laneDetectionProject

gvenkatasaik

August 2017

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

Ans. Opencv function cv2.calibrateCamera is used for finding the camera matrix and the distortion coefficients. 9 x 6 chessboards are considered for the calibration of the camera.

Figure 1: Distorted Image

Figure 2: 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.

Ans.

1. The image color channel is converted from the RGB color channel to the HLS color channel.
2. As discussed in the lectures, S channel resulted in better recognition of the lane lines.
3. For gradients, Sobel operator is applied only for the gradients in the x direction which highlights the vertical lines.
4. Threshold values used on the S channel are: [170,255]

Figure 3: Undistorted image

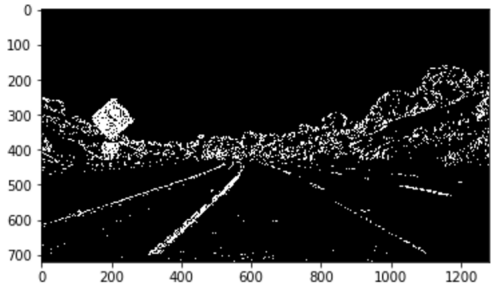

Figure 4: After applying thresholds

5. Threshold values used on the gradient image are: [20,100].

```
def edges(image):
    # HLS Conversion
    image_hls = cv2.cvtColor( image, cv2.COLOR_RGB2HLS)
    # Gray scale conversion
    gray_image = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY )
    # Threshold values for the gradient image
    thresh = [20,100]
    # Threshold values for the S channel thresholding.
    threshS = [170,255]
    S = image_hls[:,:,2]
    binary_image_cc = np.zeros_like(S)
    binary_image_cc[(S>threshS[0])&(S<threshS[1])]=1
    ## Application of the sobel operator in the x direction.
    sobelx = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0)
    abs_sobelx = np.absolute(sobelx)
    scaled_sobel = np.uint8(255*abs_sobelx/np.max(abs_sobelx))
    sxbinary = np.zeros_like(scaled_sobel)
    sxbinary[(scaled_sobel>thresh[0])&(scaled_sobel<thresh[1])]=1
    color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, binary_image_cc ))
    final_img = np.zeros_like(sxbinary)
    final_img[(binary_image_cc==1)|(sxbinary==1)]=1
    return final_img
```

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

Ans. Perspective transform is applied using an OpenCv function cv2.getPerspectiveTransform. The implementation is in the fucntion "warp" in the ipython notebook.

Figure 5: Undistorted image

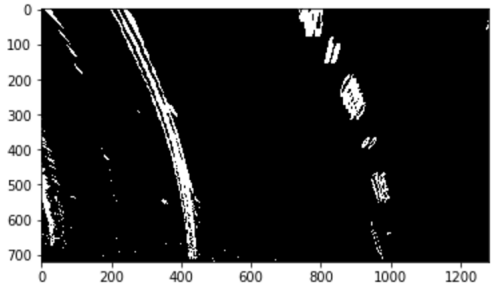

Figure 6: After applying thresholds and the perspective transform

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

Ans. This is implemented in the function "findLaneLines". The function is commented with explanations.

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.

Ans. The radius of curvature is calculated in the function "find_curve".

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

Ans. The results are plotted in (Figure 7) and (Figure 8).

7. 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!)

Ans. The video is in the Github repository.

8. 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?

Ans.

1. When the search space is not confined to the window space around the curves found initially, the recognition has gone bit off track, when the pipeline could not find definitive lane lines from the histogram.

Figure 7: Undistorted image

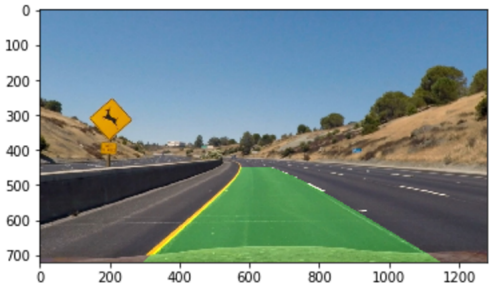

Figure 8: Image with the result plotted on to the road.

2. This pipeline may not guide the vehicle when there are no lane lines on the road.
3. This pipeline doesn't work in real time. Each iteration is taking around 3.12 seconds. Every second has approximately 26 frames. Therefore this cannot be applied in real time.
4.