National Institute of Technology Karnataka

Surathkal, Mangalore- 575025

ELECTRONICS AND ENGINEERING COMMUNICATION DEPARTMENT

Advanced Lane Detection Using Hough Lines

EC861 Image Processing And Computer Vision

Submitted to:-
Dr. AMARESWARARAO KAVURI

Submitted by :- Pratyush Raj , 221EC141.

1. Problem 1

Our aim is to improve the quality of the video sequence provided. Our input video sequence is a video recording of a highway during night. The goal is to enhance the contrast and improve the quality for the given video sequence. Our approach for this problem was as follows:

1.1 Gamma correction

Gamma correction, or often simply gamma, is a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems. Gamma correction is defined by the following power-law expression:

$$V_{out} = A * V_{in}^{\gamma} \tag{1}$$

where the non-negative real input value V_{in} is raised to the power γ and multiplied by the constant A, to get the output value V_{out}. In the common case of A = 1, inputs and outputs are typically in the range 0–1. A gamma value $\gamma < 1$ is sometimes called an encoding gamma, and the process of encoding with this compressive power-law nonlinearity is called gamma compression; conversely a gamma value $\gamma > 1$ is called a decoding gamma and the application of the expansive power-law nonlinearity is called gamma expansion.

In our case, for improving the lighting levels for our image, we are going to increase the value of gamma to a value of about 2.5. This was an experimentally determined value and we calculate the inverse of gamma and use it to create a look-up table (LUT) of size 256 and datatype 'unit8'. The values stored in the LUT are an operation involving the inverse gamma value and each element is stored as follows:

$$E_i = 255 * \frac{i}{255}^{\gamma^{-1}} \tag{2}$$

Once this table is created with associated values, we are applying this LUT to our existing image. As mentioned above, we are setting the gamma value to 2.5 and doing our element calculation. Once the Look Up table is applied to the image sequence, we do the denoising operation. We have omitted the denoising operation here as it significantly increases run time while making the video look laggy. Other than the gamma correction operation, we also tried out Histogram equalisation for our video.

1.2 Contrast Limited Adaptive Histogram Equalisation (CLAHE)

Adaptive histogram equalization (AHE) is a computer image processing technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes

```
[  0  22  29  34  38  42  45  48  51  53  55  58  60  62  63  65  67  69
  70  72  73  75  76  77  79  80  81  83  84  85  86  87  88  90  91  92
  93  94  95  96  97  98  99 100 101 101 102 103 104 105 106 107 107 108
 109 110 111 112 112 113 114 115 115 116 117 118 118 119 120 120 121 122
 123 123 124 125 125 126 127 127 128 128 129 130 130 131 132 132 133 133
 134 135 135 136 136 137 138 138 139 139 140 140 141 141 142 143 143 144
 144 145 145 146 146 147 147 148 148 149 149 150 150 151 151 152 152 153
 153 154 154 155 155 156 156 157 157 158 158 159 159 160 160 160 161 161
 162 162 163 163 164 164 164 165 165 166 166 167 167 168 168 168 169 169
 170 170 170 171 171 172 172 173 173 173 174 174 175 175 175 176 176 177
 177 177 178 178 179 179 179 180 180 180 181 181 182 182 182 183 183 183
 184 184 185 185 185 186 186 186 187 187 188 188 188 189 189 189 190 190
 190 191 191 191 192 192 192 193 193 194 194 194 195 195 195 196 196 196
 197 197 197 198 198 198 199 199 199 200 200 200 201 201 201 202 202 202
 203 203 203 204]
```

Figure 1: Look Up Table (LUT) elements

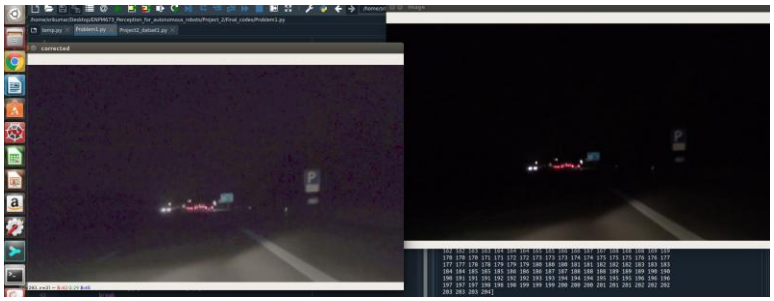

Figure 2: Comparison for video analysis

several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image, therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image.
Ordinary Histogram equalisation tends to over amplify contrasts in near-constant regions of the image, since the histogram in this region is highly concentrated. CLAHE is a variant of the adaptive HE in which this contrast amplification is limited. This solves the problem of noise amplification.

In our version of CLAHE implementation, we take the video obtained from gamma correction and use Gaussian blur filter. Now, we split the new image obtained into Hue, Saturation and Value channels and apply CLAHE operations. Once the CLAHE operations have been applied for all the channels, we go ahead and merge the three channels. After we merge the three channels, we convert the image from HSV channels to BGR channel and display. If there is a lot of noise in the image, the image is denoised using the denoising function - cv2.fastNlMeansDenoisingColored(). The final image is then used for further operations.

2 Problem 2

This is an implementation of lane detection used in the self driving cars for the application of the Lane Departure Warning System. The pipeline for the same is as follows:

Homography:
The view of the camera will be a straight meaning that the lanes will seem to meet at a vanishing point whereas, in reality, the lanes are two parallel lines. This is corrected by using Homography. Four points are chosen from one every frame of the video and we perform Homography so as to obtain a top view or Bird's view of the image captured by the camera.

Preparing the input data:
But, before performing the Homography, we need to prepare the input data. Initially the image might possibly have some distortion and we need to know the intrinsic parameters of the camera. This is given to us

and using these matrices the distortion is removed. After this now, we have an undistorted, noiseless image on which we perform edge detection. We have used the Gaussian filter to remove the unwanted disturbances and noises. And then we detected the edges using Canny edge detector.
We now need to obtain the Region of Interest (ROI) for the frames as we have a lot of unwanted details in our frames like the sky. We have written a function to remove the noise and then we have a function to detect the yellow or white lane lines, depending on whether we are using the dataset 1 or the challenge video. We then have written a function to extrapolate these lines (if they are dotted) and draw a bounded ROI. After calculating the left and right gradients, fitting a polynomial curve and drawing the mesh between the lines and then we fill it using convex Polyfill() so as to differentiate between the road and the region.

Detecting and refining the lanes:
We have used the Hough lines approach to detect the lines. In this approach, we use the polar coordinates of the image. We represent a line in the polar coordinate system as follows:

$$r_\vartheta = x_0 \cdot \cos\vartheta + y_0 \cdot \sin\vartheta \tag{3}$$

We now start plotting all the points and we if two points tend to intersect then they are on the same line. But we consider only points that are follow the below constraints:

$$r > 0 \tag{4}$$

$$0 < \vartheta < 2\pi \tag{5}$$

This means that in general, we can detect a line by finding the number of intersections between curves. We can define a threshold of the minimum number of intersections needed to detect a line. In our case we have set this threshold to be 100. So basically, we keep track of all such intersections at every point in the image and if these intersections are above the threshold then we declare it as a line. In our code we have used the Probabilistic Hough lines, this gives us the extreme points of the line.
Following this we have iterated the same thing so that these operations are performed on the all the images in dataset 1 and done for every frame in dataset 2.

Turning Prediction:
In the function that we obtain the ROI we have also written the lines that are required to detect and display if the road ahead is turning left or right or if it is straight. We have used the approach where the gradient is calculated and using this information, by taking the mean of the gradients, we check if the mean is in a particular range. Based on the range the gradients fall in we classify the direction of the car as left, right and straight.
After performing the above steps and following the pipeline we have obtained the following outputs shown step by step for dataset 1 and challenge video.

Challenges Encountered:
1. For video optimization, we tried to implement EqualizeHist() function for the split B, G, R and then merged the channels and tried to get an equalized image, but were unable to remove noise from the resultant image, satisfactorily. So, we have not implemented the equalizeHist() function.
2. We tried using CALHE function and applied that to B, G and R channels. We dint get satisfactory results (very noisy). So, we proceeded to change Hue, Saturation and Value channels using CALHE.
3. We tried to use the denoise function, but that made the video extremely laggy, making it an unoptimized video.
4. While trying to detect the white and yellow lines, we did not obtain the best results while we used the BGR images. Hence, we used the HSL form and obtained much better results.
5. We obtained better results for Hough lines when we used Canny edge detector on the masked image rather than trying to detect the lines directly using Hough lines.
6. While using Hough lines, we obtained multiple lines for a single image feature itself. We had to polyfit the lines in order to obtain a single line.

Link to the output videos:

https://drive.google.com/drive/folders/1ejW9VXp8N3V5lZbPlOCc2uSVxxau3shY?usp=sharing

Output images for dataset 1 followed by the challenge video

Figure 3: *After cropping the input image to our required ROI*

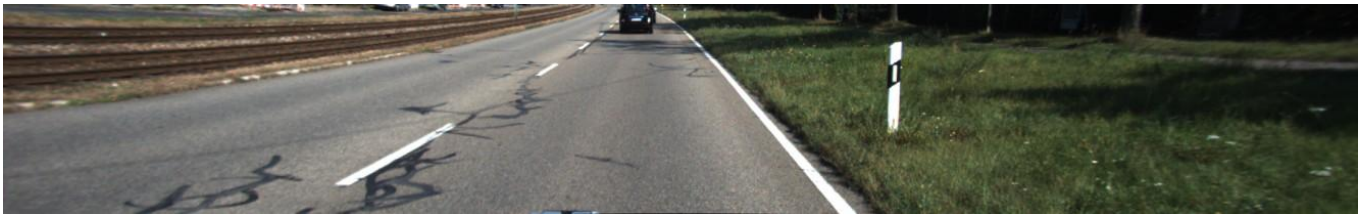

Figure 4: After undistortion on the cropped image

Figure 5: After filtering and detecting the white lines

Figure 6: After performing warping on the filtered image

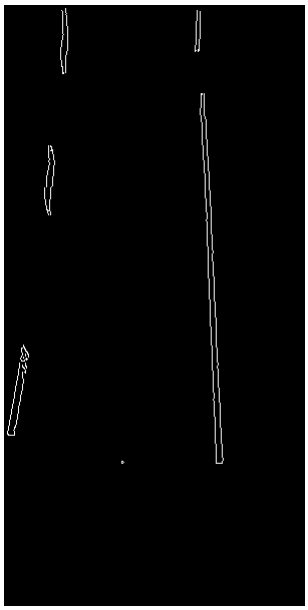

Figure 7: After detecting the edges from the warped

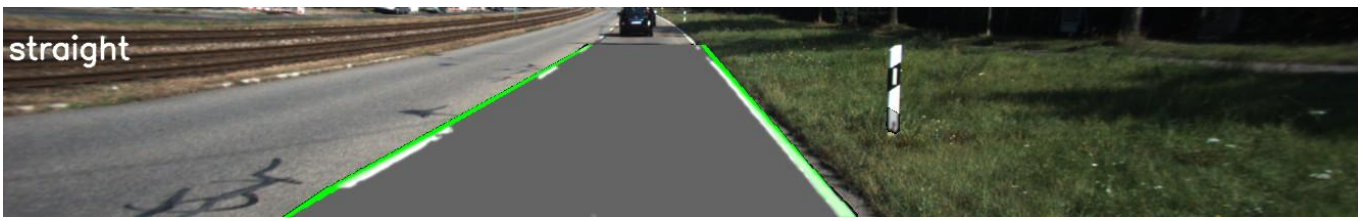

Figure 8: Final output with lane detection and direction of the car

Figure 9: After cropping the input image to our required ROI

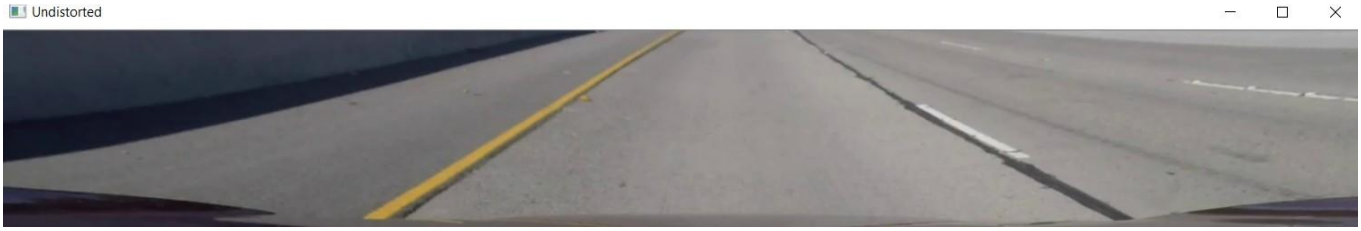

Figure 10: After undistortion on the cropped image

Figure 11: After filtering the noise

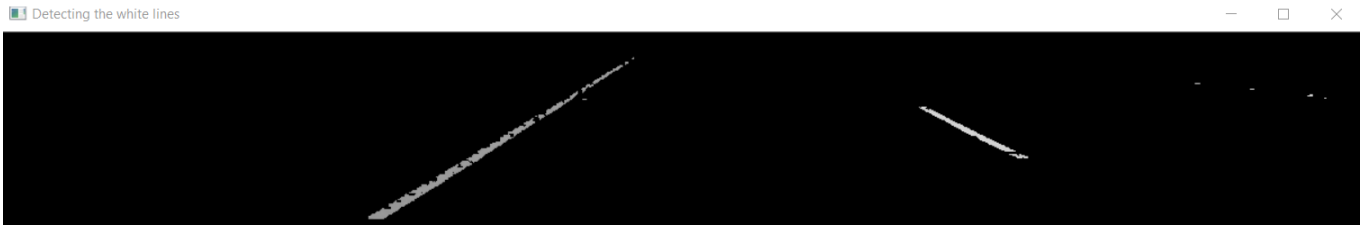

Figure 12: After detecting the white lines

Figure 13: After performing warping on the filtered image

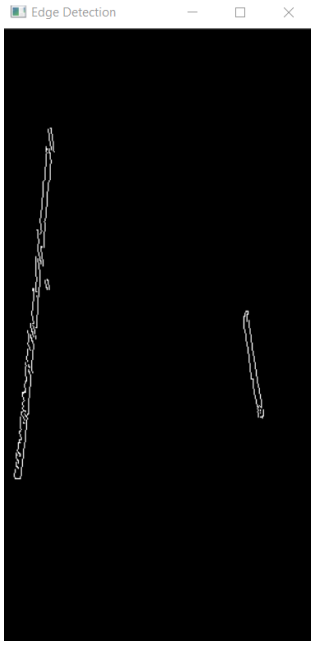

Figure 14: After detecting the edges from the warped

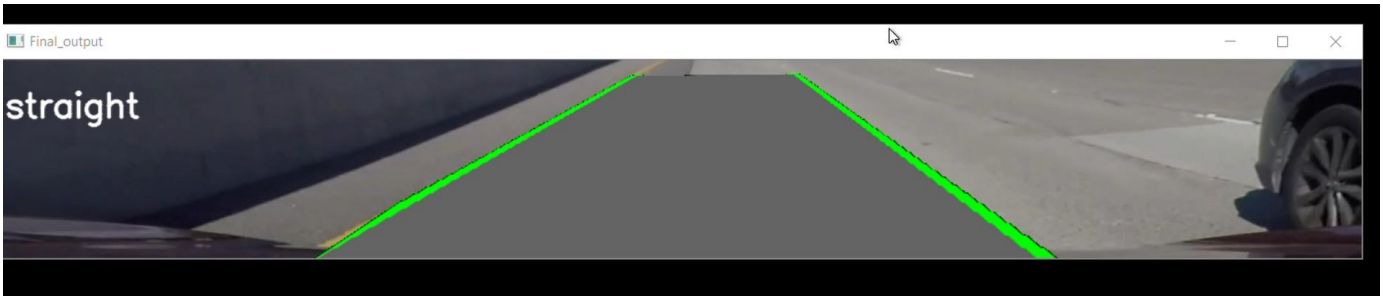

Figure 15: Final output with lane detection and direction of the car