Advanced Lane finding

The goals / steps of this project are the following:

- Explain the following steps through implementation
    - Computation of camera calibration
    - Applying a correction error through distortion
    - Create a thresholded binary image using gradients(magnitude,direction,Sobelx/y)
    - Applying a perspective transform to the birds-eye-view of the look-ahead lanes
    - Detecting lane pixels and identification of the entire lane
    - Computing the radius of curvature of the lane and the position of the vehicle w.r.t the center of the lane
    - Detecting the lane boundaries on the original image
    - Rendering the lane boundaries and displaying lane curvature and the position of the vehicle w.r.t center of the image.

Camera Calibration

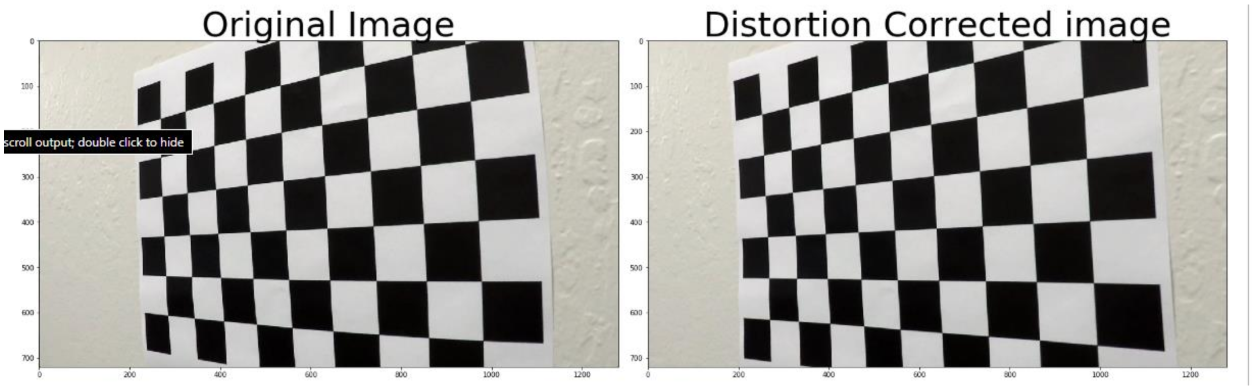
Reflection

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

Following are the steps for computing the camera matrix and distortion coefficients

- Steps: Preparation of “object points” corresponding to the coordinates of the chessboards in the world.
    - Assumption: Chessboard is placed on a plane wherein the object points are equal for each iteration of the calibration
    - Assumption: Same camera is used to capture the chessboard which is later fitted onto the car to capture test images
    - Replication of array of coordinates in Objp and appending a copy to object_points
    - Identification of the chessboard pattern of 6 rows and 9 columns.
    - Using the output obj_points and img_points to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function.
- Code: libs/camera_cal.py
    - compute_calibration_points()
    - cal_undistort(img, objpoints, imgpoints)
    - render_original_undistorted_image(orig_image,distortion_image)

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


Original Image
Distortion Corrected image
scroll output; double click to hide

3. 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.

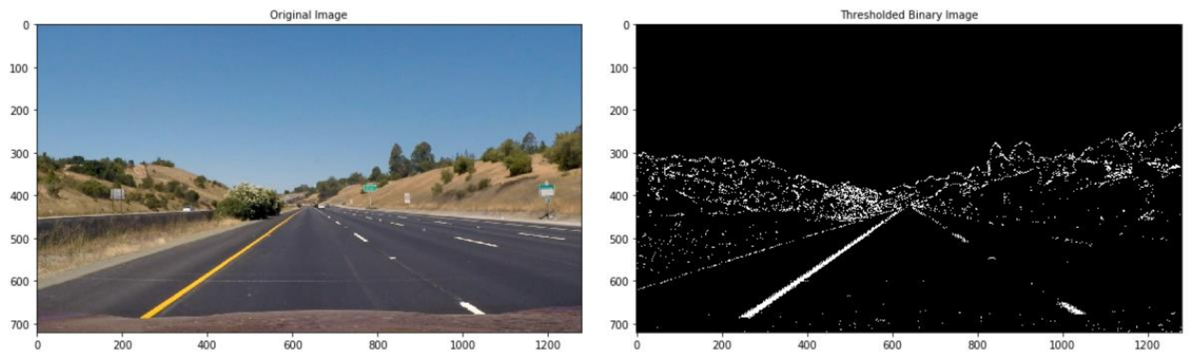
Following are the steps for creating a thresholded binary image

- Steps:
  - HLS Color space has been used.
    - S channel along with gradient combination has been used as RGB filter is sensitive to the threshold where yellow lane or change of road color exist
  - Gradient, Sobel-x, Sobel-y and magnitude thresholds are also used

- Code: libs/ dir_sobel_color_persp_warp_func.py
  - abs_sobel_thresh(img, orient='x', sobel_kernel = 3, thresh=(0,255))
  - mag_thresh(img, sobel_kernel=3, mag_thresh=(0, 255)):
  - dir_threshold(img, sobel_kernel=3, thresh=(0, np.pi/2)):
  - hls_select(img, thresh=(0, 255)):

- create_threshold_binary_image(image,ksize=3):

- Example

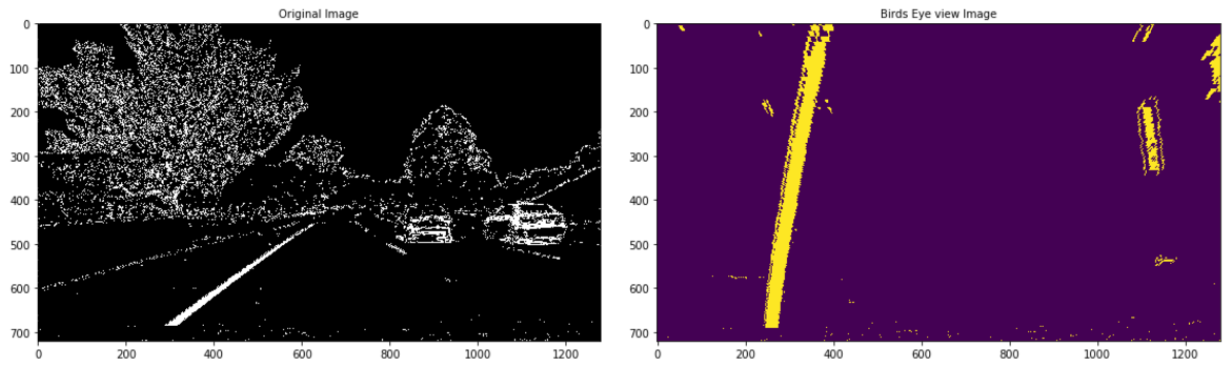


4. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

Following are the steps for performing perspective transform

- Steps:
  - The perspective transformation was performed by selecting (intuitively) the coordinates of source and destination points related to unwarped and transformed image.
  - Using cv2.getPerspectiveTransform function, the content in the source points were transformed to destination points
  - Using cv2.warpPerspective function, the matrix obtained from the previous step was applied to transform the input image to a warped image.

- Code: libs/ dir_sobel_color_persp_warp_func.py
  - birds_eye_view(image):
  - following source and destination points were used
    - src = [[490,480], [810,480], [1250,720], [140,720]]
    - dst = [[0,0], [1280,0],[1250,720], [140,720]]
- Example



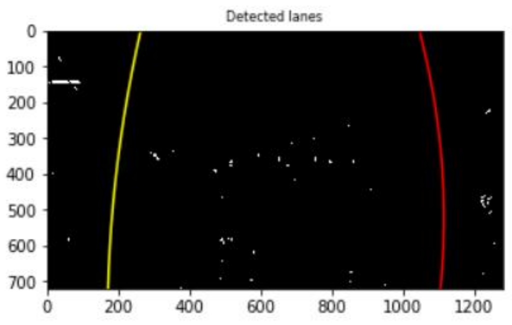
5. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Following are the steps for identification of the lane-line pixels and fitting those positions with polynomials

- Steps:
  - Calculating the histogram of non-zero x-axis in the binary image.
  - Apply sliding window approach
    - Dividing the vertical axis to 9 windows
    - Annotating the boundaries (by marking a rectangle) around the non-zero pixles in the approximated region of left and right lanes.
    - Appending non-zero pixel values of each window into left & right lines
  - Identifying a polynomial that fits all the lane pixels for each of the left and right lines
  - If the previous frame has been searched with lane pixels and the current frame has a threshold number (min_ids) of non-zero pixels in the image, reuse the previously used polynomial.
    - Else, perform the above sliding window approach and calculate the new polynomial

- Code: libs/ sliding_window_histogram.py
  - find_lane_pixels(binary_warped):
  - fit_polynomial(binary_warped):
  - optimized_fit_polynomial(binary_warped,left_fit,right_fit):
  - render_line_polynomial_orig_image(image1, image2,image2_title):

- Example

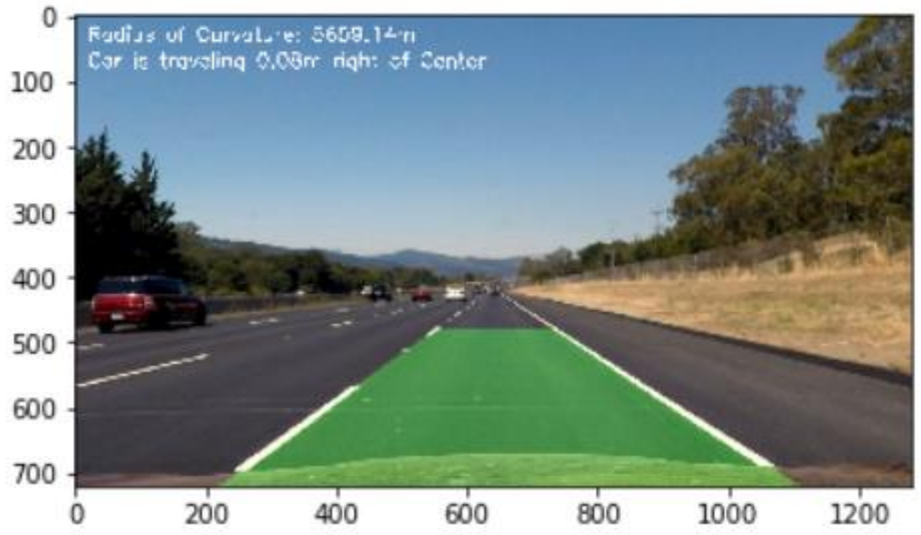


6. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

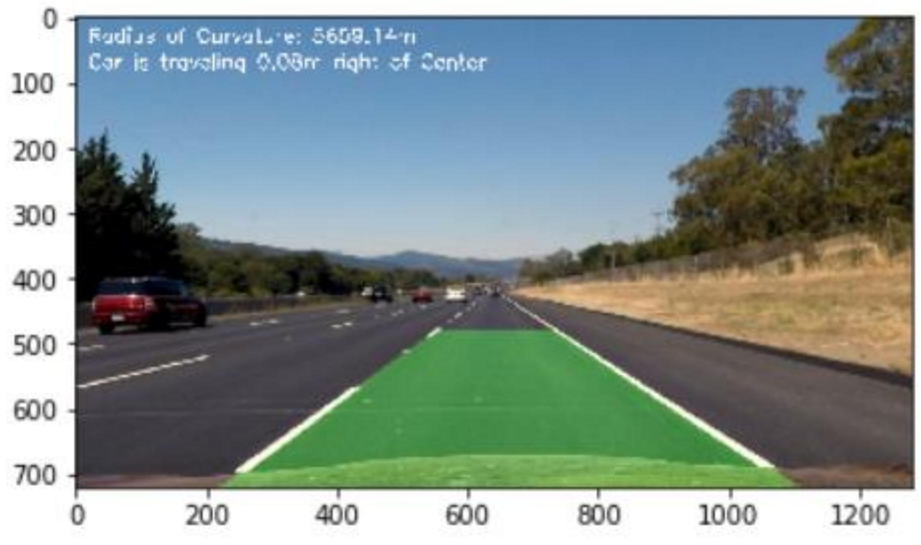
Following are the steps for computing the radius of curvature of the lane and offset distance of the car from the center

- Steps:
    - Source - . https://www.intmath.com/applications-differentiation/8-radius-curvature.php
    - Assumption: Use the coursework’s mapping criteria based on US govt regulations-
        - 30 meters per 720 pixels in the vertical direction, and 3.7 meters per 700 pixels in the horizontal direction.
    - Apply the formula above for all the lane pixels found in the polynomial fit for the left and right lane lines.
    - Asumption: Camera is mounted at the center of the car and the center of the image is the center of the car
    - From the polynomial fit of the left and right lane lines, compute the mean x value of the bottom x value of the left lane line, and bottom x value of the right lane line that provides the lane’s center value
    - Offset is the difference between vehicle's center x value (i.e. center x value of the image) and the lane's center x value.

- Code: libs/ sliding_window_histogram.py
    - compute_radius(binary_warped):
    - compute_offset(binary_warped):
- Example


```
Radius of Curvature: 5609.14m
Car is traveling 0.08m right of Center
```

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


Radius of Curvature: 5609.14m
Car is traveling 0.08m right of Center

8. 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!)

Refer out.mp4 in the workspace.

9. 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?

Following are the issues faced during the implementation of the project

- Non-zero line pixel calculation for split road conditions - Distinguishing lanes from the
  - A split road adding additional line
  - Long stretches of concrete boundaries that are very close the lane lines
- Lighting conditions during color space transforms
  - Extreme brightness on the camera
  - Under bridge conditions that completely masks yellow or white lines due to the absence of light
  - Night and extreme weather conditions

- Curvature and vehicle offset computation
    - Sharp turn scenarios –This changes the leftx and rightx values that are hardcoded for the base values for computing the radius.
    - Sharp turn scenarios with changing light conditions –
    - Merging of lanes
- Processing time for videos

Next steps for robustness –
- Averaging coefficients of previous few frames when there are no non-zero pixels found due to various lighting conditions.
- Addition of lane width as a hyperparameter to tune the polynomial coefficients during search window
- Implementation of quick search techniques for polynomial to reduce the video processing technique instead of lengthy iteration of array indexing for each frame