CSE 4128: Image processing and Computer Vision Laboratory

Dynamic Image Gestures: Zoom in and out and Rotate

By,

Mahmuda Afrin Tuli

Roll:1907019

Date of Submission: 30-06-2024

Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

Contents

1 Introduction

2 Objective

3 Project Idea

4 Project Overview

5 Methodology

6 Programming Environment

7 Programming Language

8 Key Features

9 Final Output

10 Limitations

11 Discussion

12 Conclusion

Introduction

In the era of touchless interfaces and natural user interfaces (NUIs), gesture-based control systems have gained significant traction. These systems provide a more natural way for users to interact with technology. This project focuses on leveraging computer vision techniques to recognize hand gestures and manipulate an image accordingly. The system is designed to detect hand gestures using a webcam, segment the hand region, and apply the corresponding image transformation (zoom or rotate) based on the detected gestures.

Objective

The objective of this project is to develop a dynamic image manipulation system using hand gestures captured via a webcam. The system allows users to zoom in/out and rotate an image by detecting specific hand movements and gestures. The aim is to create an intuitive and seamless interaction between the user and the digital image, enhancing user experience in applications requiring image manipulation.

Project Idea

The core idea behind this project is to use hand gestures to control the zoom and rotation of an image displayed on the screen. By implementing computer vision algorithms, the system captures real-time video input, processes the frames to detect and segment the hand, and interprets the gestures to perform the desired image manipulations. This approach provides an innovative way to interact with images without the need for traditional input devices like a mouse or keyboard.

Project Overview

The project utilizes OpenCV, a powerful library for computer vision, to process video frames captured by a webcam. The key steps involved in the project include:

1. Capturing video input from the webcam.
2. Defining a region of interest (ROI) for hand gesture detection.
3. Calibrating the background for accurate hand segmentation.
4. Detecting and segmenting the hand region.
5. Identifying hand gestures to determine the type of image manipulation (zoom or rotate).
6. Applying the corresponding transformation to the image.

Methodology

1 Video Capture and Preprocessing

Frames are captured from a webcam using OpenCV (cv2.VideoCapture). Then parameters such as frame dimensions (FRAME_HEIGHT, FRAME_WIDTH), calibration time (CALIBRATION_TIME), background update weight (BG_WEIGHT), object threshold (OBJ_THRESHOLD), and buffers for gesture recognition are initialized. An initial image (test.jpg) is loaded using cv2.imread() function from local disk for manipulation and the image is resized to a smaller size (150x150) for overlaying on camera frames using cv2.resize() function.

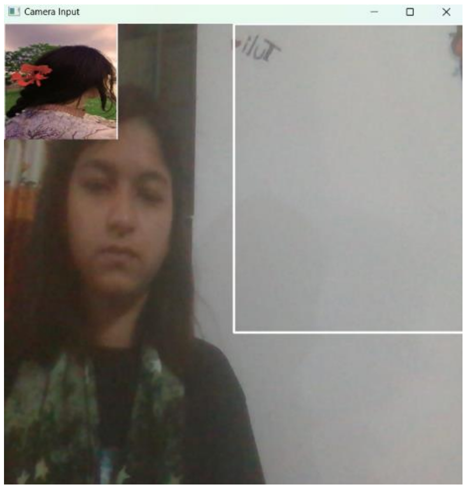

Figure 1: Initial Frame and Test Image.

2 Region of Interest (ROI) Definition

A specific region in the frame is designated to detect hand gestures, reducing computational load and focusing processing on the area where hand gestures are expected. `A function named` get_region(frame) function is used to extract the region from the frame and converts it to grayscale using cv2.cvtColor. Gaussian blur using cv2.GaussianBlur is used to reduce noise, aiding in more robust background subtraction.

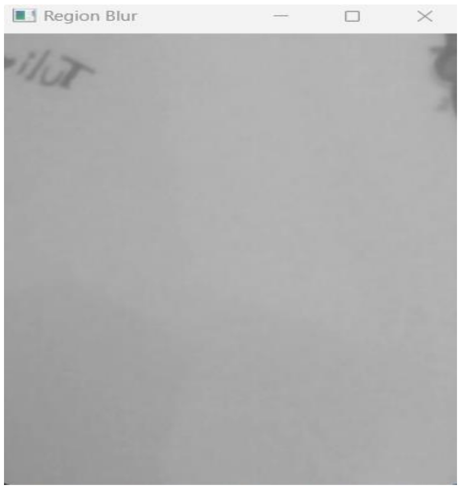

Figure 2: Region of Interest.

3 Background Calibration

The get_average(region_blur) function calculates the weighted average of the blurred region to build a model of the background over the calibration period (CALIBRATION_TIME). This helps in later steps to differentiate between the hand and the static background using cv2.accumulateWeighted.

4 Background Subtraction

The segment(region, region_blur) function computes the absolute difference between the current frame and the background. Binary thresholding using cv2.threshold function is used to create a binary image where the hand (foreground) appears white, and the background is black. Morphological operations (cv2.morphologyEx) are applied to remove noise and close gaps, ensuring clear contour detection.

5 Skin Detection

5.1 Skin Mask Creation

The get_skin_mask(frame) function converts the frame to YCrCb and HSV color spaces. These spaces are chosen because they separate chrominance (color) from luminance (brightness), making it easier to detect skin tones. Skin color falls within specific ranges in YCrCb and HSV, which are used to create binary masks (cv2.inRange). The masks from both color spaces are combined using cv2.bitwise_or to improve robustness in different lighting conditions. Further morphological operations and median blurring are applied to refine the mask.

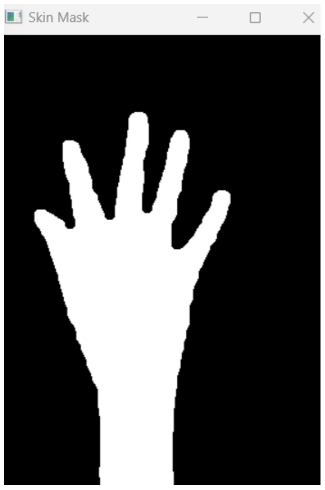

Figure 3: Skin Mask.

6 Combining Masks

6.1 Mask Combination

The combine_masks(background_mask, skin_mask) function is used a binary mask from background subtraction which is combined with the skin mask to isolate the hand more accurately. This ensures that only the detected skin regions that differ from the background are considered.

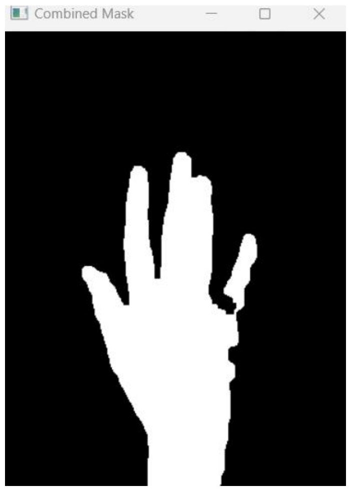

Figure 4: Combined Mask.

7 Contour Detection and Hand Structure Confirmation

7.1 Find Contours

Contours are found by using cv2.findContours() in the binary image.cv2.RETR_EXTERNAL retrieves only the outermost contours, which is ideal for detecting the outline of the hand. cv2.CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments, keeping only their endpoints, thus reducing memory usage.

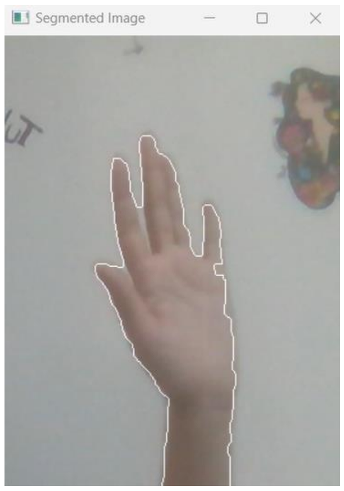

Figure 5: Contour Image.

7.2 Hand Structure Confirmation

The max_contour = max(contours, key=cv2.contourArea) function is used to select the maximum contour. The selected area is assumed the hand. This is based on the assumption that the hand is the most significant object in the region. The bounding rectangle and aspect ratio is calculated using cv2.boundingRect() function. The bounding rectangle around the largest contour helps in verifying the contour's validity. Aspect ratio and Extent is calculated by,

aspect_ratio = w / float(h) and extent = area / float(rect_area) ………….(1)

The aspect ratio and extent (ratio of contour area to bounding rectangle area) are used to confirm the hand's structure. Acceptable ranges ensure the detected contour is likely a hand (neither too elongated nor too compact).

The cv2.drawContours(region, [hand_contour], -1, (255, 255, 255)) function is used to draw the detected hand contour on the region for visual confirmation. This step aids in debugging and ensures that the hand's outline is correctly identified.

8 Feature Extraction

The get_hand_data(segmented_image, hand) function extracts hand features by identifying the fingertips using the convex hull of the hand contour.

8.1 Convex Hull Calculation

The cv2.convexHull() function computes the convex hull of the detected hand contour. The convex hull is the smallest convex shape that encloses all contour points, effectively outlining the hand's outer boundary.This helps in identifying fingertips and filtering out concave points, which are not part of the finger tips.

8.2 Identifying Fingertips

The hand.top_points = [] resets the list of fingertip coordinates. Each point in the convex hull is considered a potential fingertip.Fingertips are grouped based on their horizontal position within the region of interest (ROI). This is done by dividing the width of the ROI into segments, where each segment corresponds to a finger.

finger_index = x // (region_width // 5) ………….(2)

The hand is divided into five segments (one for each finger). Points within the same segment are grouped together in a dictionary (finger_tips), with the segment index as the key.The topmost point (minimum y-coordinate) is selected as the fingertip for that segment. This is based on the assumption that fingertips are the highest points in their respective segments.

topmost_point = min(points, key=lambda point: point[1]) ………..(3)

It finds the point with the smallest y-value, which represents the fingertip.Each identified fingertip is appended to the hand.top_points list, which stores the coordinates of the detected fingertips for further analysis and gesture recognition.

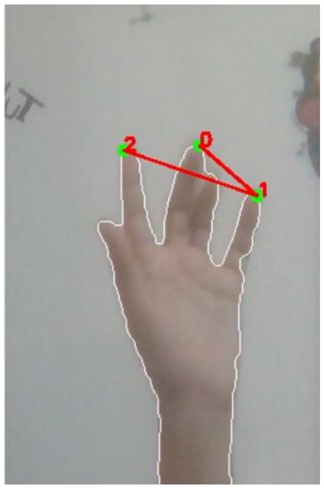

Figure 6: Hand Top Tip Points.

9 Gesture Analysis

9.1 Distance Calculation

Distance and angle are calculated to interpret hand gestures for controlling image zoom and rotation based on the relative positions of the thumb and index finger. The distance between two points (thumb tip and index tip) is calculated using the Euclidean distance formula:

d_thumb_index = int(np.sqrt((index_tip[0] - thumb_tip[0]) ** 2 + (index_tip[1] - thumb_tip[1]) ** 2)) ………………..(4)

This distance helps determine the zoom level. A significant change in distance indicates a pinch gesture, used to zoom in or out on the image.

9.2 Angle Calculation

The calculate_angle(point1, point2) function calculates the angle between two points using the arctangent function. The difference in x and y coordinates between the two points is determined by,

dx = point2[0] - point1[0] ………… (5)

dy = point2[1] - point1[1] ………… (6)

The arctangent function is used to calculate the angle in radians.

angle = np.arctan2(dy, dx) ………….. (7)

This angle is converted from radians to degrees for easier interpretation. The calculated angle helps in determining the rotation gesture. Changes in angle between frames indicate a rotational movement, which is used to rotate the image accordingly.

10 Gesture Interpretation

10.1 Zoom Mode

A significant change in the distance between the thumb and index finger indicates a zoom gesture.If the distance increases, it suggests zooming in; if it decreases, it suggests zooming out. The median distance is compared to the initial distance (startDistance) and scale is calculated by following formula:

scale = ((median_distance - startDistance) // 2) .... (8)

10.2 Rotation Mode

The change in angle between the thumb and index finger is used to determine the rotation of the image. The difference between the current angle and the initial angle (startAngle) indicates the degree of rotation.

11 Mode Selection and Action

The zoom mode adjusts the image size based on the distance between fingertips and the rotation mode rotates the image based on the angle between fingertips. The current mode is switched based on user input, providing flexibility in interaction.

12 Visualization

The segmented region displays the portion of the frame where the hand is detected. It helps verify if the hand is correctly segmented from the background.

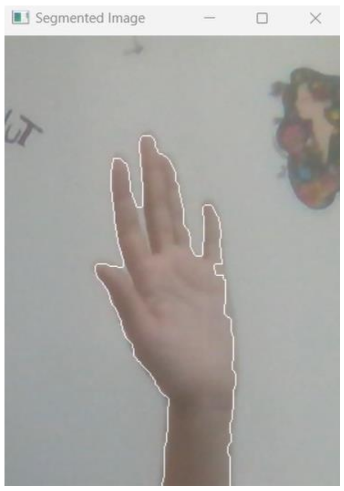

Figure 7: Segmented Image.

The skin mask shows the result of skin color detection, highlighting areas likely to contain skin. The background mask displays the binary mask of the detected hand contour.

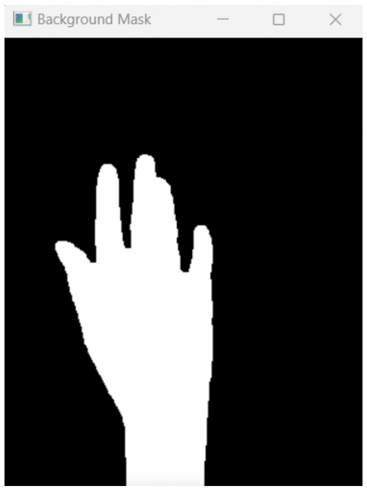

Figure 8: Background Mask.

The combined mask merges the skin and background masks to isolate the hand more accurately. The final segmented image displays the segmented hand with contours drawn, aiding in visual confirmation of hand detection.

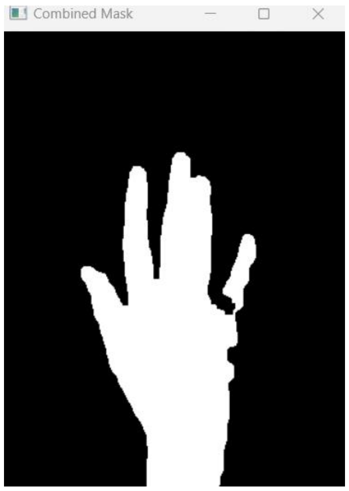

Figure 9: Combined Mask.

The main image (img) is resized or rotated based on the detected gestures (zoom or rotate) and displayed on the screen. Resizing adjusts the scale based on the distance between fingertips. Rotation adjusts the angle based on the change in angle between fingertips.

The keyboard input allows the user to control the program interactively by switching modes or adjusting parameters. Key presses and actions increase OBJ_THRESHOLD, which adjusts the sensitivity of background subtraction. Higher values make the system more selective in detecting hand regions. OBJ_THRESHOLD is decreased making the system more sensitive and likely to include more background noise. The main loop continuously captures frames, processes hand gestures, and updates the display until the user presses the exit key. It keeps the application responsive to user input and updates the visual feedback in real-time, enhancing interactivity and user experience.

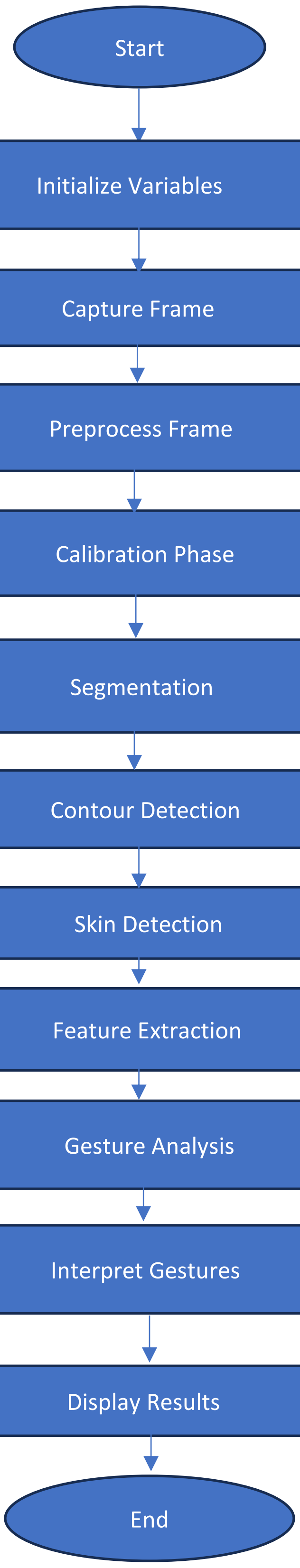

Figure 10: Flowchart of Dynamic Image Gestures: Zoom in and out and Rotate.

Programming Environment

1. Visual Studio Code
2. Spider

Programming Language

1. Python 3.11.3

Key Features

1. **Real-time Gesture Recognition**: The system processes video frames in real-time to detect and interpret hand gestures.
2. **Image Manipulation**: Users can zoom in/out and rotate the image using simple hand gestures.
3. **Background Calibration**: The system calibrates the background to enhance hand segmentation accuracy.
4. **Gesture Interpretation**: The system differentiates between zoom and rotate gestures based on the relative positions of fingertips.

Final Output

The final output of the project is a dynamic image manipulation system that allows users to zoom in/out and rotate an image using hand gestures. The system captures real-time video input, processes the frames to detect hand gestures, and applies the corresponding transformations to the image.

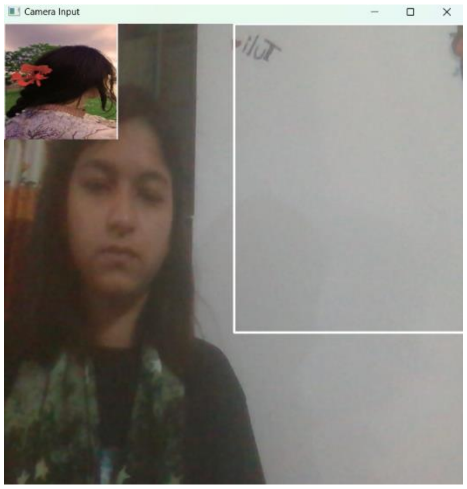

Figure 11: Initial Frame.

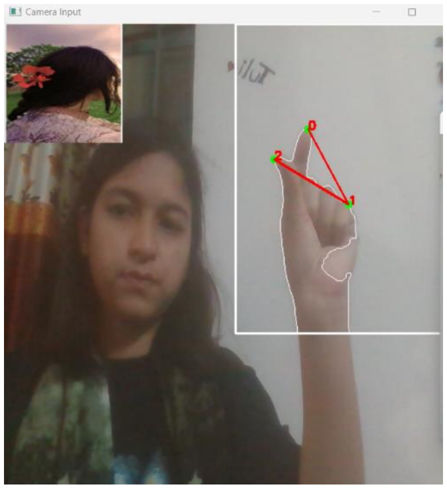

Figure 12: Zoom in Operation.

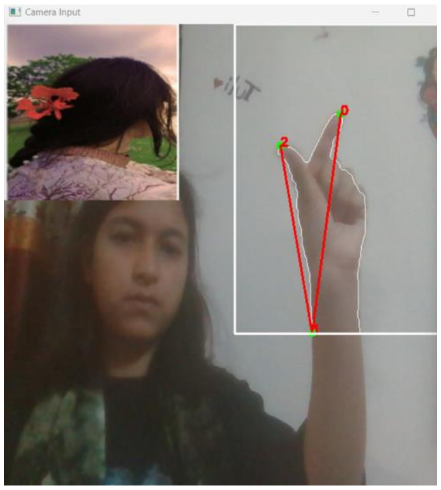

Figure 13: Zoom out Operation.

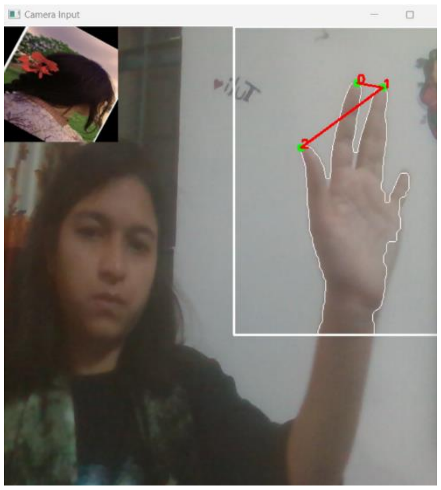

Figure 14: Rotation Operation.

Limitations

1. **Lighting Conditions**: The accuracy of hand segmentation and gesture recognition can be affected by varying lighting conditions.
2. **Background Clutter**: Complex or cluttered backgrounds can interfere with hand detection and segmentation.

3. **Gesture Complexity**: The system may have difficulty recognizing complex gestures or gestures performed too quickly.

Discussion

The development of this project highlights the potential of gesture-based interfaces in enhancing user interaction with digital content. By leveraging computer vision techniques, the system provides an intuitive way to manipulate images. However, the limitations identified suggest areas for future improvement, such as enhancing robustness to different lighting conditions and improving gesture recognition accuracy.

Conclusion

This project demonstrates the feasibility of using hand gestures to control image manipulation operations such as zooming and rotating. The system successfully integrates real-time video processing, hand gesture recognition, and image transformation to provide an intuitive and interactive user experience. Future work could focus on expanding the range of gestures recognized and improving the system's robustness to environmental variations.