Report on Image Dehazing using Bilateral Filtering and Histogram Equalization

Abstract

This report details the development and implementation of an image dehazing algorithm that leverages bilateral filtering and histogram equalization techniques. The algorithm is designed to enhance the quality of images affected by haze, making them clearer and more visually appealing. The entire pipeline is implemented using Python and the OpenCV library, processing images from an input directory and saving the enhanced images to an output directory.

Introduction

Image dehazing is a critical preprocessing step in many computer vision applications, particularly those involving outdoor scenes. Haze, caused by particles such as dust and smoke in the air, degrades the quality of images by reducing contrast and obscuring details. This report discusses an approach to dehazing that combines bilateral filtering, histogram equalization, and additional image processing techniques to enhance image clarity.

Methodology

The proposed image dehazing algorithm is composed of several key steps, implemented in a Python script utilizing the OpenCV library. The steps include:

1. **Bilateral Filtering**: Reduces noise while preserving edges.
2. **Image Sharpening**: Enhances edges to make the image appear clearer.
3. **Histogram Equalization**: Improves the contrast of the image.
4. **Saturation Adjustment**: Increases color intensity to make the image more vivid.

Bilateral Filtering

Bilateral filtering is a non-linear, edge-preserving, and noise-reducing smoothing filter. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels. This weight is determined by the spatial distance and the intensity difference, ensuring that only pixels with similar intensity values contribute to the final result.

Image Sharpening

Image sharpening is achieved using a convolutional kernel designed to enhance the edges. The kernel used in this project is:

```
lua
kernel = np.array([[-1, -1, -1],
                   [-1,  9, -1],
                   [-1, -1, -1]])
```

This kernel highlights the differences in intensity between a pixel and its neighbors, thus sharpening the image.

Histogram Equalization

Histogram equalization is performed on the luminance channel of the YUV color space. It redistributes the intensity values to span the entire range, enhancing the overall contrast of the image.

Saturation Adjustment

To enhance the visual appeal of the image, the saturation in the HSV color space is increased. This step makes the colors more vibrant, compensating for any dullness caused by the haze.

Implementation

Setup

The project processes images from an input directory (`./test/low/`) and saves the enhanced images to an output directory (`./test/predicted/`). The list of image files is obtained, and a progress bar is used to track the processing of each image.

Code

```
python
import cv2
import numpy as np
import os
from tqdm import tqdm

path = r"./test/low/"
output_path = r'./test/predicted/'

img_files = [f for f in os.listdir(path) if f.endswith('.png')]
tot_img = len(img_files)

def solve():
    with tqdm(total=tot_img) as pqs:
        for i, name in enumerate(img_files):
            img_path = os.path.join(path, name)
            img = cv2.imread(img_path)
            img = cv2.bilateralFilter(img, 9, 75, 75)
            kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
            img = cv2.filter2D(img, -1, kernel)
            img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
            img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])
            img = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
            img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
            img_hsv[:,:,1] = np.uint8(np.clip(img_hsv[:,:,1] * 1.7, 0, 255))
            img = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2BGR)
            op_path = os.path.join(output_path, name)
            cv2.imwrite(op_path, img)
            pqs.update(1)
            pqs.set_postfix({"Progress": f"{(i + 1) / tot_img * 100:.1f}%"})

solve()
```

Detailed Explanation

1. **Reading the Image**:
   - The script reads each image from the specified input directory.
2. **Bilateral Filtering**:
   - The function `cv2.bilateralFilter` is used with parameters: diameter = 9, sigmaColor = 75, and sigmaSpace = 75.
   - This reduces noise while preserving edges.
3. **Image Sharpening**:
   - The function `cv2.filter2D` is used with the sharpening kernel to enhance edges.
4. **Histogram Equalization**:
   - The image is converted to the YUV color space using `cv2.cvtColor`.
   - Histogram equalization is applied to the Y channel (luminance) using `cv2.equalizeHist`.
5. **Saturation Adjustment**:
   - The image is converted to the HSV color space.
   - The saturation channel (S) is increased by multiplying it by 1.7 and clipping values to the range [0, 255].
   - The image is converted back to the BGR color space.
6. **Saving the Enhanced Image**:
   - The processed image is saved to the specified output directory using `cv2.imwrite`.
7. **Progress Tracking**:
   - The `tqdm` library is used to display a progress bar and update it after processing each image.

Results and Discussion

The implemented algorithm effectively enhances the quality of hazy images by reducing noise, sharpening edges, improving contrast, and increasing color saturation. The combined use of bilateral filtering and histogram equalization results in clearer and more visually appealing images.

Sample Output

Original Image:

Enhanced Image:

Performance

The algorithm processes images efficiently, as indicated by the progress bar. Each step in the pipeline is optimized for performance while maintaining high image quality.

Conclusion

This project successfully demonstrates an effective approach to image dehazing using bilateral filtering and histogram equalization. The combination of noise reduction, edge sharpening, contrast enhancement, and saturation adjustment significantly improves the visual quality of hazy images. Future work could explore additional techniques for further enhancement and real-time processing capabilities.

References

- OpenCV Documentation: https://docs.opencv.org/
- Tqdm Documentation: https://tqdm.github.io/

Note: Replace the sample image paths with actual images from your input and output directories to visualize the results.

This report provides a comprehensive overview of the image dehazing project, detailing the methodology, implementation, and results. By following the steps and understanding the code, one can replicate and potentially extend this work to achieve even better image enhancement.