ENPM673 - Project 2
Lane Detection

Gnyana Teja Samudrala(115824737)
Dinesh Kadirimangalam(116353564)
Sai Chaitanya Balasankula(116137294)

March 12, 2019

Introduction

The pipeline chosen for implementing the lane detection is as follows:

- Preprocess(processImage(image,crop))
  - Undistort the image
  - GaussianBlur to denoise
- Warp to get bird's view(loadHomo(image))
- Sobel filter(sobelfilt(gray_ img,l_ lim,u_ lim))
- Color thresholding(colorSpace(image,l_ lim,u_ lim))
- Blend sobel and color thresholded images(blend(im1,im2))
- Histogram to get lane candidates(identLane(image))
- Warp back
- Overlay lanes and original image(overlay(image,mask,turn,org_ img))

(The names followed in each step are the functions created for the process)
In addition to the above mentioned functions, a VideoUtils.py was created which has the function which takes in the video an saves the frames into the given directory. This was used to create the frames of both the videos for first testing out different algorithms before moving to the entire video.
In the initial trails tried to use Hough lines for the detection but due to the limitations on Region Of Interest(ROI) considered, were not able to filter out the other lanes being detected. So decided to use the histogram method.

Figure 1: The original images taken to show the pipelane

Preprocessing

processImage(image,crop)

In this the camera matrix and distortion coefficients are used to undistort the image and the resultant undistorted image is cropped to remove the black patches around. Then to get the ROI which only covers the lower half of the image that is the road part, we cropped the first 48% of the height from the undistorted image. This image is passed through Gaussian filter to denoise the image. This can be seen in the fig. 2.

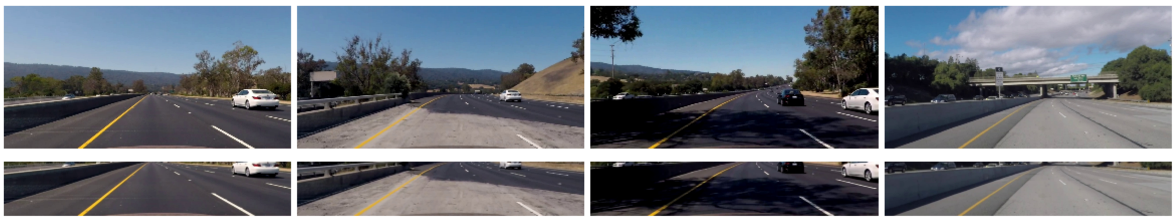

Figure 2: The first row is after removing distortions and second row is after cropping the ROI and applying Gaussian blur

Homography

loadHomo(image)

The function first tries to load the numpy array file of the homography saved as 'estHomo.npy', but if not found then tries to calculate the homography. It asks to select four points from the image being displayed using opencv. After selecting four points two from each lane in the form of a square, by clicking 'a' to confirm the point after every click and Esc key to close the window. Then from these points the homography to warp the image to top view is achieved. This homography is saved as a numpy file for further usage. The warped image using this homography is seen in the fig. 3.

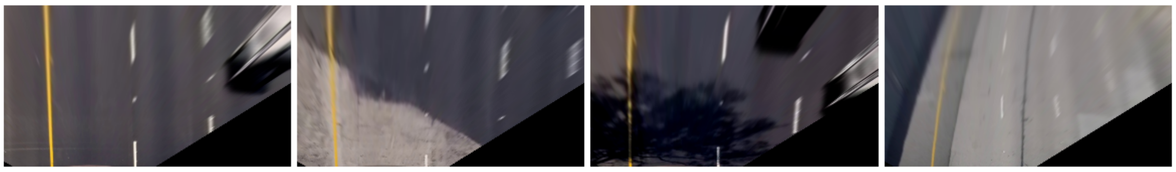

Figure 3: The images after warping to bird's eye view

Filtering and Color thresholding

sobelfilt(gray_ img,l_ lim,u_ lim)

The warped image is converted to a grayscale image and then passed through two different filters of Sobel of both x and y. These two images are converted to absolute scale and 50% of each component is added together. Then values in the range of (l_ lim,u_ lim) are made 255 and rest as zeros. This output can be seen in the figure. 4.

colorSpace(image,l_ lim,u_ lim)

Also, the normal warped color image is converted into HSL color space(Hue, Saturation and Lightness). Then from this, the saturation component is used to threshold the lanes in the warped image. Again the values in the given range are thresholded to 255. The saturation component of the image can be seen in the fig. 5 and the output after thresholding is seen in fig. 6.

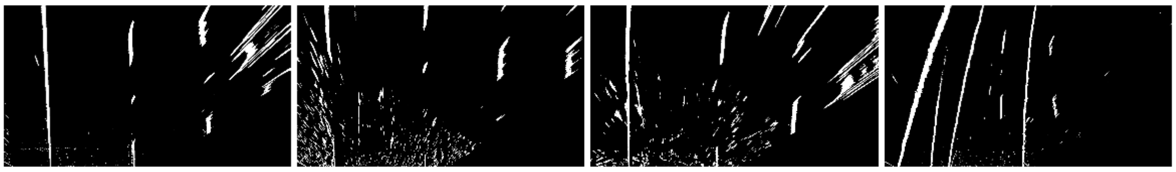

Figure 4: After sobel filtering and thresholding

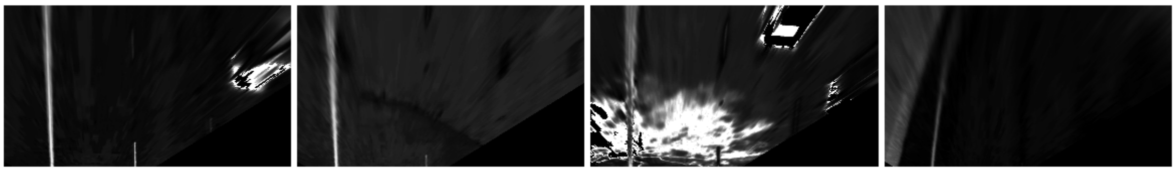

Figure 5: The 'S' component after changing color space of HSL

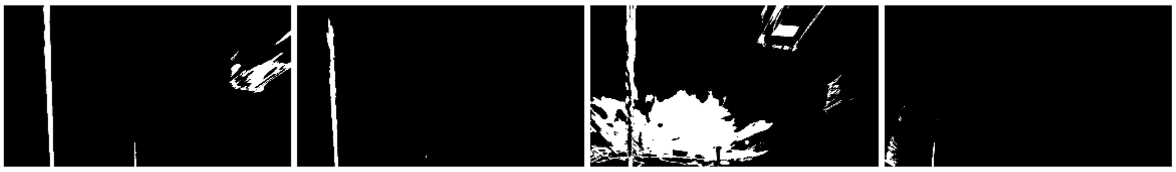

Figure 6: After color space change and thresholding

blend(im1,im2)

The thresholded images in fig. 4 and 6 are combined by performing an OR operation. This combination is done because standalone Sobel or color segmentation fails to detect the lane in few regions like when the background is changed or in presence of shadows, so the combination of these two covers almost all the parts of the lane in all the frames. The output of this step is shown in the fig. 7

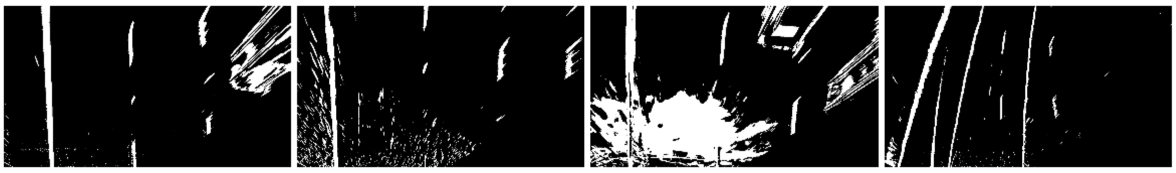

Figure 7: OR operated images between sobel and color threshold images

Histogram Lane candidates

identLane(image)

The mixed image is given as the input. A histogram of the image is made by summing up the values in the y-axis direction. There will be peaks of maxima in the graph at the x-intercept of the lane. The first maxima is searched in the range of 100 away from the starting position and the mid point of the image or x-value. We get the left lane x-intercept in the first search range. Depending on which the right lane search range is decided. The histogram in the range of +300 and +500 from the left lane x-intercept is searched to get the next maxima location where the right lane is located. These search ranges are given basing on the truth that the distance between the lanes is always fixed in the given perspective. So whenever the left lane is detected the right lane x-intercept value range can be obtained. The plots of the histogram can be visualized in the fig. 8.
After getting the x-intercept of left and right lane, two lists of (x,y) values of all the pixels which have the value of 255 in a certain specified neighborhood value of x-intercept of both left and right lanes over the entire y-range of the image is made. Then all these locations in the list are our candidate lane pixels. So these are fitted to a 2nd order polynomial one each for the lane. Then a black mask of size 500pixels more than the warped image in the y-direction is made. The extra size is just to make sure the lane detected covers the entire frame when warped back to the original image. In this black mask, the points from the 2nd order polynomial are plotted in red for entire range of y and using the command fillPoly the entire region between left and right lanes in painted green. This can be seen in the fig. 9.
To predict the turns, the slope of the line which is made by the extreme points of y of the

image and their corresponding x-values obtained from the curve fitted is used. This measure of slope value is used to predict the direction and stored to the turn variable and returned.

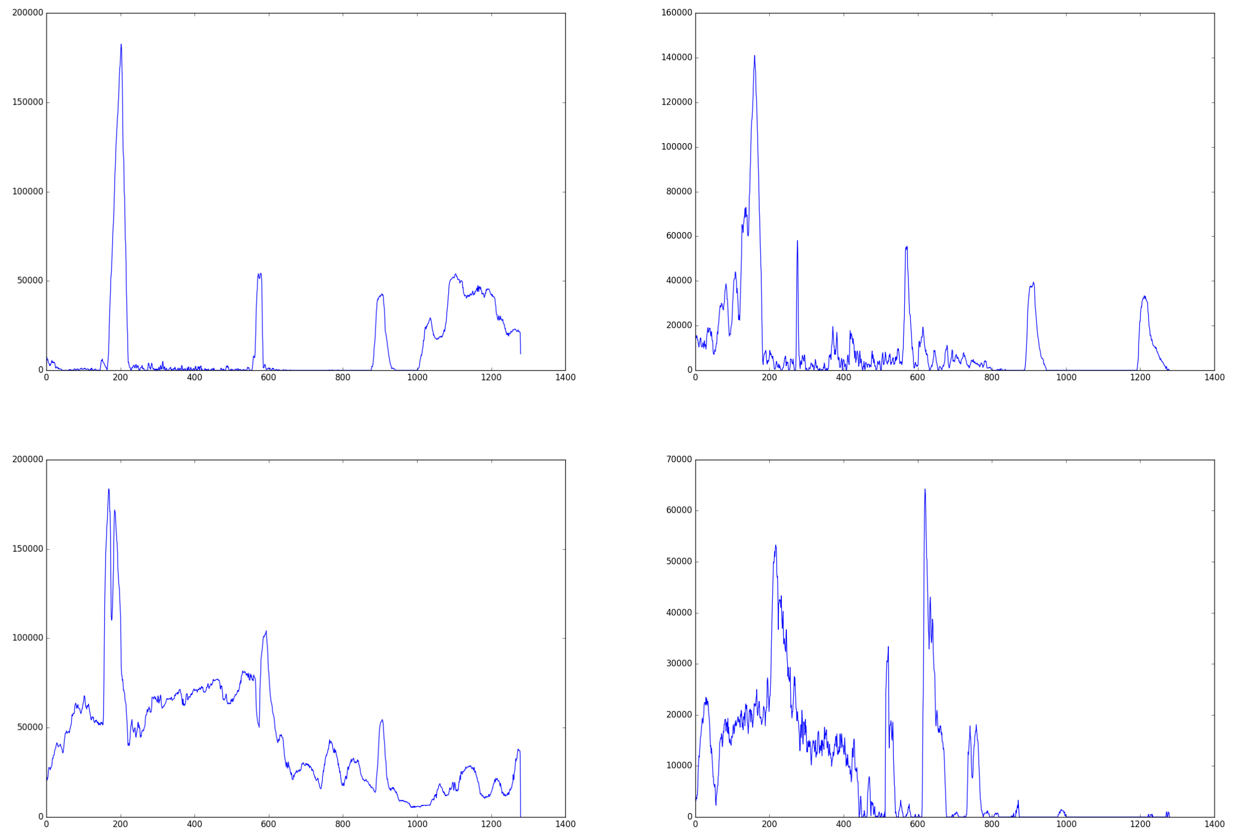

Figure 8: The histogram plots of the images in order

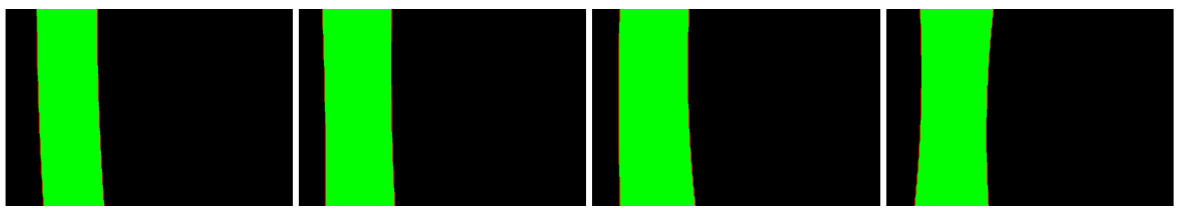

Figure 9: The lanes fitted, mask used to overlay before warpping back

Warping back and blending

overlay(image,mask,turn,org_ img)

This function takes in the warped back mask as shown in the fig. 10, and also the image on which it is to be overlayed. This is also given the variable which contains the predicted turn as a string. At first the mask is got to the same size as that of the image i.e. before cropping the ROI. Then both the images are of same dimension so they are blending by taking 50% from both the immages. Then the arrow is drawn using the arrowLine function from the centre of the lane, the direction is adjusted depending on the parameter stored in the turn variable. The final output returned by the function is visualized in fig. 11

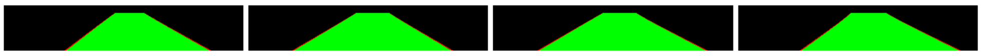

Figure 10: The lanes fitted, mask used to overlay after warpping back

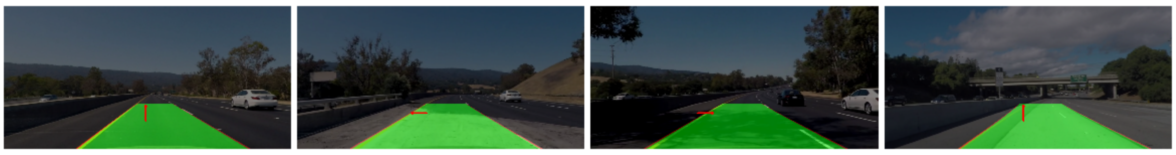

Figure 11: The lanes and direction detected

Conclusion

This pipeline will work in most of the cases given any lighting conditions and color of the road. This also works and can remain stable while changing the lanes. But may fail to be stable in the cases where the road is very patchy like in the challenge video.