Swapnil Daingade (sdaingad)

Advanced Lane Finding Project

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

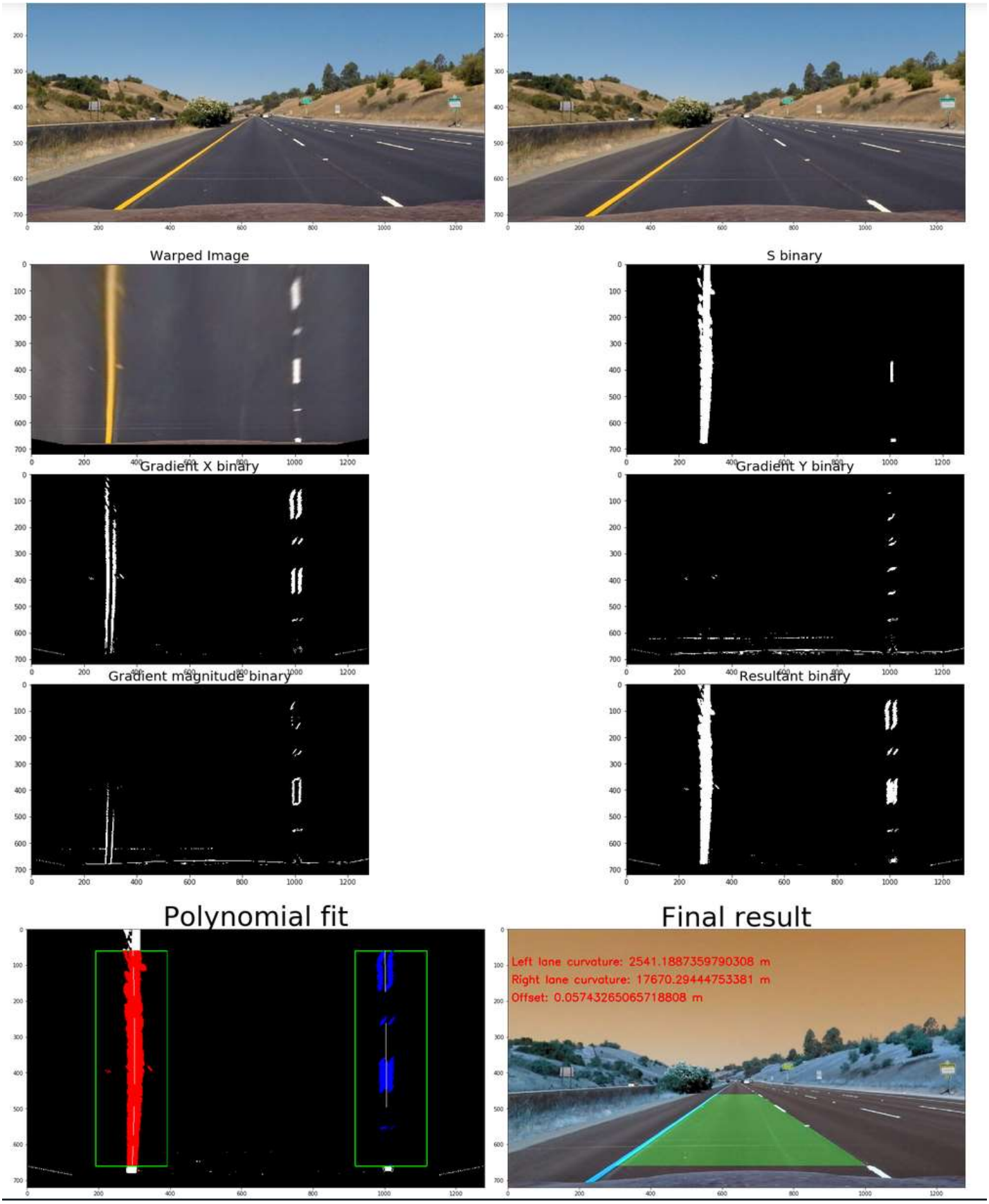
For code please refer to P2.ipynb

Please refer to ./test_images_output for sample output for test images

Please refer to ./test_videos_output/project_video.mp4 for output video

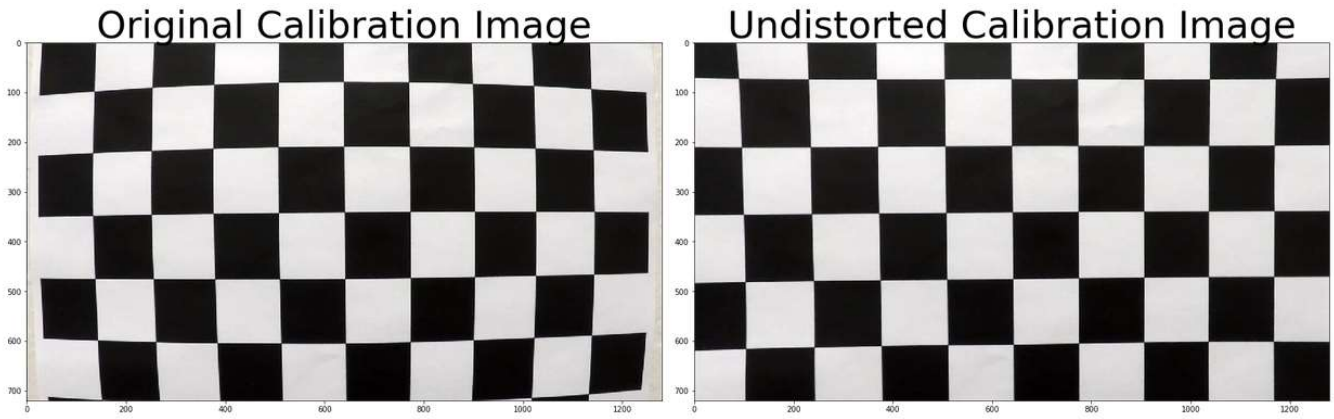
Sample image (stringht_lines1.jpg) displaying the pipeline

Original Image Undistorted Image



Camera Calibration

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



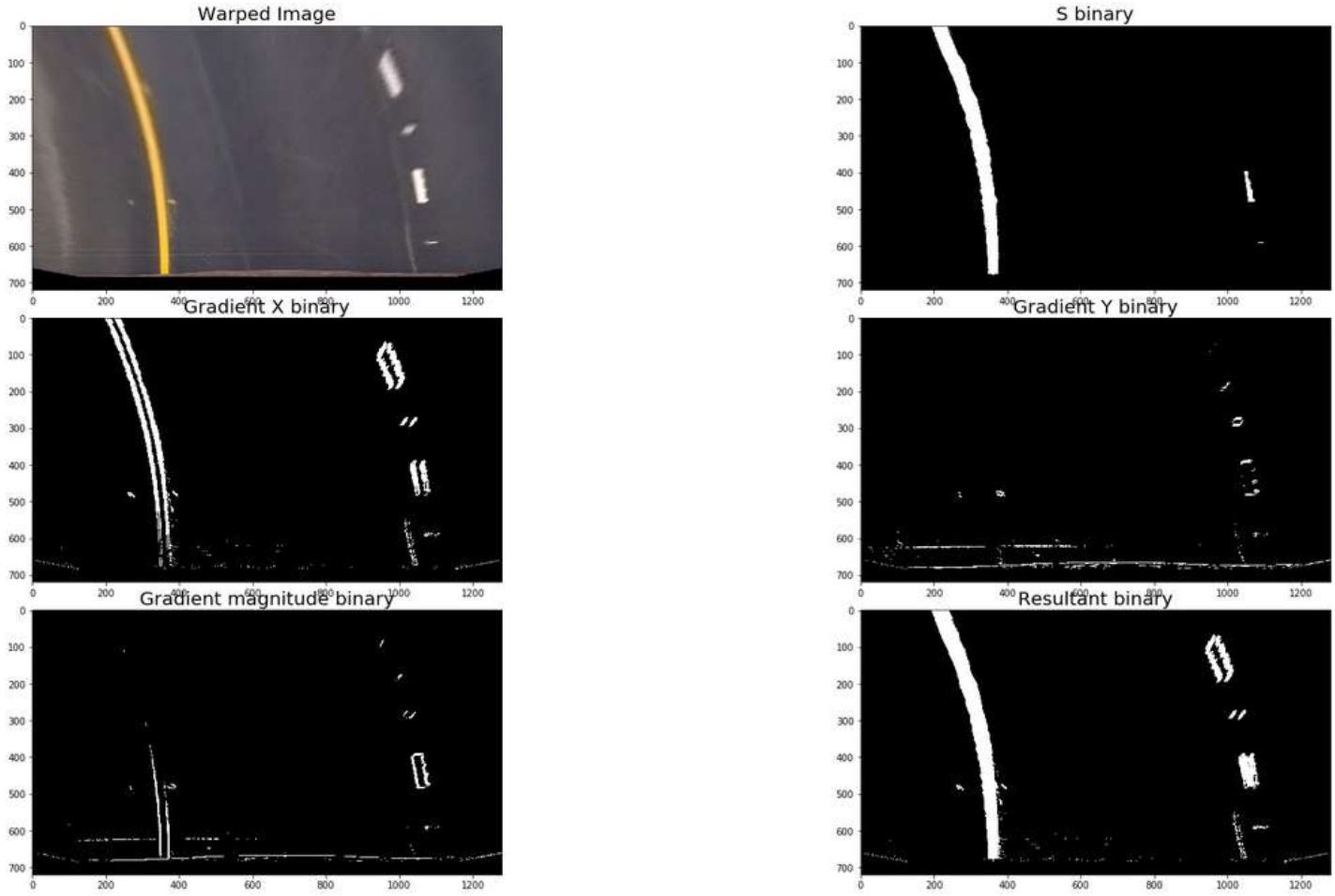
- I finding corners using the images provided for calibration in ./camera_cal folder
- I assume a chessboard size of 9x6 for this project
- Please refer to method find_corners() in code
- I compute the camera matrix and distortion coeffecients using cv2.calibrateCamera
- Please refer to method calibrate_camera() in code
- I use cv2.undistort along with the cameraMatrix and distortion coefficients computed earlier to get the undistorted image

Pipeline (single images)

1. Provide an example of a distortion-corrected 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.

Multiple thresholds created for the image above

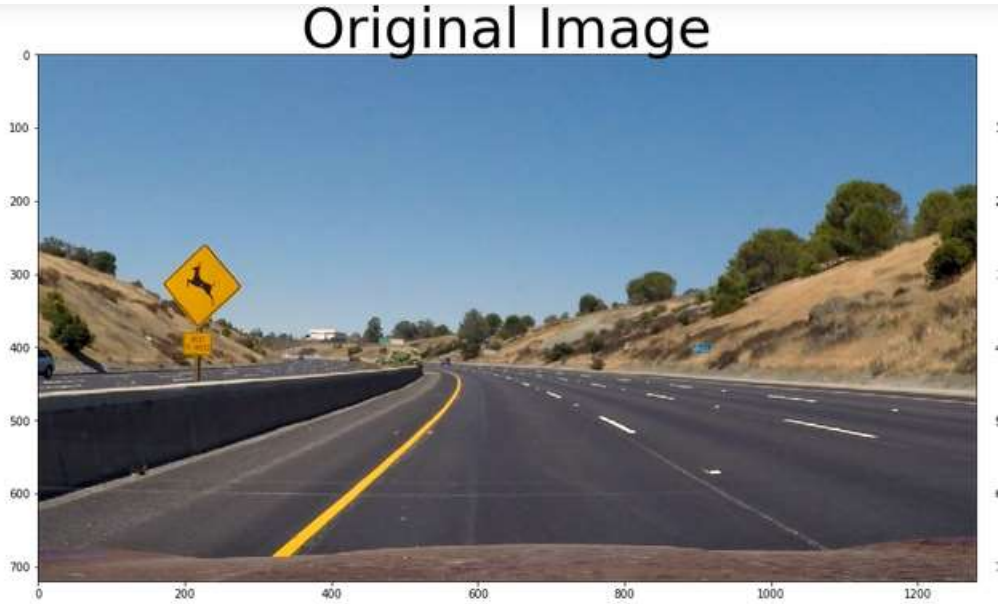


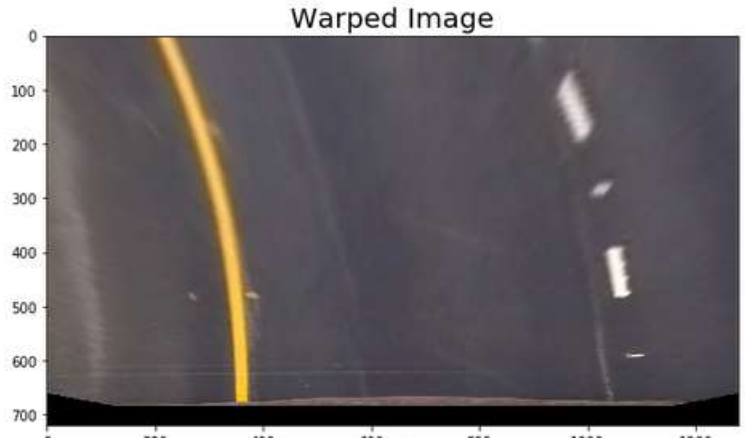
I create 5 different threshold binaries here, they are as follows

Please refer to method apply_color_and_gradient_threshold() in code

- s channel threshold binary
- gradient in x direction threshold binary
- gradient in y direction threshold binary
- magnitude threshold binary based on gradients in x and y direction
- direction threshold binary based on gradients in x and y direction
- All these threshold binaries are displayed in table format if silent=False
- **I decided that my resultant threshold binary will be a local OR of the s channel threshold binary and gradient in x direction threshold binary**

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

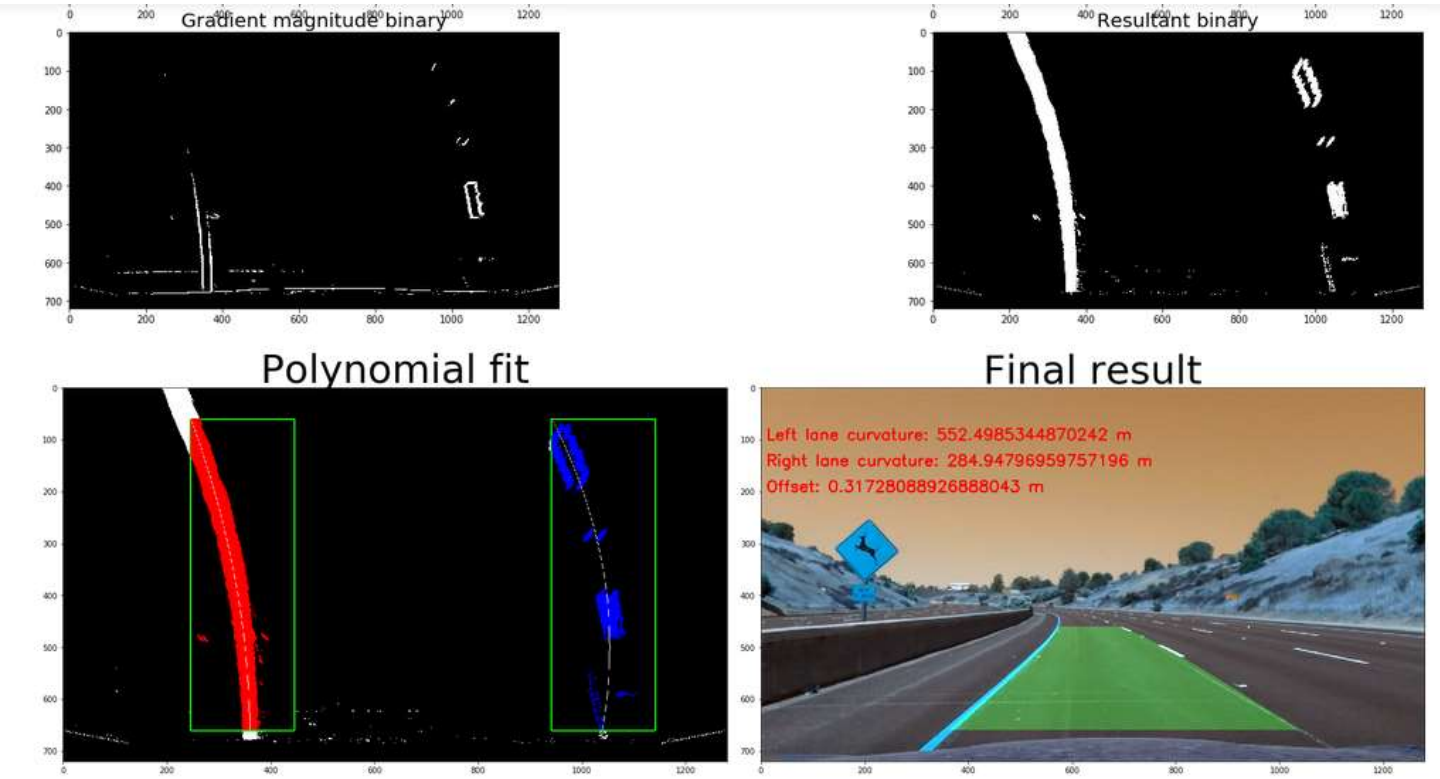




- Please see the method warp()
- I warp a trapezoidal space in the image to a rectangular space in the birds-eye view
- I compute the perspective transform mapping (M) and inverse perspective transform mapping (Minv) using cv2.getPerspectiveTransform

```
# Four source coordinates
   src = np.float32(
      [
       [535, 460],  # top left
       [740, 460],  # top right
       [1231,660],  # bottom right
       [30, 660]    # bottom left
      ])

   dst = np.float32(
      [
       [30, 60],  # top left
       [1231,60], # top right
       [1231,660], # bottom right
       [30, 660]   # bottom left
      ])
```

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



See method find_lane_pixels() in code
Identify pixels that belong to left and right lane lines

- CREDITS: The code is taken from examples discussed in lessons and adapted to my implementation
- Takes a histogram of pixels from y=60 to y=720
- Divides the image down the middle
- Peak in the left half of histogram is assumed to be the x value for left lane line
- Peak in the right half of histogram is assumed to be the x value for right lane line
- We have margin of +/- 100 pixels in either direction of left histogram peak. y values are from y=60 to y=660 We consider points in this box to be part of left line.
- We have margin of +/- 100 pixels in either direction of right histogram peak. y values are from y=60 to y=660 We consider points in this box to be part of right line.

See method fit_polynomial() in code
Fit polynomial through points identified as belonging to left and right lane lines

- CREDITS: The code is taken from examples discussed in lessons and adapted to my implementation
- Fit a polynomial through points identified as belonging to left and right lane lines
- Once equations for left and right lane lines are found, use the equations to generate x co-ordinates for left and right lane lines for y ranging from y=60 to y=660
- For determining radius of curvature in meters we need to convert the above points from x,y space to real world space
- We fit polynomials through left and right points in the real world space and calculate the radius of curvature
- We also calculate the offset of the car from the center of lane by comparing the x values of center of image with x values for center of the lane lines
- We plot points considered for left lane in red and points considered for right lane in blue
- The actual lane lines are plotted in white

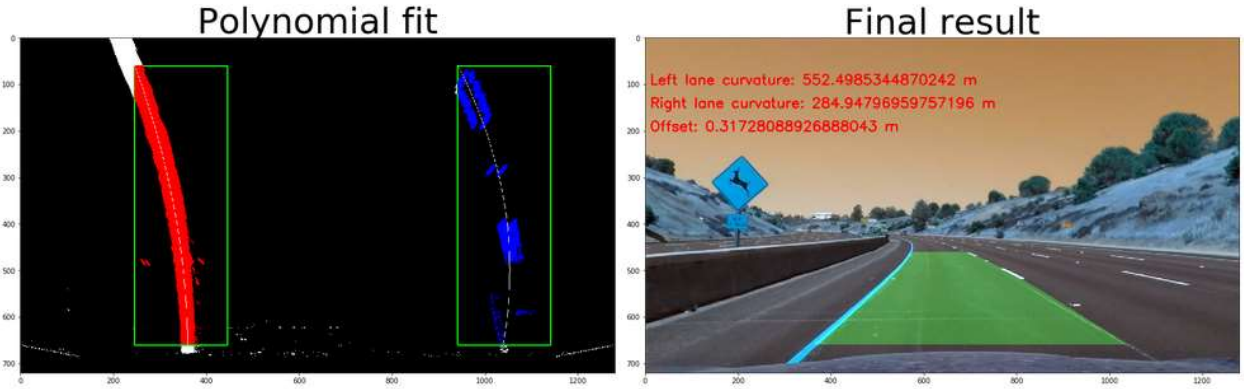
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

See method fit_polynomial() in code

```
    # For determining radius of curvature
    y_eval = 660

    ## Defines conversions in x and y from pixels space to meters
    ym_per_pix = 30.0/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension

    ## Fit new polynomials to x,y in world space
    left_fit_cr = np.polyfit(ploty*ym_per_pix, left_fitx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty*ym_per_pix, right_fitx*xm_per_pix, 2)

    left_curverad = ((1.0 + (2.0*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) /
np.absolute(2.0*left_fit_cr[0])
    right_curverad = ((1.0 + (2.0*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) /
np.absolute(2.0*right_fit_cr[0])

    # Determining offset at the center of the image
    max_y = len(ploty) - 1
    lane_mid_point_x = (right_fitx[max_y] + left_fitx[max_y])/2
    camera_x = 1280//2
    offset_pixels = np.abs(camera_x - lane_mid_point_x)
    offset_m = offset_pixels * xm_per_pix
```

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

More images in ./test_images_output folder
Here is one sample image



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!).

Please refer to ./test_videos_output/project_video.mp4 for output video

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 focus on the gradient in the x direction and s channel for thresholding and binary image generation. Any kind of prominent vertical feature has a probability
of confusing the histogram peaks that we identify for left and right line.