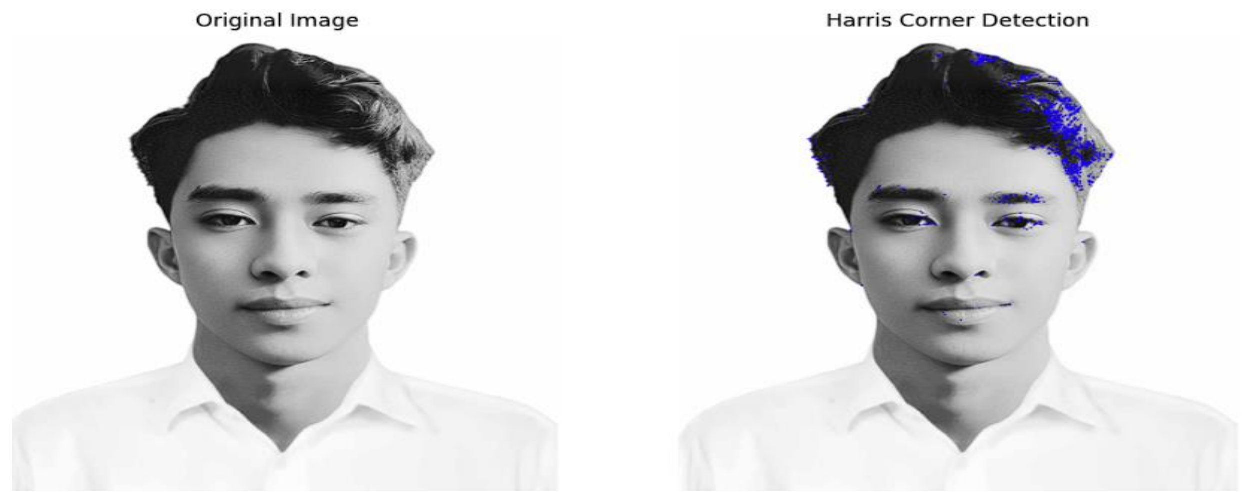
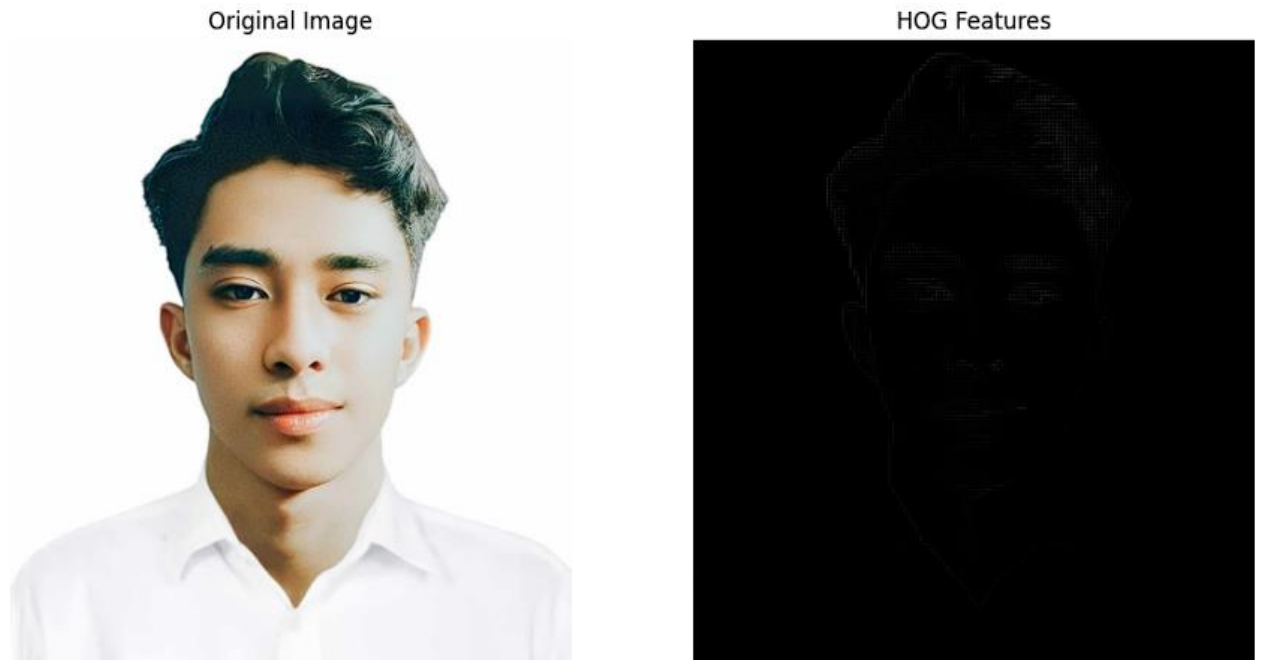
Performance Analysis

Task 1: Harris Corner Detection



In this implementation, the Harris Corner Detection algorithm is applied to a grayscale image to identify and mark corners. The process begins by loading and converting the image to a `float32` data type, a necessary step for the Harris algorithm to function correctly. The algorithm itself is configured with a block size of 2x2, a Sobel operator aperture parameter of 3, and a Harris detector free parameter of 0.04, parameters that balance computational efficiency with the sensitivity of corner detection. Post-processing involves dilating the corner response image to enhance corner points, making them more visible. A thresholding technique is used to mark significant corners in red on the original image, ensuring only the most prominent features are highlighted. The results are visually assessed by displaying the original image alongside the marked corners

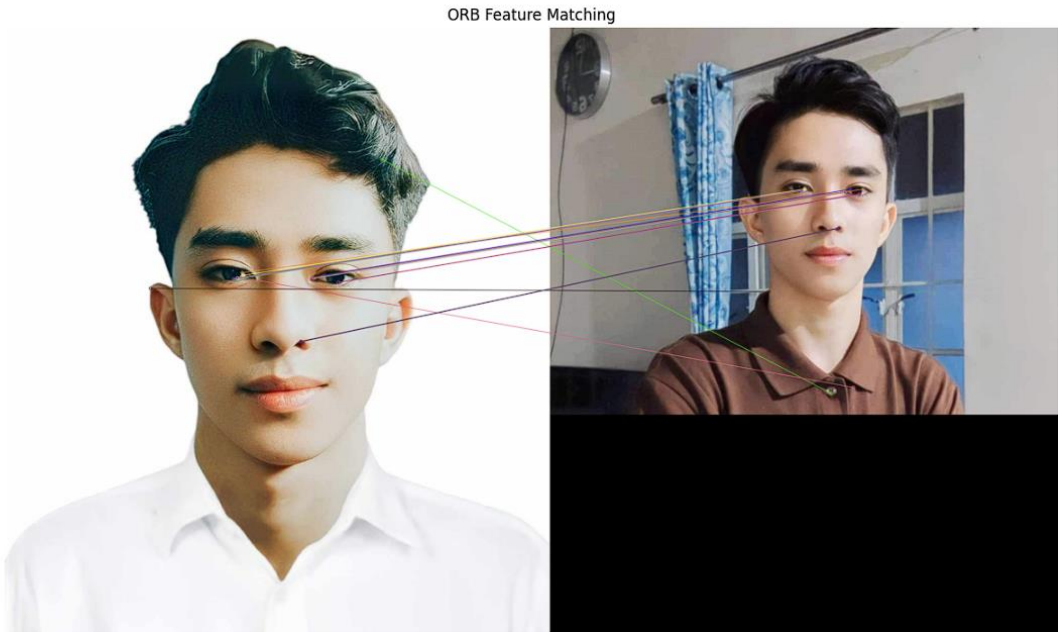
Task 2: HOG Feature Extraction



This code effectively loads an image, converts it to grayscale, and extracts its Histogram of Oriented Gradients (HOG) features. The HOG method captures important shape and texture information by analyzing the gradient directions of pixel intensities. Key parameters include 9 orientations, a cell size of 8x8 pixels, and a block size of 2x2 cells, which help balance detail and robustness to lighting changes.

The HOG visualization is generated to show the extracted features, and it's enhanced for better clarity using intensity rescaling. Finally, the original image and the HOG features are displayed side by side with Matplotlib, allowing for easy comparison. Overall, this implementation is effective for visualizing image features.

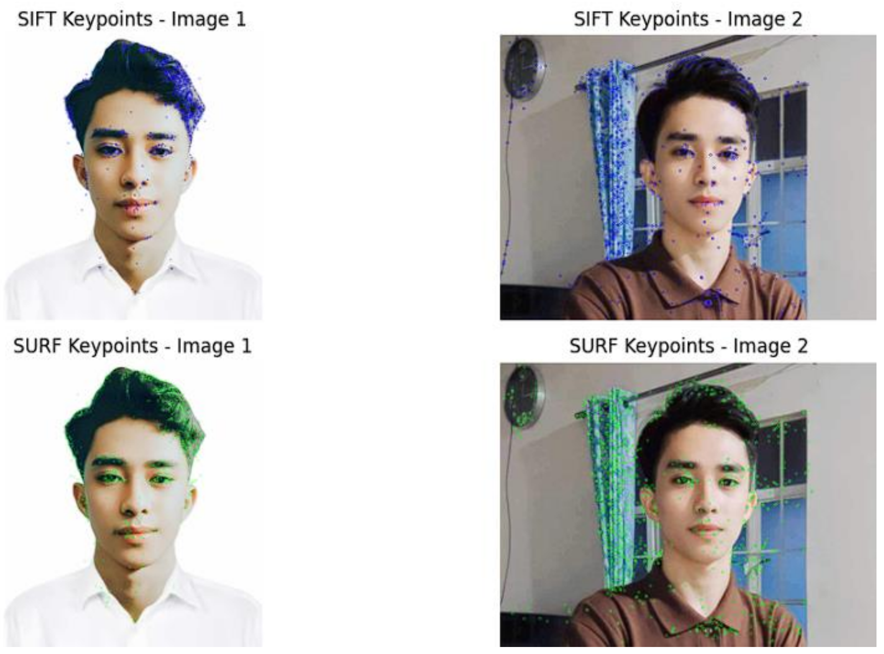
Task 3: ORB Feature Extraction and Matching



In this task I loads two images and converts them to grayscale for processing. It initializes the ORB detector to identify keypoints and compute their descriptors in both images. Using a FLANN-based matcher, it finds matches between the descriptors of the two images. The parameters for the FLANN matcher are set to optimize performance for the ORB features.

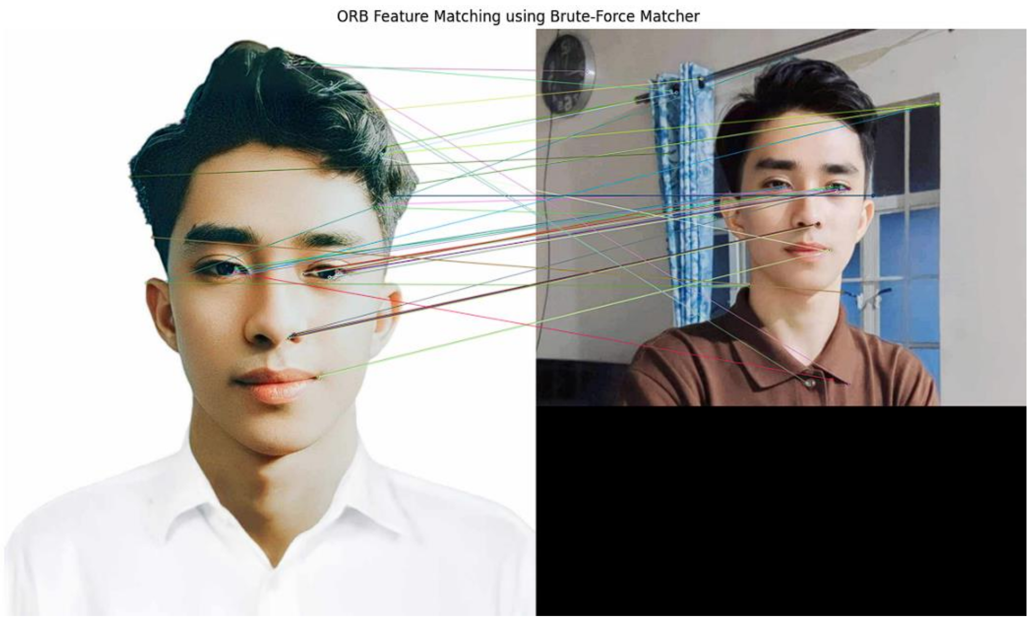
To filter out the best matches, Lowe's ratio test is applied, which retains only those matches where the closest match is significantly better than the second closest. This helps ensure that the matches are reliable. Finally, the matched keypoints are drawn onto a single image for visualization, allowing for an easy comparison of how well the two images align based on their features. The result is displayed using Matplotlib, providing a clear view of the matched features between the two images. Overall, this implementation effectively demonstrates how to use ORB for feature matching in computer vision tasks

Task 4: SIFT and SURF Feature Extraction



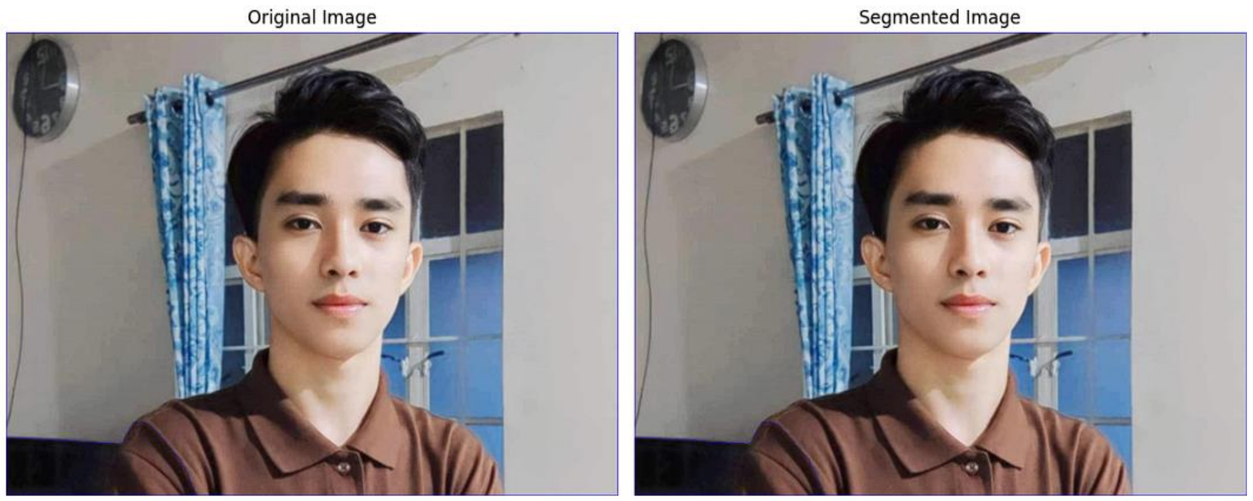
In this task I initializes the SIFT detector, which identifies keypoints and computes their descriptors in both images. The SIFT algorithm is well-known for its robustness to scale and rotation, making it effective for matching features in varying conditions. Similarly, the SURF detector is initialized to extract keypoints and descriptors. SURF is designed to be faster than SIFT while still providing reliable keypoint detection and description. Keypoints from both algorithms are drawn on the respective images, with SIFT keypoints displayed in red and SURF keypoints in green. This color-coding helps distinguish between the two methods visually.

Task 5: Feature Matching using Brute-Force Matcher



This code efficiently matches features between two images using the ORB algorithm and Brute-Force Matcher. By visualizing the matches, it highlights how well the algorithm identifies corresponding points, which is useful for tasks like image stitching and object recognition. The approach is effective and straightforward compared to other method.

Task 6: Image Segmentation using Watershed Algorithm



This code effectively segments an image into distinct objects using the Watershed algorithm. By employing preprocessing steps such as thresholding and morphological operations, it enhances the segmentation quality. The final visualization clearly highlights the boundaries of the detected objects