Assignment 1: Computer Vision

Madhia Shabih

August 2024

1 Question 1

1.1 Description

The `remove_greenscreen` function replaced the green screen background with a new background. A mask was created where the green channel exceeded a threshold of 160 and the red channel was below 60, isolating the actors. The mask was then used to replace the green screen pixels with corresponding pixels from a new background image.

(a) Green Screen Image

(b) The Mask

Figure 1: Green Screen Removal Process

Figure 2: Image with New Background

1.2 Discussion

Initially, the mask included parts of the woman's jacket. After consulting with a classmate, I adjusted the threshold values for the red and blue channels. By setting the red channel threshold to be less than 60, the jacket was successfully excluded, leading to a better isolation of the actors.

2 Question 2: Image Processing

2.1 Description

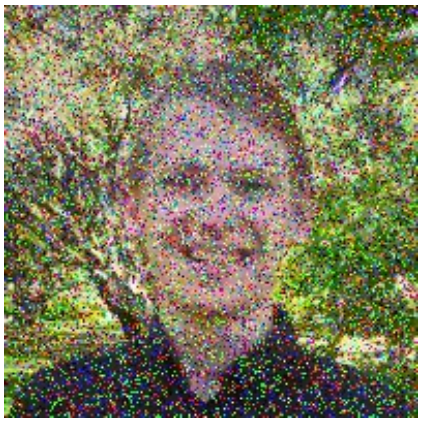
The `process_image` function converted the input image to grayscale and added salt and pepper noise using the `add_salt_and_pepper` function. The same was done for the color image. Finally, a median filter was applied to the noisy color image using the `median_filter` function.

(a) Noise Density of 0.05

(b) Noise Density of 0.2

Figure 3: Gray Scale Image with Varying Noise Densities.

(a) Noise Density of 0.05



(b) Noise Density of 0.2

Figure 4: Colour Image with Varying Noise Densities.

(a) Noise Density of 0.05

(b) Noise Density of 0.2

Figure 5: Noised and Filtered Colour Images (Filter Size of 2).

(a) Noise Density of 0.05

(b) Noise Density of 0.2

Figure 6: Noised and Filtered Colour Images (Filter Size of 4).

2.2 Discussion

As the filter size increases, the effectiveness of the median filter improves, particularly at lower noise densities. This is evident when comparing the filtered images for noise densities of 0.05 and 0.2. Larger filter sizes, such as 4, further enhance the noise reduction.

3 Question 3: Unsharp Masking

3.1 Description

The `apply_unsharp_mask` function converted the image to grayscale, applied a blur, and created a mask from the difference between the original and blurred images. The mask was normalized and combined with the original image to enhance its edges.

3.2 Results

(a) Original Image (b) Image with unsharp masking

Figure 7: Original and Sharpened Image

Figure 8: The mask

4 Question 4: Image Resizing

4.1 Description

The `nearest_neighbor_resize` function calculated the new dimensions of the resized image and assigned pixel values by finding the closest pixel in the original image. The `bl_resize` function used bilinear interpolation to estimate pixel values by averaging the four nearest neighboring pixels.

4.2 Results

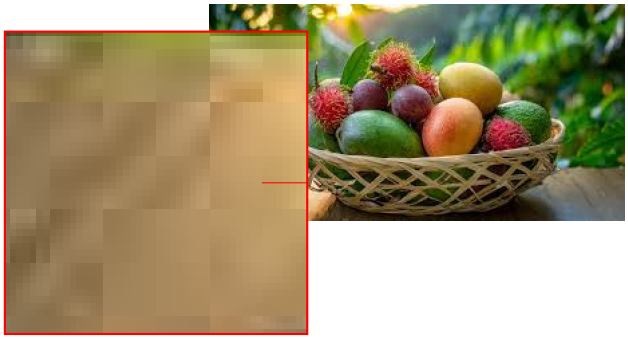


Figure 9: Image Resizing with a scale factor of 2 for Nearest Neighbour Interpolation

Figure 10: Image Resizing with a scale factor of 3 for Nearest Neighbour Interpolation

Figure 11: Image Resizing with a scale factor of 3 for Bilinear Interpolation

4.3 Discussion

Nearest neighbor interpolation results in a pixelated appearance because it assigns the value of the closest pixel to each output pixel. Bilinear interpolation, however, produces smoother transitions by averaging the values of the four nearest pixels, resulting in a more refined image.

5 Question 5

5.1 Description

This script used the SIFT (Scale-Invariant Feature Transform) algorithm to detect and match keypoints between two images, 'semper1.jpg' and 'semper2.jpg'. It matched the descriptors using a Brute-Force Matcher and highlighted the top 100 matches.

5.2 Results

Figure 12: Feature Detection on Train Image

Figure 13: Feature Detection on Query Image

Figure 14: Matching on Train Image

Figure 15: Matching on Query Image

5.3 Discussion

The SIFT algorithm is effective for detecting and matching features between images, even when there are changes in rotation. The Brute-Force Matcher provides a straightforward approach to match these features by comparing de-

scriptors from two images.

6 Question 6

6.1 Description

This question involved testing the robustness of the SIFT (Scale-Invariant Feature Transform) algorithm by matching keypoints between an original image and its resized versions. The code implemented a custom bilinear resizing function (`bl_resize`) and evaluated the accuracy of keypoint matches across various scaling factors. The `calculate_accuracy` function was used to determine how closely the rescaled keypoints in the resized images matched those in the original.

6.2 Results

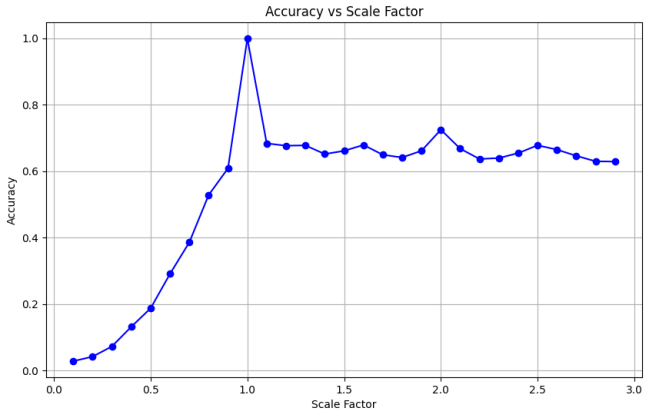


Figure 16: Accuracy of Keypoint Matching Across Different Scales

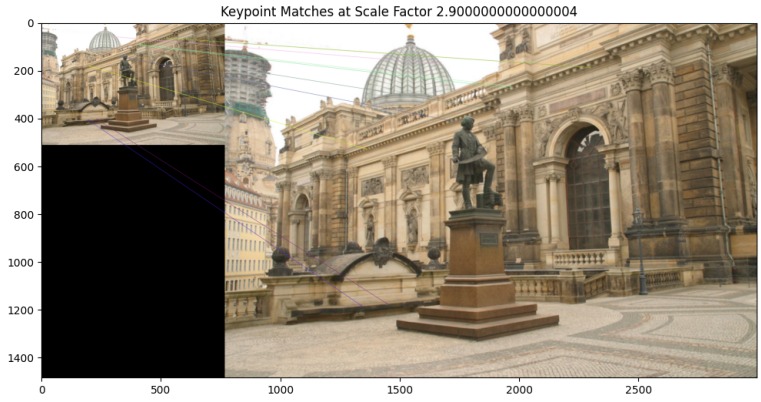


Figure 17: Keypoint Matches on Resized Image

6.3 Discussion

The experiment demonstrated that SIFT is effective in matching keypoints across different scales. The accuracy of keypoint matching remained high for moderate scaling factors. However, as the scale deviated significantly, accuracy decreased due to loss of detail in the resized images. Smaller scales led to fewer detected keypoints, while larger scales increased keypoints but sometimes reduced matching accuracy due to misalignment.