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.

Camera Calibration

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

Camera matrix and distortion coefficient is obtained by implementing a function named as camera_calibration_function() which taken in chess board images. I have used object_points, which is the coordinates of chessboard in real world. The object points are same for each of the calibration images. I append these object_points to my array whenever the chess board corners are detected. The image points with be appended with pixel position of each of the corners with successful detection of chess board.

These two arrays are used to compute distortion coefficient and distortion matrix using cv2.calibrateCamera() function. I applied distortion correction using cv2.undistort() function.

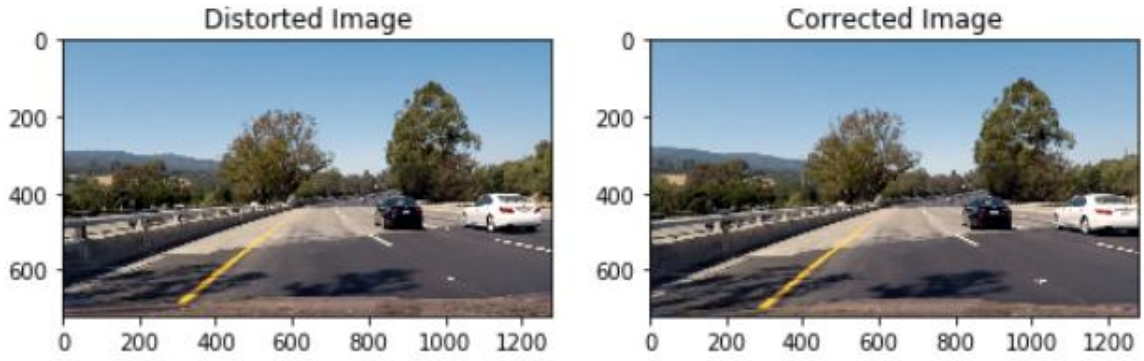
The code used for the calculations are as follows:

```
def distortion_correction(test_image):
    # read image
    #img = cv2.imread(image)
    image_size = (test_image.shape[1], test_image.shape[0])
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(object_point, image_point, image_size, None, None)
    undist_image = cv2.undistort(test_image, mtx, dist, None, mtx)
    return undist_image
```

Pipeline (single images)

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

Distortion correction is obtained as below:



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.

In binary transform section, different color spaces and gray scale transform is applied on the input image to detect lane lines. I had tried gray scale, LUV color space and LAB color space to monitor the result.

I observed that use of L channel from LUV color space helps to detect lane lines. It worked great for white line but poor for yellow lane line. So I had to

find a means to detect yellow lane lines. I tried B channel from LAB color space for this. It worked great for yellow line but poor for white lane line.

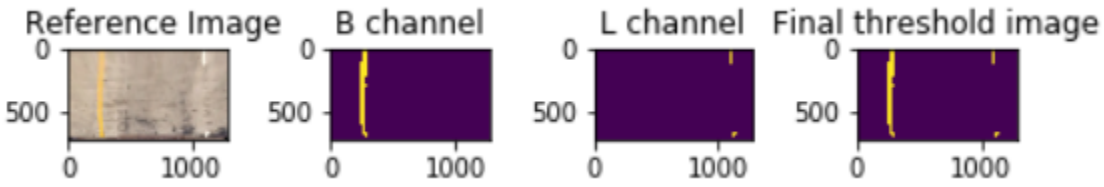
So the best means to detect lane lines are by combining L channel from LUV color space and B channel from LAB color space. The threshold values are as follows for each of the color space.

1. L channel (LUV color space): minimum threshold = 228, maximum threshold = 255.
2. B channel (LAB color space): minimum threshold = 154, maximum threshold = 204

The code for the section is as follows:

```
def threshold_function(image_1):
    undistorted_image, image, M = bird_view(image_1)


    # 1. Gray scale transform
    thresh_gray = (180, 255)
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    binary_thresh = mirror_function(gray)
    binary_thresh[(gray > thresh_gray[0]) & (gray <= thresh_gray[1])]


    # 2. LUV color space
    # I use L channel to detect lane lines.
    #It worked great for white line but poor for yellow lane line
    L_ch = cv2.cvtColor(image, cv2.COLOR_BGR2LUV)[:,:,0]
    U_ch = cv2.cvtColor(image, cv2.COLOR_BGR2LUV)[:,:,1]
    V_ch = cv2.cvtColor(image, cv2.COLOR_BGR2LUV)[:,:,2]


    L_thresh = mirror_function(V_ch)
    L_thresh[(L_ch > 227) & (L_ch < 256)] = 1
```

```
# 3. LAB color space
# I use B channel to detect lane lines.
#It worked great for yellow line but poor for white lane line
L_ch = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)[:,:,0]
A_ch = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)[:,:,1]
B_ch = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)[:,:,2]
B_thresh = np.zeros_like(A_ch)
B_thresh[(B_ch > 153) & (B_ch < 205)] = 1

# Combined binary threshold
final_thresh = mirror_function(L_thresh)
final_thresh[(L_thresh == 1) | (B_thresh == 1)] = 1
return final_thresh, B_thresh, L_thresh, image
```



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

The function bird_view is used to calculate perspective transform of the input image. function_view helps to convert image to bird's view such that we can measure lane lines and it's radius of curvature can be measures easily.

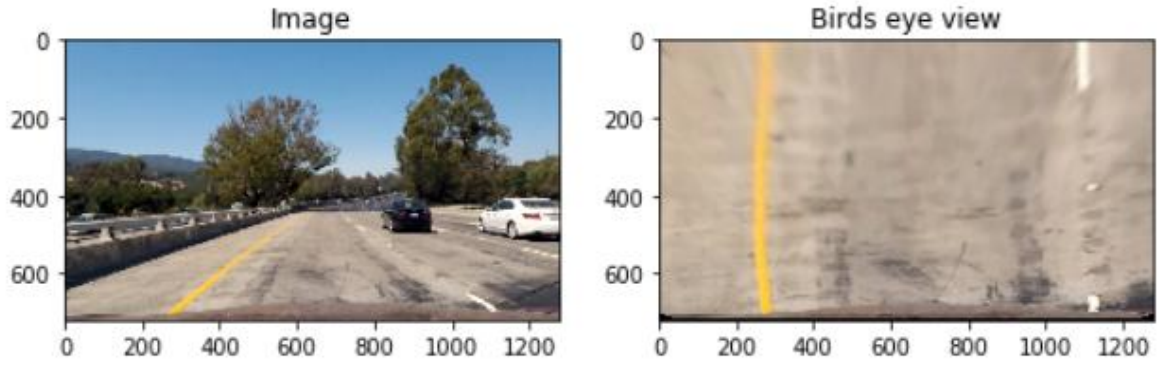
The code is as following:

```
def bird_view(img):
    undistorted_image = distortion_correction(img)
    image_size = (undistorted_image.shape[1], undistorted_image.shape[0])
```

```
source_image = np.float32([[500, 480],[800, 480],[1250, 700],[40, 700]])

destination_image = np.float32([[0, 0], [1280, 0], [1250, 700],[40, 700]])

#source_image = np.float32([[490, 482],[810, 482],[1250, 720],[40, 720]])

#destination_image = np.float32([[0, 0], [1280, 0], [1250, 720],[40, 720]])

M = cv2.getPerspectiveTransform(source_image, destination_image)

M_inverse = cv2.getPerspectiveTransform(destination_image, source_image)

warped_image = cv2.warpPerspective(undistorted_image, M, image_size)

return undistorted_image, warped_image, M
```

The function takes in 4 source points and 4 destination points to obtain M matrix and also M inverse matrix. This is obtained through cv2.getPerspectiveTransform() function. We then use cv2.warpPerspective() function to get warped image as required.

The perspective transformed image is as follows:



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

This section deals with lane_detector function() which helps to detect lane lines for both left and right and fit a polynomial into each of them. The lane

lines get detected by identifying the peaks in histogram transform of the image and monitoring pixels near these peaks. We then fir a polynomial of order two to these peak points to get the required lane lines for left and right lanes. The lane_detector() function also help to find radius of curvature, which will be explained in next section. The area between the lane line is filled and highlighted for ease of understanding to the user

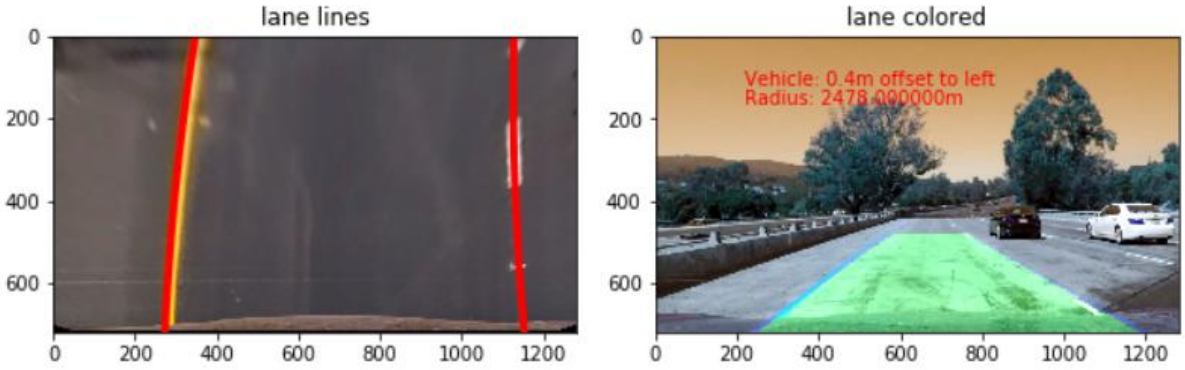
The code is as follows:

```
while band_begin > -2:
    band_sum = np.sum(final_thresh[band_begin:band_end,:], axis=0)
    argmax_l = np.argmax(band_sum[:mid_row])
    x_index = np.where((((argmax_l - band_thresh) < x_nonzero)&(x_nonzero < (argmax_l + band_thresh))&((y_nonzero > band_begin) & (y_nonzero < band_end))))
    x_window = x_nonzero[x_index]
    y_window = y_nonzero[x_index]
    if (np.sum(x_window) != 0):
        lft_x.extend(x_window.tolist())
        lft_y.extend(y_window.tolist())
    argmax_r = np.argmax(band_sum[mid_row:]) + mid_row
    x_index = np.where((((argmax_r - band_thresh) < x_nonzero)&(x_nonzero < (argmax_r + band_thresh))&((y_nonzero > band_begin) & (y_nonzero < band_end))))
    x_window = x_nonzero[x_index]
    y_window = y_nonzero[x_index]
    if (np.sum(x_window) != 0):
        rgt_x.extend(x_window.tolist())
        rgt_y.extend(y_window.tolist())

    band_begin = band_begin - 90
```

```
    band_end = band_end - 90

lft_y = np.array(lft_y)
lft_x = np.array(lft_x)
rgt_y = np.array(rgt_y)
rgt_x = np.array(rgt_x)

# fitting polynomial along the point
fit_lft = poly_func(lft_y, lft_x)
fit_lft_x = fit_lft[0]*lft_y**2 + fit_lft[1]*lft_y + fit_lft[2]
fit_rgt = poly_func(rgt_y, rgt_x)
fit_rgt_x = fit_rgt[0]*rgt_y**2 + fit_rgt[1]*rgt_y + fit_rgt[2]
rgt_x_t = fit_rgt[0]*720**2 + fit_rgt[1]*720 + fit_rgt[2]

rgt_x = np.append(rgt_x, rgt_x_t)
rgt_y = np.append(rgt_y, 720)
rgt_x = np.append(rgt_x, fit_rgt[0]*0**2 + fit_rgt[1]*0 + fit_rgt[2])
rgt_y = np.append(rgt_y, 0)

lft_x_t = fit_lft[0]*720**2 + fit_lft[1]*720 + fit_lft[2]
lft_x = np.append(lft_x, lft_x_t)
lft_y = np.append(lft_y, 720)
lft_x = np.append(lft_x,fit_lft[0]*0**2 + fit_lft[1]*0 + fit_lft[2])
lft_y = np.append(lft_y, 0)

srt_lft = np.argsort(lft_y)
srt_rgt = np.argsort(rgt_y)
```

```
lft_y = lft_y[srt_lft]
lft_x = lft_x[srt_lft]
rgt_y = rgt_y[srt_rgt]
rgt_x = rgt_x[srt_rgt]
fit_lft = poly_func(lft_y, lft_x)
fit_lft_x = fit_lft[0]*lft_y**2 + fit_lft[1]*lft_y + fit_lft[2]
fit_rgt = poly_func(rgt_y, rgt_x)
fit_rgt_x = fit_rgt[0]*rgt_y**2 + fit_rgt[1]*rgt_y + fit_rgt[2]
```

The resultant image is shown below:



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 calculated the radius of curvature using the formula mentioned in the Udacity course. By fitting points near a section of a curve, we can draw a circle. We can measure the radius of curve as radius of this circle. As we traverse along the curve, the radius changes too accordingly. We can calculate the radius at point x for curve y(x) as following:

$$\text{Radius of curvature} = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

[Reference: https://www.intmath.com]

The code is as follows:

ym_per_pix = 0.0416666667 # equation to calculate distance per pixel in y axis

xm_per_pix = 0.0052857143# equation to calculate distance per pixel in x axis

*left_fit_cr = poly_func(lft_y*ym_per_pix, lft_x*xm_per_pix)*

*right_fit_cr = poly_func(rgt_y*ym_per_pix, rgt_x*xm_per_pix)*

*curve_radius_lft = ((1 + (2*left_fit_cr[0]*np.max(lft_y) + left_fit_cr[1])**2)**1.5)/np.absolute(2*left_fit_cr[0])*

*curve_radius_rgt = ((1 + (2*right_fit_cr[0]*np.max(lft_y) + right_fit_cr[1])**2)**1.5)/np.absolute(2*right_fit_cr[0])*

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

The lane_detector() function helps to plot the lane lines and also fill them so that users can identify them clearly.

The code is as below:

```
def lane_detector(image, image_link):
    final_thresh, B_thresh, L_thresh, image_1 = threshold_function(image)
    rgt_x,rgt_y, lft_x, lft_y = [], [], [], []
    x_nonzero, y_nonzero = np.nonzero(np.transpose(final_thresh))
    band_end = 720
    band_begin = 630
    band_thresh = 23
    mid_row = 640
    while band_begin > -2:
        band_sum = np.sum(final_thresh[band_begin:band_end,:], axis=0)
```

```
        argmax_l = np.argmax(band_sum[:mid_row])
        x_index=np.where((((argmax_l-band_thresh)<x_nonzero)&(x_nonzero
<(argmax_l+band_thresh))&((y_nonzero>band_begin)&(y_nonzero<band_
end))))
        x_window = x_nonzero[x_index]
        y_window = y_nonzero[x_index]
        if (np.sum(x_window) != 0):
            lft_x.extend(x_window.tolist())
            lft_y.extend(y_window.tolist())
        argmax_r = np.argmax(band_sum[mid_row:]) + mid_row
        x_index = np.where((((argmax_r - band_thresh) <
x_nonzero)&(x_nonzero < (argmax_r + band_thresh))&((y_nonzero >
band_begin) & (y_nonzero < band_end))))
        x_window = x_nonzero[x_index]
        y_window = y_nonzero[x_index]
        if (np.sum(x_window) != 0):
            rgt_x.extend(x_window.tolist())
            rgt_y.extend(y_window.tolist())
        band_begin = band_begin - 90
        band_end = band_end - 90
    lft_y = np.array(lft_y)
    lft_x = np.array(lft_x)
    rgt_y = np.array(rgt_y)
    rgt_x = np.array(rgt_x)

    # fitting polynomial along the point
    fit_lft = poly_func(lft_y, lft_x)
    fit_lft_x = fit_lft[0]*lft_y**2 + fit_lft[1]*lft_y + fit_lft[2]
    fit_rgt = poly_func(rgt_y, rgt_x)
```

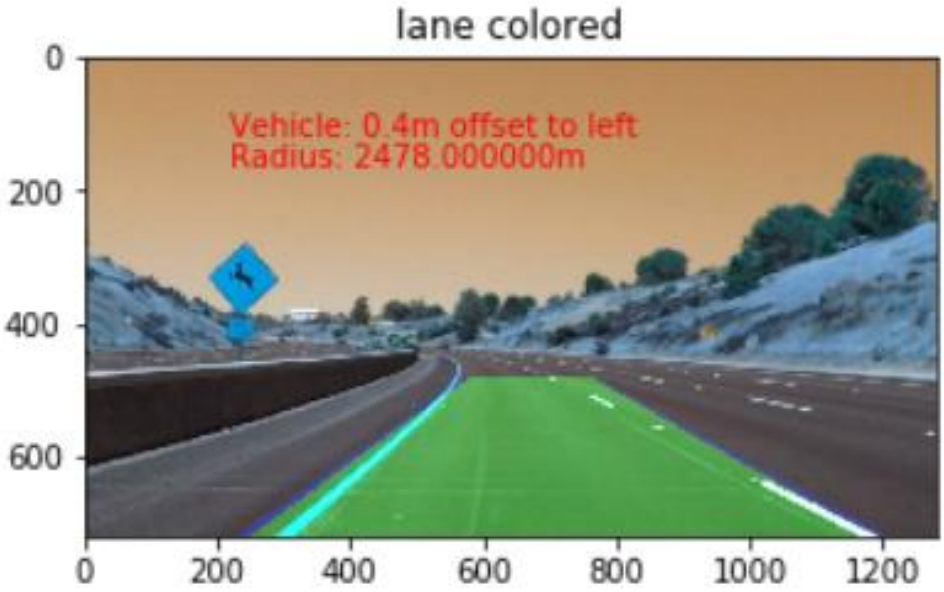
```
    fit_rgt_x = fit_rgt[0]*rgt_y**2 + fit_rgt[1]*rgt_y + fit_rgt[2]
    rgt_x_t = fit_rgt[0]*720**2 + fit_rgt[1]*720 + fit_rgt[2]
    rgt_x = np.append(rgt_x, rgt_x_t)
    rgt_y = np.append(rgt_y, 720)
    rgt_x = np.append(rgt_x, fit_rgt[0]*0**2 + fit_rgt[1]*0 + fit_rgt[2])
    rgt_y = np.append(rgt_y, 0)
    lft_x_t = fit_lft[0]*720**2 + fit_lft[1]*720 + fit_lft[2]
    lft_x = np.append(lft_x, lft_x_t)
    lft_y = np.append(lft_y, 720)
    lft_x = np.append(lft_x,fit_lft[0]*0**2 + fit_lft[1]*0 + fit_lft[2])
    lft_y = np.append(lft_y, 0)
    srt_lft = np.argsort(lft_y)
    srt_rgt = np.argsort(rgt_y)
    lft_y = lft_y[srt_lft]
    lft_x = lft_x[srt_lft]
    rgt_y = rgt_y[srt_rgt]
    rgt_x = rgt_x[srt_rgt]
    fit_lft = poly_func(lft_y, lft_x)
    fit_lft_x = fit_lft[0]*lft_y**2 + fit_lft[1]*lft_y + fit_lft[2]
    fit_rgt = poly_func(rgt_y, rgt_x)
    fit_rgt_x = fit_rgt[0]*rgt_y**2 + fit_rgt[1]*rgt_y + fit_rgt[2]
    # section to calculate the curvature radius of the lane
    ym_per_pix = 0.0416666667
    xm_per_pix = 0.0052857143
    left_fit_cr = poly_func(lft_y*ym_per_pix, lft_x*xm_per_pix)
    right_fit_cr = poly_func(rgt_y*ym_per_pix, rgt_x*xm_per_pix)
    curve_radius_lft=((1+(2*left_fit_cr[0]*np.max(lft_y)                    +
left_fit_cr[1])**2)**1.5)/np.absolute(2*left_fit_cr[0])
```

```
    curve_radius_rgt = ((1 + (2*right_fit_cr[0]*np.max(lft_y) + right_fit_cr[1])**2)**1.5)/np.absolute(2*right_fit_cr[0])


    # vehicle is positioned in the center of the lane
    center = abs(mid_row - ((rgt_x_t+lft_x_t)/2))
    source_image = np.float32([[500, 480],[800, 480],[1250, 700],[40, 700]])
    destination_image = np.float32([[0, 0], [1280, 0], [1250, 700],[40, 700]])
    #source_image = np.float32([[490, 482],[810, 482],[1250, 720],[40, 720]])
    #destination_image = np.float32([[0, 0], [1280, 0], [1250, 720],[40, 720]])
    M_new = cv2.getPerspectiveTransform(destination_image, source_image)
    undistorted_image_dummy, warped_image_dummy, M_new1 = bird_view(image)
    zero_warped = mirror_function(final_thresh).astype(np.uint8)
    warp_3_layers = np.dstack((zero_warped, zero_warped, zero_warped))
    lft_points = np.array([np.flipud(np.transpose(np.vstack([fit_lft_x, lft_y])))])
    rgt_points = np.array([np.transpose(np.vstack([fit_rgt_x, rgt_y]))])
    complete_points = np.hstack((lft_points, rgt_points))
    cv2.polylines(warp_3_layers, np.int_([complete_points]), isClosed=False, color=(0,0,240), thickness = 35)
    cv2.fillPoly(warp_3_layers, np.int_([complete_points]), (0,240, 0))
    final_thresh_x_value = final_thresh.shape[1]
    final_thresh_y_value = final_thresh.shape[0]
    warp_final = cv2.warpPerspective(warp_3_layers, M_new, (final_thresh_x_value, final_thresh_y_value))
    final_image = cv2.addWeighted(cv2.imread(image_link), 1, warp_final, 0.37, 0)
```

undistorted_image_1, warped_image_1, M_1 = bird_view(image)

return warped_image_1, fit_lft_x, lft_y, fit_rgt_x, rgt_y, final_image, center, curve_radius_lft, curve_radius_rgt

The resultant image is as follows:



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!).

The video is attached in the attachement.

Discussion

There is still a small trouble with the maximum value and minimum values for radius of curvature. There are problems with curves. Performance is much better than first project of udacity nanodegree program. It is not still clear how long should the lane line project to. It is only a arbitrary value chosen by user and research should be done to choose the optimal value.