**Leaf Morphometric Analysis**

1. **Threshold to isolate objects of interest**

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

Leaf\_coin = imread('Leaf\_coin.jpg'); % correct

figure('Name','Leaf Coin'); % new window

imshow(Leaf\_coin);

title("Leaf Coin");

[coin\_sec, coin\_bin\_sec] = colorThreshold(Leaf\_coin, 65, 188, 17, 67, 0, 61);

% Segment leaf

[leaf\_sec, leaf\_bin\_sec] = colorThreshold(Leaf\_coin, 0, 52, 0, 170, 0, 105);

%% Create union of original masks

leaf\_coin\_union = coin\_bin\_sec | leaf\_bin\_sec; % logical OR to combine objects

leaf\_coin\_union\_u8 = repmat(uint8(leaf\_coin\_union), [1 1 3]);

Leaf\_coin\_section = Leaf\_coin .\* leaf\_coin\_union\_u8; % entry-wise multiplication

function [selected\_section, mask\_section\_uint8] = colorThreshold(input\_image, Rmin, Rmax, Gmin, Gmax, Bmin, Bmax)

% Extract individual RGB channels

R = input\_image(:, :, 1);

G = input\_image(:, :, 2);

B = input\_image(:, :, 3);

% Applying Band Pass

mask\_section = (R >= Rmin & R <= Rmax) & (G >= Gmin & G <= Gmax) & (B >= Bmin & B <= Bmax);

% Apply mask to original image (set background to 0)

selected\_section = input\_image;

selected\_section(repmat(~mask\_section, [1 1 3])) = 0;

% Convert mask to uint8 for saving

mask\_section\_uint8 = uint8(mask\_section) \* 255;

end

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input:** Input Image (Original Photograph) | **Output 1:** Binary Mask for Leaf |
|  |  |
| **Output 2:** Binary Mask for Coin | **Output 3:** Entry-wise Product (Union) |

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

The threshold values (Rmin, Rmax, Gmin, Gmax, Bmax, Bmin) adopted are derived from the “Color Threshold App”. Additionally the code comprises of a “color Threshold” function, which can be (and is) re-used further in the assignment.

1. **Use morphological and/or non-linear filters to clean binary masks**

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

% Apply morphological closing to the union of original masks

leaf\_bin\_sec\_closed = holeCorrections(leaf\_bin\_sec, 'disk', 13);

coin\_bin\_sec\_closed = holeCorrections(coin\_bin\_sec, 'disk', 13);

leaf\_coin\_union\_closed = holeCorrections(leaf\_coin\_union,'disk', 13);

% Manual union after closing individual masks

leaf\_coin\_union\_closed\_u8 = repmat(uint8(leaf\_coin\_union\_closed), [1 1 3]);

Leaf\_coin\_section\_closed = Leaf\_coin .\* leaf\_coin\_union\_closed\_u8;

% Function: Morphological Hole Correction

function[closed\_output] = holeCorrections(input\_image, strl\_type, strel\_size)

diskStrel = strel(strl\_type, strel\_size);

closed\_output = imclose(input\_image, diskStrel);

end

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1:** Original Binary Mark from Task C for Leaf | **Input 2:** Original Binary Mask from Task C for Coin |
|  |  |
| **Output 1:** Cleaned Binary Mask for Leaf | **Output 2:** Cleaned Binary Mask for Coin |

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

The masks derived via Task C were “corrected” via filling holes and false negatives through a function “holeCorrections” via using Closing operations along with strel type and size set via several manual trail and error.

1. **Normalised RGB histograms of segmented objects. (0..5 Marks)**

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

[R\_Lc, G\_Lc, B\_Lc] = extractColors(Leaf\_coin);

Lc\_leaf = logical(leaf\_bin\_sec\_closed);

Lc\_coin = logical(coin\_bin\_sec\_closed);

% Extract only selected pixels for each object

R\_L = R\_Lc(Lc\_leaf);

G\_L = G\_Lc(Lc\_leaf);

B\_L = B\_Lc(Lc\_leaf);

R\_C = R\_Lc(Lc\_coin);

G\_C = G\_Lc(Lc\_coin);

B\_C = B\_Lc(Lc\_coin);

figure('Name','Histogram Plot')

ax1 = subplot(2,1,1);

plotHistogram(R\_C, G\_C, B\_C, ax1,'Histogram for coin in Leaf\_coin')

ax2 = subplot(2,1,2);

plotHistogram(R\_L, G\_L, B\_L, ax2,'Histogram for leaf in Leaf\_coin')

function [R, G, B] = extractColors(input\_image)

R = double(input\_image(:,:,1));

G = double(input\_image(:,:,2));

B = double(input\_image(:,:,3));

end

function plotHistogram(R, G, B, ax, plottitle)

edges = 0:1:255;

axes(ax);

hold (ax,'on');

plot(ax,edges(1:end-1), histcounts(R, edges, 'Normalization', 'probability'), 'r', 'LineWidth', 1.5);

plot(ax,edges(1:end-1), histcounts(G, edges, 'Normalization', 'probability'), 'g', 'LineWidth', 1.5);

plot(ax,edges(1:end-1), histcounts(B, edges, 'Normalization', 'probability'), 'b', 'LineWidth', 1.5);

title(ax,plottitle);

xlabel(ax,'Pixel Intensity (0–255)');

ylabel(ax,'Normalized Frequency');

legend(ax,{'Red','Green','Blue'});

xlim(ax,[0 255]);

grid (ax,'on');

hold (ax,'off');

end

Input and Output Images:

|  |  |
| --- | --- |
|  |  |
| **Input 1:** Entry-wise Product (Leaf Only) | **Input 2:** Entry-wise Product (Coin Only) |
|  |  |
| **Output 1:** Normalised RGB Histogram for Leaf | **Output 2:** Normalised RGB Histogram for Coin |

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

The histograms are created via using functions “extract colors” and “plotHistogram” which are made for extracting the RGB channels of an image and plotting the histogram, and furthermore, m=for better reusability.

1. **Annotate object boundaries.**

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

leaf\_outline = outline\_section(leaf\_bin\_sec\_closed, 'disk', 13 );

coin\_outline = outline\_section(coin\_bin\_sec\_closed, 'disk', 13);

union\_outline = leaf\_outline | coin\_outline;

Lc\_annotated = annotate(union\_outline, Leaf\_coin);

function [outlined\_img] = outline\_section(input\_image, strl\_type, strl\_size)

disStrel = strel(strl\_type, strl\_size);

outlined\_img = input\_image - imerode(input\_image, disStrel);

end

function [img\_annotated] = annotate(union\_outline, original\_image)

img\_annotated = original\_image; % Copy the original image

boundary\_color = uint8([58, 19, 125]);

for c = 1:3

channel = img\_annotated(:,:,c);

channel(union\_outline) = boundary\_color(c);

img\_annotated(:,:,c) = channel;

end

end

Output Images:

|  |  |
| --- | --- |
|  |  |
| **Output 1:** Boundary around Cleaned Leaf Binary Mask | **Output 2:** Boundary around Cleaned Coin Binary Mask |
|  |  |
| **Output 3:** Input Image with Overlaid Object Boundaries |

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

Boundary masks and final annotated image are created via re-usable functions “outline section” and “annotate”. Outline Section function utilises “Internal Gradient” method.

1. **Calculate object centroid, medoid and GLI**

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

%Number of componenets to look at per mask

N = 1

N = 1

[leaf\_centroids, leaf\_medoids] = computeLeafCentroidMedoid(leaf\_bin\_sec\_closed,N);

[coin\_centroids, coin\_medoids] = computeLeafCentroidMedoid(coin\_bin\_sec\_closed,N);

leaf\_centroid = leaf\_centroids(1,:);

leaf\_medoid = leaf\_medoids(1,:);

coin\_centroid = coin\_centroids(1,:);

coin\_medoid = coin\_medoids(1,:);

GLI\_leaf\_pixels = (2\*G\_L - R\_L - B\_L) ./ (2\*G\_L + R\_L + B\_L);

function medoid\_coord = computeMedoid(binaryMask, sampleSize)

if nargin < 2

sampleSize = 300;

end

[y, x] = find(binaryMask);

coords = [x, y];

N = size(coords,1);

if N == 0

medoid\_coord = [NaN NaN];

return;

end

% Downsample pixels (random)

idx = randperm(N, min(sampleSize, N));

sample = coords(idx, :);

% Compute squared distance matrix

D = pdist2(sample, sample, 'euclidean').^2;

% Sum distances for each candidate

[~, medoidIdx] = min(sum(D, 2));

medoid\_coord = sample(medoidIdx, :);

end

function [centroids, medoids] = computeLeafCentroidMedoid(mask, N)

mask = logical(mask);

if nnz(mask)==0

centroids = [];

medoids = [];

return;

end

% Label components

[L, numObj] = bwlabel(mask);

stats = regionprops(L, 'Area', 'Centroid');

% Get top N by area

areas = [stats.Area];

[~, idx] = sort(areas, 'descend');

N = min(N, numObj);

idx = idx(1:N);

centroids = zeros(N,2);

medoids = zeros(N,2);

for i = 1:N

compID = idx(i);

compMask = (L == compID);

centroids(i,:) = stats(compID).Centroid;

medoids(i,:) = computeMedoidFast(compMask);

end

end

Output Images:

|  |
| --- |
|  |
| **Output 1:** Input Image with Overlaid Boundaries, Centroid and Medoid Markers, and GLI Value for Leaf |

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

Medoid and Centroid is (again) produced via re-usable functions, computeMedoid and computeCentroidMedoid, which has a numerical parameter, to determine the top N submasks (by area) in a single mask (Useful when calculating for images with multiple leaves (in Task I) )

1. **Morphometric analysis**

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

N = 1; % Number of largest leaves to analyze

COIN\_DIAMETER\_MM = 25.9; % Verify this value for your coin!

% Extract the top N largest leaves

leafMasks = getTopNComponents(leaf\_bin\_sec\_closed, N);

% Error checking

if isempty(leafMasks)

error('No leaves found in the closed binary mask! Check your segmentation.');

end

fprintf('Found %d leaf component(s)\n', length(leafMasks));

Found 1 leaf component(s)

% Initialize results

results = cell(length(leafMasks), 1);

% Measure each leaf

for i = 1:length(leafMasks)

fprintf('\n----- Measuring Leaf %d of %d -----\n', i, length(leafMasks));

% Extract current leaf mask and ensure it's logical

currentLeafMask = logical(leafMasks{i});

currentCoinMask = logical(coin\_bin\_sec\_closed);

% Validate mask is not empty

if sum(currentLeafMask(:)) == 0

warning('Leaf %d mask is empty. Skipping.', i);

continue;

end

% Measure leaf

results{i} = measureLeafUsingCoin(currentLeafMask, currentCoinMask, ...

Leaf\_coin, COIN\_DIAMETER\_MM);

end

function results = measureLeafUsingCoin(leafMask, coinMask, origImg, coinDiameterMM)

% ---------------- Input Validation ----------------

if ~islogical(leafMask) || ~islogical(coinMask)

error('leafMask and coinMask must be logical (binary) images');

end

if coinDiameterMM <= 0

error('coinDiameterMM must be positive');

end

if ~isequal(size(leafMask), size(coinMask))

error('leafMask and coinMask must have the same dimensions');

end

% ---------------- Extract Coin Diameter ----------------

coinStats = regionprops(coinMask, 'MajorAxisLength', 'Area');

if isempty(coinStats)

error('No coin detected in coinMask.');

elseif length(coinStats) > 1

[~, idx] = max([coinStats.Area]);

coin\_diam\_pix = coinStats(idx).MajorAxisLength;

else

coin\_diam\_pix = coinStats.MajorAxisLength;

end

% Scale factor mm/pixel

scale = coinDiameterMM / coin\_diam\_pix;

% ---------------- Extract Leaf Stats ----------------

leafStats = regionprops(leafMask, 'MajorAxisLength', 'MinorAxisLength', ...

'Area', 'Perimeter', 'Centroid', 'Orientation', 'BoundingBox');

if isempty(leafStats)

error('No leaf detected in leafMask.');

elseif length(leafStats) > 1

% use largest leaf

[~, idx] = max([leafStats.Area]);

leafStats = leafStats(idx);

end

% ---------------- Measurements in mm ----------------

leaf\_length\_mm = leafStats.MajorAxisLength \* scale;

leaf\_width\_mm = leafStats.MinorAxisLength \* scale;

leaf\_area\_mm2 = leafStats.Area \* (scale^2);

% ---------------- TRUE PERIMETER (Contour-Based) ----------------

boundaries = bwboundaries(leafMask);

boundary = boundaries{1};

perim\_px = sum( sqrt( sum( diff(boundary).^2, 2 ) ) );

leaf\_perim\_mm = perim\_px \* scale;

% ---------------- Major Axis Endpoints ----------------

c = leafStats.Centroid;

theta = deg2rad(leafStats.Orientation);

halfLen = leafStats.MajorAxisLength / 2;

x1 = c(1) + halfLen \* cos(theta);

y1 = c(2) - halfLen \* sin(theta);

x2 = c(1) - halfLen \* cos(theta);

y2 = c(2) + halfLen \* sin(theta);

% ---------------- Visualization ----------------

figHandle = figure('Name','Leaf Real-World Measurements', 'NumberTitle','off');

clf;

imshow(origImg);

hold on;

% Draw leaf contour

plot(boundary(:,2), boundary(:,1), 'c-', 'LineWidth', 2);

% Draw major axis line

plot([x1 x2], [y1 y2], 'r-', 'LineWidth', 3);

% Draw centroid

plot(c(1), c(2), 'yx', 'MarkerSize', 14, 'LineWidth', 3);

% Dynamic text placement

[H, W, ~] = size(origImg);

textX = max(20, W\*0.02);

textY = max(40, H\*0.05);

ax = gca;

set(ax, 'Units','normalized', 'Position',[0 0 1 1]);

txt = sprintf(['Leaf Measurements:\n','Length: %.2f mm\n', 'Width : %.2f mm\n', 'Area : %.2f mm^2\n', 'Perim : %.2f mm\n', 'Scale : %.4f mm/pixel'],leaf\_length\_mm, leaf\_width\_mm, ...

leaf\_area\_mm2, leaf\_perim\_mm, scale);

text(0.02, 0.98, txt, ...

'Units','normalized', ...

'HorizontalAlignment','left', ...

'VerticalAlignment','top', ... % anchor text at top-left corner

'Color','yellow', ...

'BackgroundColor','black', ...

'FontSize',12, ...

'FontWeight','bold', ...

'Margin',5, ...

'EdgeColor','white');

legend({'Leaf Boundary','Major Axis','Centroid'}, 'Location','southeast', ...

'TextColor','white','Color','black');

title('Leaf Measurements with True Contour Perimeter', ...

'Color','white','FontSize',14,'FontWeight','bold');

hold off;

% ---------------- Return Struct ----------------

results.length\_mm = leaf\_length\_mm;

results.width\_mm = leaf\_width\_mm;

results.area\_mm2 = leaf\_area\_mm2;

results.perimeter\_mm = leaf\_perim\_mm;

results.scale\_mm\_per\_pix = scale;

results.centroid = c;

results.orientation\_deg = leafStats.Orientation;

results.aspect\_ratio = leaf\_length\_mm / leaf\_width\_mm;

results.coin\_diameter\_px = coin\_diam\_pix;

Output Images:

|  |
| --- |
|  |
| **Output 1:** Input Image with Overlaid Major/Minor Axes (mm), Area (mm2), and Perimeter (mm). |

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

The process is (as shown in previous tasks) is taking top N components in a given mask (Here 1, but useful while doing task I. Overall the task is coupled with function “measureLeafUsingCoin” to use the given coin (a 2p coin) with a set diameter, using it to calculate pixels per mm, to measure axi of the leaf.

1. **Multi-leaf analysis (0..40 marks)**

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

% Task I – Leaf Analysis Pipeline (Improved Version)

clear; clc; close all;

% --- CONSTANTS & SETTINGS ---

IMAGE\_FOLDER = '';

OUTPUT\_FOLDER = fullfile(IMAGE\_FOLDER, 'TaskI\_Results');

DOWNSAMPLE\_STEP = 5; % Step for medoid computation

MIN\_AREA\_PX = 200; % Minimum area for coin detection

COIN\_DIAMETER\_MM = 25.9; % Coin diameter in mm

if ~exist(OUTPUT\_FOLDER, 'dir')

mkdir(OUTPUT\_FOLDER);

end

% --- Load images ---

imageFiles = dir(fullfile(IMAGE\_FOLDER, '\*.jpg'));

if isempty(imageFiles)

error('No .jpg images found in folder: %s', IMAGE\_FOLDER);

end

fprintf('Found %d images to process.\n', length(imageFiles));

% --- Initialize container for CSV output ---

globalRows = {};

% Loop through images

for imgIdx = 1:length(imageFiles)

imgName = imageFiles(imgIdx).name;

[~, baseName, ~] = fileparts(imgName);

imgPath = fullfile(IMAGE\_FOLDER, imgName);

fprintf('\n========================\nProcessing (%d/%d): %s\n========================\n', ...

imgIdx, length(imageFiles), imgName);

% --- Read image ---

currentImage = im2uint8(imread(imgPath));

% 1) Segment Yellow & Green leaves

[yellowSection, yellowMask] = getLeafMask(currentImage, 'Y');

[greenSection, greenMask] = getLeafMask(currentImage, 'G');

yellowMask = logical(yellowMask);

greenMask = logical(greenMask);

unionMask = yellowMask | greenMask;

% 2) Morphological cleaning

yellowMaskClosed = holeCorrections(yellowMask, 'disk', 13);

greenMaskClosed = holeCorrections(greenMask, 'disk', 13);

unionMaskClosed = holeCorrections(unionMask, 'disk', 13);

% Save cleaned masks

imwrite(uint8(yellowMaskClosed)\*255, fullfile(OUTPUT\_FOLDER, [baseName '\_Y\_closed.png']));

imwrite(uint8(greenMaskClosed)\*255, fullfile(OUTPUT\_FOLDER, [baseName '\_G\_closed.png']));

imwrite(uint8(unionMaskClosed)\*255, fullfile(OUTPUT\_FOLDER, [baseName '\_union\_closed.png']));

% 3) Compute histograms

[Rchan, Gchan, Bchan] = extractColors(currentImage);

R\_Y = Rchan(yellowMaskClosed); G\_Y = Gchan(yellowMaskClosed); B\_Y = Bchan(yellowMaskClosed);

R\_G = Rchan(greenMaskClosed); G\_G = Gchan(greenMaskClosed); B\_G = Bchan(greenMaskClosed);

histFig = figure('Visible','off');

subplot(2,1,1);

plotHistogram(R\_G, G\_G, B\_G, gca, [baseName ' - GREEN histogram']);

subplot(2,1,2);

plotHistogram(R\_Y, G\_Y, B\_Y, gca, [baseName ' - YELLOW histogram']);

saveas(histFig, fullfile(OUTPUT\_FOLDER, [baseName '\_histograms.png']));

close(histFig);

% 4) Detect coin automatically

[LabeledImg, numComponents] = bwlabel(unionMaskClosed);

if numComponents == 0

coinMask = false(size(unionMaskClosed));

else

stats = regionprops(LabeledImg, 'Area', 'Perimeter');

areas = [stats.Area];

perimeters = [stats.Perimeter]; perimeters(perimeters==0) = eps;

circularity = (4\*pi\*areas)./(perimeters.^2);

circNorm = rescale(circularity);

areaNorm = rescale(1./areas);

score = 0.7\*circNorm + 0.3\*areaNorm;

score(areas < MIN\_AREA\_PX) = -Inf;

[~, coinIdx] = max(score);

coinMask = (LabeledImg == coinIdx);

end

imwrite(uint8(coinMask)\*255, fullfile(OUTPUT\_FOLDER, [baseName '\_coin\_mask.png']));

fprintf('Detected coin component\n');

% 5) Determine top-N leaves

topN = computeNFromFileName(baseName);

fprintf('Top-N components = %d\n', topN);

% 6) Extract top-N leaf components

leafMasks = getTopNComponents(unionMaskClosed, topN);

if isempty(leafMasks)

warning('No top components returned for %s — skipping', baseName);

continue;

end

fprintf('Found %d top components\n', length(leafMasks));

% 7) Compute centroids & medoids

[leafCentroids, leafMedoids] = computeLeafCentroidMedoid(unionMaskClosed, topN);

% 8) Measure & save per-leaf data

perImageRows = {};

for leafIdx = 1:length(leafMasks)

leafMaskCurrent = logical(leafMasks{leafIdx});

if nnz(leafMaskCurrent) == 0

continue;

end

% Centroid & medoid

if leafIdx <= size(leafCentroids,1)

centroid = leafCentroids(leafIdx,:);

else

centroid = regionprops(leafMaskCurrent,'Centroid').Centroid;

end

if leafIdx <= size(leafMedoids,1)

medoid = leafMedoids(leafIdx,:);

else

medoid = computeMedoid(leafMaskCurrent, DOWNSAMPLE\_STEP);

end

% GLI computation

GLI\_pixels = (2\*Gchan(leafMaskCurrent)-Rchan(leafMaskCurrent)-Bchan(leafMaskCurrent)) ./ ...

(2\*Gchan(leafMaskCurrent)+Rchan(leafMaskCurrent)+Bchan(leafMaskCurrent));

GLI\_mean = mean(GLI\_pixels,'omitnan');

% Leaf damage

damagePct = computeLeafDamage(leafMaskCurrent);

% Check if coin

isCoin = nnz(leafMaskCurrent & coinMask)/max(1, nnz(coinMask)) > 0.3;

% Leaf measurements using coin reference

if ~isCoin

try

meas = measureLeafUsingCoin(leafMaskCurrent, coinMask, currentImage, COIN\_DIAMETER\_MM);

leafLength\_mm = meas.length\_mm;

leafWidth\_mm = meas.width\_mm;

leafArea\_mm2 = meas.area\_mm2;

leafPerim\_mm = meas.perimeter\_mm;

leafOrientation = meas.orientation\_deg;

catch

leafLength\_mm=NaN; leafWidth\_mm=NaN; leafArea\_mm2=NaN;

leafPerim\_mm=NaN; leafOrientation=NaN;

end

else

leafLength\_mm=NaN; leafWidth\_mm=NaN; leafArea\_mm2=NaN;

leafPerim\_mm=NaN; leafOrientation=NaN;

end

% Crop & save leaf

try

cropLeafAndSave(leafMaskCurrent, currentImage, fullfile(OUTPUT\_FOLDER, sprintf('%s\_comp%02d', baseName, leafIdx)));

catch

warning('Cropping failed for component %d', leafIdx);

end

% Store data for CSV

perImageRows{leafIdx,1} = baseName;

perImageRows{leafIdx,2} = leafIdx;

perImageRows{leafIdx,3} = isCoin;

perImageRows{leafIdx,4} = centroid(1);

perImageRows{leafIdx,5} = centroid(2);

perImageRows{leafIdx,6} = medoid(1);

perImageRows{leafIdx,7} = medoid(2);

perImageRows{leafIdx,8} = GLI\_mean;

perImageRows{leafIdx,9} = leafLength\_mm;

perImageRows{leafIdx,10} = leafWidth\_mm;

perImageRows{leafIdx,11} = leafArea\_mm2;

perImageRows{leafIdx,12} = leafPerim\_mm;

perImageRows{leafIdx,13} = leafOrientation;

perImageRows{leafIdx,14} = damagePct;

globalRows = [globalRows; perImageRows(leafIdx,:)]; % append to global

end

% 9) Annotated overlay figure with per-leaf info + overall GLI

annFig = figure('Visible','off');

imshow(currentImage);

hold on;

% Draw leaf outlines

unionOutline = outline\_section(unionMaskClosed, 'disk', 5);

[boundaries, ~] = bwboundaries(unionOutline,'noholes');

for k = 1:length(boundaries)

plot(boundaries{k}(:,2), boundaries{k}(:,1), 'c', 'LineWidth', 0.7);

end

% Overlay centroids, medoids, damage %, GLI

for k = 1:length(leafMasks)

c = cell2mat(perImageRows(k,4:5)); % Centroid

m = cell2mat(perImageRows(k,6:7)); % Medoid

plot(c(1), c(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);

plot(m(1), m(2), 'bo', 'MarkerSize', 9, 'LineWidth', 2);

dmg = perImageRows{k,14};

GLI\_val = perImageRows{k,8};

text(m(1)+6, m(2), sprintf('D: %.1f%%\nGLI: %.2f', dmg, GLI\_val), ...

'Color','white','BackgroundColor','black','FontSize',9,'Margin',2,'FontWeight','bold');

end

% Overall GLI

overallGLI = mean((2\*Gchan(unionMaskClosed)-Rchan(unionMaskClosed)-Bchan(unionMaskClosed)) ./ ...

(2\*Gchan(unionMaskClosed)+Rchan(unionMaskClosed)+Bchan(unionMaskClosed)), 'omitnan');

text(size(currentImage,2)-10, 20, sprintf('GLI (overall) = %.3f', overallGLI), ...

'Color',[0 0.4 0],'FontSize',14,'FontWeight','bold','HorizontalAlignment','right');

hold off;

exportgraphics(annFig, fullfile(OUTPUT\_FOLDER, [baseName '\_annotated\_all.png']), 'Resolution',150);

close(annFig);

end

% 10) Save final CSV

if ~isempty(globalRows)

headers = {'Image','Component','IsCoin','CentroidX','CentroidY','MedoidX','MedoidY', ...

'GLI\_mean','Length\_mm','Width\_mm','Area\_mm2','Perimeter\_mm','Orientation\_deg','DamagePercent'};

Tglobal = cell2table(globalRows,'VariableNames',headers);

writetable(Tglobal, fullfile(OUTPUT\_FOLDER,'leaf\_summary.csv'));

fprintf('Saved leaf summary CSV to %s\n', fullfile(OUTPUT\_FOLDER,'leaf\_summary.csv'));

end

fprintf('\n=== BATCH COMPLETE — Results in %s ===\n', OUTPUT\_FOLDER);

function cropLeafAndSave(mask, imgRGB, prefix)

% mask: logical mask same size as imgRGB

% imgRGB: uint8 RGB image

CC = bwconncomp(mask);

S = regionprops(CC, 'BoundingBox');

if isempty(S)

return;

end

% Bounding box of component

bb = S(1).BoundingBox;

x = max(1, floor(bb(1)));

y = max(1, floor(bb(2)));

w = round(bb(3));

h = round(bb(4));

x2 = min(size(imgRGB,2), x + w - 1);

y2 = min(size(imgRGB,1), y + h - 1);

% Just crop original image

cropped = imgRGB(y:y2, x:x2, :);

% Save only ONE output: raw crop

imwrite(cropped, [prefix '\_crop.png']);

end

function damagePercent = computeLeafDamage(leafMask)

% Compute damage as the fraction of internal hole area inside a leaf component.

% leafMask must be logical.

leafMask = logical(leafMask);

if ~any(leafMask(:))

damagePercent = NaN;

return;

end

% Fill holes to get the "ideal" leaf

filled = imfill(leafMask, 'holes');

% Holes are pixels in filled but not in original

holeMask = filled & ~leafMask;

holeArea = bwarea(holeMask); % uses subpixel area if needed

leafArea = bwarea(filled);

if leafArea == 0

damagePercent = NaN;

else

damagePercent = (holeArea / leafArea) \* 100;

end

end

Input and Output :

Task I takes a batch of multiple images, from a specified folder “imageFolder” as mentioned in the code. It reuses functions like colorThresholding, outLine, measureLeafFromCoin etc. as shown from the previous tasks.

Due to the colors of leaves selected, each images are divided into two types of masks: Yclosed, Gclosed, for yellow and green color components respectively. The coin has been separated via using the “Circularity” value of a mask.

The hole corrections on each mask has not been done “excessively” so as to retain “damage-area” pixels for damage area % info.

The topNcomponents, meant to get the biggest compnents per image are calculated via the filename,   
the number of digits in the filename are is used to get “N”

The functions used here for the first time are “damagePercent” and cropLeafandSave detecting damage percentage perleaf, and cropping each leaf section as accurately as possible from a given image.

Most of the outputs are to be written in a file in the folder of the leaves, names “filename\_results”.

Below Given are outputs for a select few images:

A Csv file is also outputted to record metrics like, damage percent, area etc. per leaf segment

Original Images:

**A group of leaves on a white surface

AI-generated content may be incorrect.**

**Masks (Closed) :**

**Yellow:**

A black and white image of a couple of animals

AI-generated content may be incorrect.A white shapes on a black background

AI-generated content may be incorrect.A white and black image of a plant

AI-generated content may be incorrect.A group of white shapes

AI-generated content may be incorrect.

A white flower and a moon

AI-generated content may be incorrect.

**Green:**

A white face with a black background

AI-generated content may be incorrect.A white silhouette of a person

AI-generated content may be incorrect.A white silhouette of a person

AI-generated content may be incorrect.A white flower on a black background

AI-generated content may be incorrect.

A white stars on a black background

AI-generated content may be incorrect.

**Union:**

A white silhouette of leaves and a flower

AI-generated content may be incorrect.A white silhouettes of animals

AI-generated content may be incorrect.A white silhouettes of leaves

AI-generated content may be incorrect.

A white silhouettes of leaves

AI-generated content may be incorrect.A white silhouettes of plants and a circle

AI-generated content may be incorrect.

**Histograms:**

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

**Damage Areas, Centroid, medoid and GLI :**

A close up of leaves

AI-generated content may be incorrect.A close-up of several leaves

AI-generated content may be incorrect.A group of leaves with different colors

AI-generated content may be incorrect.

A close-up of different types of leaves

AI-generated content may be incorrect.A close-up of different types of leaves

AI-generated content may be incorrect.

**Sample Major Axis examples from above images:**

**A close up of leaves

AI-generated content may be incorrect.**A close up of leaves

AI-generated content may be incorrect.A close-up of several leaves

AI-generated content may be incorrect.

**A close up of leaves

AI-generated content may be incorrect. A close up of leaves

AI-generated content may be incorrect.**

**Cropped image examples from above images:**

A close-up of a leaf

AI-generated content may be incorrect.A close up of a leaf

AI-generated content may be incorrect.A close up of a leaf

AI-generated content may be incorrect.

A yellow leaf with a hole in it

AI-generated content may be incorrect.A close up of a coin

AI-generated content may be incorrect.A close-up of a leaf

AI-generated content may be incorrect.

A yellow leaf with brown spots

AI-generated content may be incorrect.

Final Comments:

* The file overall is .mlx so as to use global functions created across several tasks in the same environment.
* The code itself is divided into several executable “Sections” to run the code efficiently.
* Currently, for some images, there are “buggy” outputs, majorly due to the way “topNcomponents”, color thresholding for Task I is set, leading to some misplacement of either axi, centroid medoid or damage percentage calculation which have scope of improvement in the future.
* Helper functions and sources:
  + - 1. Color Threshold App from Matlab to gather R, G, B values.
      2. Usage of AI tools for general tasks such as comment usage, formatting and try/catch blocks for Task I.

## APPENDIX

### Coin threshold values using Color threshold App:

function [BW,maskedRGBImage] = createMask(RGB)

%createMask Threshold RGB image using auto-generated code from colorThresholder app.

% [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using

% auto-generated code from the colorThresholder app. The colorspace and

% range for each channel of the colorspace were set within the app. The

% segmentation mask is returned in BW, and a composite of the mask and

% original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 25-Oct-2025

%------------------------------------------------------

% Convert RGB image to chosen color space

I = RGB;

% Define thresholds for channel 1 based on histogram settings

channel1Min = 65.000;

channel1Max = 188.000;

% Define thresholds for channel 2 based on histogram settings

channel2Min = 17.000;

channel2Max = 67.000;

% Define thresholds for channel 3 based on histogram settings

channel3Min = 0.000;

channel3Max = 61.000;

% Create mask based on chosen histogram thresholds

sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...

(I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...

(I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

BW = sliderBW;

% Initialize output masked image based on input image.

maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.

maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

### Leaf threshold values using Color threshold App:

function [BW,maskedRGBImage] = createMask(RGB)

%createMask Threshold RGB image using auto-generated code from colorThresholder app.

% [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using

% auto-generated code from the colorThresholder app. The colorspace and

% range for each channel of the colorspace were set within the app. The

% segmentation mask is returned in BW, and a composite of the mask and

% original RGB images is returned in macanskedRGBImage.

% Auto-generated by colorThresholder app on 25-Oct-2025

%------------------------------------------------------

% Convert RGB image to chosen color space

I = RGB;

% Define thresholds for channel 1 based on histogram settings

channel1Min = 0.000;

channel1Max = 52.000;

% Define thresholds for channel 2 based on histogram settings

channel2Min = 0.000;

channel2Max = 170.000;

% Define thresholds for channel 3 based on histogram settings

channel3Min = 0.000;

channel3Max = 105.000;

% Create mask based on chosen histogram thresholds

sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...

(I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...

(I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

BW = sliderBW;

% Initialize output masked image based on input image.

maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.

maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

### Yellow section threshold values using Color threshold App for Task I:

function [BW,maskedRGBImage] = createMask(RGB)

%createMask Threshold RGB image using auto-generated code from colorThresholder app.

% [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using

% auto-generated code from the colorThresholder app. The colorspace and

% range for each channel of the colorspace were set within the app. The

% segmentation mask is returned in BW, and a composite of the mask and

% original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 10-Dec-2025

%------------------------------------------------------

% Convert RGB image to chosen color space

I = RGB;

% Define thresholds for channel 1 based on histogram settings

channel1Min = 88.000;

channel1Max = 255.000;

% Define thresholds for channel 2 based on histogram settings

channel2Min = 31.000;

channel2Max = 181.000;

% Define thresholds for channel 3 based on histogram settings

channel3Min = 0.000;

channel3Max = 62.000;

% Create mask based on chosen histogram thresholds

sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...

(I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...

(I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

BW = sliderBW;

% Initialize output masked image based on input image.

maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.

maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

### Green section threshold values using Color threshold App for Task I:

function [BW,maskedRGBImage] = createMask(RGB)

%createMask Threshold RGB image using auto-generated code from colorThresholder app.

% [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using

% auto-generated code from the colorThresholder app. The colorspace and

% range for each channel of the colorspace were set within the app. The

% segmentation mask is returned in BW, and a composite of the mask and

% original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 10-Dec-2025

%------------------------------------------------------

% Convert RGB image to chosen color space

I = RGB;

% Define thresholds for channel 1 based on histogram settings

channel1Min = 0.000;

channel1Max = 81.000;

% Define thresholds for channel 2 based on histogram settings

channel2Min = 50.000;

channel2Max = 148.000;

% Define thresholds for channel 3 based on histogram settings

channel3Min = 0.000;

channel3Max = 105.000;

% Create mask based on chosen histogram thresholds

sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...

(I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...

(I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

BW = sliderBW;

% Initialize output masked image based on input image.

maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.

maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end