Image Processing

Module Name: <COMP2271>

Date: <26/02/2022>

Submitted as part of the degree of Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

1 INTRODUCTION

This project uses image processing to improve image frames from videos taken around durham. Two videos and methods are used to test our results. We use a validation test to test our ground truth images in comparison to our corrupted image. We then use our test set image to find a yolo object detection score. We next use both of these methods to assess our best final results in our conclusion.

2 METHODOLOGY

Given our test images we used several techniques to improve the quality of our images. To do this we will be using OpenCV functions. We first dewarped our image to correct their distortions. Next, we removed noise from our image to smoothen it and remove "black dots". We then proceeded to change the contrast and brightness of our image and finish off by sharpening the image to reduce the blurriness from the noise removal. To assess how good my image processing is I will be comparing a file of corrupted images with validation images to see how good my dewarping and noise removal is. I will then also use yolo object detection to assess the contrast and brightness functions. Though both methods are useful for testing our program, no noise removal, for example, would increase our object detection but would obviously not be good when comparing images. Below we will be discussing the different methodologies for our image processing.

2.1 Dewarping

Dewarping is when we adjust the dimensions of the image to get an appropriate fit. In order to do this, we will maneuver the coordinates of the image and the width and height and find these using paint. We will then find a matrix and apply these coordinates to form a matrix using getPerspectiveTransform() and make the final transformation including the width and height using warpPerspective().

2.2 Noise Removal

When removing noise, we intend to smooth the image by removing salt and pepper noise from the corrupted image. To do this we will be using median blur, however, if we remove too much noise we lose detail, therefore we try to find a balance,

2.3 Contrast

For contrast we will then use the functions convertScaleAbs, np.clip, and clahe.

2.4 Brightness

We then proceeded to increase the brightness similar to contrast: using convertScaleAbs and np.clip. We attempted to use gamma correction, however we found that this has no real affect on the performance of the image.

2.5 Sharpening

When applying noise removal, we can see that detail is lost and it becomes quite blurry, therefore, we will attempt to sharpen our image using canny detection. For canny, we detect our edges and minus it from our original blurry image. Here we also need to be careful with our parameters as ones that are too low will not preserve detail, however, if we use too much sharpening then we get ringing, and our image will no longer be of good quality. We then tested two different sharpening methods filter2D and high pass and low pass filters. For filter2D we specify a 3 by 3 matrix filled with -1s except we edit the middle parameter to change the sharpness. This matrix is specific for edge detection when using 2D filtering. From our table we can see the parameters we used that maximises sharpness while minimizing the ringing. For high pass and low pass filtering we will first use Fourier transformation on the image and then edit the parameters of high pass and low pass filtering to get the best image possible. High pass controls the sharpness while low pass controls the noise removal. So once we multiply these with the Fourier transformation our result is the addition of the two results.

3 RESULTS

Below shows what parameters we chose as well as the the stages of our images:

Table 1: Hyper Parameters of functions

Function	Affects	Hyper Parameter Tuning
Median Blur	Noise removal	ksize = 3
Gaussian Blur	Noise removal	
getPerspectiveTransform	Dewarping	Coordinates = [[20,20], [945,7],[965,375], [16,385]]
WarpPerspective	Dewarping	Width = 1024 Height = 394
Clahe	Contrast	Cliplimit = 0.6, tileGridSize = (1,1)
Np.clip	Contrast	Alpha = 0.51, Beta = 10
Np.clip	Brightness	Gamma = -1
ConvertScaleAbs	Brightness	Beta = 6
High Pass	Sharpen	d =387, n= 3
Low Pass	Noise Removal	d =421, n = 2
Kernel	Sharpen	Middle_n = 10
Canny Edge Detection	Sharpen	Number of edges = 500

Original image

After dewarping:

After Contrasting

After Noise Removal

After Brightness Correction

5 Conclusions

In the end I got a 0.85 comparison score, a 0.654 object detection score with 1014 detections and a 0.62 mean confidence score. To improve my score. I regret using too many functions at once as this affected the images a lot therefore, if I had more time, I would have filtered out the right functions. Secondly, I would have liked to split my test set in 2 and change the brightness and contrast in each as I feel the functions edited the images in different ways. I would have also liked to have edited my yolo file to make it more targeted for traffic object detection.

References

[1] atapour/ip-python-opencv: Examples and code demonstrations for the Image Processing module at Durham University (github.com)

[2] Python OpenCV - Filter2D() Function - GeeksforGeeks

[3] Fourier Transform — OpenCV-Python Tutorials 1 documentation (opencv24-python-tutorials.readthedocs.io)

[4] python - How to apply a LPF and HPF to a FFT (Fourier transform) - Stack Overflow