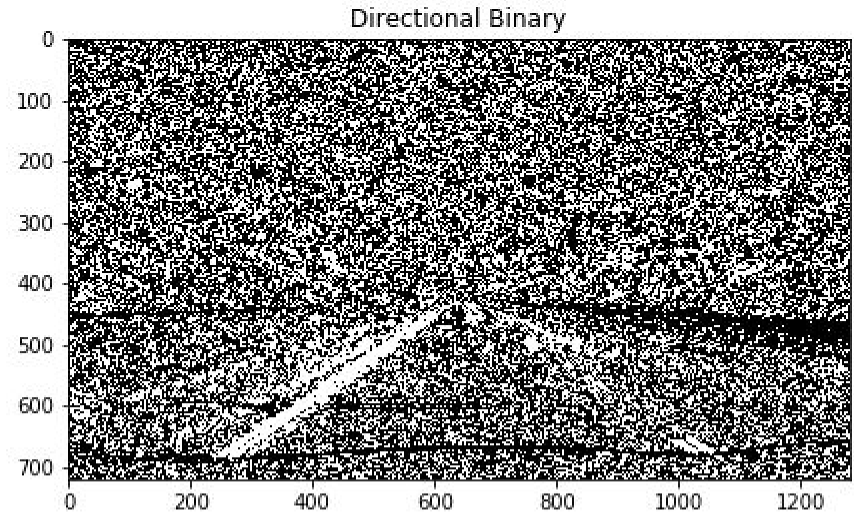




These images visualize how the S channel and L channel binaries of the original image look.

More visualizations of magnitude and directional binaries:





The “***extract_combined_binary(current_image)***” performs this function.

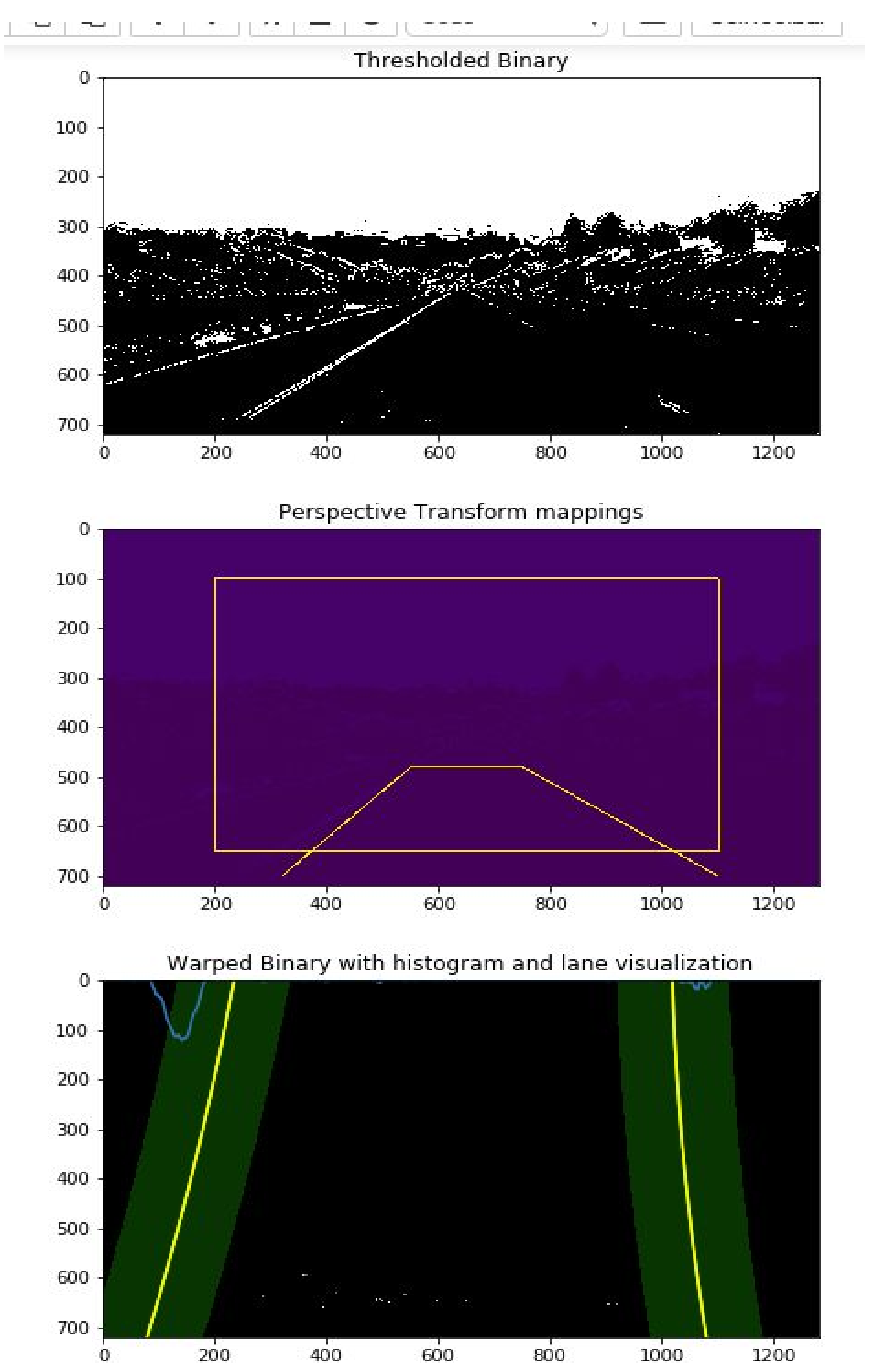
Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

This is done in the “***perform_binary_warp(binary_image)***” function.

Method to perform perspective transform:

- Get the binary thresholded image
- Mark an area of the image that represents an area where the lane lines are present (this will be a trapezoid)
- Mark a rectangular box showing where the above trapezoid should transform to.

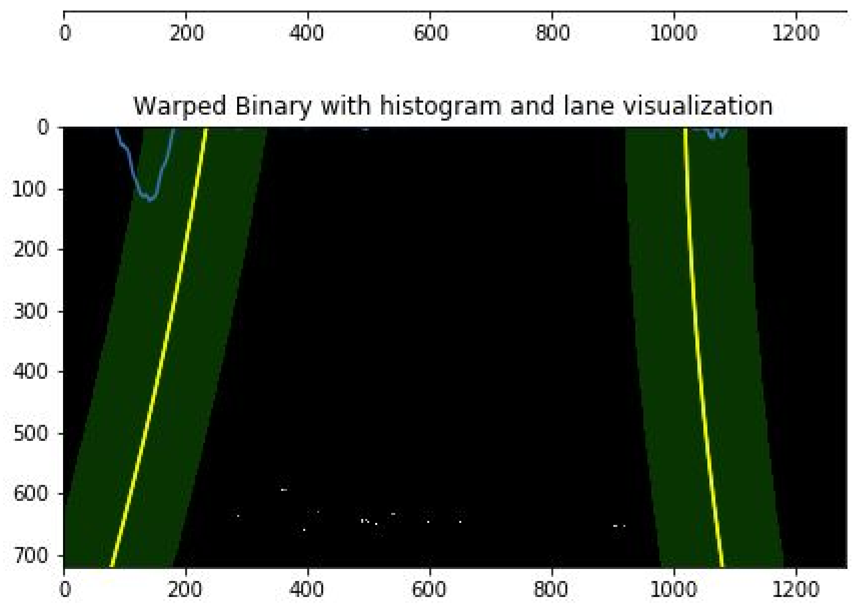
- Use the OpenCV warpPerspective function to perform perspective transform on the image. This is visualized as follows:



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

We perform this function in the “***fit_polynomial_with_sliding_windows(binary_warped)***” function. The method is as follows:

- We get the perspective transformed / warped binary image
- We then extract sum of all points at various points along the Y axis on the lower half of the image
- We plot this sum as a histogram on the image
- There will be two peaks in the histogram representing two lane lines
- The bases of these two peaks represent the start points of the left and right lane lines.
- We then divide the image along the Y axis into several “windows”
- We use the sliding window technique to identify high density points within a rectangular window from start of the lane along the Y axis toward the end. This is effectively searching for high density points along the lane lines. We do this for all windows.
- We extract nonzero points from above method and feed it to Numpy’s polyfit function to get a polynomial function fitting the points along the lane lines. This function will help us draw lane lines for subsequent lines.
- The “***poly_fit_fast***” function can be used to speed up this process by skipping the sliding window once we have a good fit. We are running sliding window over all images for purposes of this project to keep it simple.
- This is visualized as follows:



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.

- We do this in the “***measure_radius_of_curvature(binary_warped, left_fit, right_fit, leftx, lefty, rightx, righty)***” function

- This function uses the input image (binary warped) and generates evenly spaced points along the Y axis.
- We then use the polynomial function to fit the left and right lane line points
- We use a margin of 100 points to keep a window of 100 pixels on all four sides of the point to find lane points and their direction. We use these formulae for the getting the radius:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

In the case of the second order polynomial above, the first and second derivatives are:

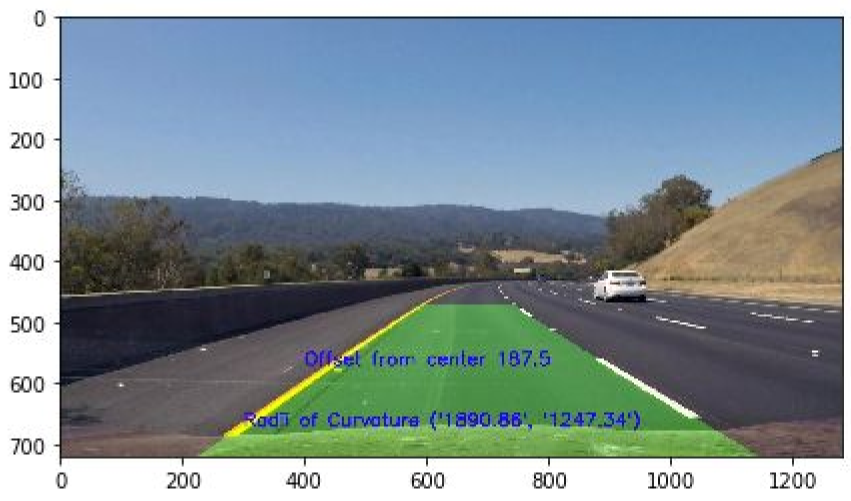
$$f'(y) = \frac{dx}{dy} = 2Ay + B$$

$$f''(y) = \frac{d^2x}{dy^2} = 2A$$

So, our equation for radius of curvature becomes:

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

- To calculate the position of vehicle from the center, we assume camera is mounted in the center of the car. We calculate center of lane, center of image and the different of image center and lane center gives us the offset from center of the car.



The above image also answers “**Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.**”

Video output:

The video output is available at https://github.com/chmod600/carnd1-project4/blob/master/p4.mp4

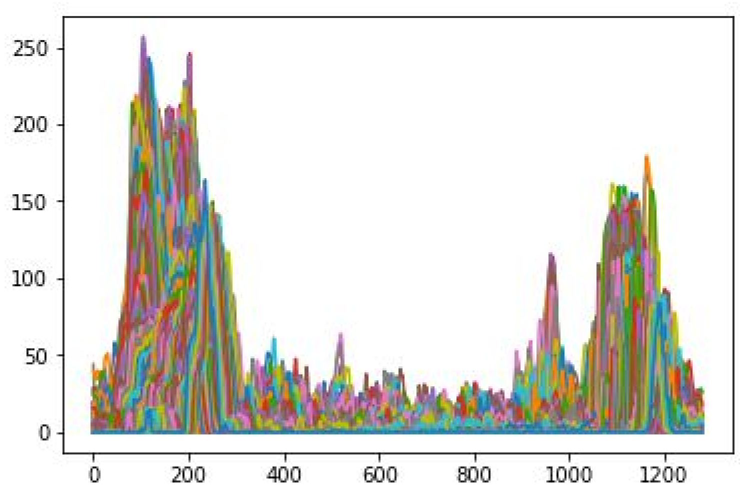
Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

Presently, the pipeline is tuned WRT to the project video. One assumption being the left lane is always solid (not broken) and gets plotted very easily and does not require special attention. The S channel of the image does a good job of clearly detecting it in the binary. To make it more robust, this pipeline will need improvements in detection of the left lane image via various thresholding methods and smoothing techniques.

Another assumption is that the lane width remains the same. We can also use auto polygon detection to perform a perspective transform of the image rather than hardcoding the values.

Speed improvements are also skipped for purposes of this project. Several ways like skipping sliding windows can be used to optimize for speed.

Interesting Histogram of all lane detections:



References/Attributions:

1. Code taken from the lessons, quizzes and my previous SDCND projects.
2. Stackoverflow, github provided with good examples and explanations of how stuff worked in python.
3. Udacity forms and Q&A sessions