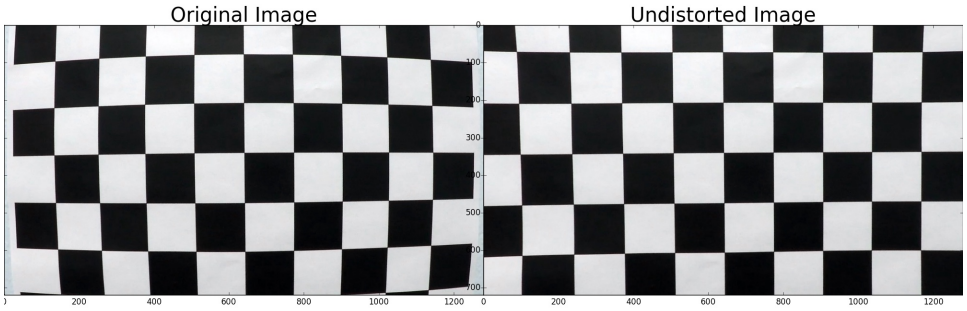
Advanced Lane Finding Project 4

by: Sung H. Yoon

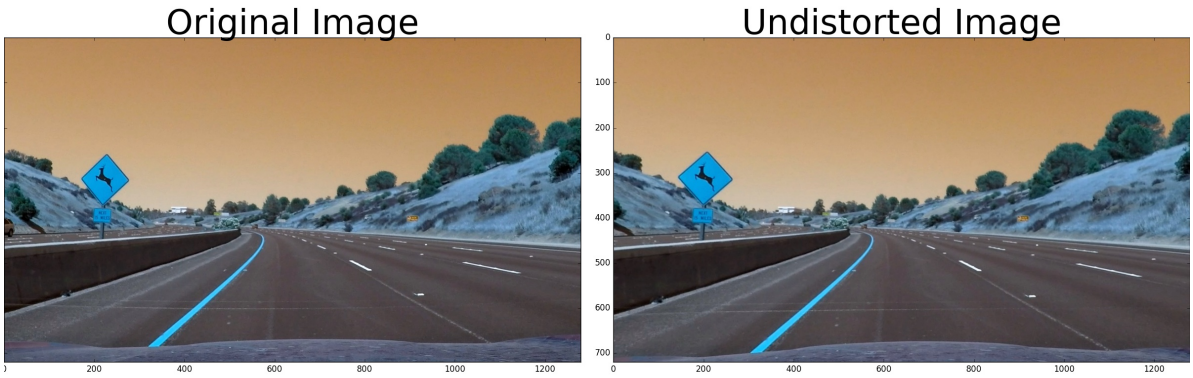
February 13, 2017

1. Camera Calibration
   i. I processed each chessboard images in camera_cal sub_directory.
   ii. Computed the camera calibration matrix & distortion coefficient in cameraCalibration.py, which starting point was given as class exercise, but
      - Changed x,y = 9,6 instead of hard coding (line 10)
      - objp is now flexible as I got Rid of the hard coding on matrix size (line 13 &14)
      - Created pickle file cameraCalibPickle.p
      - As in exercise using objpoint & imgpoints, camera calibration & distortion coefficients were computed using cv2.calibrateCamera() function.
      - I applied the coefficients using cv2.undistort() to test image as seen here:
      - **File: output_images/**undistChessBoard.jpeg



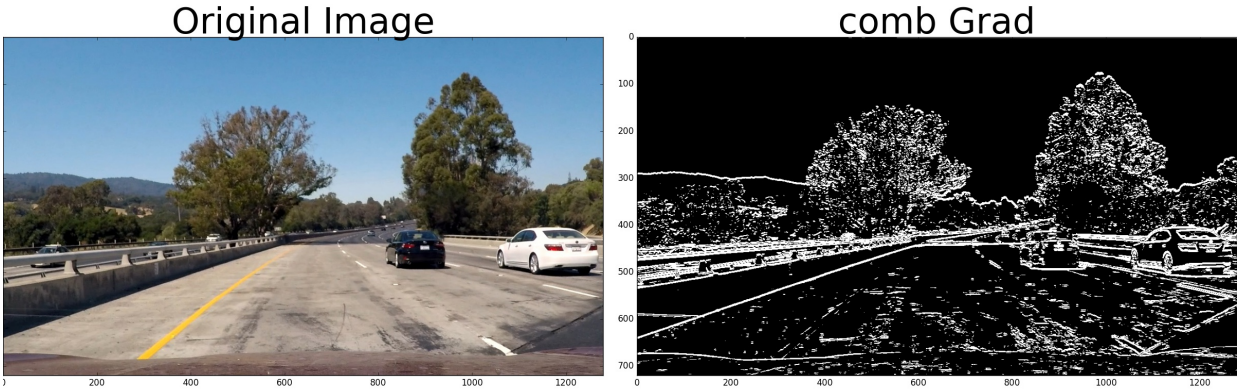
2. Pipeline (single images)
   i. Has the distortion correction been correctly applied to each image?
      - Using undistortTestImgs.py, I have undistored test images. One of them is here

- **File: output_images**/undistTestImg.jpeg



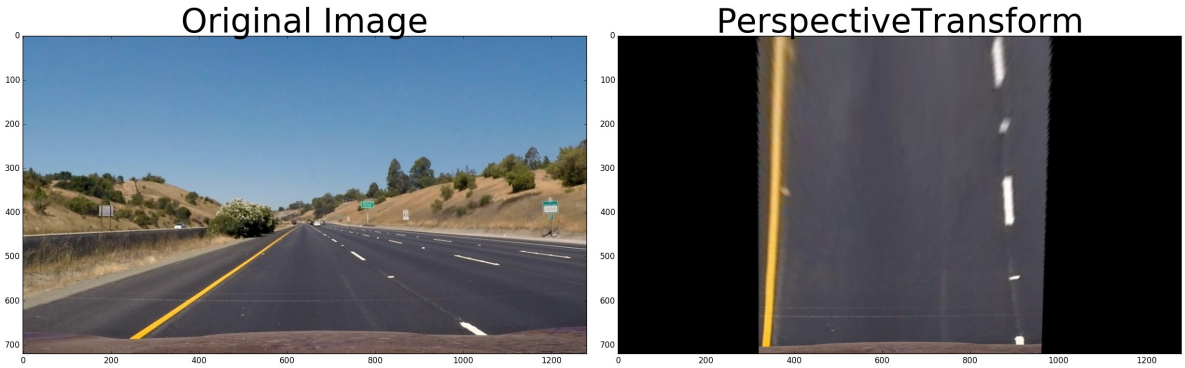
- Notice that the sign on left appears to be a lot smaller than right
- Has a binary image been created using color transforms, gradients or other methods?

ii. Please take a look at ./output_images/binary.jpeg



- The guts are in file **AdvLnUtils.py**
- but this section's implementation is in **binaryImg.py**
- The **imgUtils.py** contains the support files for **AdvLnUtils.py** and AdvLnUtils functions routinely calls the various functions from it. These are either general functions that graph, mask, draws lines, or slices images that I may be able to utilize in the future (well hope to anyway). I will move the specific functions from **AdvLnUtils**, once I get those to be more generalized.

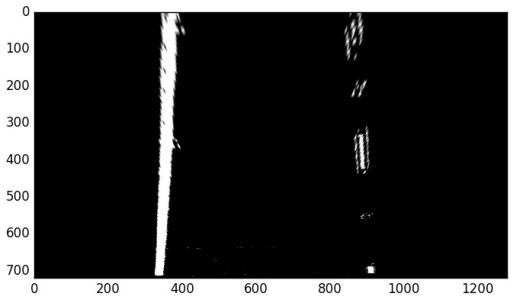
3. Has a perspective transform been applied to rectify the image?
   i. Transformed by warp function in AdvLnUtils.py using perspectTrans.py

- 

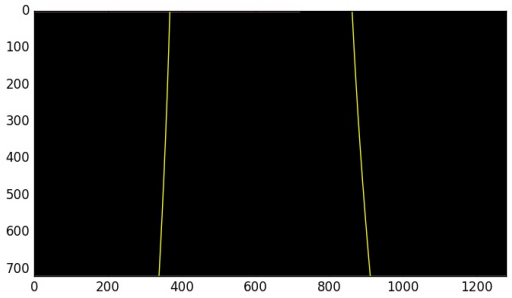
- 

- Hard Code accomplish this is on function **def warp(Img)** in **AdvLnUtils.py**
  - `y_bot = Img.shape[0] #=720`
  - `src = { 'tR': [730, 460],  'tL': [570, 460],`
  - `'bR': [1180, y_bot], 'bL': [180,  y_bot] }`
  - `vertices = np.array([[ src['bL'], src['tL'], src['tR'], src['bR'] ]], dtype=np.int32)`
  - `regionOfInterest = outlineRegionOfIntest(Img, vertices)`
  - `# src coordinates`
  - `src = np.float32([ src['tR'], src['bR'], src['bL'], src['tL'] ], dtype=np.int32)`
  -
  - `dst = { 'tR': [980, 0],  'tL': [320, 0],`
  - `'bR': [960, y_bot],'bL': [320, y_bot],}`
  - `# Dst coordinates`
  - `Dst = np.float32([dst['tR'], dst['bR'], dst['bL'], dst['tL'] ], dtype=np.int32)`

4. Have lane line pixels been identified in the rectified image and fit with a polynomial?



i. this Image is transformed with



ii. the polynomial

- Taking the course's approach
- I have used, getLineCurvature & getLineCurvature function, which uses formula

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

- Afterward take 1st & 2nd derivative of the polynomials

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

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

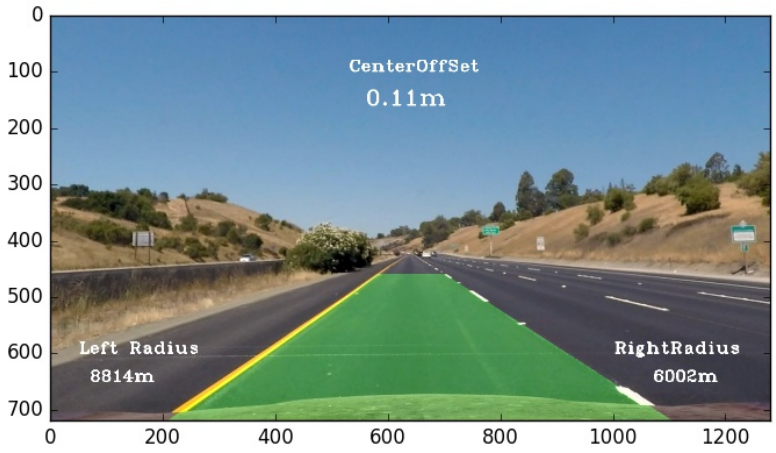
- Results in

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

- The results are then converted into meters
- All above were taken copied & directly from lecture website <Measuring Curvature>

-

5. Having identified the lane lines, has the radius of curvature of the road been estimated?
   i. And the position of the vehicle with respect to center in the lane?
      - The position of the vehicle with center, left, & right radius has been calculated into the image itself.
      - This is shown here:



      - And continues through out the video, as white letter CenterOffSet, Left Radius, & Right Radius in meters.

6. Pipeline (video)
   i. Does the pipeline established with the test images work to process the video?
      - Link to my video Result

7. README
   i. Has a README file been included that describes in detail the steps taken to construct the pipeline, techniques used, areas where improvements could be made?
      - This document will serve as README
   ii. Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.
      - The techniques used in this project followed the Udacity lectures closely. The polynomial coefficients were used and this works reasonably well in the project video to identify the lanes; however, as shown in the challenge video the result is a bit erratic.
        - Here's Link to my video on the 1st Challenge
      - The few things that would need improvements that I can immediately think of are
        - 1. The lines on birds eye view is less than parallel. This hard coded may have contributed to jumping in the lane detection.
        - 2. The bigger problem is likely that the road surface had multiple tones in the middle that would confuse the line coefficients. Adjusting and improving the various threshold and color calibration should help.

- 3. The coefficient and equation may be too simplistic for more twisting roads in the harder challenge. A piecewise linear should improve result quite a bit. Plus, the road right in front of the car does NOT all of sudden turn into less than a meter should be embedded into the logic.
  The Link for the harder Challenge is here, and it does reflect my concern.
- 4. The video clip transform script speed is painfully show. This will cause problems in real life. Either off load the calculations to GPU or better algorithms would be necessary. Either case, all these for loops must be taken out. GPU nor numpy would be able to function well with those around. And, might be better to dial down the image size as I did in the prior projects.