**EXERCISE 3**

## Exercise 1: Harris Corner Detection

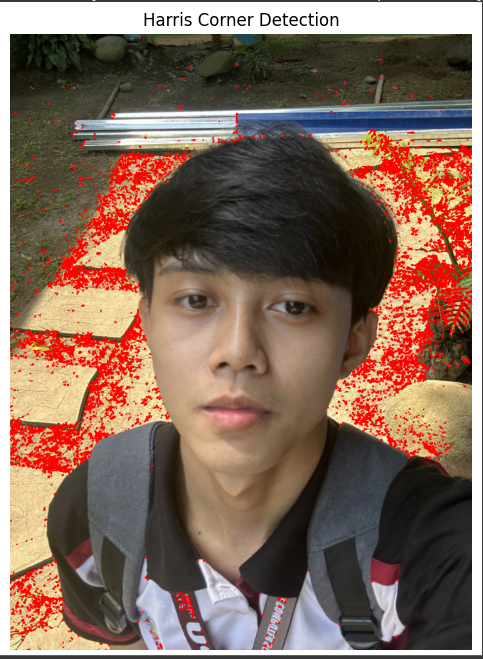
**Purpose: To detect corners in an image using the Harris Corner Detection algorithm.**

**Process:**

1. **Load the Image**: Reads the specified image from your drive.
2. **Convert to Grayscale**: Converts the image to grayscale, a necessary step for corner detection.
3. **Apply Harris Corner Detection**: The algorithm calculates where corners are present based on intensity changes in multiple directions.
4. **Mark Detected Corners**: Corners are highlighted in red on the original image by thresholding the response values.
5. **Display the Result**: Shows the image with red markings at each detected corner.

**Expected Output:**

You’ll see the original image with red dots marking the corners. Corners are typically at intersections of edges or changes in structure within the image.



**Exercise 2: HOG (Histogram of Oriented Gradients) Feature Extraction**

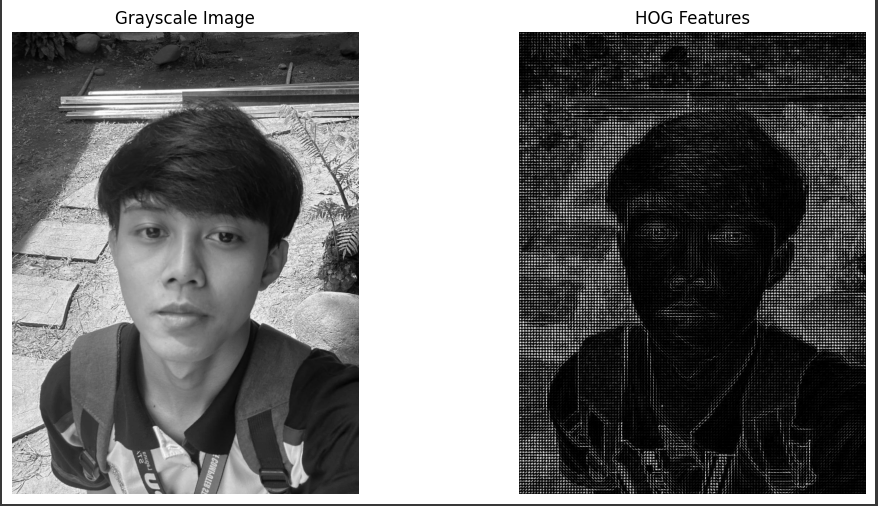
**Purpose: To extract HOG features, which capture the structure of objects in an image, and visualize the gradient orientations.**

**Process:**

1. **Load the Image**: Reads the specified image from your drive.
2. **Convert to Grayscale**: Converts the image to grayscale, which is required for HOG processing.
3. **Apply HOG Descriptor**: Computes the HOG features, returning both the feature descriptor and a visualization of gradient orientations.
4. **Rescale the HOG Image for Better Contrast**: Adjusts the contrast of the HOG visualization for clearer visualization.
5. **Display the Result**: Shows the grayscale image alongside the HOG visualization.

**Expected Output:**

You’ll see two images side by side: the original grayscale image and the HOG visualization. The HOG image will show gradient orientations, where each small block represents the direction and magnitude of edges, emphasizing shapes and structures within the image.



**Exercise 3: FAST (Features from Accelerated Segment Test) Keypoint Detection**

**Purpose: To detect keypoints quickly using the FAST algorithm and display them on the image.**

**Process:**

1. **Load the Image**: Reads the specified image from your drive.
2. **Convert to Grayscale**: Converts the image to grayscale, as FAST operates on single-channel images.
3. **Apply FAST Keypoint Detector**: Detects keypoints (interesting points in the image) using the FAST algorithm, which is known for its speed.
4. **Draw Keypoints on the Image**: Draws each detected keypoint as a green dot on the original image.
5. **Display the Result**: Shows the image with keypoints highlighted.

**Expected Output:**

You’ll see the original image with green dots marking the keypoints. These are typically located on corners, edges, and other significant points in the image, providing useful features for further tasks like object recognition or tracking.



**Exercise 4: Feature Matching using ORB and FLANN**

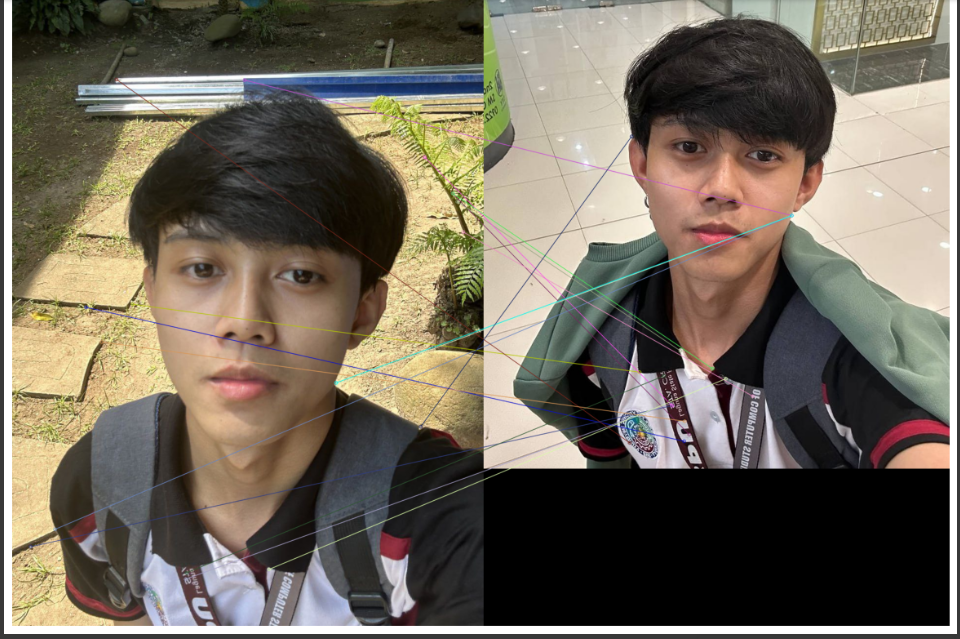
**Purpose: To match features between two images using ORB descriptors and FLANN-based matching.**

**Process:**

1. **Load the Images**: Reads two images to be compared and matched.
2. **Convert to Grayscale**: Converts both images to grayscale, as ORB is applied to single-channel images.
3. **Extract ORB Keypoints and Descriptors**: Detects keypoints and computes feature descriptors using ORB for both images.
4. **Set Up FLANN Matcher**: Configures FLANN for fast matching, using settings optimized for binary descriptors like ORB.
5. **Perform Feature Matching**: Matches descriptors between the two images.
6. **Apply Ratio Test**: Filters matches to retain only the best ones, improving accuracy.
7. **Draw Matched Features**: Draws lines connecting matched keypoints between the two images.
8. **Display the Result**: Shows the two images side-by-side with lines connecting matched points.

**Expected Output:**

You’ll see two images displayed side by side with lines connecting matched features. This visualization helps you identify similar structures or objects between the images. Good matches are often clustered around areas with similar patterns or objects.



**Exercise 5: Image Segmentation using Watershed Algorithm**

**Purpose: To segment an image into distinct regions using the Watershed algorithm, which is especially useful for separating overlapping objects.**

**Process:**

1. **Load the Image**: Reads the specified image from your drive.
2. **Convert to Grayscale**: Converts the image to grayscale, a necessary step for thresholding and segmentation.
3. **Apply Binary Threshold**: Creates a binary version of the image, where foreground and background are separated based on pixel intensity.
4. **Remove Noise with Morphological Operations**: Cleans up small noise in the background and foreground using morphological transformations.
5. **Identify Foreground and Background**: Uses distance transform to identify sure foreground and background areas.
6. **Label Markers for Watershed**: Labels distinct regions to be used by the Watershed algorithm.
7. **Apply Watershed Algorithm**: Segments the image into regions, marking boundaries between them.
8. **Mark Boundaries**: Highlights boundaries on the original image with a red color.
9. **Display the Result**: Shows the segmented image with red lines marking the boundaries between distinct regions.

**Expected Output:**

You’ll see the original image with red lines marking the boundaries between segmented regions. This helps in separating overlapping or touching objects, making it useful for applications like object counting or shape analysis.

