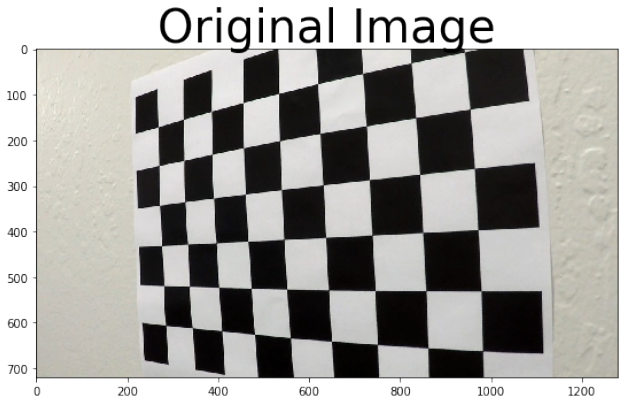
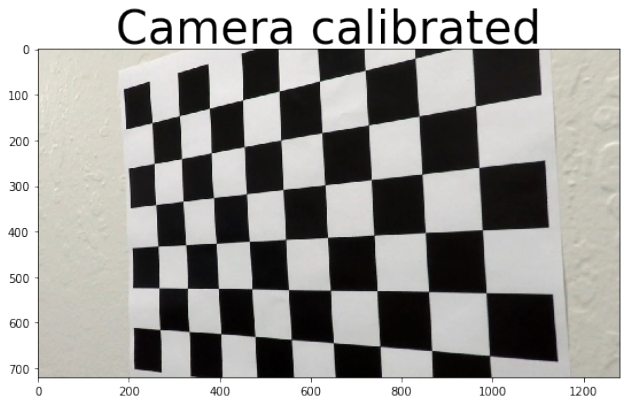
Writeup project 4. Advanced lane finding

1.- Camera calibration

1. Have the camera matrix and distortion coefficientes been computed correctly and checked on one of the calibration images as text?

The code for this step is contained in the first code cell of the IPython notebook (or in lines # through # of the file called some_file.py).
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. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:





I saved the mtx and dist values of the calibrate_camera() function with pickle. Then I just used those values in the function cv2.undistort() to undistort the image.

2.- Pipeline

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:

I saved the mtx and dist values of the calibrate_camera() function with pickle. Then I used those values in the function cv2.undistort() to undistort the image

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 reused some functions from the first project to do that:
- grayscale() to transform to grayscale an image
-gaussian_blur() to smooth the images
-region_of_interest() to select the region I want to take care of.

Furthermore I added some functions like:
-hls_s() to get the s_channel of an image
-abs_sobel_thresh to calculate directional gradient
-mag_thresh() to calculate the gradient magnitude
-dir_threshold applies Sobel x and y, then computes the direction of the gradient and applies a threshold.
-paint_lines() to recognize the region of interest

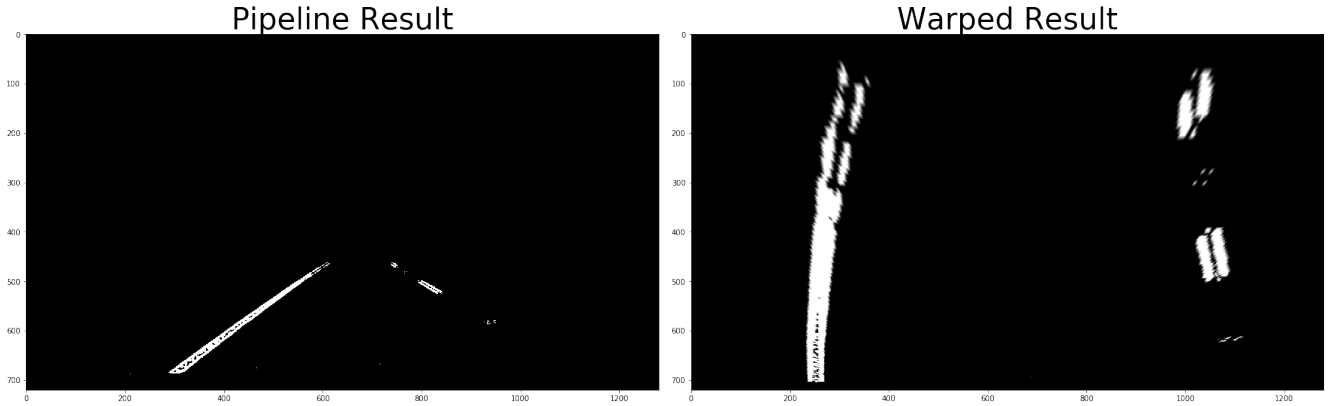
You can find the code in the 3rd, 4th and 5th cells of the advanced_lanes_finding.ipynb.
I started by smoothing the image and transforming it to grayscale. Then I extracted the S channel and applied all the thresholds.
Finally I combined the thresholds and selected the region of interest. I got this result:

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

To do this I implemented a function called corners_unwarp() (6th cell) where I selected the points that I wanted to "move" and fit in another image.

```
src = np.float32([[589,457],[750,457],[1200,720],[145,720]])
dst = np.float32([[offset, offset2],
                  [img_size[0]-offset, offset2],
                  [img_size[0]-offset, img_size[1]-offset2],
                  [offset, img_size[1]-offset2]])
```

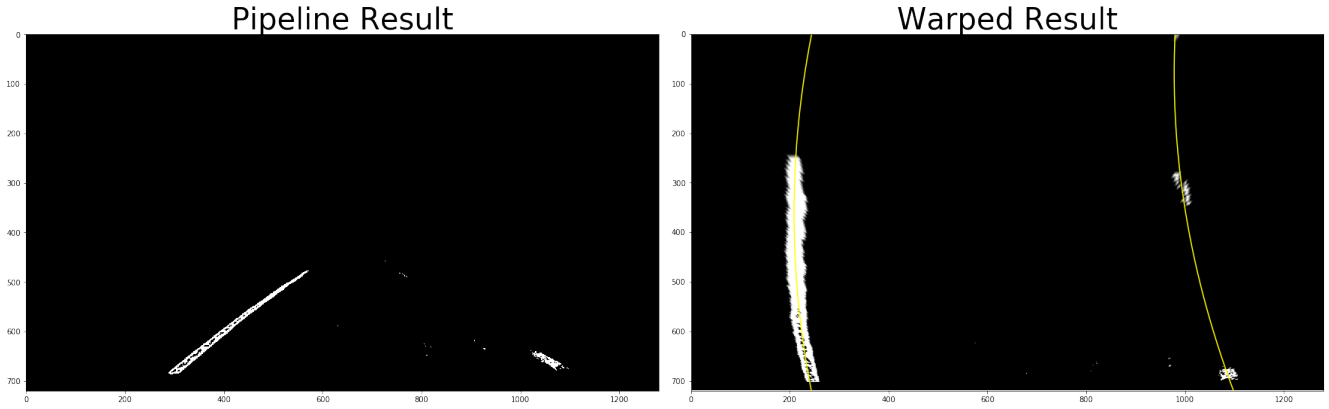
And the result looks like this:



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

In the 9th cell of the advanced_lanes_finding.ipynb file I write two functions to fit the lines with a polynomial. The first is needed when you don't have data from before (sometimes you can not find the lane in the image, or just in the beginning), the second one uses the data known from previous images to determine where to fit the line.

Here is the 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.

I did this by using information extracted from the find_lanes() and fit_lines() functions (10th cell). For the radius of curvature function I used all the pixels on the y side and the pixels from the x side where there was a line. For the car position I got the indices of the line points and the previous car position.

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 the 11th cell. I used a function called draw_lane() that uses the inverse value got in the warping step. Then I just drown the lane in the main image. The result looks like this:

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!).

My video is called project.mp4, take a look at it.

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?

I would like to take a few more hours to improve the pipeline for finding the lines. I found that my pipeline is failing when no point are found meeting the requirements. I also think I have to improve the radius of curvature function, I think it can be much more robust.