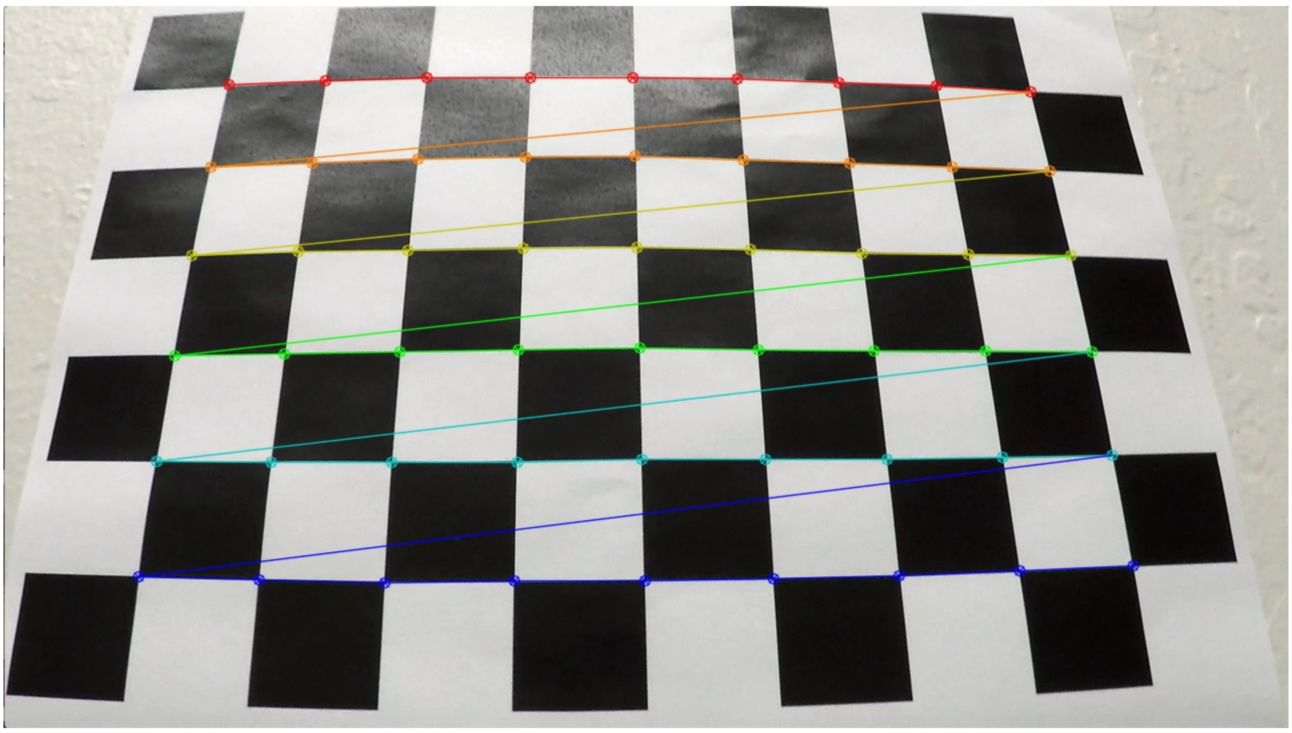
Camera Calibration

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

In order to calibrate camera, we are going to use chessboard. Chessboard is good for camera calibration. High contrast cells make it easier to detect the edges. We are going to use openCV for image calibration. OpenCV functions findChessboardCorners() and drawChessboardCorners() are used to find and draw corners in an image of a chessboard pattern.

We need the number of inside corners in rows and columns, When we have this information openCV could find and draw the corners. In our test set, 9 corners exist in a row and 6 corners exists in a column.

```
nx= 9 # corner count in x axis
ny = 6 # corner count in y axis

# Find the chessboard corners
ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)

if ret == True:
    # Draw and display the corners
    cv2.drawChessboardCorners(img, (nx, ny), corners, ret)
```



we need objpoints and imgpoints to calibrate the camera image.

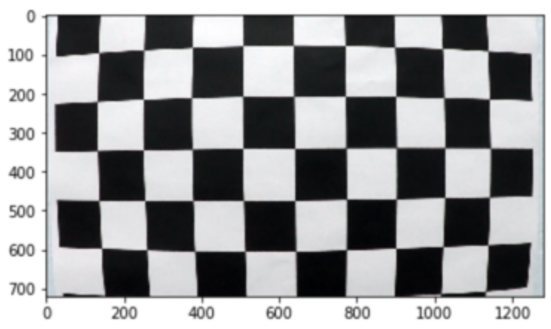
Camera calibration, given object points, image points, and the shape of the grayscale image:

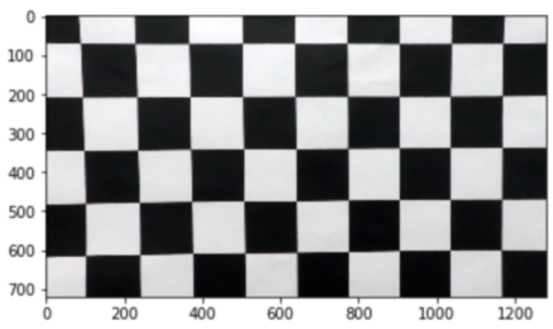
```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
```

Undistorting a test image:

```
dst = cv2.undistort(img, mtx, dist, None, mtx)
```

You can see original image at the left side, and undistorted image at the right side.

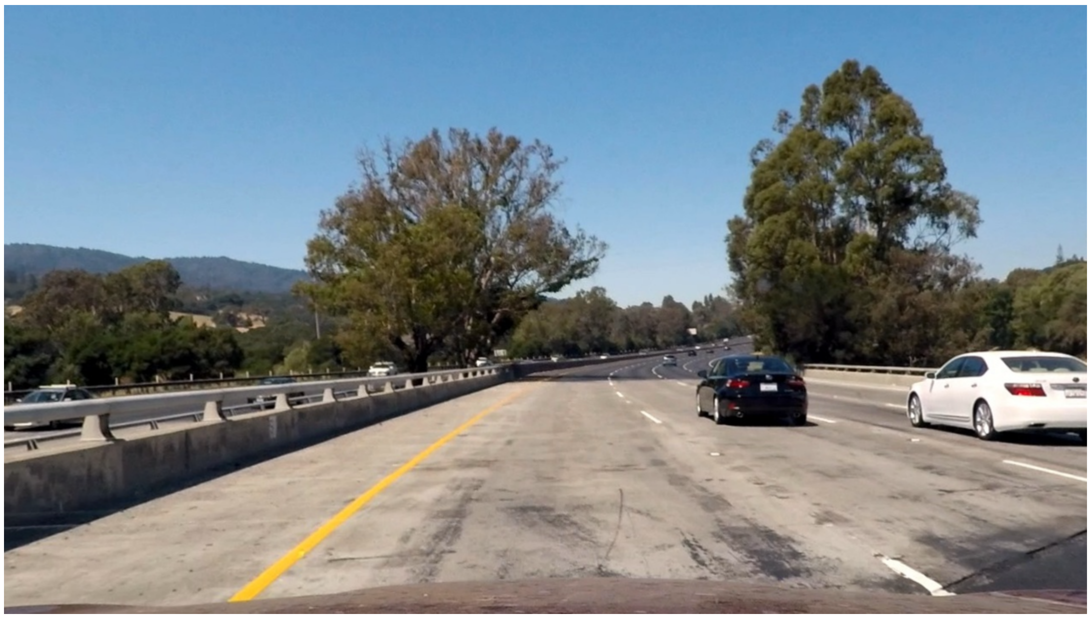




Pipeline (Test images)

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

You can find a distortion-corrected image 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.

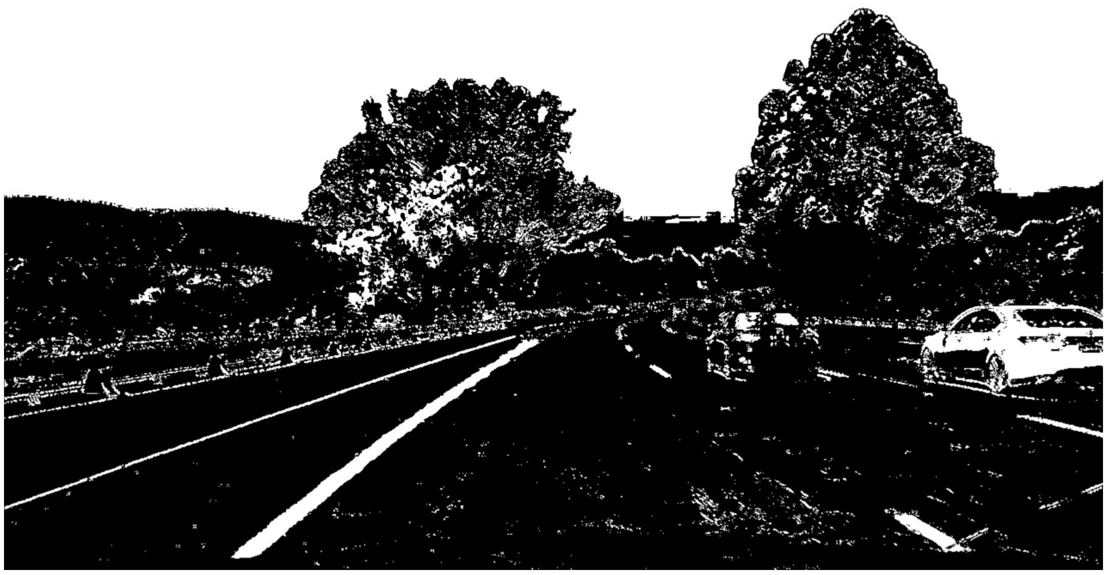
You can find a function which undistort and convert an image to a binary image.

```
def binary_image(img):

    img = cv2.undistort(img, mtx, dist, None, mtx)

    #process image and generate binary ppixels of
    preprocessImage = np.zeros_like(img[:,:,0])
    gradx = abs_sobel_thresh(img, orient='x', thresh=(12,255)) #12
    grady = abs_sobel_thresh(img, orient='y', thresh=(25, 255)) #15
    c_binary = color_threshold(img, sthresh=(100, 255), vthresh=(50, 255))
    preprocessImage[((gradx == 1) & (grady == 1) | (c_binary == 1))] = 255 # TODO

    return preprocessImage
```

in the image_gen.py file, you can find the code which generate binary pixels between line 87 and 92. Also, I used abs_sobel_thresh(line:13) and color_threshold(line: 55) function.

I have applied openCV sobel threshold for x and y orientation. I have also applied color threshold for HLS and HSV. If one of the sobel or color threshold is one, the pixel in the final image will be one, otherwise it will be zero.

You can find result image below:



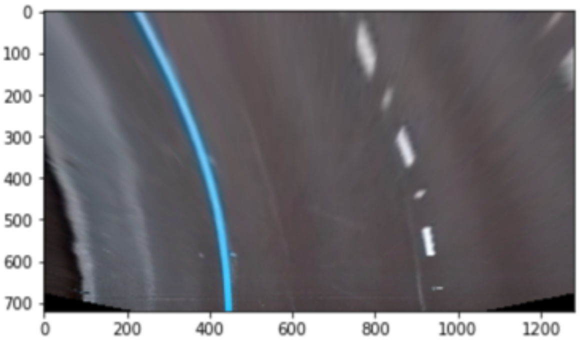
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

You can find a function which transform the perspective of the image to bird's-eye view.

```
def perspective_transform_image(img):

    img = cv2.undistort(img, mtx, dist, None, mtx)

    #mark on defining perspective transformation area
    img_size = (img.shape[1], img.shape[0])
    bot_width = .76
    mid_width = .08
    height_pct = .62
    bottom_trim = .935
    src = np.float32(
          [
             [img.shape[1]*(.5-mid_width/2), img.shape[0]*height_pct],
             [img.shape[1]*(.5+mid_width/2), img.shape[0]*height_pct],
             [img.shape[1]*(.5+bot_width/2), img.shape[0]*bottom_trim],
             [img.shape[1]*(.5-bot_width/2), img.shape[0]*bottom_trim]
          ]
       )
    offset = img_size[0]*.25
    dst = np.float32([ [offset, 0], [img_size[0]-offset, 0], [img_size[0]-offset, img_size[1]], [offset,
img_size[1]]])

    #perform the transform
    M = cv2.getPerspectiveTransform(src, dst)
    Minv = cv2.getPerspectiveTransform(dst, src)
    warped = cv2.warpPerspective(img, M, img_size, flags=cv2.INTER_LINEAR)

    return warped
```

First, you need to define the source and destination points. Four points is enough to make a perspective transform. Then you can use openCV getPerspectiveTransform function to get transformation matrix. Finally you can use openCV warpPerspective function to transform the perspective and get the warped image. You can find the codes in image_gen.py at the line 94-114.

Left-side you can see an perspective transformed image to bird's-eye view, Right-side it is also converted to binary.



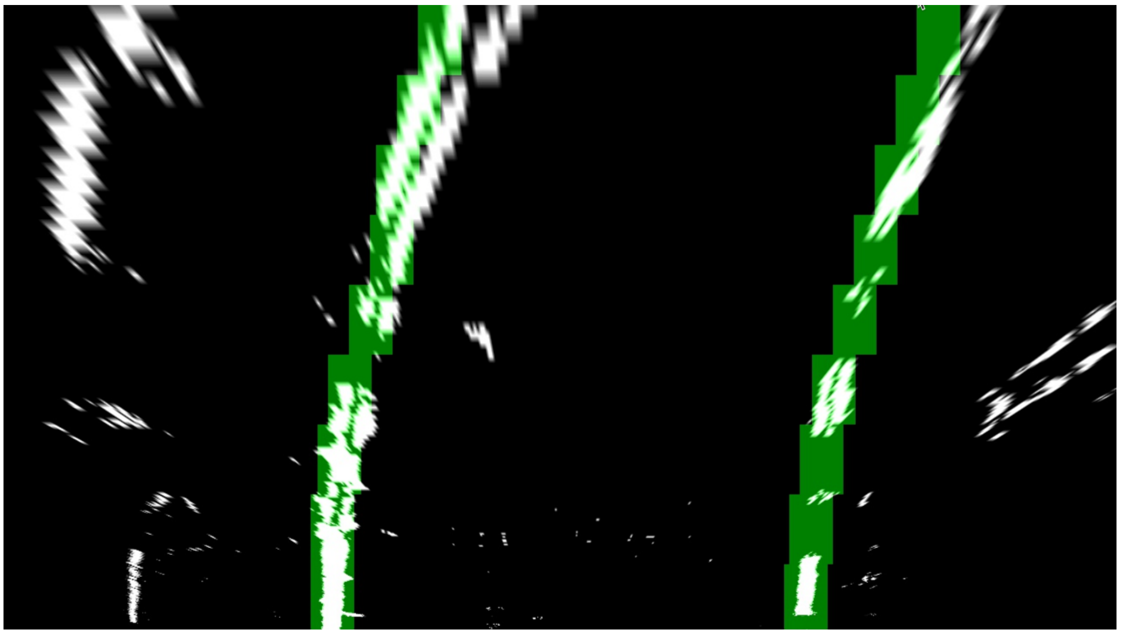
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Since the lane lines are curved, they can be represented with a polynomial. After perspective transform, we can identify the lane lines with a histogram and then we can apply sliding window places around the line centers, to find and follow the lines up to the top of the frame. We can use openCV polyfit function to fit the detected lines to a polynomial. The implementation can be found in image_gen.py file line from 119 to 178, also from tracker.py file find_window_centroids functions is used.

fit to polynomial:

```
    left_fit = np.polyfit(res_yvals, leftx, 2)
    left_fitx = left_fit[0] * yvals * yvals + left_fit[1] * yvals + 
left_fit[2]
    left_fitx = np.array(left_fitx, np.int32)

    right_fit = np.polyfit(res_yvals, rightx, 2)
    right_fitx = right_fit[0] * yvals * yvals + right_fit[1] * yvals 
+ right_fit[2]
    right_fitx = np.array(right_fitx, np.int32)
```

sliding window:



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.

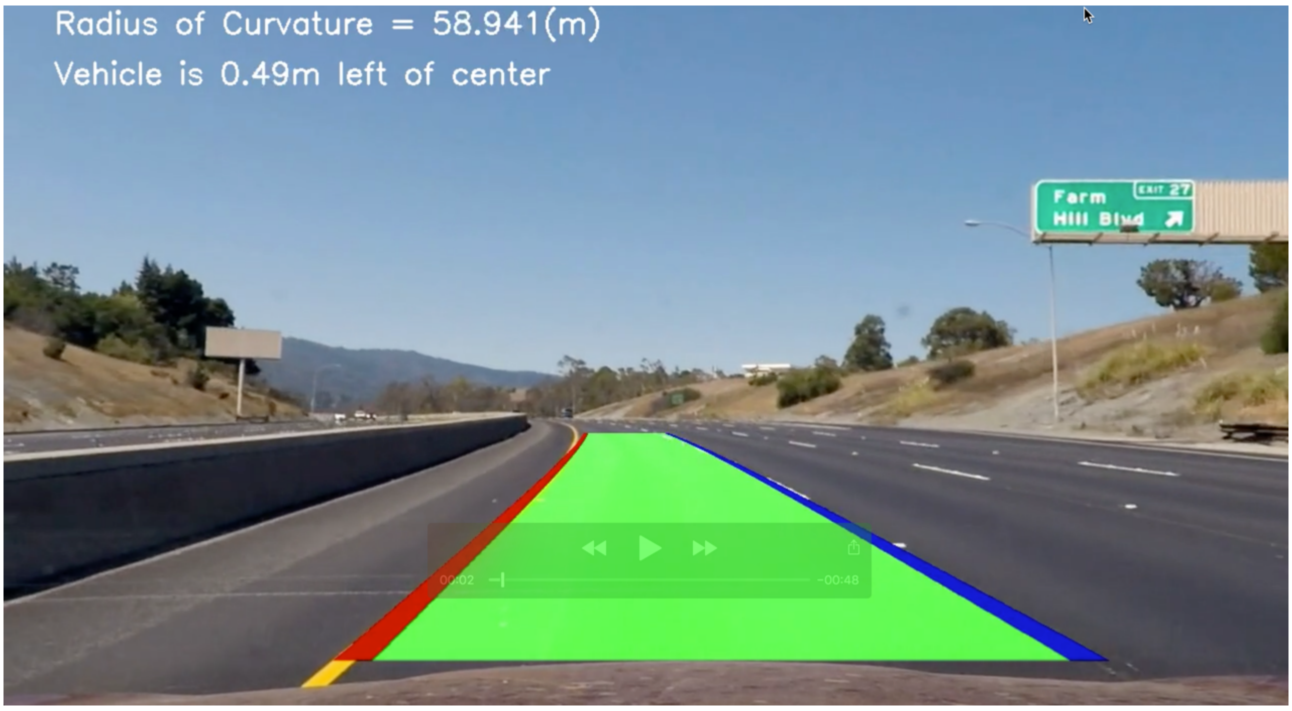
I used following lines to calculate the radius of the curvature and the position of the vehicle. I also drew them on the video. This code can be found in image_gen.py file at line 186-200.

```
    ym_per_pix = curve_centers.ym_per_pix # meters per pixel in y dimension
    xm_per_pix = curve_centers.xm_per_pix # meters per pixel in x dimension
    curve_fit_cr = np.polyfit(np.array(res_yvals, np.float32) * ym_per_pix, np.array(leftx,
np.float32)*xm_per_pix, 2)
    curverad = ((1 + (2 * curve_fit_cr[0] * yvals[-1] * ym_per_pix + curve_fit_cr[1]) **2)**1.5) /
np.absolute(2 * curve_fit_cr[0])

    #calculate the offset of the car on the road
    camera_center = (left_fitx[-1] + right_fitx[-1])/2
    center_diff = (camera_center-warped.shape[1]/2)*xm_per_pix
    side_pos = 'left'
    if center_diff <= 0:
        side_pos = 'right'

    #draw the text showing curvature, offset and speed
    cv2.putText(result, 'Radius of Curvature = '+ str(round(curverad, 3)) + '(m)', (50,50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
    cv2.putText(result, 'Vehicle is '+ str(abs(round(center_diff, 3))) + 'm ' + side_pos + ' of center',
(50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
```

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

You can find related code in image_gen.py at line 156-184.

Here is an example :


Radius of Curvature = 58.941(m)
Vehicle is 0.49m left of center

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 is a link: https://vimeo.com/262753970

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?

First, During perspective transform I have set source and destination points manually, it is not the best approach. It could be better to generate source and destination points for perspective transform. Other parameters for perspective transform could be optimized. Color and gradient threshold could be optimized.