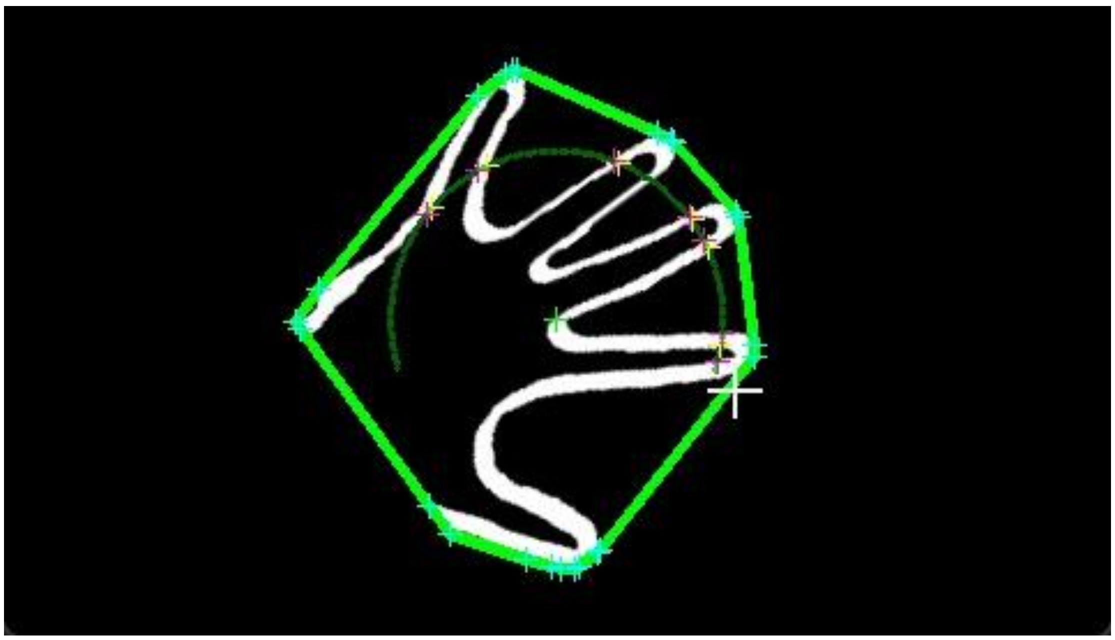
CS585 Assignment 2
Video Processing & Shape Analysis



Munir Siddiqui

Teammate: Muhammad Aseef Imran

14 February 2024

1. Problem Definition

i. **Description:**
The problem at hand is to design and implement an algorithm that recognizes hand shapes or gestures, and to create a graphical display that responds to the recognition of the hand shapes or gestures. This is a task with broad applications in human-computer interaction, accessibility, gesture-based control systems, entertainment, and healthcare. By enabling users to interact with devices and systems using natural hand movements, gesture recognition enhances user experience, accessibility for individuals with disabilities, and usability in various environments where traditional input methods are impractical.

ii. **Our Implementation:**
The primary challenge is to accurately detect and interpret hand gestures captured by a webcam in real-time. We decided to implement an algorithm that detects the number of fingers raised in one hand by a person, and then use that algorithm to play a fun memory game. Our goal is to recognize hand gestures captured by a webcam, interpret them as numbers, and then compare the guessed number with a randomly generated number for each round of the game. This involves several key aspects:

a. Hand Detection: The algorithm needs to identify and isolate the hand region from the background and other objects in the webcam feed. Accurate hand detection is essential for subsequent analysis and recognition of gestures.
b. Finger Detection: The algorithm must accurately count the number of fingers visible in the detected hand region. Finger detection involves analyzing the contours and shapes within the hand region to infer the positions and orientations of individual fingers.
c. Gesture Interpretation: Once the fingers are detected, the system must interpret their configuration as numeric values. Each numeric value corresponds to a specific hand gesture, representing numbers from one to five. The accuracy of gesture interpretation is crucial for the player to correctly guess the randomly generated number in the memory game.
d. Real-time Performance: The system must operate in real-time, processing webcam frames at a sufficient speed to provide a smooth and interactive user experience. Achieving real-time performance requires efficient algorithms and optimized code implementation.

iii. **Assumptions:**
Our algorithm makes the following assumptions, all of which are related to the hand-detection part of the algorithm:

a. The background does not contain any skin color, such as wood or skin-colored walls.
b. The lighting condition is ideal, so that the hand is illuminated properly and can be detected as having skin color.
c. The user's face is not in the frame. If it is, then it must not overlap with the hand and in perspective, it must be smaller than the user's hand. In other words, the user's hand must be the largest isolated skin-colored object in the frame.
d. The user is wearing full sleeves, as the wrist causes problems with the hand-detection part of the algorithm.
e. The user's hand is rotated between 0° and 90° clockwise/anticlockwise.

iv. **Anticipated Difficulties:**
We anticipated that the following tasks would be challenging:

a. Determining where the hand ends and the wrist begins.
b. Finding out the orientation of the hand and rotating it back to 0°.
c. Identifying the number of fingers raised in the hand.

2. Method and Implementation

The implemented method involves several steps. The helper function which implements each step is written in parentheses after each step:

i. **`count_fingers()`:**

a. Preprocessing:
- Blurring to remove noise. (`apply_smoothing`)
- Masking to isolate skin-colored objects. (`mask_image`)
- Conversion to a binary image. (`convert_to_binary`)
- Dilation to smoothen the edges of the binary image. (no separate helper function)

b. Object Extraction:
- Using contouring, finding the largest object in the binary image, assumed to be the hand. (`binary_img_extract_largest_obj`)
- Refining the object by reducing contour complexity and removing defects. (`angle_contour_reducer`, `defects_remover_via_angle_checking`)
- Dilation once again to reduce further defects. (no separate helper function)

- Finding largest object in the binary image once again – this helps to fill in the holes inside the binary image of the hand. (`binary_img_extract_largest_obj`)

c. Analysis:

- Scaling the object so that the hand does not get cropped when it is later rotated. (`scale_obj`)
- Finding the axis of least inertia to determine the orientation of the hand. (`find_axis_of_least_inertia`)
- Translating the hand to the center of the image to facilitate rotation. (`move_obj_to_center`)
- Rotating the hand to 0° based on the calculated orientation. (`rotate_at_center`)

d. Recognition:

- Finding the largest object yet again in the scaled and rotated image. (`get_fingers_in_frame`)
- Finding the bounding box that completely bounds the hand. (`get_fingers_in_frame`)
- Drawing an arc of radius one third of the height of the bounding box (determined experimentally), centered at the centroid of the hand. (`get_fingers_in_frame`)
- Counting the number of fingers intersecting with the arc by considering transitions from background to foreground pixels and vice versa. (`get_fingers_in_frame`)
- Calculating the circularity of the hand to distinguish between four and five fingers. (`calc_roundness`)
- Using the mean and standard deviation of the number of fingers counted to eliminate anomalies. (no separate helper function)

ii. `main()`:

a. Game Logic:

- Displaying the graphics for the game
- Generating a random number for the player to memorize.
- Capturing the player's guessed number based on hand gestures. (`count_fingers`)
- Comparing the guessed number with the generated number to determine correctness.

Here are some images of how the hand looks during each step of the algorithm:

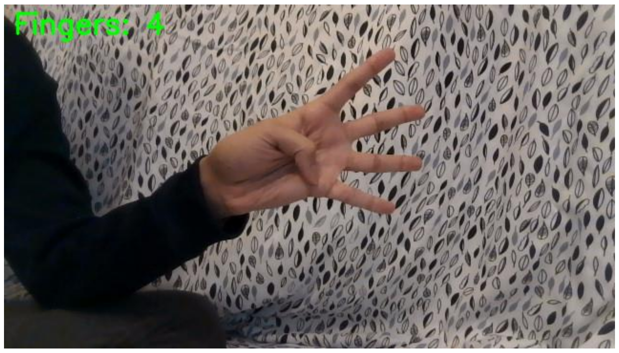


Fig 2.1 - Original frame; user sees this during the game

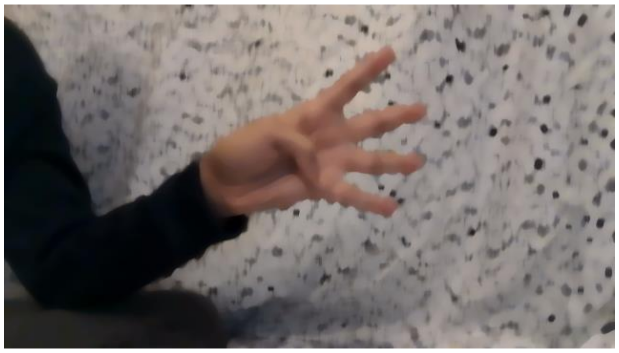


Fig 2.2 - Image is blurred to reduce noise

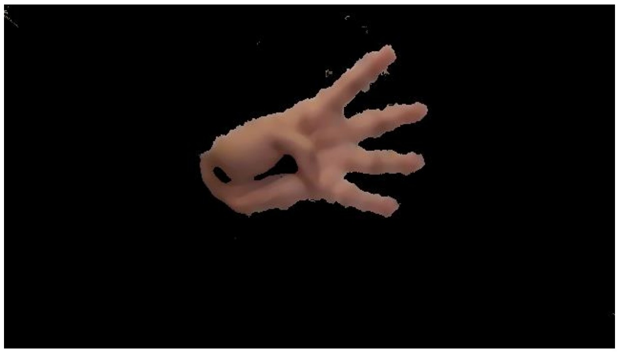


Fig 2.3 - Masked image with only skin-color pixels remaining

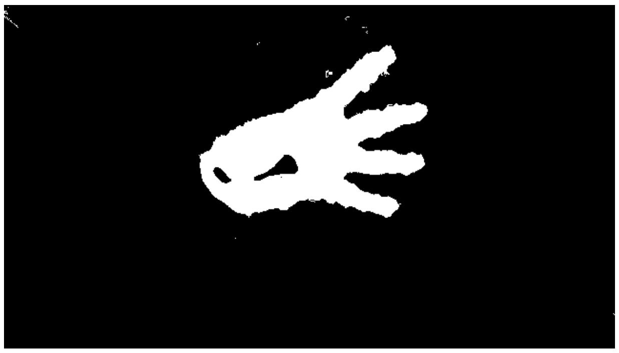


Fig 2.4 - Colored image is converted to a binary image

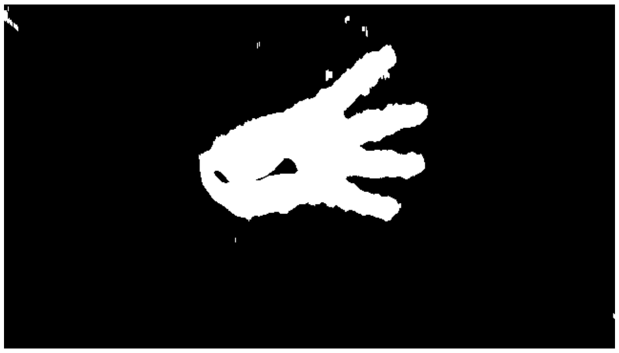


Fig 2.5 - Image after first dilation

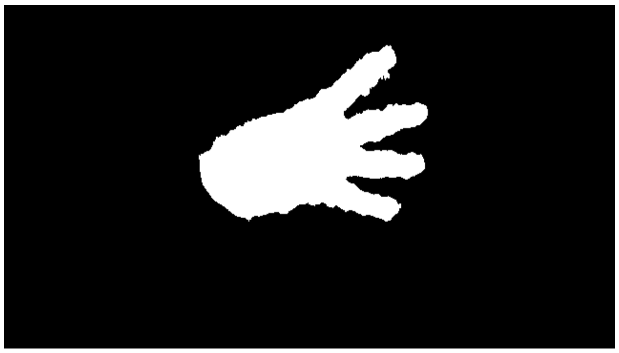


Fig 2.6 - Largest object extracted from the image

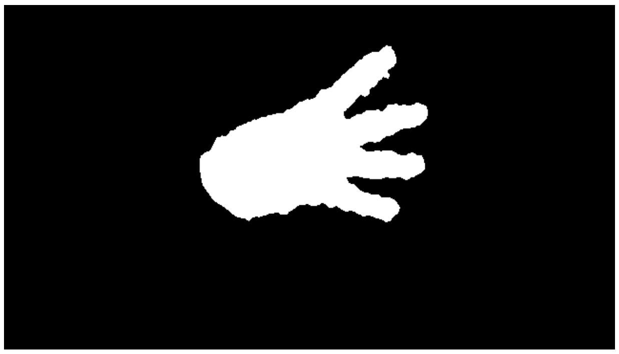


Fig 2.7 - Image after reducing contours and removing defects

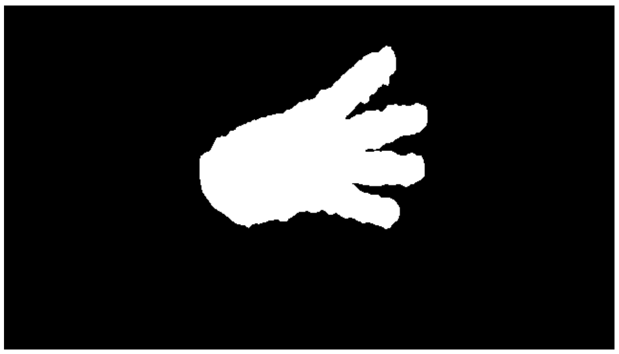


Fig 2.8 - Image after second dilation

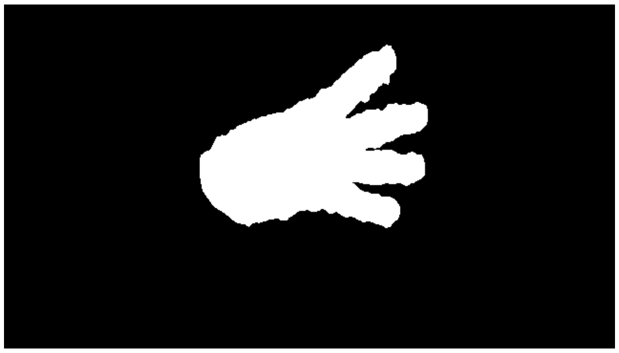


Fig 2.9 - Largest object extracted from the image again

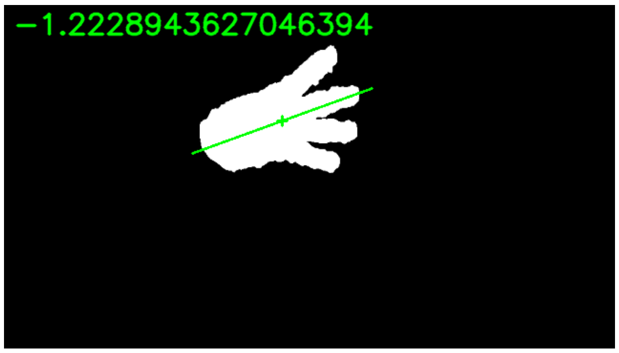


Fig 2.10 - Scaled image with axis of least inertia marked

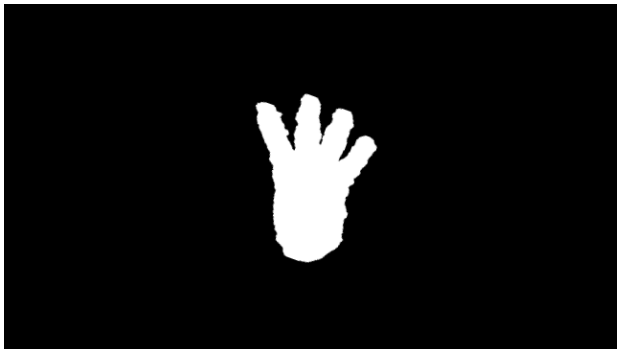


Fig 2.11 – Image is centered and rotated to 0°

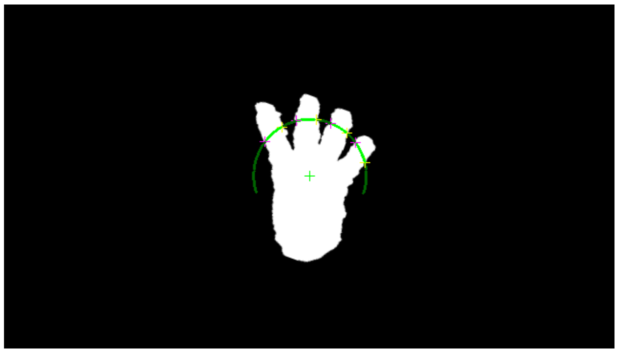


Fig 2.12 – Final image with arc drawn to count fingers

3. Experiments

i. Description:
In order to test the performance of our algorithm, we used a sample size of 250, with 50 trials for each of the five gestures. We wrote a *Python* script to automate this process. Each trial was conducted by the same person, but with different hand orientations, distances from the camera, and finger combinations.

ii. Evaluation Metrics:
We used *sklearn.metrics* to create a confusion matrix for the 250 trials. Using the confusion matrix, we calculated the accuracy of the algorithm by adding up the values along the diagonal and dividing by the total number of trials.

4. Results

We obtained the following confusion matrix:

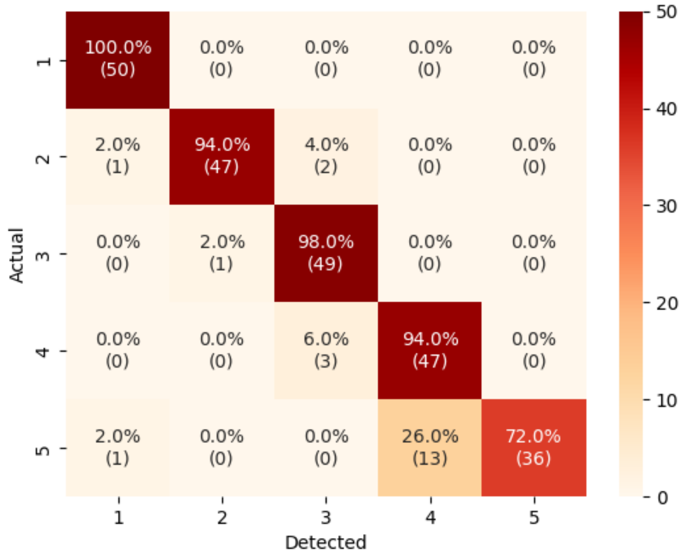


Fig 4.1 - Confusion matrix for the number of fingers detected by the algorithm

We can see that the algorithm is effective in detecting one to four fingers. However, the algorithm's ability to detect five fingers is 22 to 28 percent points worse, and it often (13/50 times) incorrectly classifies five fingers as four. It rarely miscounts two to three fingers, which might be due to anomalies such as poor lighting, distance from the camera, or unusual combinations of fingers.

The accuracy of the algorithm is:

$$Accuracy = \frac{50 + 47 + 49 + 47 + 36}{250} \times 100 = 91.6\%$$

5. Discussion

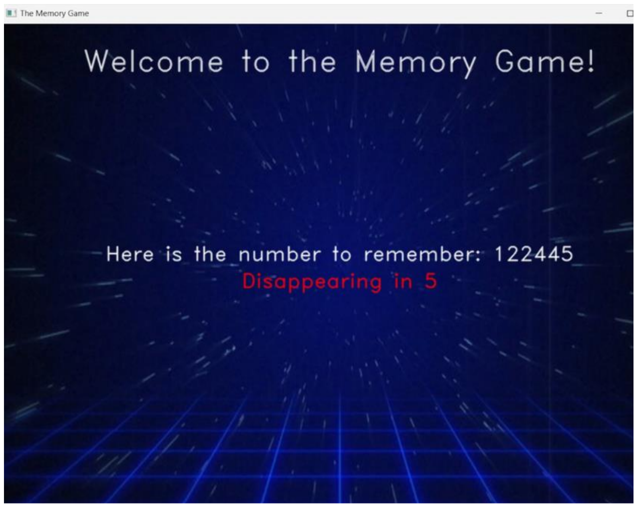
i. **Strengths:**
   a. The algorithm has a high overall accuracy of 91.6%.
   b. The algorithm performs extremely well for one to four fingers.
   c. The algorithm is fairly efficient as it detects gestures in an adequate amount of time.

ii. **Weaknesses:**
   a. The algorithm is not as accurate when detecting five fingers as there are scenarios where both `get_fingers_in_frame` and `calculate_roundness` fail.
   b. The algorithm only works with certain backgrounds where the skin color detection works well.
   c. The algorithm misbehaves when the user's wrist is visible as that impacts the angle of least inertia calculation.

iii. **Potential Future Work:**
   a. Come up with a method to determine the boundary between the hand and the wrist.
   b. Improve skin-color recognition or use an alternative method such as template matching so that the algorithm performs better with various backgrounds.
   c. Test the algorithm with different hand shapes and skin colors to get a more accurate performance analysis.
   d. Find a way to better distinguish between four and five fingers in order to improve the accuracy of the algorithm.

6. The Graphical Display

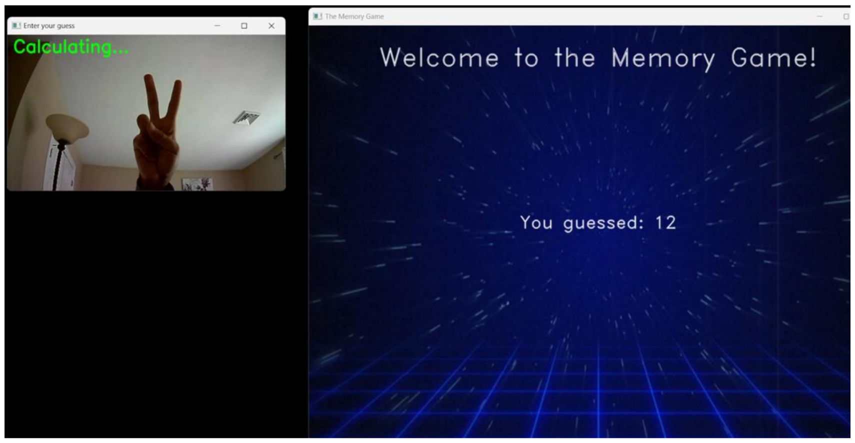
We decided to create a fun graphical display to show our algorithm in action. As mentioned in section 1, subsection ii, we created a memory game. The game begins by giving the user a few seconds to memorize a sequence of numbers.



Here is the number to remember: 122445
Disappearing in 5

Fig 6.1 - The game begins by displaying a sequence of numbers

Then, the user uses hand gestures to recreate that sequence, one digit at a time.



You guessed: 12

Fig 6.2 - The user uses gestures to input numbers

Based on the sequence input by the user, the user is shown the results screen, where they can choose to replay the game or quit.

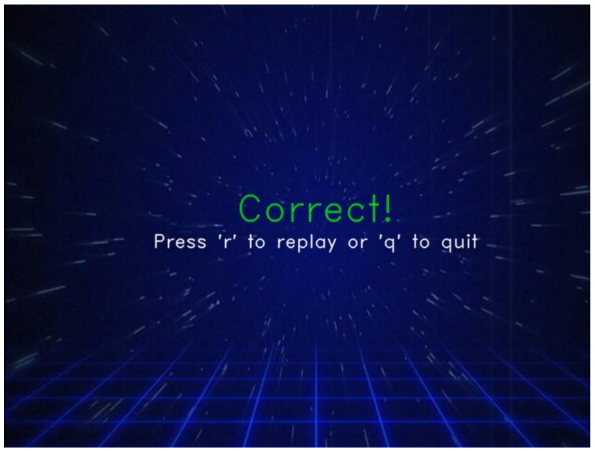


Fig 6.3 - The user correctly recreates the sequence

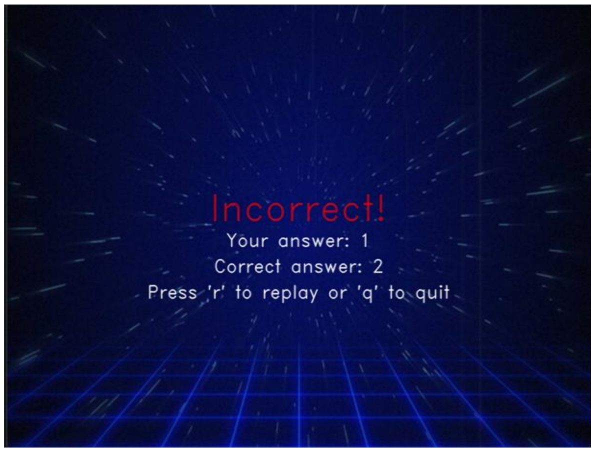


Fig 6.4 - The user is not able to recreate the sequence

A demo of the gameplay can be found here: youtube.com/watch?v=Y1NxWTVGxM0.

7. Conclusion

I am very satisfied with the outcomes achieved by our algorithm. Despite areas with potential for much improvement, the overall performance, especially in terms of accuracy, surpasses our expectations. Beyond the current application, we envision numerous other potential uses for this algorithm, including educational counting games for young children and various scenarios where precise hand gesture recognition is essential.

8. Sources

i. Djamila Dahmani, Mehdi Cheref, Slimane Larabi, “Zero-sum game theory model forsegmenting skin regions”, *Image and Vision Computing*, Volume 99, 2020, 103925, ISSN 0262-8856, doi.org/10.1016/j.imavis.2020.103925.

ii. Nayar, Shree, lecturer. “Geometric Properties | Binary Images. First Principles of Computer Vision.” *Youtube*, 1 Mar. 2021, youtube.com/watch?v=ZPQiKXqHYrM.

iii. Computer Science. “YCbCr and RGB Colour.” *YouTube*, 23 Nov. 2019, youtube.com/watch?v=3dET-EoIMM8.

iv. The Scientist Algorithm. “Color Models | RGB, HSL, HSV and CMYK.” *YouTube*, 21 Mar. 2021, youtube.com/watch?v=cKaRxd6tdHY.

v. freeCodeCamp.org. “OpenCV Course - Full Tutorial with Python.” *YouTube*, 3 Nov. 2020, youtube.com/watch?v=oXlwWbU8l2o.

vi. Background image for graphical display. stock.adobe.com/images/white-particle-coming-from-the-background-above-squared-floor-4k/454107697?prev_url=detail.

vii. GitHub Copilot.
Prompt: Rewrite this code using numpy:

```
m10 = 0
m01 = 0
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if img[i, j] > 0:
            m10 += i
            m01 += j
m11 = 0
m20 = 0
m02 = 0
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if img[i, j] > 0:
            m11 += (i - x) * (j - y)
            m20 += (i - x) ** 2
            m02 += (j - y) ** 2
```

Output:

```
m10 = np.sum(np.where(img > 0)[0])
m01 = np.sum(np.where(img > 0)[1])
m20 = np.sum((np.where(img > 0)[0] - x) ** 2)
m11 = np.sum((np.where(img > 0)[0] - x) * (np.where(img >
      0)[1] - y))
m02 = np.sum((np.where(img > 0)[1] - y) ** 2)
```