Advanced Lane Finding

Self Driving Car Nano Degree Program

Udacity

Robin Redhu

27 June 2021

The goals / steps of this project are the following:

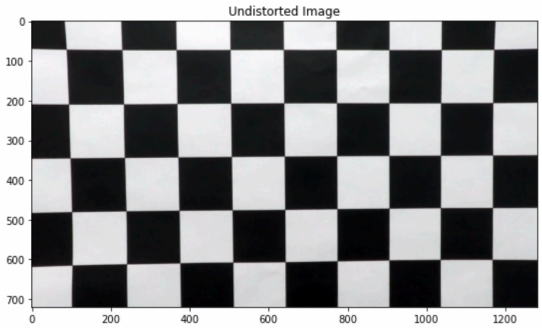
1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
2. Apply a distortion correction to raw images.
3. Use color transforms, gradients, etc., to create a thresholded binary image.
4. Apply a perspective transform to rectify binary image ("birds-eye view").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

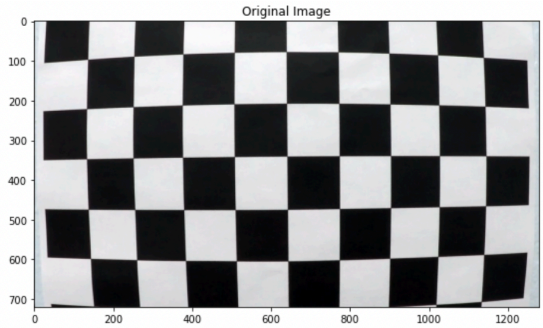
Camera Calibration

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

I started the calibration by creating the object points which represent the image points in in real world with x, y, z coordinates where z is for depth. I have kept z as 0 for all the points because in our 2D image depth will always be zero. Next I created object points and image points arrays to store the pixel location/ corners. For

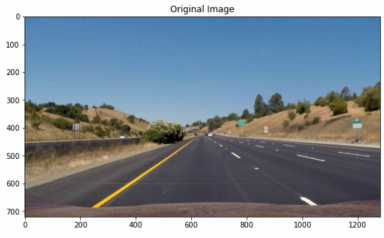
Caliberating we need a list of images with the help of which we find out the image points. And then iterating through each image we store the object points and image points respectively. Then I fed this data to cv2.findChessboardCorners() function. But for that we have to convert our image to grayscale first. Then finally we calibrated our image using cv2.calibrateCamera() function which returned a matrix to transform the 3d Image to 2D image, distortion coefficients, radial and tangential vectors. And with the help of matrix and distortion coefficients using cv2.undistort() method we have got the undistorted image.

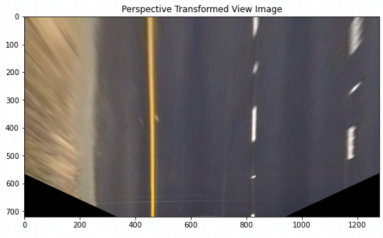




2. Provide an example of a distortion-corrected image.

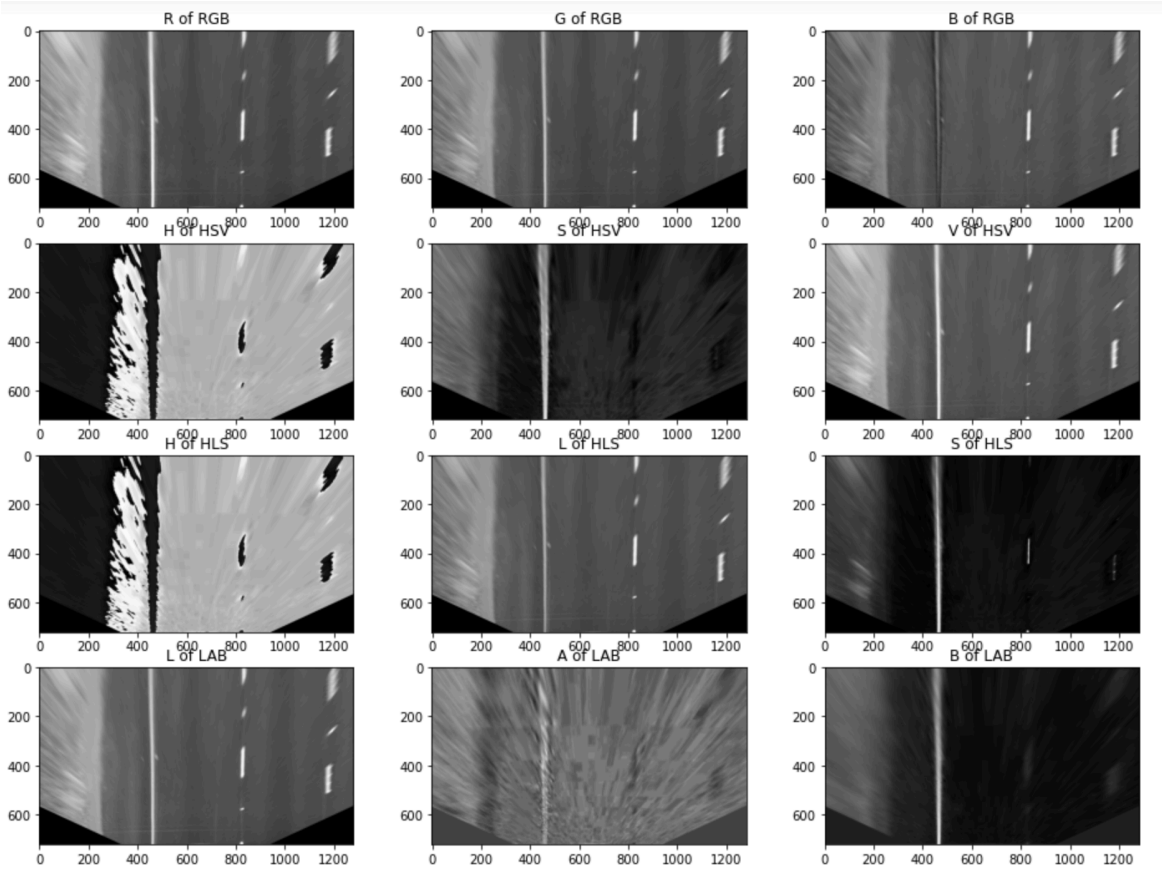
To demonstrate the above step, moreover, I will apply the distortion correction and then will get the bird's eye view (using cv2.getPerspectiveTransform() and cv2.warpPerspective()) to one of the test images like this one:





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.

I have used a variety of color spaces to create the thresholded binary image. At first, I examined RGB, HSV, HLS and LAB color spaces out of which I selected L color channel of HLS to determine the white stripped lanes and for yellow one I chose S of HSV first and then B of LAB. Check below images for your reference.



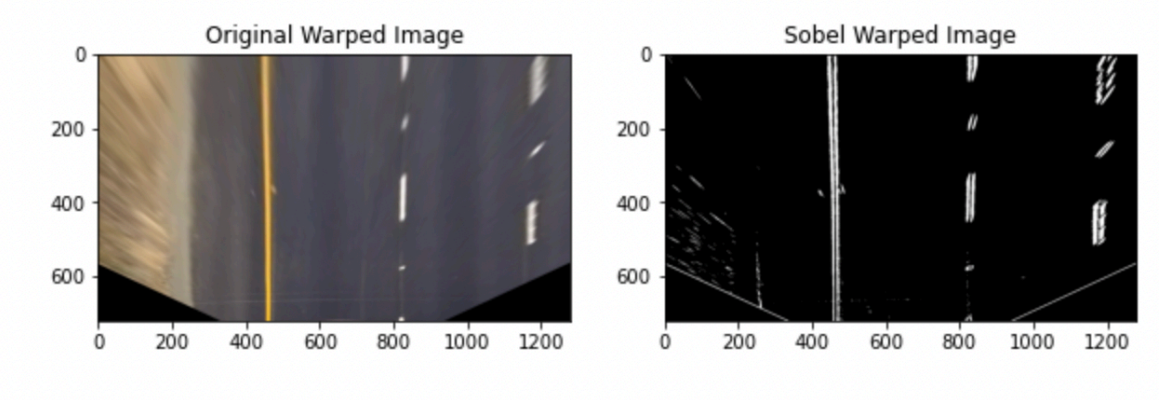
But after checking these results for other images with different lightning condition I rejected S of HSV for yellow lanes because it was creating lot of noise under shaded region and B color channel of LAB was working just fine. 🙂

Below is the piece of code used for getting different color channel

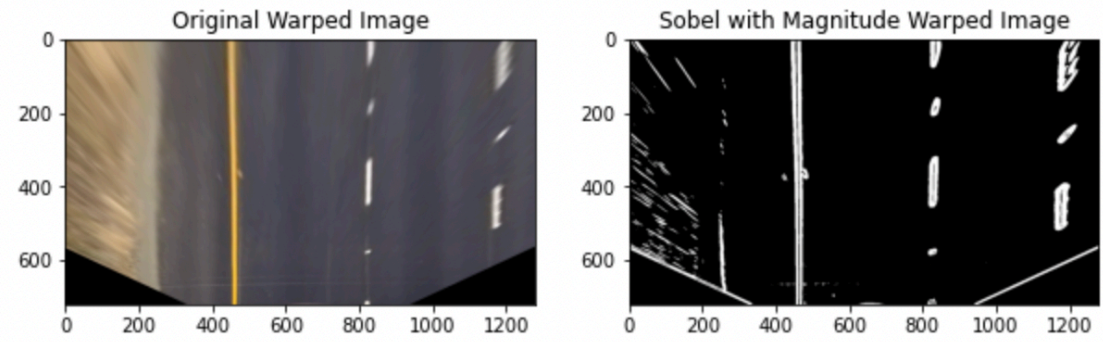
```
# Visualizing RGB color space
unwarped_R = unwarped[:, :, 0]
unwarped_G = unwarped[:, :, 1]
unwarped_B = unwarped[:, :, 2]
```

```
# Visualizing HSV color space
unwarped_hsv = cv2.cvtColor(unwarped,
cv2.COLOR_RGB2HSV)
unwarped_hsv_H = unwarped_hsv[:, :, 0]
unwarped_hsv_S = unwarped_hsv[:, :, 1]
unwarped_hsv_V = unwarped_hsv[:, :, 2]
# Visualizing HLS color space
unwarped_hls = cv2.cvtColor(unwarped,
cv2.COLOR_RGB2HLS)
unwarped_hls_H = unwarped_hls[:, :, 0]
unwarped_hls_L = unwarped_hls[:, :, 1]
unwarped_hls_S = unwarped_hls[:, :, 2]

# Visualizing LAB color space
unwarped_lab = cv2.cvtColor(unwarped,
cv2.COLOR_RGB2LAB)
unwarped_lab_L = unwarped_lab[:, :, 0]
unwarped_lab_A = unwarped_lab[:, :, 1]
unwarped_lab_B = unwarped_lab[:, :, 2]
```

Next I defined Sobel Absolute Threshold method to detect the edges using the unwrapped perspective transformed image. I have used threshold filters to separate the edges from the noisy data.

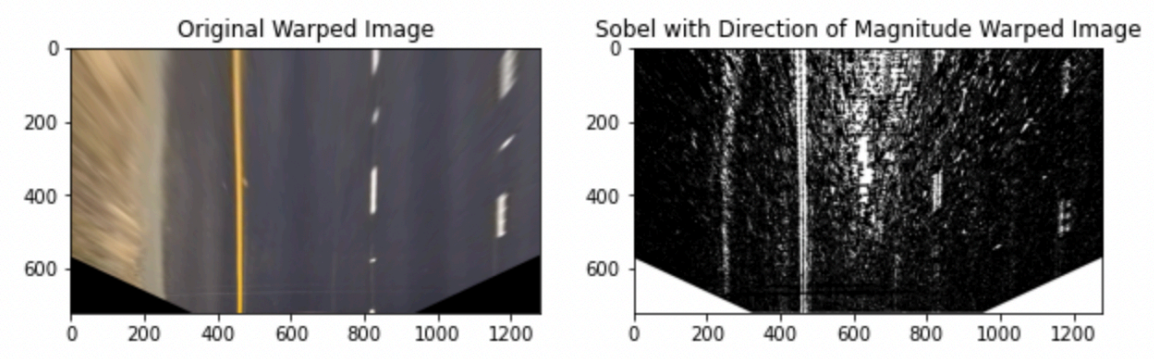


Then for better results I calculated magnitude of both sobelx and solely in both the orientation. And then again checked the results.

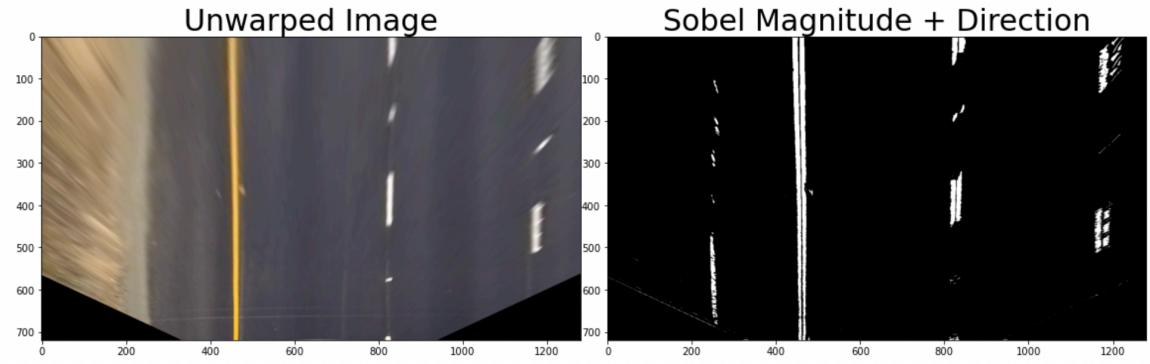


This time I got more prominent results.

And to got one more step ahead, I included direction of magnitude to detect the edges and got below results.



Yes, I know this looks horrible 😅 but what if we combine this with magnitude as well 🤨. Let's see below.



Looks good right 😟, ya I know there are some unwanted data as well. But what if focus on some particular area on the road like just in front of our vehicle as of now then? 😀 Ahh, yes that works.

But you know what, for some images it worked fine but for some, this method was not that good as our old using color channels 🥲

So finally after trying with multiple combinations I decided to go with L channel and B color channel to create the binary image. 😬

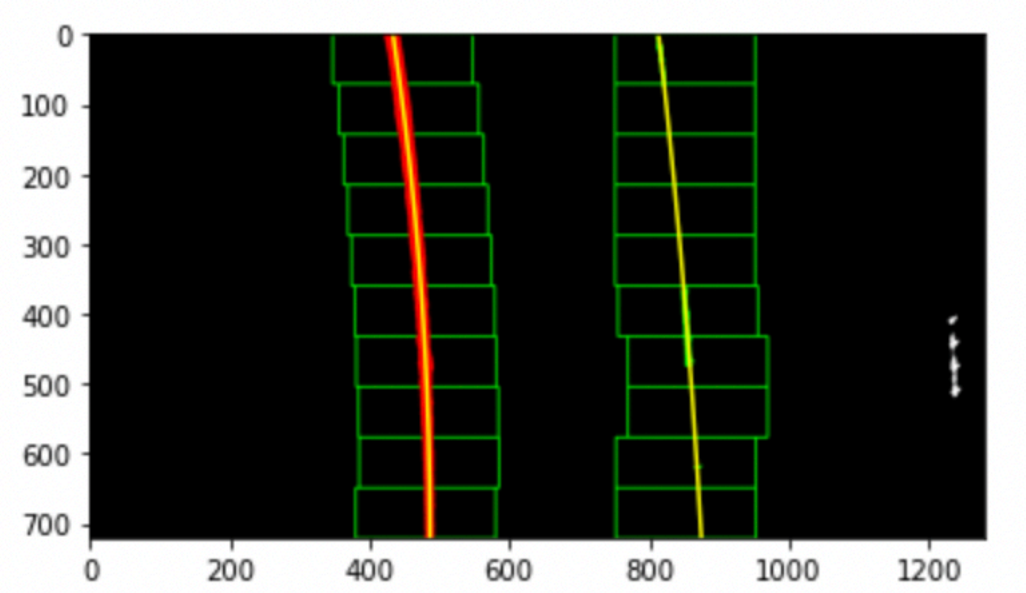
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Umm, I must say, that's an interesting question. To be honest I did nothing new just implemented the concept of applying a polynomial of 2nd degree to calculate the lane line pixels 😜 studied in Udacity SDC.

```
ploty = np.linspace(0, binary_mask.shape[0] - 1,
binary_mask.shape[0])

left_fit_x = left_fit[0] * (ploty**2) +
left_fit[1]*ploty + left_fit[2]

right_fit_x = right_fit[0] * (ploty**2) +
right_fit[1]*ploty + right_fit[2]
```

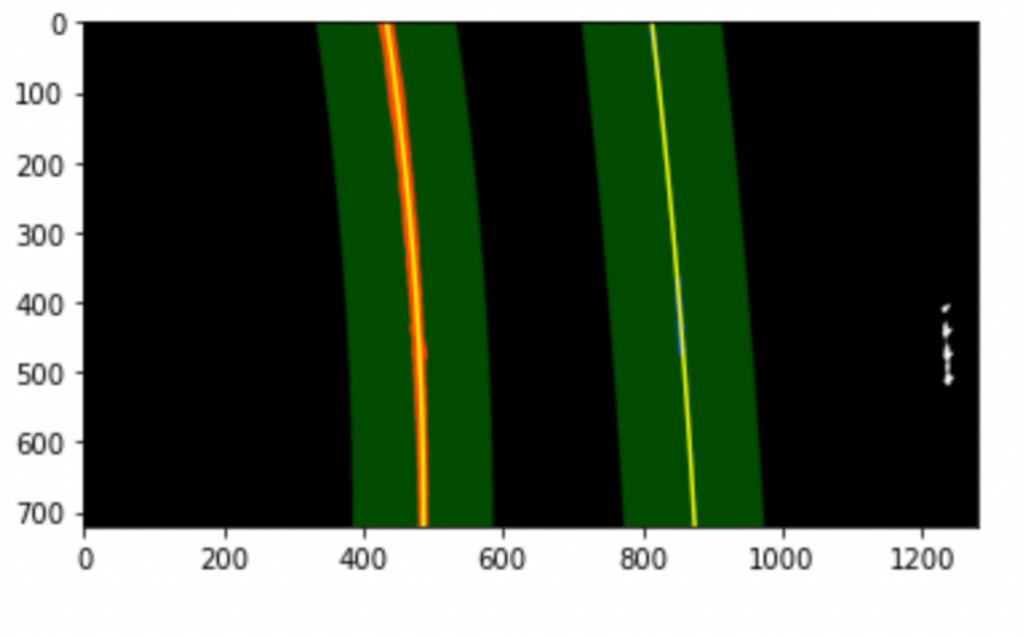
And finally I plotted them on the binary image that we got in our previous step. Ok to go in more detail, I have used 2 methods first is Sliding Window Polyfit (poly_fit()), which will detect the lane lines and fits the polynomial equation to both left lane and right lane in the first lane or when the lane is not detected in the new frame. Afterwards, our second method Poly_fit_using_ previous frame will be used. Although we can use Sliding window for whole video but that won't be an efficient way of writing code 👨🏻‍💻

Sliding Window Polyfit



Why?? Because in sliding window if you see, we have to check through all frame if our algorithm found any lane or not, however in later one, we just check in the margin are around the previous frame so this makes our code quicker and efficient🥳

Polyfit using Previous Frame



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.

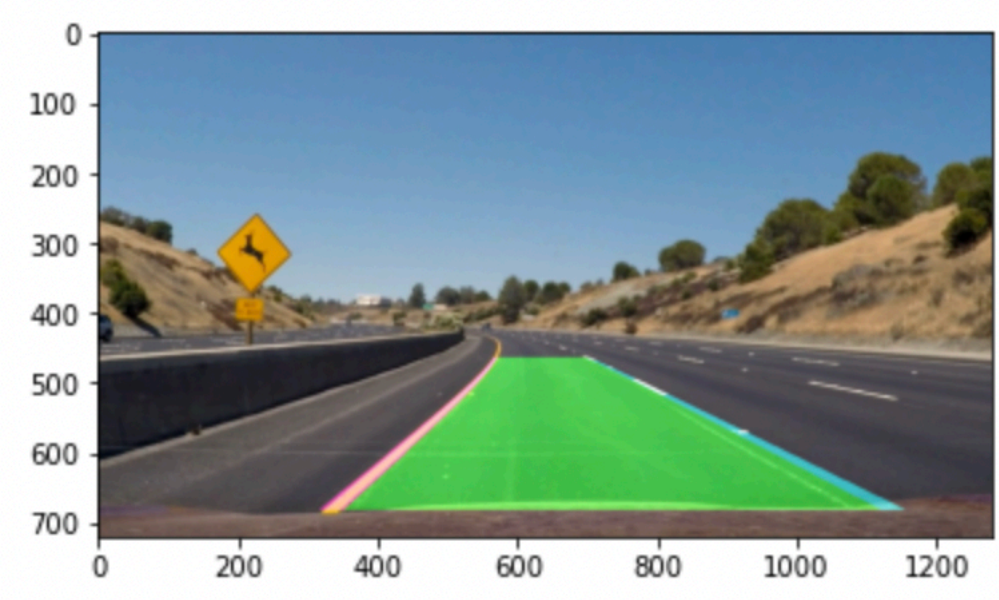
For this first I calculated metre per pixels. Then I chose the point nearest to our car where we want to calculate the radius of curvature and position of our vehicle 🚗 with respect to centre.
Next I find out the non zero pixels in the image for left lane and right lane indices to extract the left and right lane pixel coordinates.

Then using the 2nd order polynomial equation I mapped those pixel coordinates to the real world scale on road to get the exact location of our detected lane. Sounds confusing 😵, yea I know but check below, you will get it 🙂

```python
def measure_curvature_real(img, left_fit, right_fit, left_lane_inds, right_lane_inds):
    # meters per pixel in y dimension, as per given information lane line is 10 ft = 3.048 meters
    ym_per_pix = 3.048/100
    # meters per pixel in x dimension, as per given information lane width is 12 ft = 3.7 meters
    xm_per_pix = 3.7/378
    left_curverad, right_curverad, center_dist = (0, 0, 0)
    # Define y-value where we want radius of curvature. Here I have chosen y_eval the value nearest to our car
    # which is bottom of the image
    h = img.shape[0]
    ploty = np.linspace(0, h-1, h)
    y_eval = np.max(ploty)

    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = img.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    # Extract left and right line pixel cordinates
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    if len(leftx) != 0 and len(rightx) != 0:
        # Fit new polynomials to x,y in world space. Here we have first conversted thge scale to real world
        # and then found the polynomial equation of degree 2
        left_fit_cr = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
        right_fit_cr = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)
        # Calculate the new radii of curvature
        left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) /
                                                    np.absolute(2*left_fit_cr[0])
        right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) /
                                                    np.absolute(2*right_fit_cr[0])
        # Now our radius of curvature is in meters

    # Distance from center is image x midpoint - mean of left_fit and right_fit intercepts
    if right_fit is not None and left_fit is not None:
        car_position = img.shape[1]/2
        left_fit_x_int = left_fit[0]*h**2 + left_fit[1]*h + left_fit[2]
        right_fit_x_int = right_fit[0]*h**2 + right_fit[1]*h + right_fit[2]
        lane_center_position = (right_fit_x_int + left_fit_x_int) /2
        center_dist = (car_position - lane_center_position) * xm_per_pix
```

And for more details on radius of curvature check this page . This is an awesome tutorial on radius of curvature.

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



I have defined this in final function on my code and result of which looks like above 😃. Looks 🆒 right, I know.

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!).**

    Here's a link to my video result

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?

At the onset, I faced issues with selecting among the different approach to create the binary image. First, I chose Sobel threshold with magnitude and direction combined but it worked for few images and when it comes to shady area and low lightning conditions it fails terribly. Next I chose to go with color space selecting L from HLS and S from HSV color space. L was used to detect white lanes and S was for detecting yellow lanes. Initially it worked well but again with the images having shadows S color channel was failing to detect the lane. Then I decided to go with B color Channel of LAB and it does the job.

Next, in the starting it was really difficult to go with hit and trial method to get the accurate threshold values. But later I found this this method called **interact()** which literally saved my hours. 😀

Another problem I had was dealing with the lanes which were literally tortuous. My algorithm though fails terribly 🥺
Even I increased the number of windows in sliding window to 100 and 500, but wait is it worth?? 🧐 bcz it's gonna take hell lot of time. So Even I am curious to know how to deal with those fascinating roads 🛤️