```
% Specify the filenames of the images
imageFilenames = {'Hand_1.png', 'Hand_2.png', 'Hand_3.png', 'Hand_4.png',
'Hand_5.png'};

% Initialize a matrix to store perpendicular line lengths
fingerWidthsMatrix = zeros(10, 5);

% Loop over each hand image
for imageIdx = 1:5

% Load the hand image
originalImage = imread(imageFilenames{imageIdx});

% Convert the image to grayscale
handImage = rgb2gray(originalImage);

% Use the Roberts edge detection to obtain the edge image
edgeImage = edge(handImage, 'Roberts');

% Create a figure for displaying the images
figure('Name', 'Hand Image and Edge Image', 'NumberTitle', 'off');

% Display the original hand image
subplot(1, 2, 1);
imshow(handImage);
title('Original Hand Image');

% Display the edge image
subplot(1, 2, 2);
imshow(edgeImage);
title('Edge Image');

% Create a figure for setting points, lines, and perpendicular lines
figure('Name', 'Set Points', 'NumberTitle', 'off');
imshow(handImage);
title('Click on each finger joint and knuckles (12 points total)');

% Initialize the list to store points
points = zeros(12, 2);
% Initialize a matrix to store perpendicular line lengths
perpLengthsMatrix = zeros(10, 5);
% Loop to set 12 points
for i = 1:12
    % Wait for the user to click on the image
    [x, y] = ginput(1);

    % Store the clicked point in the 'points' array
    points(i, :) = [x, y];

    % Display a circle on the clicked point
    hold on;
    plot(x, y, 'go', 'MarkerSize', 10);
end

% Close the figure
close;

% Create separate lines for fingers and palm
thumbLine = extendLine(points(1:2, :));
```

```
indexLine = extendLine(points(3:4, :));
middleLine = extendLine(points(5:6, :));
ringLine = extendLine(points(7:8, :));
pinkyLine = extendLine(points(9:10, :));
palmLine = extendLine(points(11:12, :));

% Display the image with lines
figure;
imshow(handImage);
hold on;
plot(points(:, 1), points(:, 2), 'go', 'MarkerSize', 10);
plot(thumbLine(:, 1), thumbLine(:, 2), 'r-', 'LineWidth', 2);
plot(indexLine(:, 1), indexLine(:, 2), 'r-', 'LineWidth', 2);
plot(middleLine(:, 1), middleLine(:, 2), 'r-', 'LineWidth', 2);
plot(ringLine(:, 1), ringLine(:, 2), 'r-', 'LineWidth', 2);
plot(pinkyLine(:, 1), pinkyLine(:, 2), 'r-', 'LineWidth', 2);
plot(palmLine(:, 1), palmLine(:, 2), 'b-', 'LineWidth', 2);

% Draw perpendicular lines at each point (1-10) to the edge of the hand
for i = 1:10
    point = points(i, :);

    % Calculate direction of the line between consecutive points
    if rem(i,2)~=0
       lineDir = points(i + 1, :) - points(i, :);
    else
        lineDir = points(i , :) - points(i-1, :);
    end

    % Calculate perpendicular direction
    perpDir = [lineDir(2), -lineDir(1)];  % Rotate the direction vector 90 degrees

    % Extend perpendicular line to the edge of the hand
    perpLine = extendPerpendicularLine(point, perpDir, edgeImage);

    % Plot perpendicular line
    plot(perpLine(:, 1), perpLine(:, 2), 'k--', 'LineWidth', 1);

    % Calculate and print the length of the perpendicular line
    perpLength = norm(perpLine(2, :) - perpLine(1, :));
        fingerWidthsMatrix(i, imageIdx) = perpLength;


    % Determine the finger information
    if i == 1 || i == 2
        fingerInfo = 'Thumb';
    elseif i == 3 || i == 4
        fingerInfo = 'Index Finger';
    elseif i == 5 || i == 6
        fingerInfo = 'Middle Finger';
    elseif i == 7 || i == 8
        fingerInfo = 'Ring Finger';
     elseif i == 9 || i == 10
        fingerInfo = 'Pinky Finger';

    end

    % Display the length and finger information
    fprintf('Perpendicular Line %d (from %s): Length = %.2f pixels\n', i,
```

```
fingerInfo, perpLength);
end

% Name the original lines
text(thumbLine(1, 1), thumbLine(1, 2), 'f2', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');
text(indexLine(1, 1), indexLine(1, 2), 'f3', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');
text(middleLine(1, 1), middleLine(1, 2), 'f4', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');
text(ringLine(1, 1), ringLine(1, 2), 'f5', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');
text(pinkyLine(1, 1), pinkyLine(1, 2), 'f6', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');
text(palmLine(1, 1), palmLine(1, 2), 'f1', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');

title('Manually set points, lines, and perpendicular lines on the hand image');

% Print the manually set points
disp('Manually set points:');
disp(points);

end

% Calculate Euclidean distance difference matrix for individual finger widths
euclideanDiffMatrix = zeros(5, 5);

for i = 1:5
    for j = i+1:5
        % Calculate the Euclidean distance difference between individual finger
widths
        diffVector = fingerWidthsMatrix(:, i) - fingerWidthsMatrix(:, j);
        euclideanDiffMatrix(i, j) = norm(diffVector);
    end
end

% Display the results
disp('Euclidean Distance Difference Matrix (Individual Finger Widths):');
disp(euclideanDiffMatrix);
function linePoints = extendLine(points)
    % Calculate the direction vector of the line
    direction = points(2, :) - points(1, :);

    % Extend the line towards both directions by 20 pixels (you can adjust this
value)
    extendedPoints = [points(1, :) - 1 * direction; points(2, :) + 1 * direction];

    % Combine original and extended points to form the line
    linePoints = extendedPoints;
end


function perpLine = extendPerpendicularLine(point, perpDir, edgeImage)
    % Extend perpendicular line to the edge of the hand

    % Normalize the perpendicular direction vector
    perpDir = perpDir / norm(perpDir);
```

```
    % Set the maximum extension distance (adjust as needed)
    maxExtension = 50;

    % Initialize the perpendicular line
    perpLine = zeros(2, 2);

    % Extend the line in one direction
    for i = 1:maxExtension
        extendedPoint = round(point + i * perpDir);

        % Check if the extended point is outside the image boundaries
        if any(extendedPoint < 1) || extendedPoint(1) > size(edgeImage, 2) ||
extendedPoint(2) > size(edgeImage, 1)
            break;
        end

        % Check the edge pixel value
        if edgeImage(extendedPoint(2), extendedPoint(1)) > 0
            perpLine(1, :) = extendedPoint;
            break;
        end
    end

    % Extend the line in the opposite direction
    for i = 1:maxExtension
        extendedPoint = round(point - i * perpDir);

        % Check if the extended point is outside the image boundaries
        if any(extendedPoint < 1) || extendedPoint(1) > size(edgeImage, 2) ||
extendedPoint(2) > size(edgeImage, 1)
            break;
        end

        % Check the edge pixel value
        if edgeImage(extendedPoint(2), extendedPoint(1)) > 0
            perpLine(2, :) = extendedPoint;
            break;
        end
    end
end
```