VR Assignment-1 Report

Chinmay Parekh
IMT2020069

The libraries used for this assignment are as follows:
OpenCV
Matplotlib
Numpy

a) Use line fitting to detect road lane in images taken from the front of a car

Step 1: edge detection results are highly sensitive to image noise. One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it.(Canny edge detection uses linear filtering with a Gaussian kernel to smooth noise and then computes the edge strength and direction for each pixel in the smoothed image however smoothening it showed better results).

Step 2: Use canny edge detection to detect the edges.

Step 3: Take all those areas of the image concerning the road. For this I have made use of the region_of_interest function which will crop the remaining part of the image which isn't useful.

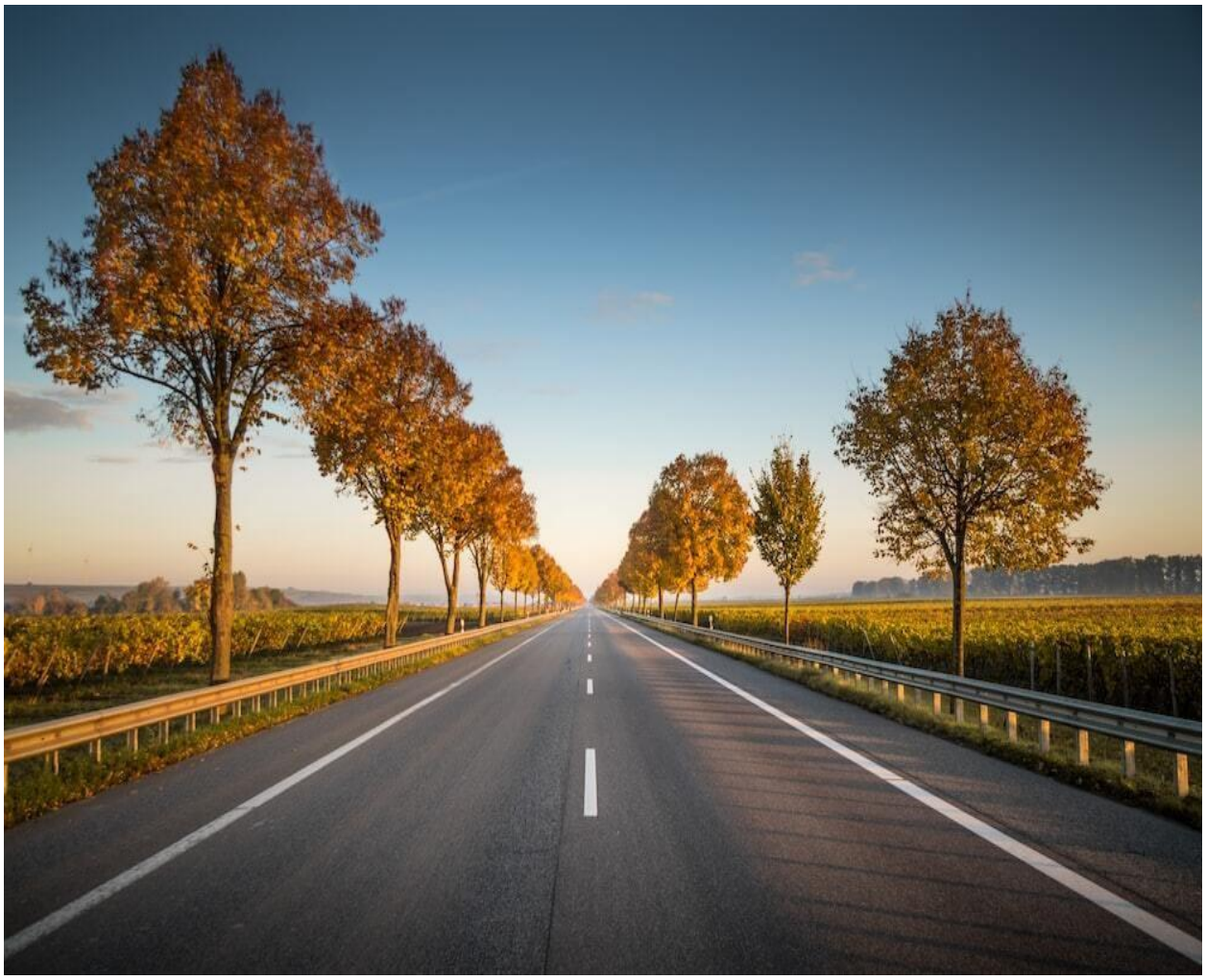
Step 4:Perform bitwise and between the edges detected by the canny edge detector and the mask.

Step 5: the Hough Transform algorithm detects lines by finding the (ρ, θ) pairs that have a number of intersections larger than a certain threshold. Use Hough Transform to convert the edges into lines.

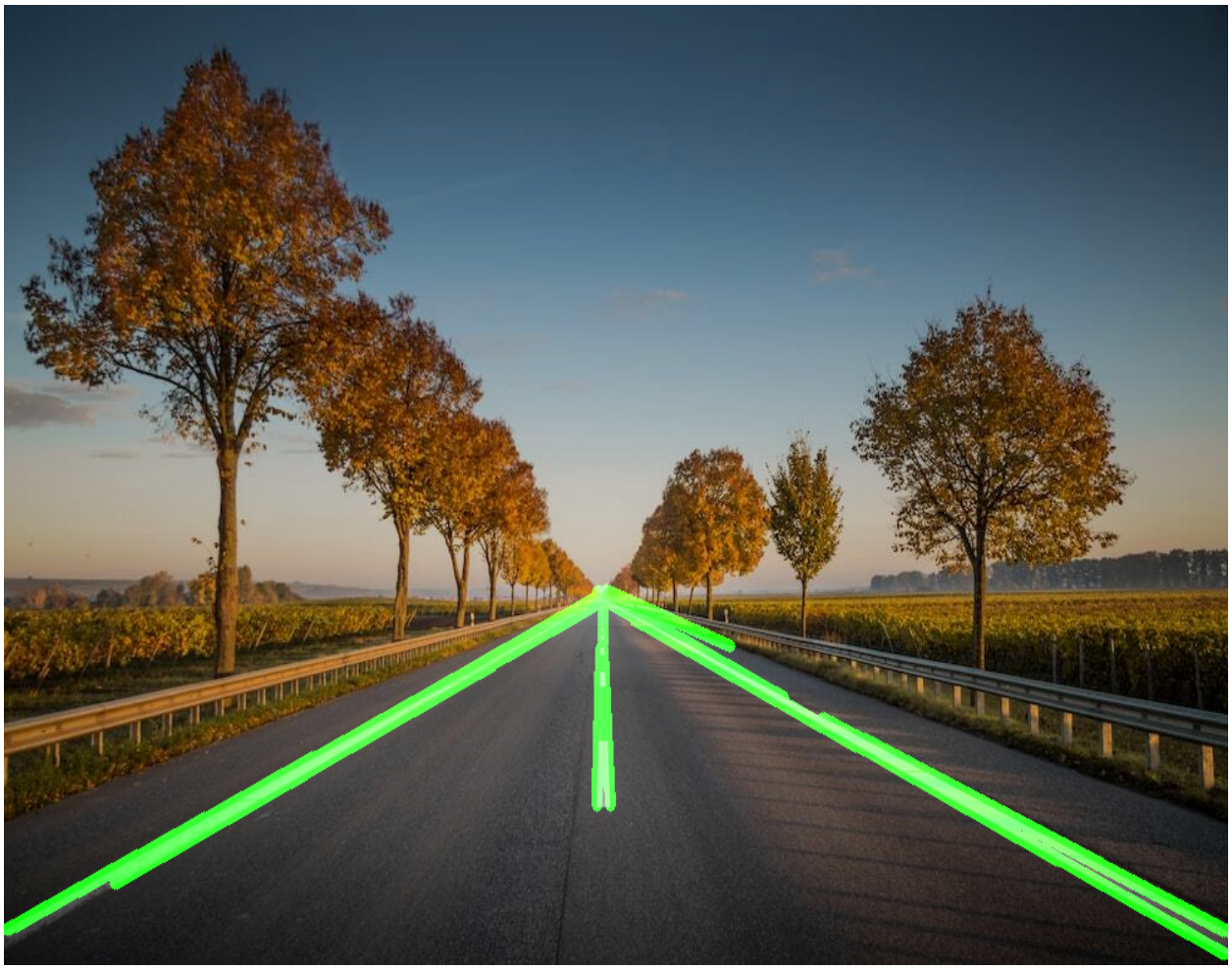
Its paramaters are as follows:

rho :Distance resolution of the accumulator in pixels.
theta:Angle resolution of the accumulator in radians.
threshold :Only lines that are greater than threshold will be returned.
minLineLength:Line segments shorter than that are rejected.
maxLineGap :Maximum allowed gap between points on the same line to link them
cv.HoughLinesP(mask, 1, np.pi/180, 20, np.array([]), minLineLength=20, maxLineGap=5) gave the best results after observing the edges of the image.
Step 6: Make use of the addWeighted() function to add the hough transformed image with the coloured image which will give us the desired output.

I made use of the following image:



I got the following output after following the steps mentioned above:



I tried this on different images but did not get proper outputs because generalising this approach is difficult due to the mask coordinates.

b) **Detect shadow regions on the road and inpaint it**

Step 1: Take all those areas of the image concerning the road. For this I have made use of the region_of_interest function which will crop the remaining part of the image which isn't useful.
Step 2: For the given two images, I identified the regions of interest by

checking the common regions for both the images and made a mask for it.
Step 3:Convert the image into HSV Space because HSV color space gives a better separation of chromaticity and intensity and so it becomes easier to detect and remove shadow.

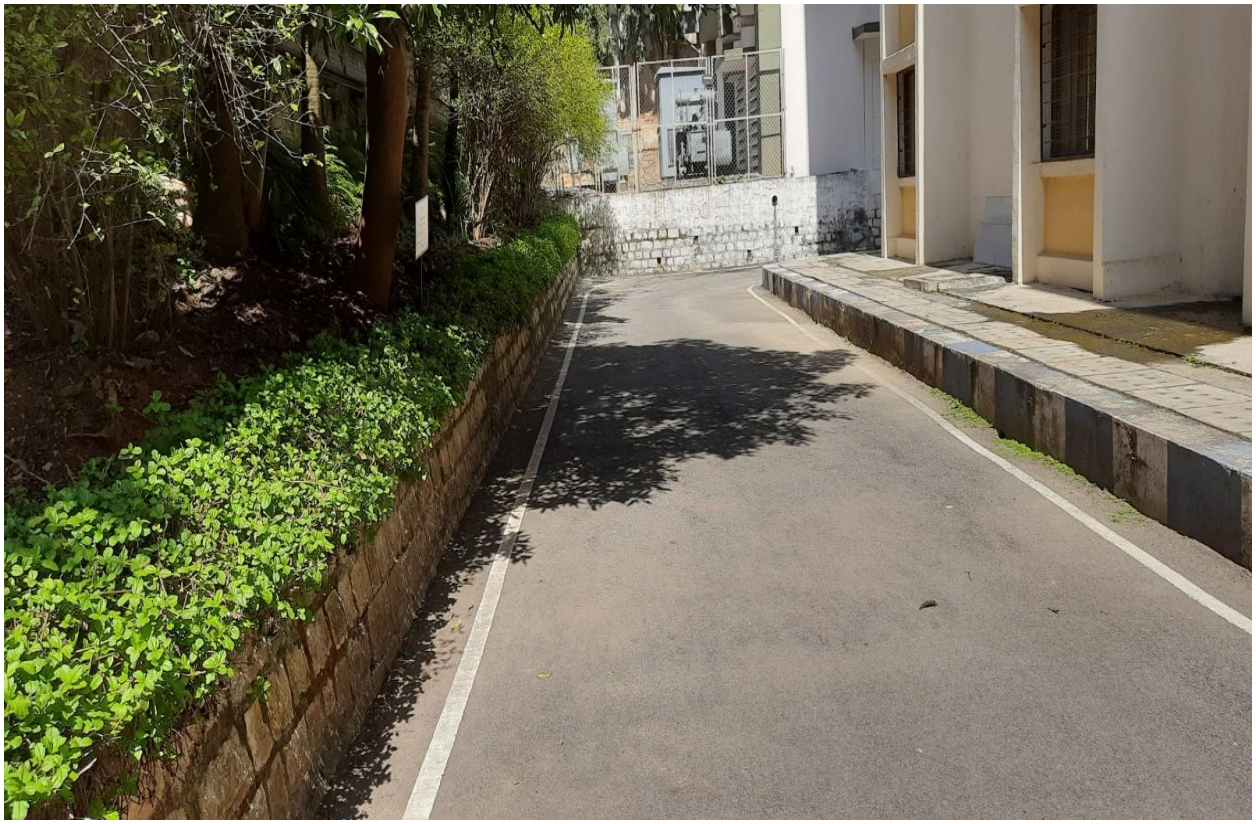
Step 4:After observing the values, I noted down the colours for the surrounding road, the shadow and the region surrounding the border(the faint border).

Step 5: Iterate over the pixes and check if the fall within a threshold close to the colours noted down in step 4. If they are within the threshold, the pixel value is replaced with the colour of the surrounding road.

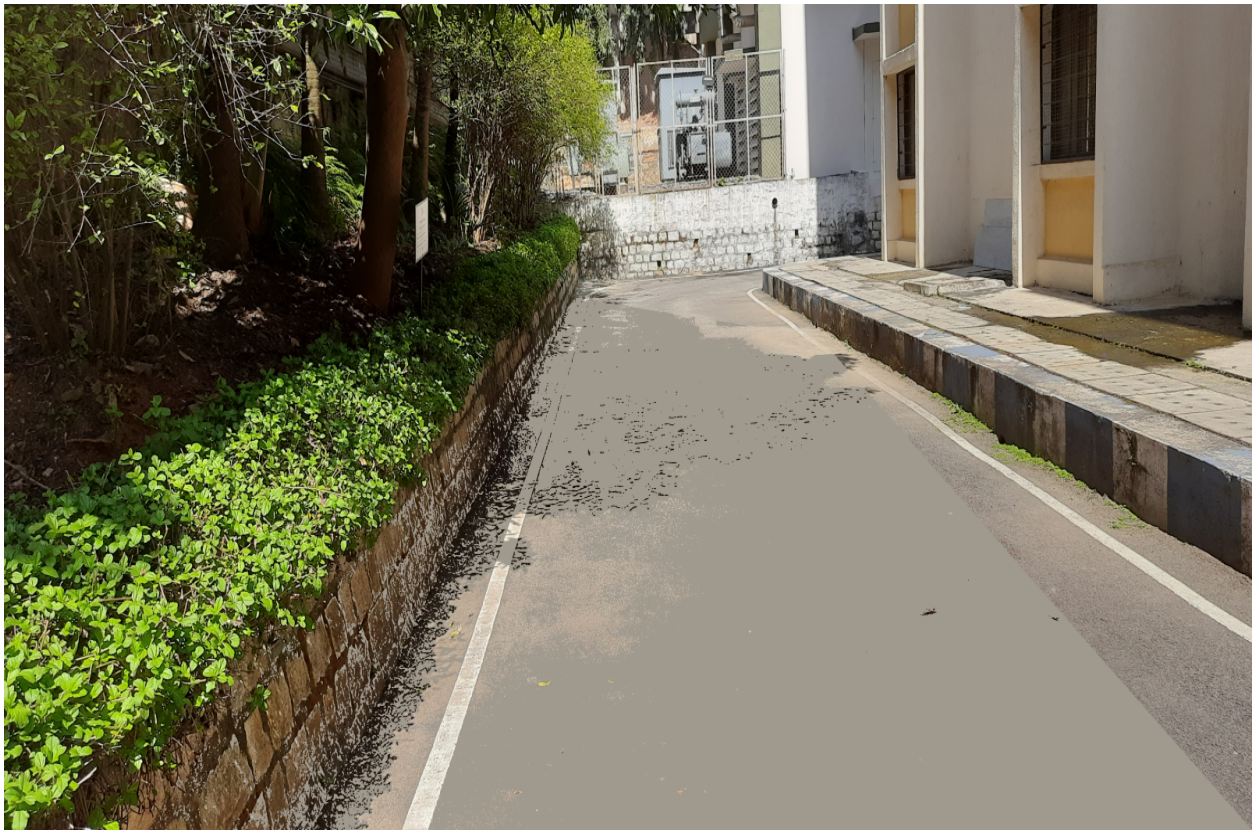
Step 6: Converted all white pixels into 1 and the black pixes were kept intact for the mask created in step 2. I also made use of ~mask which acts as its inverse image.

Step 7: For the regions of interest ,it is multiplied with the mask and the regions which are not important are multiplied by the inverse of the mask. The resultant image generated after this is our desired image.

I used the following image:



The output generated after following the above steps is as follows



I tried this on different images but did not get exact outputs because generalising this approach is difficult due to the mask coordinates.

Extra: I also tried a different approach to get better results. I tried using kmeans within a region and then replacing the shadow cluster with the centroid rgb values of the surrounding road cluster. Another approach tried by me was to make use of the inRange() function followed by inPaint() function but I was unable to fine tune the parameters of the inRange() function.