Logo

Description automatically generated**MOD002643 Image Processing**

Coursework Brief for Trimester 1, 2024-25

Student ID: 2179610

**Gesture Recognition**

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Marks Available** |
| **A**. Complete the three MATLAB Training Certificates. | An all-or-nothing mark awarded only if all three training certificates are 100% completed. You are required to supply hyperlinks to your certificates bearing your full name (the same name you are known by at ARU). PDFs are not accepted (see next page). Each certificate takes ~2 hours. | 15 |
| **B**. Complete 11 self-test quizzes with a mark of 12 or higher. | An all-or-nothing mark awarded only if all 11 weekly self-test quizzes are completed with a mark of 12 or higher. Your lecturer will verify completion on Canvas, so you only need to tick to signify completion (see next page). | 15 |
| **C**. Threshold static image to isolate foam finger. | Take a static photograph of yourself holding up your foam finger against a plain background (e.g., blank white wall). Include your face in the image. Using multiband thresholding (only) in RGB space segment the foam finger as neatly as possible, demonstrating your results with a binary image. | 5 |
| **D**. Use morphological and/or non-linear filters to clean thresholded image. | Attempt to convert FNs to TPs and FPs to TNs using morphological operators and/or non-linear filters. Try to close any interior holes and remove marks or writing on your foam finger and remove fragments of noise, demonstrating your results with a binary image and product with RGB image. | 5 |
| **E**. Annotate foam finger boundaries on original image. | Use the non-linear range filter or a linear edge detection filter to trace the boundary around the foam finger. Use this to annotate the foam finger in the original image with a coloured border. | 5 |
| **F**. Calculate centroid and medoid and annotate image. | Determine the centroid and medoid of the foam finger using the binary images produced in Task D. Display the centroid and medoid using two different markers. | 5 |
| **G**. Adapt for real-time webcam use. | Install the webcam Support Package to collect snapshots images from your webcam in a loop in real-time. Refine and apply Tasks C..F to enable the boundary, centroid and medoid to be displayed on the original image in real-time, also displaying the pre- and post-cleaned binary image. | 10 |
| **H**. Real-time gesture recognition. | Add functionality that enables you to discriminate between the following four gestures: up swipe, down swipe, left swipe, right swipe. Annotate the original image stream with the intended gesture, recording video evidence of each. Punctuate between gestures by removing the foam finger from view. An excellent submission would also allow different numeric symbols to be recognised (0..9). | 40 |
| **TOTAL** | | **100** |

**NOTES**: (**1**) The pass mark is 40%, so as a minimum complete A..F. Do not just complete A..D as it is unlikely that you will score full-marks for programming tasks as your answer is unlikely to be objectively perfect. (**2**) You must upload your work in this this template as a Word document (DOCX). See Appendix A for more detailed presentation guidelines.

**A. Training Certificates (0 or 15 Marks)**

MATLAB Onramp Certificate Hyperlink:

https://matlabacademy.mathworks.com/progress/share/certificate.html?id=3047a4e2-d57f-4946-a03b-a2897c573c88

Image Processing Onramp Certificate Hyperlink:

https://matlabacademy.mathworks.com/progress/share/certificate.html?id=19645c93-98dd-472f-884e-9e6105f85e5a

Computer Vision Onramp Certificate Hyperlink:

https://matlabacademy.mathworks.com/progress/share/certificate.html?id=d031c0d1-25d7-4455-9bb0-ac5d30f3362f

**B. Quizzes 1..11 (0 or 15 Marks)**

Tick boxes to confirm completion with a mark of 12/15 or higher.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |

1. **Threshold static image to isolate foam finger** **(0..5 Marks)**

Source Code in Editable Plain Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

% Clear workspace, close all figures, and clear command window

clear;

close all;

clc;

% Read the input image

inputImage = imread('C:\Users\edwar\MATLAB\IP\_Sessment\IP\_Test.jpg');

% Convert the image to double precision for numerical operations

inputImageDouble = im2double(inputImage);

% Define the color thresholds for the blue foam finger

blueLowerThreshold = [0, 0, 0.39];

blueUpperThreshold = [0.39, 0.39, 1];

% Create a binary mask to isolate the blue region

binaryMask = (inputImageDouble(:,:,1) >= blueLowerThreshold(1) & ...

inputImageDouble(:,:,1) <= blueUpperThreshold(1)) & ...

(inputImageDouble(:,:,2) >= blueLowerThreshold(2) & ...

inputImageDouble(:,:,2) <= blueUpperThreshold(2)) & ...

(inputImageDouble(:,:,3) >= blueLowerThreshold(3) & ...

inputImageDouble(:,:,3) <= blueUpperThreshold(3));

% Remove noise from the binary mask

cleanedBinaryMask = bwareaopen(binaryMask, 500);

% Convert the cleaned binary mask to uint8 for display

binaryMaskUint8 = uint8(cleanedBinaryMask) \* 255;

% Display the original and cleaned binary images

figure;

subplot(1, 2, 1);

imshow(inputImage);

title('Original Image');

subplot(1, 2, 2);

imshow(binaryMaskUint8);

title('Cleaned Binary Mask');

% Save the cleaned binary mask

imwrite(binaryMaskUint8, 'cleaned\_binary\_mask.png');

% Convert the cleaned binary mask to logical

binaryMaskLogical = logical(binaryMaskUint8);

% Create a 3-channel binary mask for RGB multiplication

binaryMask3D = cat(3, binaryMaskLogical, binaryMaskLogical, binaryMaskLogical);

% Apply the binary mask to the original image to isolate the foam finger

isolatedFoamFinger = inputImage .\* uint8(binaryMask3D);

% Display the isolated foam finger

figure;

imshow(isolatedFoamFinger);

title('Isolated Foam Finger');

% Save the isolated foam finger

imwrite(isolatedFoamFinger, 'isolated\_foam\_finger.png');

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input:** RGB Image Holding Foam Finger | **Output 1:** Thresholded Binary Image |
|  |  |
| **Output 2:** RGB and Binary Image Entry-wise Product |

Any Comments on Effectiveness of Process/Threshold Values Adopted (50 words max):

The process effectively isolates the foam finger by utilising precise blue color thresholds and noise removal. However, the thresholds might not generalise well to varying lighting or similar hues. Adaptive thresholding or additional color spaces (e.g., HSV) could enhance robustness.

1. **Use morphological and/or non-linear filters to clean thresholded image (0..5 Marks)**

Source Code in Editable Plain Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

% Clear workspace, close all figures, and clear command window

clear all;

close all;

clc;

% Step 1: Load the image

imagePath = 'C:\Users\edwar\MATLAB\IP\_Sessment\IP\_Test.jpg';

foamFingerImage = imread(imagePath); % Read the image

% Step 2: Convert to double precision for processing

foamFingerImageDouble = im2double(foamFingerImage); % Convert image to double precision

% Step 3: Define RGB thresholds for blue foam finger

lowerThreshold = [0, 0, 0.39]; % Lower threshold for blue

upperThreshold = [0.39, 0.39, 1]; % Upper threshold for blue

% Step 4: Create a binary mask based on the RGB thresholds

binaryMask = (foamFingerImageDouble(:,:,1) >= lowerThreshold(1) & ...

foamFingerImageDouble(:,:,1) <= upperThreshold(1)) & ...

(foamFingerImageDouble(:,:,2) >= lowerThreshold(2) & ...

foamFingerImageDouble(:,:,2) <= upperThreshold(2)) & ...

(foamFingerImageDouble(:,:,3) >= lowerThreshold(3) & ...

foamFingerImageDouble(:,:,3) <= upperThreshold(3));

% Step 5: Remove small noise using morphological operations

cleanBinaryMask = bwareaopen(binaryMask, 500); % Remove objects smaller than 500 pixels

% Step 6: Fill black dots (holes) inside the foam finger

filledBinaryMask = imfill(cleanBinaryMask, 'holes'); % Fill any holes in the binary mask

% Step 7: Convert the filled binary mask to uint8 format for display

binaryMaskUint8 = uint8(filledBinaryMask) \* 255; % Convert logical mask to uint8 for display

% Step 8: Display the original and filled binary images

figure;

subplot(1, 2, 1);

imshow(foamFingerImage); % Display original image

title('Original Image');

subplot(1, 2, 2);

imshow(binaryMaskUint8); % Display filled binary mask image

title('Cleaned and Filled Binary Image of Foam Finger');

% Step 9: Save the cleaned and filled binary image

imwrite(binaryMaskUint8, 'cleaned\_filled\_binary\_image.png'); % Save cleaned binary mask image

% Step 10: Convert cleaned and filled binary mask to logical

binaryMaskLogical = imbinarize(binaryMaskUint8); % Ensure binary mask is logical

% Step 11: Convert binary mask to 3-channel image for RGB multiplication

binaryMask3ch = cat(3, binaryMaskLogical, binaryMaskLogical, binaryMaskLogical); % Create 3-channel mask

% Step 12: Multiply the original image by the binary mask

resultImage = uint8(double(foamFingerImage) .\* double(binaryMask3ch)); % Element-wise multiplication

% Step 13: Display the isolated foam finger (RGB masked by binary image)

figure;

imshow(resultImage); % Display result image

title('RGB Image with Binary Mask Applied (Foam Finger Isolated)');

% Step 14: Save the result image

imwrite(resultImage, 'isolated\_foam\_finger.png'); % Save isolated foam finger image

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1:** RGB Image Holding Foam Finger from Task C | **Input 2:** Binary Image from Task C |
|  |  |
| **Output 1:** Cleaned Binary Image | **Output 2:** RGB and Cleaned Binary Image Entry-wise Product |

Any Comments on Effectiveness of Process/Methods Adopted (50 words max):

The process effectively isolates the foam finger by utilising precise blue colour thresholds and noise removal. However, the chosen thresholds may not generalise under varying lighting or with similar hues in the background. Incorporating adaptive thresholding techniques or switching to alternative colour spaces such as HSV could significantly improve robustness and accuracy.

1. **Annotate foam finger boundaries on original image. (0..5 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

% Clear workspace, close all figures, and clear command window

clear all;

close all;

clc;

% Step 1: Load the cleaned binary mask

cleanedMask = imread('cleaned\_filled\_binary\_image.png');

% Step 2: Convert the binary mask to logical

cleanedMaskLogical = imbinarize(cleanedMask);

% Step 3: Enhance the binary mask using morphological operations

se = strel('disk', 2); % Structuring element

enhancedMask = imdilate(cleanedMaskLogical, se); % Dilate

enhancedMask = imerode(enhancedMask, se); % Erode

% Step 4: Smooth the binary mask using Gaussian filter

smoothedMask = imgaussfilt(double(enhancedMask), 1);

% Step 5: Detect edges using Canny edge detection

edges = edge(smoothedMask, 'Canny', [0.1 0.3]);

% Step 6: Ensure the outline is connected by dilating the edges

connectedEdges = imdilate(edges, strel('disk', 7)); % Further dilation to connect gaps

% Step 7: Create a new binary image with black background and white traces

tracedImage = zeros(size(connectedEdges));

tracedImage(connectedEdges) = 255;

% Step 8: Save the traced image

imwrite(uint8(tracedImage), 'traced\_binary\_mask.png');

% Step 9: Load the original RGB image

originalImage = imread('C:\Users\edwar\MATLAB\IP\_Sessment\IP\_Test.jpg');

% Step 10: Get the foam finger boundary

foamFingerBoundary = bwperim(cleanedMaskLogical);

% Step 11: Create a copy of the original image for annotation

annotatedImage = originalImage;

% Step 12: Define the color for the boundary

boundaryColor = [255, 0, 0]; % Red

% Step 13: Annotate the boundary on the original image

for c = 1:3

channel = annotatedImage(:,:,c);

channel(foamFingerBoundary) = boundaryColor(c);

annotatedImage(:,:,c) = channel;

end

% Step 14: Thicken the boundary for better visibility

thickBoundary = imdilate(foamFingerBoundary, strel('disk', 7)); % Larger dilation for a continuous boundary

for c = 1:3

channel = annotatedImage(:,:,c);

channel(thickBoundary) = boundaryColor(c);

annotatedImage(:,:,c) = channel;

end

% Step 15: Display the traced binary image

figure;

subplot(1, 2, 1);

imshow(uint8(tracedImage));

title('Traced Binary Mask with Thick Edges');

% Step 16: Display the annotated image

subplot(1, 2, 2);

imshow(annotatedImage);

title('Annotated Image with Thick Foam Finger Boundary');

% Step 17: Save the annotated image

imwrite(annotatedImage, 'annotated\_image.png');

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1:** RGB Image Holding Foam Finger from Task C | **Input 2:** Cleaned Binary Image from Task D |
|  |  |
| **Output 1:** Foam Finger Boundary Binary Image | **Output 2:** RGB Image with Superimposed Boundary |

Any Comments on Effectiveness of Process/Methods Adopted (50 words max):

The process effectively highlights the foam finger boundary by leveraging morphological operations and edge detection techniques. The use of dilation and erosion ensures smooth and continuous boundaries, while Gaussian smoothing reduces noise. However, boundary visibility might depend on the original image contrast. Adjusting the dilation size or boundary colour could further enhance clarity.

1. **Calculate centroid and medoid and annotate image. (0..5 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

% Clear workspace, close all figures, and clear command window

clear all;

close all;

clc;

% Step 1: Load the cleaned binary mask from Task D

final\_cleaned\_binary\_mask = imread('cleaned\_filled\_binary\_image.png'); % Load cleaned binary mask

% Step 2: Convert the cleaned binary mask to logical

binary\_mask\_logical = imbinarize(final\_cleaned\_binary\_mask); % Convert binary mask to logical

% Step 3: Calculate the centroid

stats = regionprops(binary\_mask\_logical, 'Centroid'); % Get region properties

centroid = stats.Centroid; % Get the centroid coordinates [x, y]

% Step 4: Calculate the medoid

[y\_indices, x\_indices] = find(binary\_mask\_logical); % Get the indices of the mask

medoid\_x = median(x\_indices); % Calculate median for x-coordinates

medoid\_y = median(y\_indices); % Calculate median for y-coordinates

medoid = [medoid\_x, medoid\_y]; % Create medoid coordinate [x, y]

% Step 5: Convert binary mask to RGB for visualization

cleaned\_binary\_with\_markers = cat(3, binary\_mask\_logical \* 255, binary\_mask\_logical \* 255, binary\_mask\_logical \* 255); % Convert binary to RGB

% Step 6: Add centroid marker (green circle) with enhanced visibility

marker\_size = 15; % Increased marker size

line\_thickness = 5; % Thicker lines for visibility

cleaned\_binary\_with\_markers = insertShape(cleaned\_binary\_with\_markers, 'Circle', ...

[centroid(1), centroid(2), marker\_size], 'Color', 'green', 'LineWidth', line\_thickness);

% Step 7: Add medoid marker (red cross) with enhanced visibility

line\_length = 15; % Increased length of the cross

cleaned\_binary\_with\_markers = insertShape(cleaned\_binary\_with\_markers, 'Line', ...

[medoid(1) - line\_length, medoid(2), medoid(1) + line\_length, medoid(2)], 'Color', 'red', 'LineWidth', line\_thickness);

cleaned\_binary\_with\_markers = insertShape(cleaned\_binary\_with\_markers, 'Line', ...

[medoid(1), medoid(2) - line\_length, medoid(1), medoid(2) + line\_length], 'Color', 'red', 'LineWidth', line\_thickness);

% Step 8: Display the cleaned binary image with markers

figure;

imshow(cleaned\_binary\_with\_markers);

title('Cleaned Binary Image with Centroid (Green) and Medoid (Red) Markers');

% Step 9: Save the cleaned binary image with markers

imwrite(cleaned\_binary\_with\_markers, 'cleaned\_binary\_with\_centroid\_medoid\_markers.png'); % Save the image

% Step 10: Load the original RGB image

image\_rgb = imread('IP\_Test.jpg'); % Replace 'original\_image.png' with the correct file name

% Step 11: Annotate the centroid and medoid on the original RGB image

% Add centroid marker (green circle)

image\_with\_annotation = insertShape(image\_rgb, 'Circle', ...

[centroid(1), centroid(2), marker\_size], 'Color', 'green', 'LineWidth', line\_thickness);

% Add medoid marker (red cross)

image\_with\_annotation = insertShape(image\_with\_annotation, 'Line', ...

[medoid(1) - line\_length, medoid(2), medoid(1) + line\_length, medoid(2)], 'Color', 'red', 'LineWidth', line\_thickness);

image\_with\_annotation = insertShape(image\_with\_annotation, 'Line', ...

[medoid(1), medoid(2) - line\_length, medoid(1), medoid(2) + line\_length], 'Color', 'red', 'LineWidth', line\_thickness);

% Step 12: Display the annotated original RGB image

figure;

imshow(image\_with\_annotation);

title('Original RGB Image with Centroid (Green) and Medoid (Red) Markers');

% Step 13: Save the annotated original RGB image

imwrite(image\_with\_annotation, 'rgb\_image\_with\_centroid\_medoid\_markers.png'); % Save the image

% Step 14: Output the centroid and medoid coordinates

disp(['Centroid: (', num2str(centroid(1)), ', ', num2str(centroid(2)), ')']);

disp(['Medoid: (', num2str(medoid(1)), ', ', num2str(medoid(2)), ')']);

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1:** RGB Image Holding Foam Finger from Task C | **Input 2:** Cleaned Binary Image from Task D |
|  |  |
| **Output 1:** Cleaned Binary Image with Centroid/Medoid Markers | **Output 2:** RGB Image with Superimposed Centroid/Medoid Markers |

Any Comments on Effectiveness of Process (50 words max):

The process appears effective in calculating and visualising the centroid and medoid. However, the effectiveness depends on the accuracy of the binary mask. If thresholding in Task D is inadequate, it could misidentify the object region. Consider reviewing thresholding values for optimal mask quality to enhance overall accuracy.

1. **Adapt for real-time webcam use** **(0..10 Marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

clear all;

close all;

clc;

% Initialize webcam

camera = webcam;

% Structuring elements for image processing

morph\_element = strel('disk', 10); % Morphological structuring element

edge\_element = strel('disk', 3); % Element for dilating edges

% Marker styles for geometric points

center\_marker = 'bo'; % Blue circle

median\_marker = 'm\*'; % Magenta star

figure;

% Real-time processing loop

while true

% Capture an image from the webcam

live\_image = snapshot(camera);

% Convert image to double precision

image\_scaled = im2double(live\_image);

% Threshold values for object detection

Red\_min = 0.0; Red\_max = 0.5;

Green\_min = 0.0; Green\_max = 0.5;

Blue\_min = 0.5; Blue\_max = 1.0;

% Create a binary mask based on thresholds

binary\_mask = (image\_scaled(:,:,1) >= Red\_min & image\_scaled(:,:,1) <= Red\_max) & ...

(image\_scaled(:,:,2) >= Green\_min & image\_scaled(:,:,2) <= Green\_max) & ...

(image\_scaled(:,:,3) >= Blue\_min & image\_scaled(:,:,3) <= Blue\_max);

% Clean the binary image using morphological operations

processed\_mask = imclose(binary\_mask, morph\_element); % Close small gaps

processed\_mask = imopen(processed\_mask, morph\_element); % Remove noise

processed\_mask = imfill(processed\_mask, 'holes'); % Fill internal holes

% Detect edges and dilate them for visibility

edge\_map = edge(processed\_mask, 'sobel'); % Find edges

expanded\_edges = imdilate(edge\_map, edge\_element); % Expand edge regions

% Overlay boundary on the original image

outlined\_image = live\_image; % Copy the original image

outlined\_image = im2uint8(outlined\_image); % Convert to uint8

for channel\_idx = 1:3

color\_channel = outlined\_image(:,:,channel\_idx);

color\_channel(expanded\_edges) = 255 \* (channel\_idx == 1); % Highlight edges in red

outlined\_image(:,:,channel\_idx) = color\_channel;

end

% Compute centroid using region properties

region\_properties = regionprops(processed\_mask, 'Centroid'); % Find centroid

if ~isempty(region\_properties)

centroid\_point = region\_properties.Centroid;

else

centroid\_point = [NaN, NaN];

end

% Compute medoid using distance transform

distance\_map = bwdist(~processed\_mask); % Distance transformation

[~, max\_index] = max(distance\_map(:));

[medoid\_row, medoid\_col] = ind2sub(size(processed\_mask), max\_index);

medoid\_point = [medoid\_col, medoid\_row];

% Overlay centroid and medoid markers

if ~any(isnan(centroid\_point))

outlined\_image = insertShape(outlined\_image, 'Circle', [centroid\_point, 5], 'Color', 'blue', 'LineWidth', 5); % Centroid

end

outlined\_image = insertShape(outlined\_image, 'Circle', [medoid\_point, 5], 'Color', 'magenta', 'LineWidth', 5); % Medoid

% Display images in a 2x2 grid

subplot(2, 2, 1);

imshow(live\_image);

title('Live Image');

subplot(2, 2, 2);

imshow(binary\_mask);

title('Binary Mask');

subplot(2, 2, 3);

imshow(processed\_mask);

title('Processed Mask');

subplot(2, 2, 4);

imshow(outlined\_image);

title('Highlighted Image (Edges, Centroid, Medoid)');

% Pause for visualization

pause(0.1);

end

% Release the webcam when finished

clear camera;

Input and Output Videos:

Upload video called TaskG showing (1) the original RGB image, (2) the thresholded image, (3) the cleaned thresholded image with annotations (border, centroid and medoid), and finally (4) the border, centroid and medoid labelled image applied to video stream from the webcam. This could be implemented as a stored frame-by-frame video formatted as a 2×2 subplot or montage.

Any Comments on Effectiveness of Process/Design Decisions (50 words max):

The process appears effective in calculating and visualizing the centroid and medoid. However, the effectiveness depends on the accuracy of the binary mask. If thresholding in Task D is inadequate, it could misidentify the object region. Consider reviewing thresholding values for optimal mask quality to enhance overall accuracy.

1. **Real-time gesture recognition (0..40 marks)**

Source Code in Editable Text in a Fixed Width Font (i.e., *not* a picture/screenshot):

clear all;

close all;

clc;

% Initialize webcam

camera\_device = webcam;

% Structuring elements for image processing

close\_element = strel('disk', 10); % Larger structuring element for closing gaps

expand\_element = strel('disk', 3); % Smaller structuring element for edge dilation

% Marker styles for points

center\_point\_marker = 'bo'; % Blue circle

middle\_point\_marker = 'm\*'; % Magenta star

% Variables to track gestures

previous\_center = []; % Previous centroid for motion tracking

gesture\_movement\_threshold = 20; % Minimum movement for gesture detection

gesture\_label = ''; % Text to display the detected gesture

figure;

% Real-time image processing loop

while true

% Capture a frame from the webcam

captured\_frame = snapshot(camera\_device);

scaled\_frame = im2double(captured\_frame); % Convert to double precision

% Threshold values for object detection

red\_lower = 0.0; red\_upper = 0.5;

green\_lower = 0.0; green\_upper = 0.5;

blue\_lower = 0.5; blue\_upper = 1.0;

% Generate a binary mask based on thresholds

binary\_mask = (scaled\_frame(:,:,1) >= red\_lower & scaled\_frame(:,:,1) <= red\_upper) & ...

(scaled\_frame(:,:,2) >= green\_lower & scaled\_frame(:,:,2) <= green\_upper) & ...

(scaled\_frame(:,:,3) >= blue\_lower & scaled\_frame(:,:,3) <= blue\_upper);

% Clean the binary mask using morphological operations

refined\_mask = imclose(binary\_mask, close\_element); % Close gaps

refined\_mask = imopen(refined\_mask, close\_element); % Remove small artifacts

refined\_mask = imfill(refined\_mask, 'holes'); % Fill internal holes

% Detect edges and thicken them

detected\_edges = edge(refined\_mask, 'sobel'); % Edge detection

dilated\_edges = imdilate(detected\_edges, expand\_element); % Expand edges

% Overlay boundaries on the captured frame

overlay\_image = captured\_frame; % Copy the captured frame

overlay\_image = im2uint8(overlay\_image); % Convert to uint8

for channel\_index = 1:3

color\_layer = overlay\_image(:,:,channel\_index);

color\_layer(dilated\_edges) = 255 \* (channel\_index == 1); % Highlight edges in red

overlay\_image(:,:,channel\_index) = color\_layer;

end

% Compute the centroid using region properties

regions = regionprops(refined\_mask, 'Centroid'); % Find centroid

if ~isempty(regions)

center\_point = regions.Centroid;

else

center\_point = [NaN, NaN]; % Handle case with no detected center

end

% Compute the medoid using distance transformation

distance\_field = bwdist(~refined\_mask); % Distance transformation

[~, max\_distance\_idx] = max(distance\_field(:));

[medoid\_row, medoid\_col] = ind2sub(size(refined\_mask), max\_distance\_idx);

middle\_point = [medoid\_col, medoid\_row];

% Annotate the centroid and medoid on the overlay image

if ~any(isnan(center\_point))

overlay\_image = insertShape(overlay\_image, 'Circle', [center\_point, 5], 'Color', 'blue', 'LineWidth', 5); % Mark centroid

end

overlay\_image = insertShape(overlay\_image, 'Circle', [middle\_point, 5], 'Color', 'magenta', 'LineWidth', 5); % Mark medoid

% --- Gesture Detection Logic ---

if ~isempty(previous\_center) && ~any(isnan(center\_point))

% Compute motion vector

x\_shift = center\_point(1) - previous\_center(1); % Change in x

y\_shift = center\_point(2) - previous\_center(2); % Change in y

% Detect significant motion

if abs(x\_shift) > abs(y\_shift) && abs(x\_shift) > gesture\_movement\_threshold

if x\_shift > 0

gesture\_label = 'Right Swipe';

else

gesture\_label = 'Left Swipe';

end

elseif abs(y\_shift) > abs(x\_shift) && abs(y\_shift) > gesture\_movement\_threshold

if y\_shift > 0

gesture\_label = 'Down Swipe';

else

gesture\_label = 'Up Swipe';

end

end

end

% Update the previous centroid

previous\_center = center\_point;

% Annotate the gesture label on the overlay image

overlay\_image = insertText(overlay\_image, [10, 10], gesture\_label, 'FontSize', 20, 'BoxColor', 'yellow', 'TextColor', 'black');

% --- Display the images in a 2x2 layout ---

subplot(2, 2, 1);

imshow(captured\_frame);

title('Captured Frame');

subplot(2, 2, 2);

imshow(binary\_mask);

title('Binary Mask');

subplot(2, 2, 3);

imshow(refined\_mask);

title('Refined Mask');

subplot(2, 2, 4);

imshow(overlay\_image);

title('Overlay Image (Edges, Centroid, Medoid, Gesture)');

% Pause for frame rate control

pause(0.1); % Adjust based on your webcam frame rate

end

% Release the webcam after processing

clear camera\_device;

Input and Output Videos:

Upload video called TaskH showing (1) the original RGB image, (2) the thresholded image, (3) the cleaned thresholded image with annotations (border, centroid and medoid), and finally (4) the border, centroid and medoid labelled image applied to video stream from the webcam, along with a text label reflecting the last gesture recognised. This could be implemented as a stored frame-by-frame video formatted as a 2×2 subplot or montage.

In the video, run through the different basic gestures (up, down, left and right swipe), and for an excellent mark, the extended gestures (numeric digits 0..9).

Any Comments on Effectiveness of Process/Design Decisions (50 words max):

The real-time process effectively tracks gestures by detecting and refining binary masks and computing geometric properties like the centroid and medoid. However, its success heavily relies on the threshold values and lighting conditions. Fine-tuning these parameters and incorporating dynamic adjustments can enhance the robustness and accuracy of gesture detection.

**Appendix A:** PresentationGuidelines

1. To expedite marking, your work must be submitted using this Word template ONLY.
2. You are to submit ONE DOCX file along with .m files containing your code for each task. You will also upload two video files in mp4 format if you attempt Tasks G and Task H. You cannot upload a ZIP or RAR file since this would prevent your report being parsed by Turnitin, and would also mean that it cannot be annotated with feedback by the marker during the marking process.
3. Call your code files **TaskC.m** up to **TaskH.m** so that they can be easily found. It is fine to have additional files containing functions too, if you break your code up in this way.
4. In your source code, variable names should make sense (i.e., avoid single letter variable names, except where these correspond to mathematical convention, like **L**, **M**, **N**, **i** and **j**, or where variable names correspond exactly to the variables used in equations from the course notes). Variable names should *never* be verbs. Where it is appropriate for variables to contain several words, use lower camelCaps (first letter lower case, first letter of each subsequent word in upper case).
5. Code should be well commented (in MATLAB, a comment starts with the % symbol). Where the end keyword is used to terminate a loop or if statement, place % end if, % end while, or % end for, as appropriate, to highlight what the end corresponds to. You can also use the Live Editor in MATLAB to document your code if you prefer.
6. Break large programs into functions, where appropriate, such that each function is informatively named, starting with a verb, and performs one small, well-defined task.
7. Avoid using literals in programs; either use arguments to control settings (e.g., inputs to functions), or collect settings from the user at run-time, or declare named constants (by convention, in UPPER CASE).
8. Code should be properly indented, and ordinarily one line should contain only one command. Before submitting your work, highlight all code and click “smart indent” to do this automatically. Code in your report should be presented in a fixed width font like Courier New to preserve indentation.
9. Ensure code is concise, efficient, and that unnecessary work is not done. Check that your program does not perform calculations or declare variables that are not subsequently used.
10. It is acceptable to use built-in functions provided in MATLAB or its official toolboxes, if these are adequately explained in the corresponding text. If you use MATLAB code from a third-party source, this source should be acknowledged, and the underpinning theory precisely explained in the corresponding Comments boxes.