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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Camera Calibration

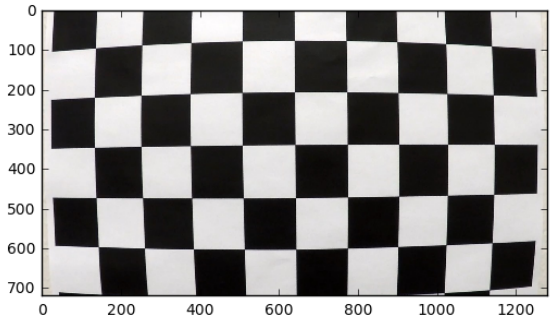
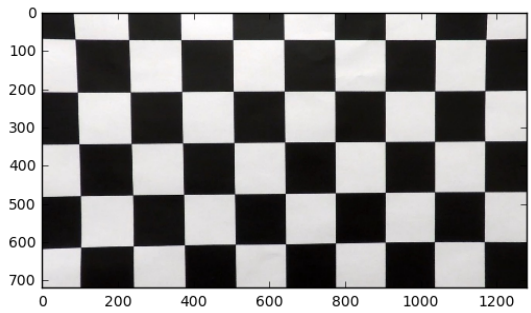
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the Camera distortion - Advanced Lane finding

Steps to undistort the image:

1. prepare "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. assuming z=0.

2. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection
3. Use objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera()
4. Apply this distortion correction to the test image using the cv2.undistort() function and obtained this result:

LEFT: Original image RIGHT: Undistorted image

Pipeline (single images)

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

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:


LEFT: Original image RIGHT: Undistorted 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.

The pipeline and video creation code is located here - Advanced Lane finding.ipynb

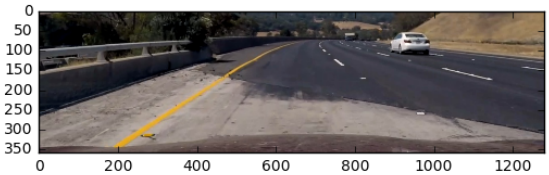
Pipeline steps:

For the pipeline lets take 2 example.

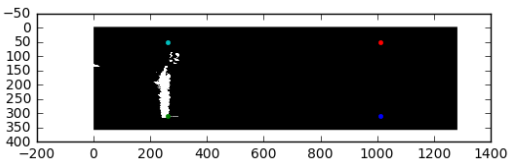
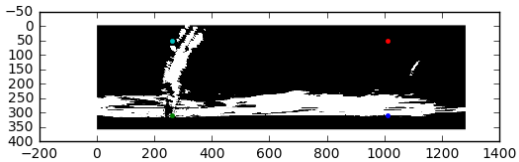
- On the LEFT - image with some color variations
- On the RIGHT - image with SHADOW

STEPS in pipeline with examples

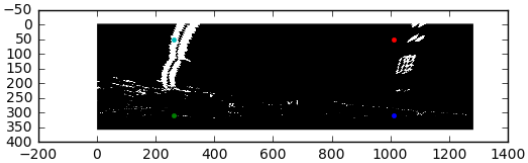
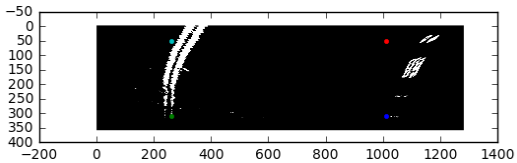
1. Undistort image (e.g shown above)
2. Crop off top half of image - example below



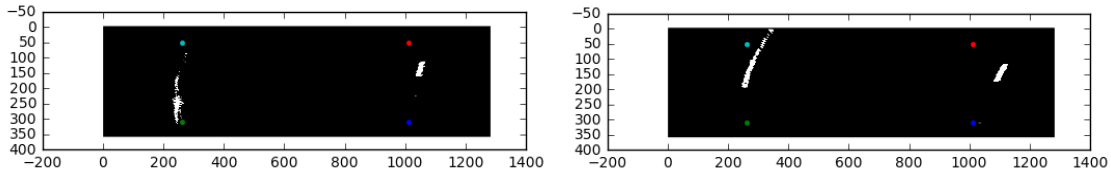

3. Do a perspective transform
4. Apply following filters (example using different image) (the dots are markers for transformation for my own example)
    - SATURATION

    - SOBEL for X axis

    - YELLOW and WHITE color filters



Important Points

- SOBEL is used to detect shadows. When we take sum of the xaxis rows, we can detect shadows when the sum is high value.

```
shadowDetected = False
        if(binaryImg['type'] == 'hls_s'):
            #detect shadow
            for tmpMultiplier in range(N_BOX//2):
                finaltmpImgHeight = tmpImg.shape[0]
                sum =
(np.sum(tmpImg[(finaltmpImgHeight-tmpMultiplier*BOX_SIZE):finaltmpImgHeight-(tmpMultiplier-
1*BOX_SIZE),:]))
                if sum > 5000:
                    shadowDetected = True
                    if debug:
                        print("SHADOW DETECTED!!!");
```

- We penalize Saturation in constructing histogram if shadow is detected

```
if shadowDetected == False:
            tmpHist = tmpHist * 2
```

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

The code for pipeline the following does perspective transform

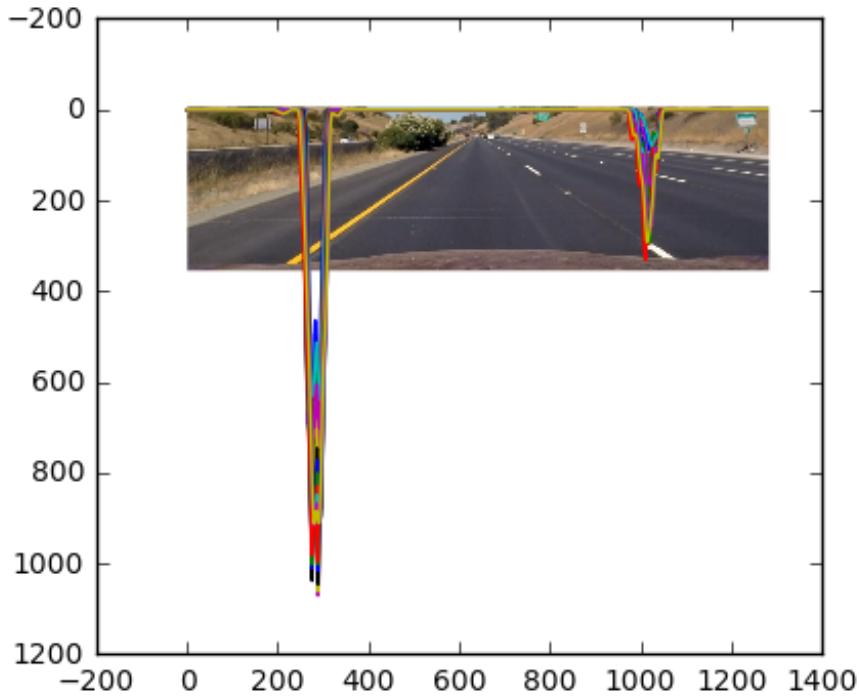
```
    binaryImg['final'] = perspectiveTransform(binaryImg['img'],False,'gray')
```

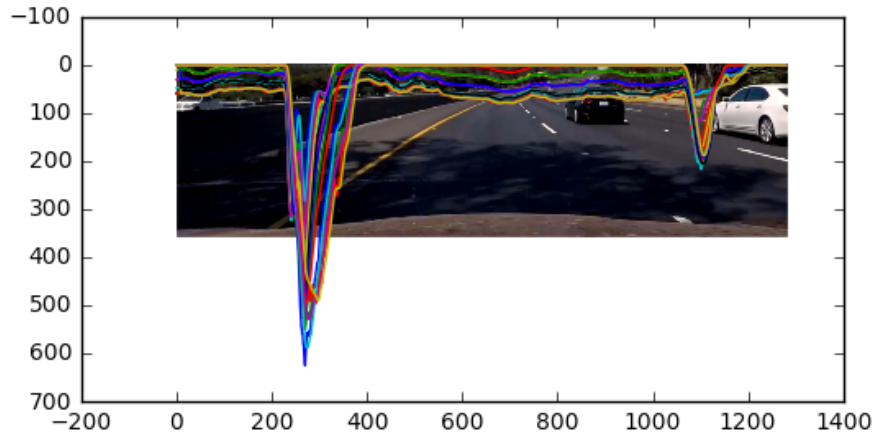
Above in (2) i provided an example on the final image. We declare source points based on image size. I put a physical dot marker and eyeballed the lane. We can improve this in future iteration. Then the destination points are derived through assuming that the points represent a straight lane

```
srcPoints =
np.array([[img_size[0]-270,img_size[1]-50],[img_size[0]-1020,img_size[1]-50],[img_size[0]-562,img_size[1]-249],[img_size[0]-718,img_size[1]-249]],np.float32)

destPoints =
np.array([[srcPoints[0][0],srcPoints[0][1]],[srcPoints[1][0],srcPoints[1][1]],[srcPoints[0][0],50],[srcPoints[1][0],50]],np.float32)
```

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

After the step (4) show in the section (2) the histogram is constructed. Below is an example of how it might look for normal and shadow images when overlayed over original image.





NOTE: These histograms are averaged out.. I.e a moving average is constructed to smooth them

```
def moving_average(a, n=3):
    ret = np.cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n
```

Later in the pipeline - below is used

```
        tmpMovingAvg = moving_average(tmpHist,n=MOVING_AVG_CONST)
```

For Lane detection the following steps are taken:

- Calculates the peaks

We use opencv **find_peaks** :

```
signal.find_peaks_cwt(histBinary[i], np.arange(1,10))
```

- identifies which of the peaks is the correct peak for left and right lane using standard deviation and mean. it assumes 1st datapoint and verifies assumption and changes the left/right based on this

For this we

1. Divide the peaks into left and right lane peaks
2. Get highest in each lane and verify that is the peak by averaging out the PEAKS of EACH of the N Histograms produced

```
#Something went wrong in our detection. Use the other values to find something
closer to std
  if(abs(leftx_current-lmean) > 100):
      if debug:
          print("Something went wrong in our detection. LEFT LANE")
      for tmp in leftPeaks[0]["val"]:
          if(abs(tmp-lmean) <= 100):
              leftx_current = tmp + lstd//2
              break

  if(abs(rightx_current-rmean) > 100):
      if debug:
          print("Something went wrong in our detection. RIGHT LANE")
      for tmp in rightPeaks[0]["val"]:
          if(abs(tmp-rmean) <= 100):
              rightx_current = tmp + rstd//2
              break
```

3. Again make sure the left and right lane identified are correct by averaging out over LAST N (N=10) iterations. It should not wobble too much. This is done using

   ```
   global leftx_current_lastN
   global rightx_current_lastN
   ```

   We track over last N frames. If its indeed wrong lets take the mean over last 10

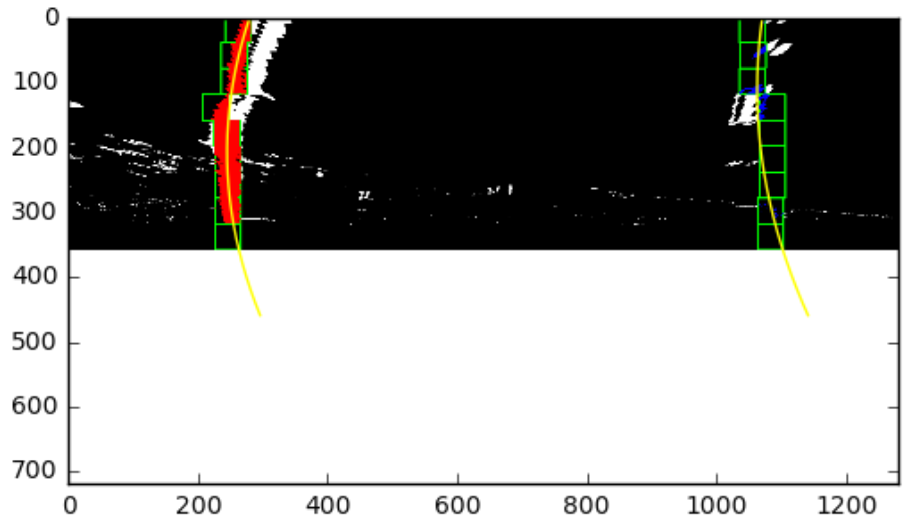
   ```
   #if Something went wrong in our detection. Lets use our mean
   if(abs(leftx_current-left_lastNmean) > 100):
       if debug:
           print("Something went wrong in our detection. LEFT LANE. taking mean")
       leftx_current = left_lastNmean

   if(abs(rightx_current-right_lastNmean) > 100):
       if debug:
           print("Something went wrong in our detection. RIGHT LANE. taking mean")
       rightx_current = right_lastNmean
   ```

4. It constructs a lane using the techniques in udacity's course by finding a set of points to FIT a 2nd order polynomial

```
# Fit a second order polynomial to each
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)
```

5. Then it calculates the radius of curvature and current position of car in the lane

```
# 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
```



After this we transform the polynomial using the Inverse transformation matrix and then fit it on the lane by drawing on it

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 did this in lines

```
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])

    lane_middle = int((rightx_current - leftx_current)/2.)+leftx_current

    img_middle = (binary_warped.shape[1]//2)
```

```
    if (lane_middle-img_middle > 0):
        vp = (lane_middle-img_middle)*xm_per_pix
    else:
        vp = (lane_middle-img_middle)*xm_per_pix

    return (left_fit,right_fit,ploty,left_fitx,right_fitx,(left_curverad+right_curverad)/2,vp)
```

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

Lane is identified as curved at ~1KM radius

Lane identified as straight. (hence radius is almost 10KM)

Lane identified as curving at ~1KM radius

Lane identified as straight

Lane identified as straight

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. It is also in the Advanced Lane finding.ipynb at last

---

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?

Following points are some of the problems/opportunities that I could use of next

1. Reliance on histogram peaks
   The cv2.find_peaks works on histogram. But it is as good as the data. The supplied data should have proper peaks at the lane. If for some reason in the combination is HLS/SOBEL/COLOR_ISOLATION combination we are still not able to differentiate the lane its a problem. We actually isolate shadows but thats just one part. There are multiple other points which might put off like road color is wrong, extra pains in middle of lane, noise in pictures etc. We should devise clever ways using a combination of further image processing and tracking lanes based on previous lane positions better
2. Choosing lane starting points
   This one is important for the lane curvature. Since we move the points inside the bounding box based on MEAN, we should consider use cases where there are lot of noise. In this case its better to use historical lane data.
3. Lanes which curve too much like in extra_challenge_video
   This is linked to above point, but when lanes curve too much, the algorithm prediction breaks because the curvature looks like a shadow to the current algorithm. We should correct this by making sure we cross examine other channels
4. Predicting speed of car to know how far to see in lane
   This will help us in knowing how far should we predict lanes. In case of FAST driving cars, we can predict a lot further since the roads wont curve too much. In case of SLOW cars, it usually means there is a turn and we should be more careful and conservative in lane prediction