Project 4 Report – Advanced Lane Finding

Rafael Barreto

INTRODUCTION

In this project each frame from the project video is processed for detection and drawing lanes on the road. For each frame, the follow steps were realized:

1. Distortion Correction
2. Combined gradient thresholds
3. Perspective Transform
4. Drawing the lines

Before processing all frames from the project video, all test images were submitted to the same pipeline to verifying if all tasks are made correctly.

CAMERA CALIBRATION AND DISTORTION CORRECTION

The first step of the project is calibrating the camera and correcting the distortion of each image. For camera calibration was used as recommended the chess board because it has high contrast pattern and is a regular shape.

The project has 20 different images of the same chess board but the OpenCV function indChessboardCorners() just had success with 17 images of this group of images. So, with the list of image points created of each image the frame of the video can be corrected decreasing the most part of distortion.

PERSPECTIVE TRANSFORM

The undistorted image was submitted to an perspective transform for a top-down view that will facilitate the location and the drawing of the lines. Four points of the undistorted imaged were chosen, called source points, and four points of desired destination. This resulted in the following source and destination points:

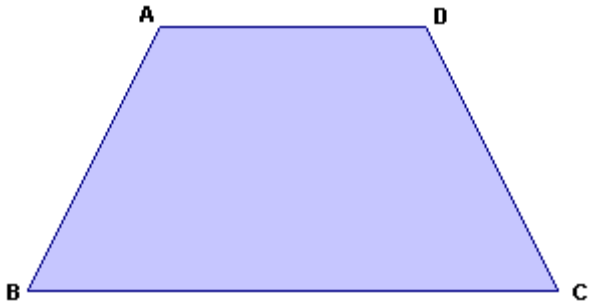

Figure 1**:** The location of the vertices.

Source points:	**Destination Points:**
A = [595, 450]	A’ = [275 - 30, 50] = [245,50]
B = [275, 720 - 50] = [275,670]	B’ = [275 - 30, 670] = [245, 670]
C = [1280 - 240, 720 - 50] = [1040, 670]	C’ = [1040+30, 670] = [1070,670]
D = [(1280 – 590), 450] = [690, 450]	D’ = [1040+30, 50] = [1070,50]

Result:

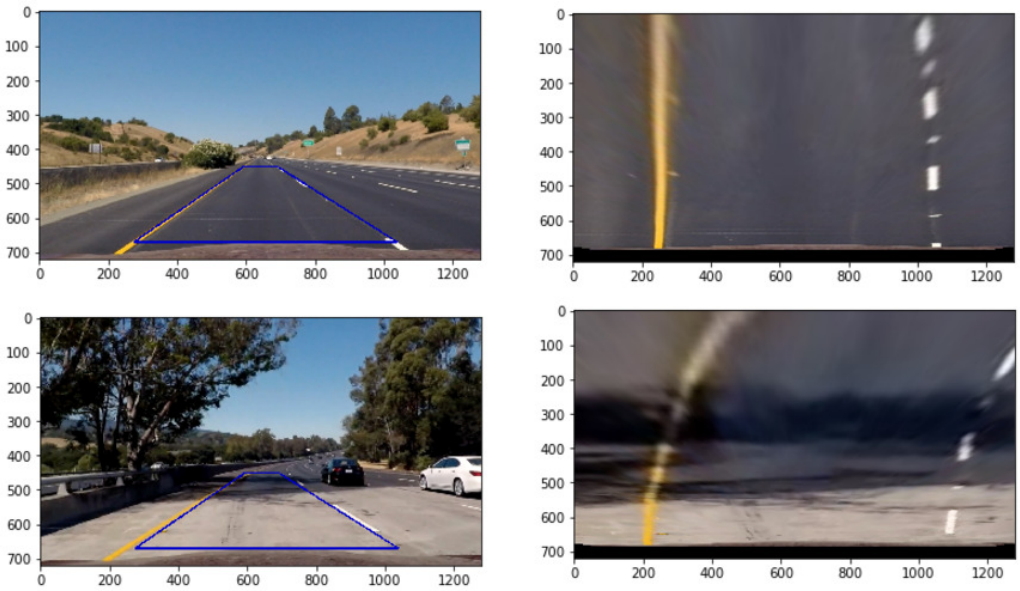

Figure 2: Result of perspective transform

EDGE DETECTION WITH GRADIENT THRESHOLDS

In this part were created 4 functions to calculate 4 different gradient thresholds. The Sobel taking the derivative of the image in the x or y direction, the magnitude of the gradient using both directions, the Direction of the Gradient threshold and the Color Thresholding using the HLS space. This last is a good option to detect yellow lanes.

The final threshold function used was a combination of the 4 previous thresholds:

```
### COMBINED THRESHOLD

# This function combines the previous types of thresholds
def My_threshold(warped_im, ksize=3, Plot=False):

    R = warped_im[:,:,0]

    thresh = (220, 255)
    binary = np.zeros_like(R)
    binary[(R > thresh[0]) & (R <= thresh[1])] = 1

    # Apply each of the thresholding functions
    gradx = abs_sobel_thresh(warped_im, orient='x', sobel_kernel=ksize, thresh=(10, 255))

    # Apply each of the thresholding functions
    yellow = select_yellow(warped_im)
    white = select_white(warped_im)

    # Combine Thresholds
    combined = np.zeros_like(yellow)
    combined[(binary == 1)] = 1
    combined[(gradx == 1) ] = 1
    combined[(yellow >= 1) ] = 1
    combined[(white >= 1)] = 1

    if Plot == True:
        histogram = np.sum(combined[combined.shape[0]//2:,:], axis=0)

        f, (ax1,ax2) = plt.subplots(1,2, figsize = (20,10))

        ax1.set_title('Binary Image')
        ax1.imshow(combined, cmap='gray')
        ax2.set_title('Histogram')
        ax2.plot(histogram)

    return combined
```

Figure 3: The Combined Threshold Function

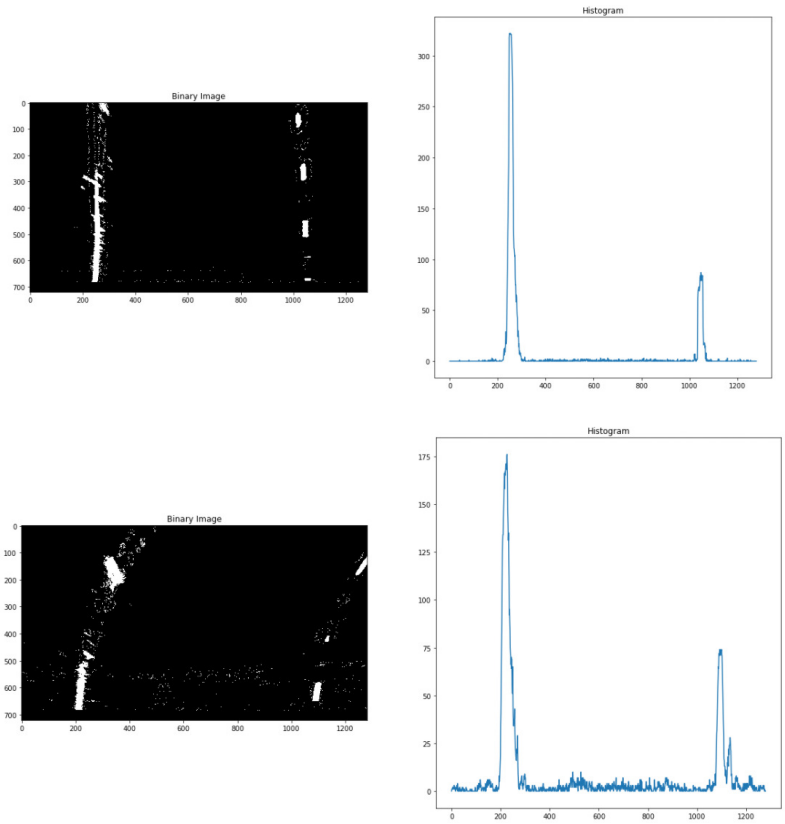

Figure 4: The binary warped images and its respective histograms.

LOCATE THE LANE LINES AND FIT A POLYNOMIAL

With the binary warped image is calculated the histogram of it, that is a graphical representation of the distribution of numerical data, in this case, the data are nonzeros pixels of this binary warped image. The two points of the histogram that represents the lines are the peaks (if a good binary warped image was generated after the combined thresholds). From those points a sliding window approach is used to find and follow the lines up to the top of the frame.

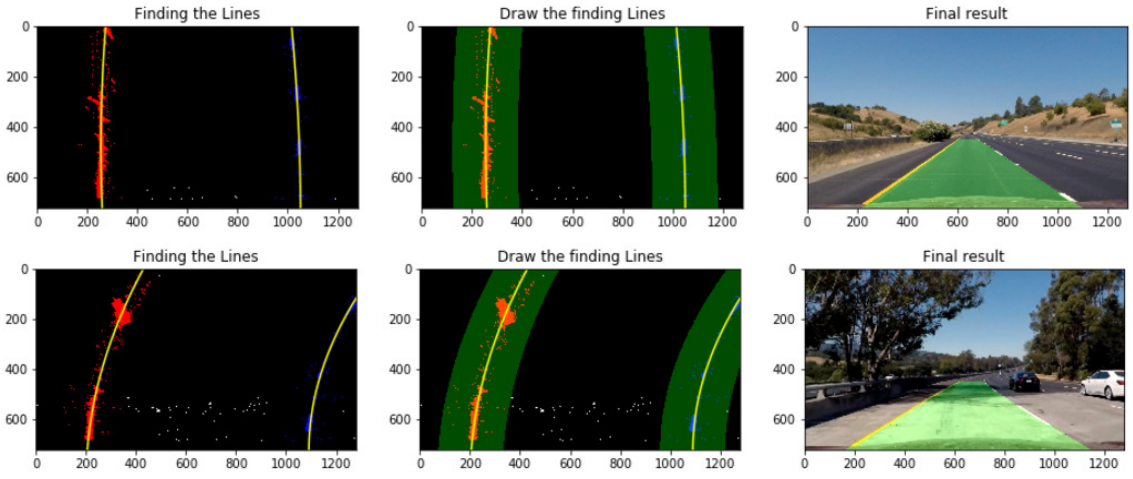

Figure 5: The results of the finding lane function.

RADIUS OF CURVATURE OF THE LANE AND THE LANE OFFSET

CURVATURE

The radius of curvature is calculated by the function:

```
### CURVATURE MEASURE

def measure_curve(leftx,rightx,ploty):

    # y_eval is in bottom of the image
    y_eval = np.max(ploty)
    bottom_index = len(ploty) - 1

    #Convert x,y pixels to meters
    ym_per_pix = 30/len(ploty)#meters per pixel in y dimension
    lane_width = np.absolute(leftx[bottom_index] - rightx[bottom_index])
    xm_per_pix = 3.7/lane_width #meters per pixel in x dimension

    #Fit new polynominals to x,y
    left_fit_new = np.polyfit(ploty*ym_per_pix, leftx*xm_per_pix, 2)
    right_fit_new = np.polyfit(ploty*ym_per_pix, rightx*xm_per_pix, 2)

    #Calculate new radius of curvature

    left_curverad = ((1 + (2*left_fit_new[0]*y_eval*ym_per_pix + left_fit_new[1])**2)**1.5) / np.absolute(2*left_fit_new[0])
    right_curverad = ((1 + (2*right_fit_new[0]*y_eval*ym_per_pix + right_fit_new[1])**2)**1.5) / np.absolute(2*right_fit_new[0])
    #print radius of curvature in meters

    curverad = (left_curverad, right_curverad)
    return curverad
```

Figure 7: Radius of curvature function.

The y values of the images increase from top to bottom, so is desired the measure of radius of curvature closest to the vehicle at y_value = image.shape[0].

LANE OFFSET

The position of the camera was considered in the middle of the image (pixel x = 640) and the lane offset is measured in the function finding_the_Lines ().The values of the lane offset calculated seems reasonable in all frames.

```
xm_per_pix = 3.7/700

if len(lefty) > 1 and len(righty) > 1:
    lane_offset = (640 - (left_fitx[-1]+right_fitx[-1])/2)*xm_per_pix
else:
    lane_offset = 0
```

Figure 8: Lane offset measure.

PROCESSING ALL FRAMES FROM VIDEO

After the successful results with all test images, all frames from the video were processed by the same pipeline and in the final, a video was rebuilt with the processed frames. The pipeline does not fail on any point but there are a few small deviations in some frames. After the deviations, the system could recover and start to find the lines correctly.

CONCLUSION

In this work, all parts of the proposed techniques in advanced lane detection were approached. Some issues come up in some frames, for instance, frames with shadows or without lines in the bottom of the image. The perspective transform distorts the lines on the top of the image and the pipeline could be more robust if this distortion were minimized.

A robust system can be achieved if the pipeline is capable to draw correct lines using day and night videos. Maybe will be necessary a different pipeline for each situation and a real autonomous car vision system.