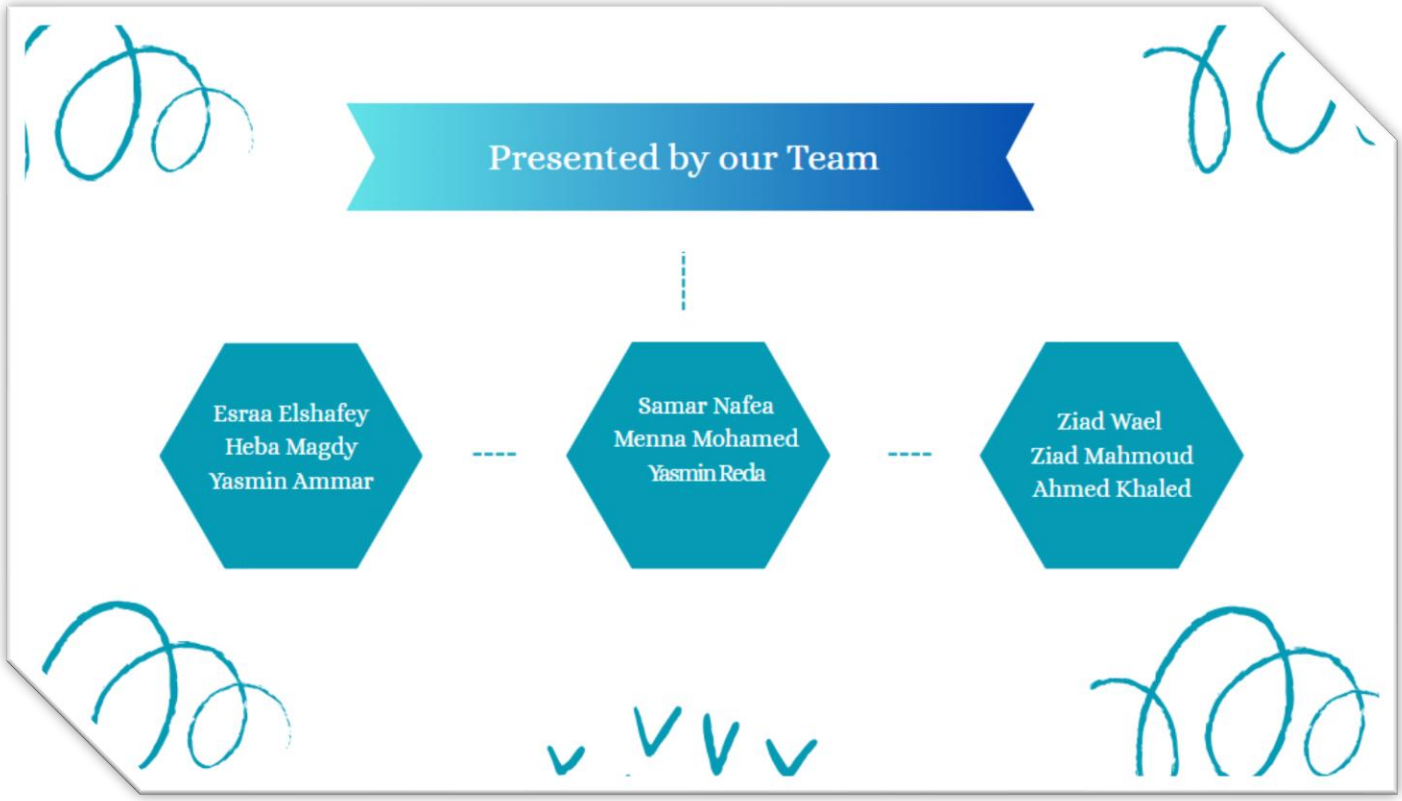
MATLAB Face Recognition System

Record new faces, save them in the database, and recognize them when new photos are entered.

Dr / Azhar Ahmed


Presented by our Team
Esraa Elshafey
Heba Magdy
Yasmin Ammar
Samar Nafea
Menna Mohamed
Yasmin Reda
Ziad Wael
Ziad Mahmoud
Ahmed Khaled

MATLAB Face Recognition System

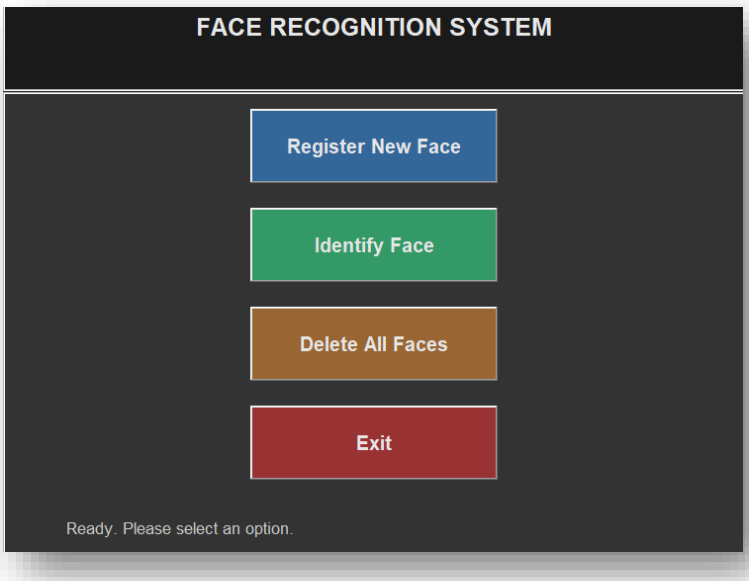
Overview

This project is a dark-themed GUI-based face recognition system implemented in MATLAB. It allows users to register new faces, identify faces from images, delete all saved faces, and visualize feature similarities.

GUI Structure and Theme

The GUI is composed of:

- **Main Window**: A dark-themed interface with a title and four buttons.
- **Buttons**:
  - Register New Face
  - Identify Face
  - Delete All Faces
  - Exit
- **Status Text**: A dynamic message panel that displays system states and updates.



GUI Theme

A function `setDarkTheme()` is used to globally apply a consistent dark mode to all GUI components, adjusting background and text colors.

Core Functionalities

1. Register New Face

- Opens a file dialog to choose an image.
- Detects and extracts a face using `detectAndExtractFace()`.
- Displays the face in a preview window.
- Prompts the user to enter a name.
- Saves the face image and extracts features using `extractImprovedFaceFeatures()`.
- Stores features and image in the saved faces folder.

2. Identify Face

- Opens a file dialog to choose an image.
- Detects and extracts a face from the image.
- Extracts features and compares them against saved faces.
- Uses `calculateImprovedFeatureDistance()` to measure similarity.
- Displays the query face, top matches, and highlights the best match if found.

3. Delete All Faces

- Confirms with the user using a modal dialog.
- Deletes all files in the `saved_faces` directory.

Face Detection Process

Implemented in detectAndExtractFace():

- Reads image and converts to grayscale.
- Applies histogram equalization (myHistEq()) and Gaussian smoothing (myGaussianSmooth()).
- Detects skin regions using thresholds in YCbCr and RGB.
- Extracts region properties and filters based on:
    - Minimum area
    - Aspect ratio (width/height)
    - Eccentricity (face should be oval)
    - Extent (area fill ratio)
- Eye detection in candidate regions using detectEyePair().
- Applies non-maximum suppression to select the best face.
- Adds a margin and resizes the face to 160x160 pixels.

Feature Extraction Methods

Defined in extractImprovedFaceFeatures():

Local Binary Patterns (LBP)

- Face is divided into 20x20 pixel cells.
- For each pixel in a cell, compares to neighbors and generates a binary code.
- Builds histogram of LBP values per cell.
- Histograms are normalized and concatenated.

Histogram of Oriented Gradients (HOG)

- Computes gradients (magnitude + direction).
- Builds histograms of directions within cells.
- Normalized histograms are added to the feature vector.

Gabor Filters

- Applies filters in 8 orientations.
- Extracts mean and standard deviation of filtered results.
- Represents texture and edge direction features.

Feature Normalization

- All features (LBP, HOG, Gabor) are concatenated.
- Final vector is normalized using L2 norm.

Face Matching and Similarity

Performed by `calculateImprovedFeatureDistance()`:

- Splits the feature vector into three parts: LBP, HOG, Gabor.
- Compares features using:
    - **LBP**: Chi-square distance
    - **HOG**: Euclidean distance
    - **Gabor**: Cosine distance
- Final score = 0.3 × LBP + 0.3 × HOG + 0.4 × Gabor
- Lower scores indicate higher similarity.

Dialog Windows

- Custom errorDialog and successDialog windows use dark color schemes.
- Display feedback for both success and failure operations.

Feature Visualization

- Uses bar charts to show feature similarity between query and matched face.
- Colors are adjusted for visibility in dark mode.

Helper Functions

- cleanNameForFilename(): Sanitizes names for file storage.
- detectSkinRegions(): Combines RGB and YCbCr logic for accurate skin masking.
- myEdgeDetection(): Simple Sobel-based edge detection used in eye detection.
- nonMaximumSuppression(): Filters overlapping detected regions.
- myHistEq() and myGaussianSmooth(): Custom contrast and smoothing functions.
- detectEyePair(): Analyzes edge density in the upper half of the detected region to confirm presence of eyes.
- visualizeFeatureComparison(): Groups and normalizes features for side-by-side comparison in bar charts.
- calculateChiSquareDistance(), cosineDistance(): Mathematical functions for comparing feature vectors.

system Structure and Flow

System Architecture

- Modular functions encapsulated by clear GUI controls.
- Image preprocessing, detection, extraction, and comparison are separate and reusable.

Data Flow

1. **Image Acquisition**: User selects an image via dialog.
2. **Face Detection**: Processed via filters and morphology.
3. **Feature Extraction**: Combines three distinct methods (LBP, HOG, Gabor).
4. **Comparison**: Against all stored features, using multiple distance metrics.
5. **Result Display**: Shows visuals, match text, and percentage similarity.

Performance Considerations

- Uses custom functions for performance and clarity (e.g., smoothing, histogram equalization).
- Designed to handle multiple faces and varied image conditions.
- Non-maximum suppression avoids duplicated detections.

Limitations

- Cannot detect many faces more than one face this is for recognition part.
- if the background is has to many colors it may miss the main face to detect.
- something it may conflict between faces that has minimal matched characteristics.

Challenges Faced

1. Manual Processing Overhead

- Many operations (e.g., filtering, morphology) were manually coded, making the system complex and slow.

2. Limited Accuracy

- Low-level features weren't reliable enough under varying lighting or image conditions.

3. Single Face Recognition Only

- The system can detect and recognize only one face at a time.

4. Background Interference

- Complex or colorful backgrounds may confuse the system, causing it to miss the main face.

5. Feature Confusion

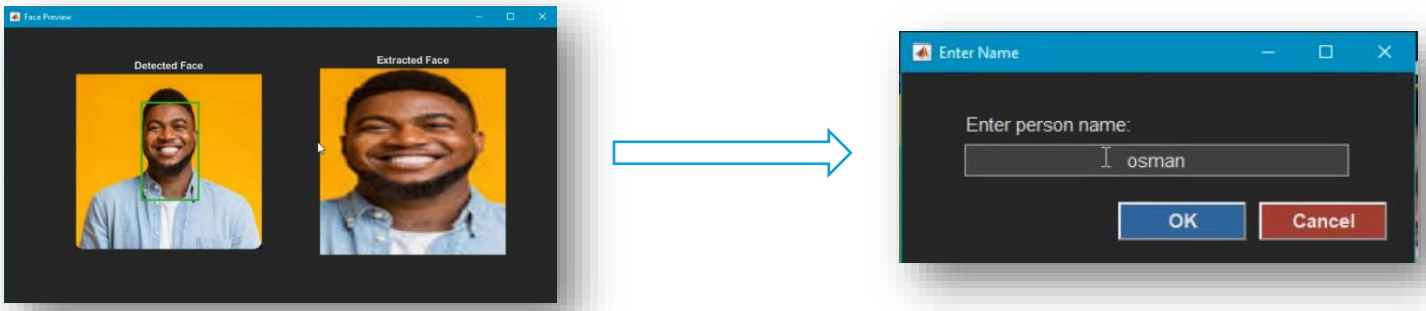
- Faces with similar low-level features may get confused or wrongly labeled.

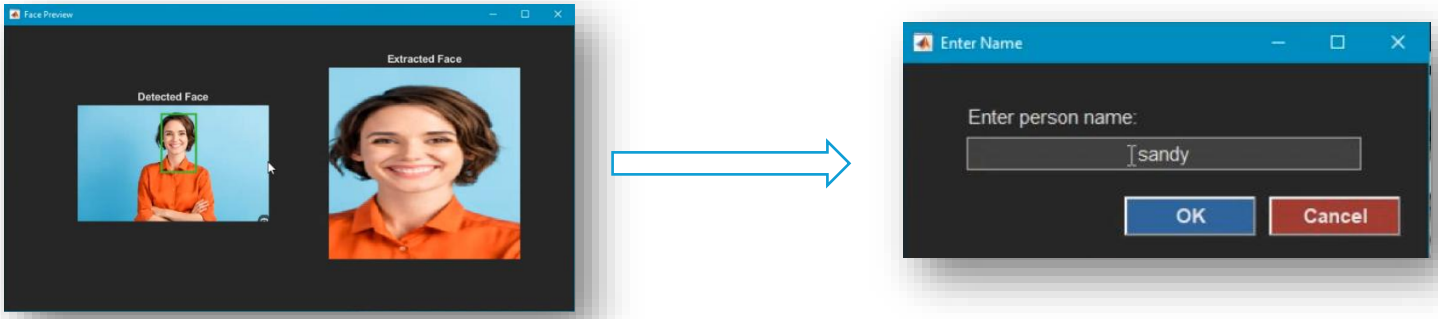
6. Performance Issues

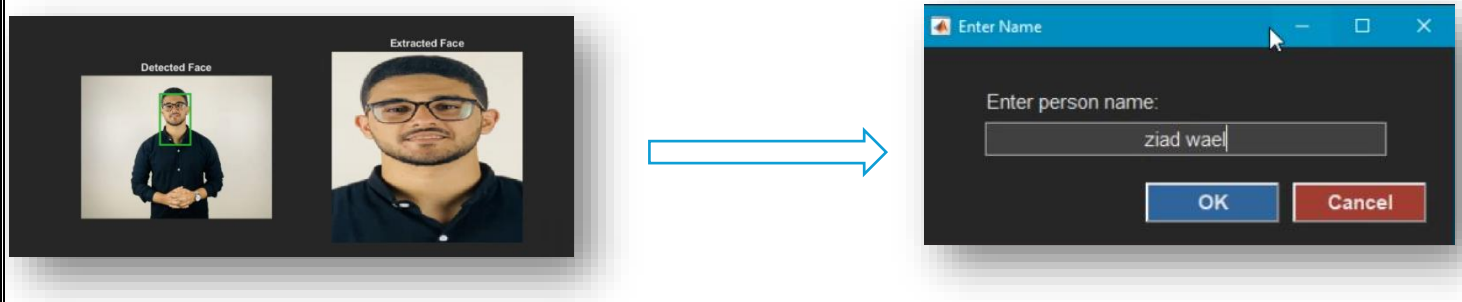
- Manual steps slow down the system, especially with larger images or high resolution.

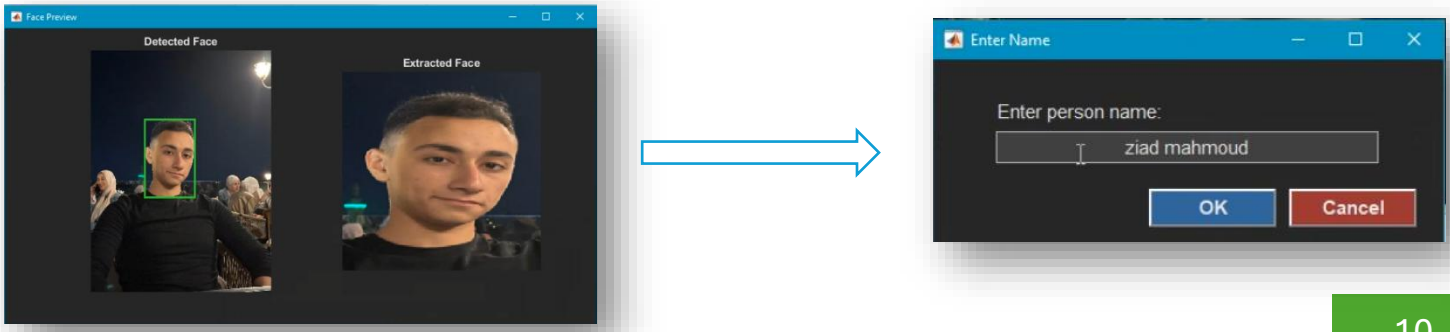
Test the code

- **Register New Face**

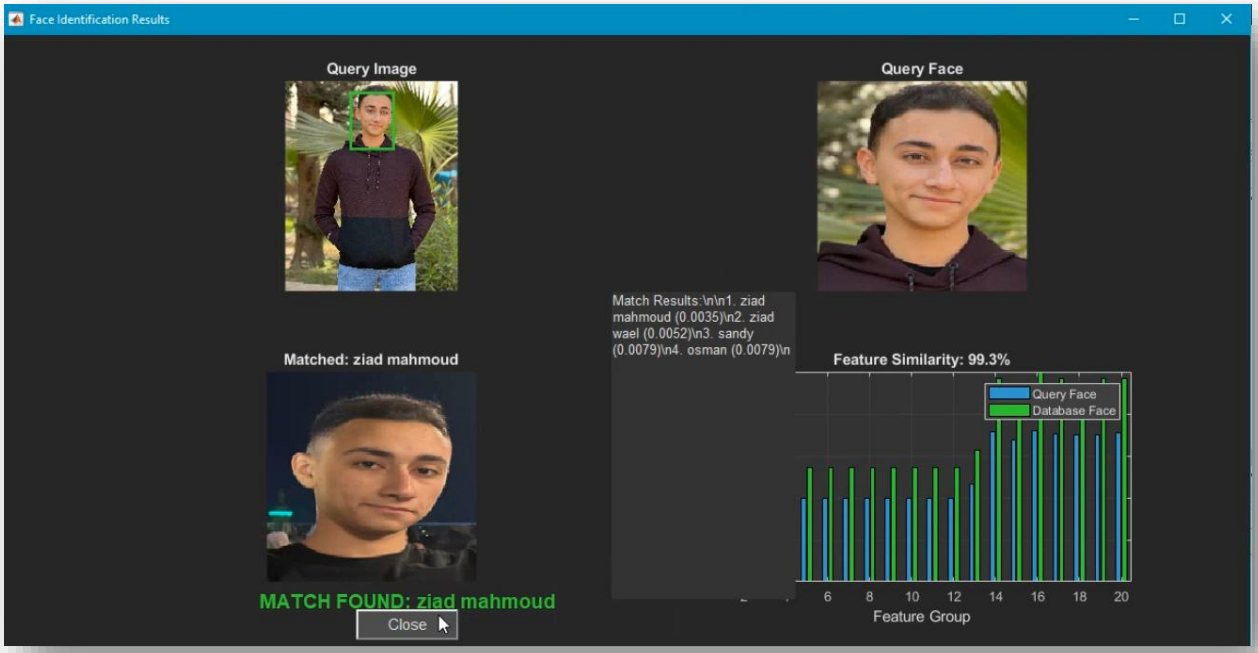




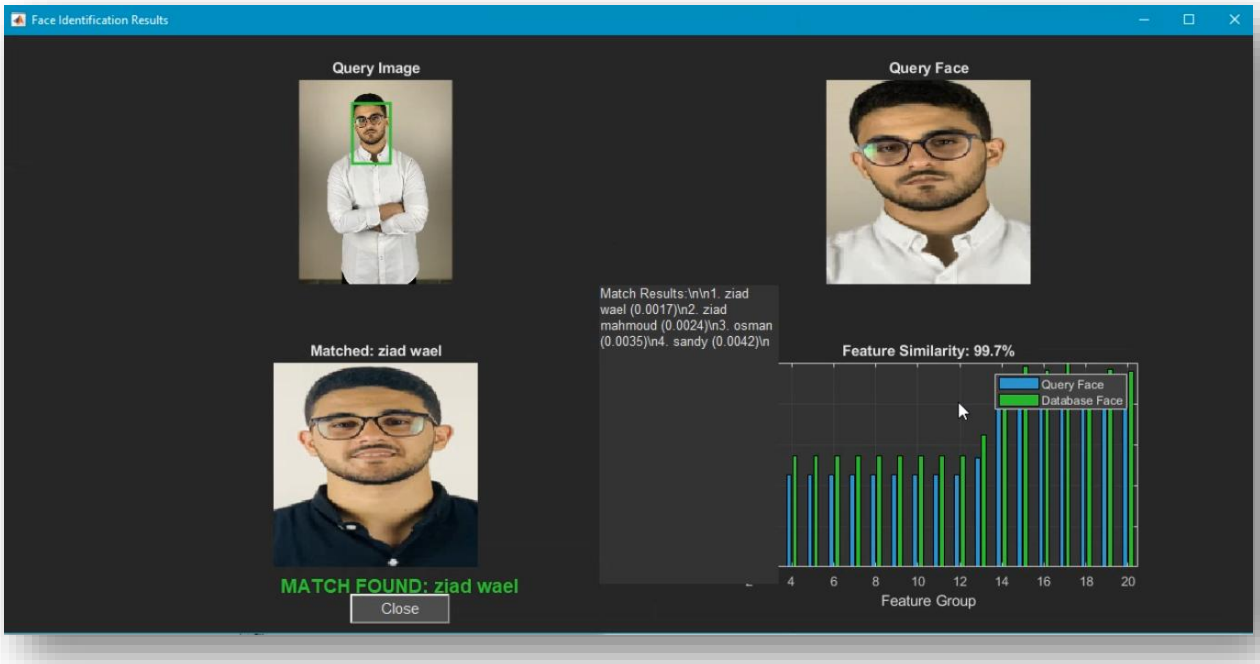




- **Identify Face**

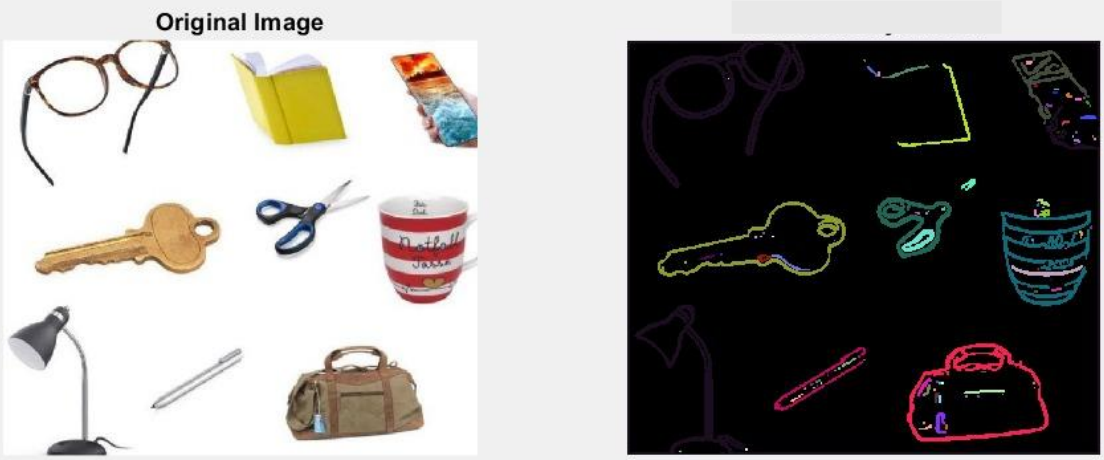


When we entered a new photo for Ziad, the program successfully recognized it.



And so here ……..

Object Detection and Labeling in an Image



This MATLAB script performs object detection and labeling using image processing techniques. The main steps include grayscale conversion, Gaussian filtering, edge detection, morphological operations, connected component labeling, and overlaying detected objects on the original image.

Load the Image

```
img = imread('ay.jpg');
```

The image 'ay.jpg' is loaded into the variable img.

Convert the Image to Grayscale Manually

```
gray_img = 0.2989 * double(img(:,:,1)) + 0.5870 * double(img(:,:,2)) + 0.1140 * double(img(:,:,3));
gray_img = uint8(gray_img);
```

- The image is manually converted to grayscale using the weighted sum formula based on human perception.
- The red, green, and blue channels are multiplied by their respective weights and summed.
- The result is converted back to an 8-bit unsigned integer format (`uint8`).

Apply a Manual Gaussian Filter

Define the Gaussian Kernel

```
sigma = 2;
filter_size = 5;
half = floor(filter_size/2);
[x, y] = meshgrid(-half:half, -half:half);
gaussian_kernel = exp(-(x.^2 + y.^2) / (2 * sigma^2));
gaussian_kernel = gaussian_kernel / sum(gaussian_kernel(:));
```

- A 5×5 Gaussian filter is created with standard deviation `sigma = 2`.
- The kernel values are computed using the Gaussian function and normalized to sum to 1.

Apply the Gaussian Filter

```
smoothed_img = zeros(size(gray_img));
padded = padarray(double(gray_img), [half half], 'replicate');
for i = 1:size(gray_img, 1)
  for j = 1:size(gray_img, 2)
    window = padded(i:i+2*half, j:j+2*half);
    smoothed_img(i,j) = sum(window .* gaussian_kernel, 'all');
  end
end
```

- The grayscale image is padded to handle edge pixels.
- A sliding window applies the Gaussian filter to each pixel, smoothing the image.

Sobel Edge Detection

Define Sobel Filters

```
sobel_x = [-1 0 1; -2 0 2; -1 0 1];
sobel_y = [-1 -2 -1; 0 0 0; 1 2 1];
```

The Sobel operators are used to detect horizontal and vertical edges.

Apply the Sobel Filter

```
edge_x = zeros(size(smoothed_img));
edge_y = zeros(size(smoothed_img));
padded = padarray(smoothed_img, [1 1], 'replicate');

for i = 1:size(smoothed_img, 1)
  for j = 1:size(smoothed_img, 2)
    window = padded(i:i+2, j:j+2);
    edge_x(i,j) = sum(window .* sobel_x, 'all');
    edge_y(i,j) = sum(window .* sobel_y, 'all');
  end
end
```

The image is padded, and a sliding window applies the Sobel filter to compute gradients along the x and y directions.

Compute Gradient Magnitude and Apply Threshold

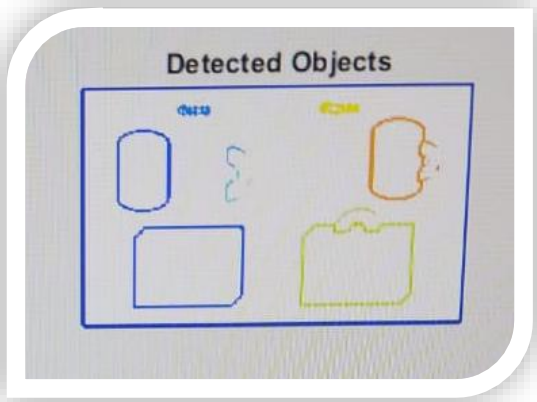
```
edge_mag = sqrt(edge_x.^2 + edge_y.^2);
mean_val = mean(edge_mag(:));
std_val = std(edge_mag(:));
threshold = mean_val + 0.5 * std_val;
binary_edges = edge_mag > threshold;
```

- The edge magnitude is computed using the Euclidean norm.
- A dynamic threshold is determined using the mean and standard deviation of gradient magnitudes.
- A binary edge map is generated.

Morphological Processing (Opening Operation)

Erosion

```
se = ones(3, 3);
eroded = zeros(size(binary_edges));
padded = padarray(binary_edges, [1 1], 0);
for i = 1:size(binary_edges, 1)
  for j = 1:size(binary_edges, 2)
    window = padded(i:i+2, j:j+2);
    eroded(i,j) = all(window(se == 1));
  end
end
```



- A 3×3 structuring element is used for erosion.
- A sliding window checks if all pixels in the window are 1; if not, the pixel is eroded.

Dilation

```
ilated = zeros(size(eroded));
padded = padarray(eroded, [1 1], 0);
for i = 1:size(eroded, 1)
  for j = 1:size(eroded, 2)
    window = padded(i:i+2, j:j+2);
    dilated(i,j) = any(window(se == 1));
```

```
  end
end
clean_img = dilated;
```

- Dilation is applied to restore object boundaries after erosion, completing the opening operation.

Connected Component Labeling

```
labeled = zeros(size(clean_img));
label = 1;
for i = 1:size(clean_img, 1)
  for j = 1:size(clean_img, 2)
    if clean_img(i,j) == 1 && labeled(i,j) == 0
      % Start flood fill
      queue = [i, j];
      while ~isempty(queue)
        r = queue(1,1);
        c = queue(1,2);
        queue(1,:) = [];
```


```
        if r >= 1 && r <= size(clean_img,1) && c >= 1 && c <= size(clean_img,2) && clean_img(r,c) == 1 &&
labeled(r,c) == 0
          labeled(r,c) = label;
          % Add neighbors
          queue = [queue; r-1, c; r+1, c; r, c-1; r, c+1];
        end
```

```
        end
        label = label + 1;
    end
  end
end
num_objects = label - 1;
```

- A flood-fill algorithm assigns unique labels to connected components.
- The `queue` stores pixel locations to be processed.

Overlay Detected Objects on the Original Image

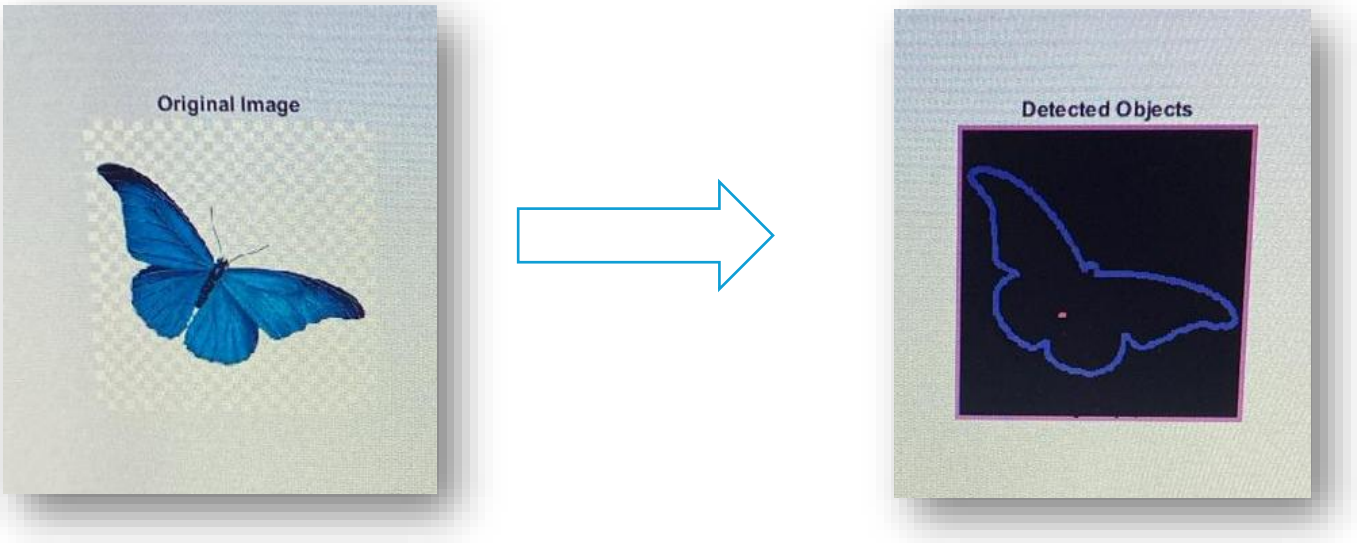
```
img_double = im2double(img);
colors = rand(num_objects, 3);
result_img = img_double; % Start with original image

for k = 1:num_objects
  mask = (labeled == k);
  for c = 1:3
    temp = result_img(:,:,c);
    temp(mask) = 0.5 * colors(k,c) + 0.5 * temp(mask);
    result_img(:,:,c) = temp;
  end
end
```

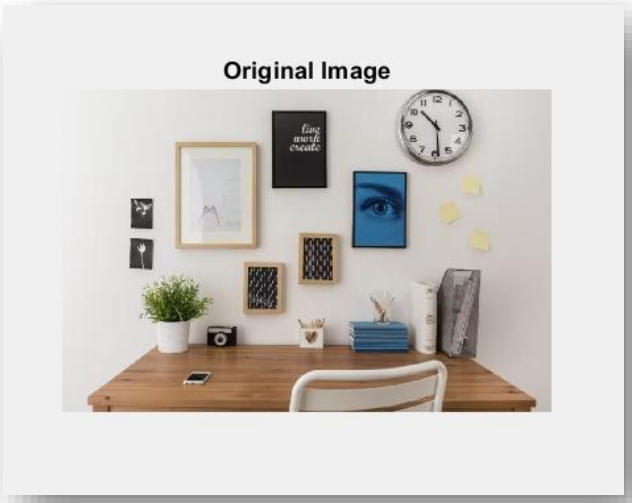
- A color overlay is applied to each detected object using random colors.
- The detected regions are blended with the original image.

Display the Results

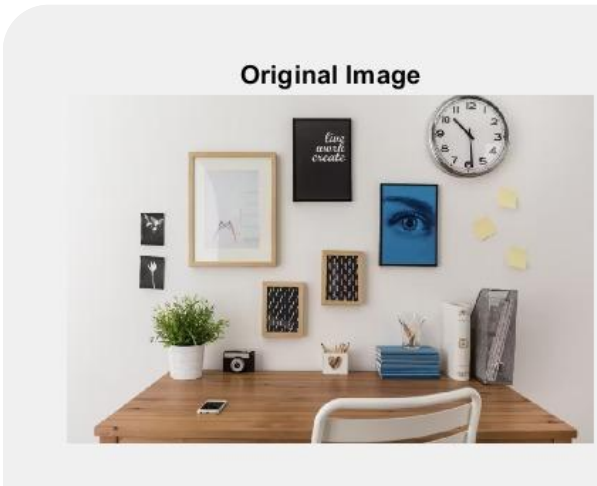
```
figure;
subplot(1,2,1); imshow(img); title('Original Image');
subplot(1,2,2); imshow(result_img); title('Detected Objects');
fprintf('Number of objects detected: %d\n', num_objects);
```

- The original image and the processed image are displayed.
- The number of detected objects is printed.

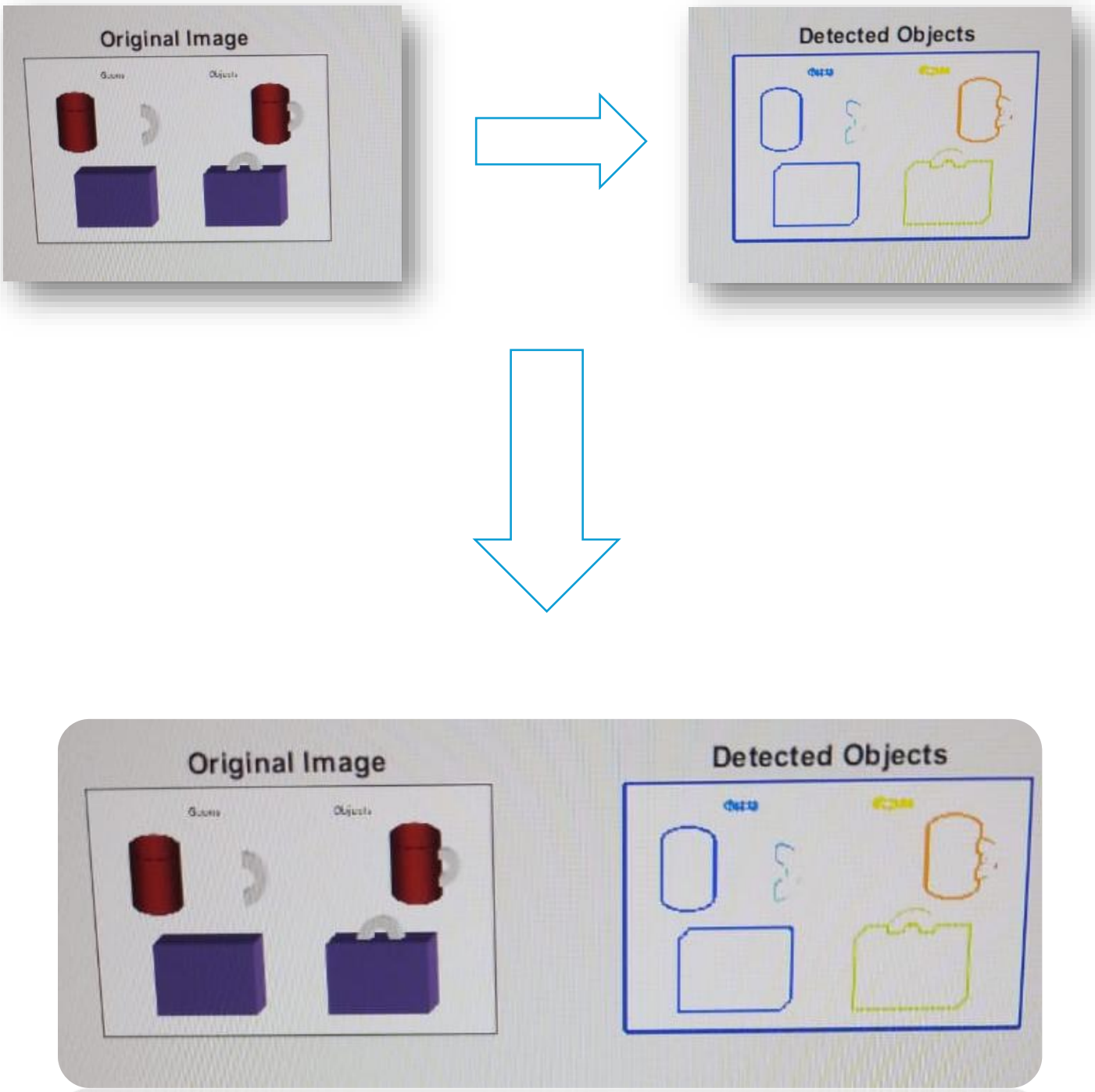



Original Image


Detected Objects


Original Image


Detected Objects: 167


Original Image
Detected Objects
Original Image
Detected Objects