**03/14/2025**

1. Circularity diganti menjadi ellipticity +frekuensi. Dataset BUSBRA, memiliki data yang lebih banyak menunjukkan ellips
2. Protokol yang digunakan :

* Brightness Adjustment : 1 Netral, <1 jika terlalu terang, tidak >1.5 supaya detail tidak hilang.
* Contras Adjustment : up to >1.5 jika perbedaan objek dan latar belakang kurang jelas. Jika terlalu tinggidan detail hilang, dilakukan uji visual dari penguji independen.
* Gaussian Filter : Kernel standar 7x7 (dapat diatur 3x3 – 9x9, dan dilihat dampak dari tresholding juga).
* Median Filter : sama dengan Gaussian
* Tresholding : nilai uji coba dari dataset yaitu 60, (dapat disesuaikan lagi dari 30-150)

1. Edge-Sharpness :

Hasil SCPM adalah berbentuk Masking, step pada paper :

1. **Edge detection** diterapkan → menghasilkan **masking** tumor.
2. Dari **masking**, kita ambil tepi tumor saja → membuat **edge map.**
3. Dari **edge map,** kita pilih titik-titik tepi → digunakan untuk mengambil **profil intensitas dari citra asli**
4. **Profil intensitas** dianalisis dengan **fungsi sigmoid** → untuk mendapatkan nilai **ketajaman tepi tumorIntensity Profile Extraction**

function [profiles, normal\_lines] = extractIntensityProfiles(original\_image, edge\_points, normal\_length)

% EXTRACTINTENSITYPROFILES Extracts intensity profiles perpendicular to edge points

%

% Inputs:

% original\_image - The original USG image (grayscale)

% edge\_points - Nx2 matrix of edge points [x, y] from SCPM

% normal\_length - Length of normal line segment (in pixels) for sampling (default: 11)

%

% Outputs:

% profiles - Cell array of intensity profiles for each edge point

% normal\_lines - Cell array of coordinates for each normal line (for visualization)

if nargin < 3

normal\_length = 11; % Default length of normal profile (11 pixels - 5 on each side)

end

% Get image dimensions

[img\_height, img\_width] = size(original\_image);

% Number of edge points

num\_points = size(edge\_points, 1);

% Initialize outputs

profiles = cell(num\_points, 1);

normal\_lines = cell(num\_points, 1);

% Distance between sampling points along the normal

sample\_step = 0.5; % Can adjust for finer sampling

% For each edge point

for i = 1:num\_points

% Get current point

x = edge\_points(i, 1);

y = edge\_points(i, 2);

% Calculate normal direction - depends on neighboring points

if i == 1

dx = edge\_points(2, 1) - edge\_points(num\_points, 1);

dy = edge\_points(2, 2) - edge\_points(num\_points, 2);

elseif i == num\_points

dx = edge\_points(1, 1) - edge\_points(num\_points-1, 1);

dy = edge\_points(1, 2) - edge\_points(num\_points-1, 2);

else

dx = edge\_points(i+1, 1) - edge\_points(i-1, 1);

dy = edge\_points(i+1, 2) - edge\_points(i-1, 2);

end

% Calculate normal vector (perpendicular to tangent)

normal\_x = -dy;

normal\_y = dx;

% Normalize the normal vector

normal\_length\_actual = sqrt(normal\_x^2 + normal\_y^2);

normal\_x = normal\_x / normal\_length\_actual;

normal\_y = normal\_y / normal\_length\_actual;

% Sample points along the normal in both directions

half\_length = (normal\_length - 1) / 2;

num\_samples = round(normal\_length / sample\_step);

% Initialize profile and line coordinates

profile = zeros(num\_samples, 1);

line\_coords = zeros(num\_samples, 2);

% Create vector of sample positions

sample\_positions = linspace(-half\_length, half\_length, num\_samples);

% Extract intensity values along the normal

for j = 1:num\_samples

% Calculate sample point coordinates

sample\_x = x + normal\_x \* sample\_positions(j);

sample\_y = y + normal\_y \* sample\_positions(j);

% Store line coordinates for visualization

line\_coords(j, :) = [sample\_x, sample\_y];

% Check if the sample point is within image boundaries

if sample\_x >= 1 && sample\_x <= img\_width && sample\_y >= 1 && sample\_y <= img\_height

% Use bilinear interpolation to get intensity at non-integer coordinates

profile(j) = interp2(double(original\_image), sample\_x, sample\_y, 'linear');

else

% Set to NaN if outside image boundaries

profile(j) = NaN;

end

end

% Store results

profiles{i} = profile;

normal\_lines{i} = line\_coords;

end

% Optional: Add function to visualize the normal lines and profiles

End

**Curve Fitting with Sigmoid Function**

function [sharpness\_metrics, fitted\_curves] = fitSigmoidToProfiles(profiles, sample\_positions)

% FITSIGMOIDTOPROFILES Fits sigmoid curves to intensity profiles and calculates sharpness

%

% Inputs:

% profiles - Cell array of intensity profiles

% sample\_positions - Vector of positions along the normal (x-axis for fitting)

%

% Outputs:

% sharpness\_metrics - Structure with sharpness parameters for each profile

% fitted\_curves - Cell array of fitted sigmoid curves for visualization

if nargin < 2

% Default sample positions if not provided

num\_samples = length(profiles{1});

sample\_positions = linspace(-num\_samples/2, num\_samples/2, num\_samples);

end

num\_profiles = length(profiles);

% Initialize outputs

sharpness\_metrics = struct(...

'slope', zeros(num\_profiles, 1), ...

'midpoint', zeros(num\_profiles, 1), ...

'amplitude', zeros(num\_profiles, 1), ...

'baseline', zeros(num\_profiles, 1), ...

'rmse', zeros(num\_profiles, 1), ...

'r\_squared', zeros(num\_profiles, 1));

fitted\_curves = cell(num\_profiles, 1);

% Sigmoid function for fitting

sigmoid\_func = @(p, x) p(1) + p(2) ./ (1 + exp(-p(3) \* (x - p(4))));

% Options for optimization

options = optimoptions('lsqcurvefit', 'Display', 'off');

% For each profile

for i = 1:num\_profiles

profile = profiles{i};

% Skip profiles with NaN values (outside image boundaries)

if any(isnan(profile))

sharpness\_metrics.slope(i) = NaN;

sharpness\_metrics.midpoint(i) = NaN;

sharpness\_metrics.amplitude(i) = NaN;

sharpness\_metrics.baseline(i) = NaN;

sharpness\_metrics.rmse(i) = NaN;

sharpness\_metrics.r\_squared(i) = NaN;

fitted\_curves{i} = NaN;

continue;

end

% Estimate initial parameters for sigmoid fit

% [baseline, amplitude, slope, midpoint]

baseline\_estimate = min(profile);

amplitude\_estimate = max(profile) - baseline\_estimate;

slope\_estimate = 1; % Initial guess

midpoint\_estimate = sample\_positions(round(length(sample\_positions)/2));

p0 = [baseline\_estimate, amplitude\_estimate, slope\_estimate, midpoint\_estimate];

% Perform curve fitting with least-squares

try

[p\_fit, ~, residuals] = lsqcurvefit(sigmoid\_func, p0, sample\_positions, profile, [], [], options);

% Store the fitted parameters

sharpness\_metrics.baseline(i) = p\_fit(1);

sharpness\_metrics.amplitude(i) = p\_fit(2);

sharpness\_metrics.slope(i) = p\_fit(3);

sharpness\_metrics.midpoint(i) = p\_fit(4);

% Calculate goodness of fit metrics

SSE = sum(residuals.^2);

SST = sum((profile - mean(profile)).^2);

sharpness\_metrics.rmse(i) = sqrt(mean(residuals.^2));

sharpness\_metrics.r\_squared(i) = 1 - SSE/SST;

% Generate fitted curve for visualization

fitted\_curve = sigmoid\_func(p\_fit, sample\_positions);

fitted\_curves{i} = fitted\_curve;

catch

% If fitting fails, set as NaN

sharpness\_metrics.slope(i) = NaN;

sharpness\_metrics.midpoint(i) = NaN;

sharpness\_metrics.amplitude(i) = NaN;

sharpness\_metrics.baseline(i) = NaN;

sharpness\_metrics.rmse(i) = NaN;

sharpness\_metrics.r\_squared(i) = NaN;

fitted\_curves{i} = NaN;

end

end

end

**Hypothesis Testing for Edge Sharpness**

function results = performHypothesisTesting(sharpness\_data, group\_labels)

% PERFORMHYPOTHESISTESTING Performs statistical analysis of edge sharpness data

%

% Inputs:

% sharpness\_data - Cell array of sharpness metrics for different images/groups

% group\_labels - Cell array of group labels

%

% Outputs:

% results - Structure with hypothesis testing results

if nargin < 2

% Generate default group labels if not provided

for i = 1:length(sharpness\_data)

group\_labels{i} = ['Group ' num2str(i)];

end

end

% Extract slopes (main sharpness metric) from each group

slopes = cell(length(sharpness\_data), 1);

for i = 1:length(sharpness\_data)

slopes{i} = sharpness\_data{i}.slope;

% Remove NaN values

slopes{i} = slopes{i}(~isnan(slopes{i}));

end

% Initialize results structure

results = struct();

% Calculate basic statistics for each group

results.stats = struct();

for i = 1:length(slopes)

results.stats(i).group = group\_labels{i};

results.stats(i).mean = mean(slopes{i});

results.stats(i).median = median(slopes{i});

results.stats(i).std = std(slopes{i});

results.stats(i).min = min(slopes{i});

results.stats(i).max = max(slopes{i});

results.stats(i).n = length(slopes{i});

end

% Prepare data for ANOVA

all\_slopes = [];

all\_groups = [];

for i = 1:length(slopes)

all\_slopes = [all\_slopes; slopes{i}];

all\_groups = [all\_groups; i\*ones(length(slopes{i}), 1)];

end

% Perform one-way ANOVA

[results.anova.p, results.anova.table, results.anova.stats] = anova1(all\_slopes, all\_groups, 'off');

% Check if ANOVA shows significant differences

results.significant\_diff = results.anova.p < 0.05;

% Perform Tukey's HSD test if ANOVA is significant

if results.significant\_diff

% Calculate critical values for Tukey's HSD

[results.tukey.c, results.tukey.means, results.tukey.h, results.tukey.names] = multcompare(results.anova.stats, 'Display', 'off');

% Format the Tukey's HSD results

results.tukey.comparisons = cell(size(results.tukey.c, 1), 1);

for i = 1:size(results.tukey.c, 1)

group1 = group\_labels{results.tukey.c(i, 1)};

group2 = group\_labels{results.tukey.c(i, 2)};

mean\_diff = results.tukey.c(i, 4);

lower\_ci = results.tukey.c(i, 3);

upper\_ci = results.tukey.c(i, 5);

p\_value = results.tukey.c(i, 6);

results.tukey.comparisons{i} = struct(...

'group1', group1, ...

'group2', group2, ...

'mean\_diff', mean\_diff, ...

'ci\_lower', lower\_ci, ...

'ci\_upper', upper\_ci, ...

'p\_value', p\_value, ...

'significant', p\_value < 0.05);

end

end

% Add box plot visualization function

results.createBoxPlot = @createBoxPlot;

function fig = createBoxPlot(slopes, group\_labels)

fig = figure;

boxplot(all\_slopes, all\_groups, 'Labels', group\_labels);

title('Edge Sharpness Comparison');

ylabel('Sharpness Metric (Sigmoid Slope)');

grid on;

end

end

**Validation of Edge Sharpness Analysis**

function results = validateEdgeSharpnessMethod(original\_images, edge\_points, options)

% VALIDATEEDGESHARPNESSMETHOD Validates the edge sharpness analysis method

%

% Inputs:

% original\_images - Cell array of original USG images

% edge\_points - Cell array of edge points from SCPM for each image

% options - Structure with validation settings (optional)

%

% Outputs:

% results - Structure with validation results

% Default options

if nargin < 3

options = struct();

end

% Set default options if not provided

if ~isfield(options, 'normal\_length')

options.normal\_length = 11;

end

if ~isfield(options, 'visualize')

options.visualize = true;

end

if ~isfield(options, 'sample\_step')

options.sample\_step = 0.5;

end

if ~isfield(options, 'group\_labels')

for i = 1:length(original\_images)

options.group\_labels{i} = ['Image ' num2str(i)];

end

end

% Initialize results structure

results = struct();

results.sharpness\_data = cell(length(original\_images), 1);

results.profiles = cell(length(original\_images), 1);

results.fitted\_curves = cell(length(original\_images), 1);

% Process each image

for img\_idx = 1:length(original\_images)

% Current image and edge points

img = original\_images{img\_idx};

edges = edge\_points{img\_idx};

% Extract intensity profiles

[profiles, normal\_lines] = extractIntensityProfiles(img, edges, options.normal\_length);

% Create sample positions for curve fitting

num\_samples = length(profiles{1});

sample\_positions = linspace(-(options.normal\_length-1)/2, (options.normal\_length-1)/2, num\_samples);

% Fit sigmoid curves to profiles

[sharpness\_metrics, fitted\_curves] = fitSigmoidToProfiles(profiles, sample\_positions);

% Store results

results.sharpness\_data{img\_idx} = sharpness\_metrics;

results.profiles{img\_idx} = profiles;

results.fitted\_curves{img\_idx} = fitted\_curves;

% Visualize if requested

if options.visualize

visualizeResults(img, edges, normal\_lines, profiles, fitted\_curves, sharpness\_metrics, options.group\_labels{img\_idx});

end

end

% Perform hypothesis testing

results.stats = performHypothesisTesting(results.sharpness\_data, options.group\_labels);

% Calculate overall metrics

all\_slopes = [];

for i = 1:length(results.sharpness\_data)

valid\_slopes = results.sharpness\_data{i}.slope(~isnan(results.sharpness\_data{i}.slope));

all\_slopes = [all\_slopes; valid\_slopes];

end

results.overall\_mean = mean(all\_slopes);

results.overall\_std = std(all\_slopes);

% Generate report

results.generateReport = @generateReport;

function generateReport()

% Create a summary report

fprintf('=== Edge Sharpness Analysis Report ===\n\n');

% Overall statistics

fprintf('Overall Statistics:\n');

fprintf('Total edges analyzed: %d\n', length(all\_slopes));

fprintf('Mean sharpness (slope): %.4f\n', results.overall\_mean);

fprintf('Standard deviation: %.4f\n\n', results.overall\_std);

% Group statistics

fprintf('Group Statistics:\n');

for i = 1:length(results.stats.stats)

fprintf('Group %s:\n', results.stats.stats(i).group);

fprintf(' Mean sharpness: %.4f\n', results.stats.stats(i).mean);

fprintf(' Median sharpness: %.4f\n', results.stats.stats(i).median);

fprintf(' Standard deviation: %.4f\n', results.stats.stats(i).std);

fprintf(' Number of valid edges: %d\n\n', results.stats.stats(i).n);

end

% ANOVA results

fprintf('Hypothesis Testing Results:\n');

fprintf('ANOVA p-value: %.6f\n', results.stats.anova.p);

if results.stats.significant\_diff

fprintf('Significant differences detected between groups (p < 0.05)\n\n');

% Tukey's HSD results

fprintf('Tukey''s HSD Test Results:\n');

for i = 1:length(results.stats.tukey.comparisons)

comp = results.stats.tukey.comparisons{i};

fprintf(' %s vs %s: Mean diff = %.4f, p = %.6f', ...

comp.group1, comp.group2, comp.mean\_diff, comp.p\_value);

if comp.significant

fprintf(' (significant)\n');

else

fprintf(' (not significant)\n');

end

end

else

fprintf('No significant differences detected between groups (p >= 0.05)\n');

end

end

function visualizeResults(img, edges, normal\_lines, profiles, fitted\_curves, sharpness\_metrics, title\_str)

% Create a figure with 2x2 subplots for visualization

figure('Name', ['Edge Sharpness Analysis: ' title\_str], 'Position', [100, 100, 1000, 800]);

% Subplot 1: Original image with edge points

subplot(2, 2, 1);

imshow(img, []); hold on;

plot(edges(:,1), edges(:,2), 'r.', 'MarkerSize', 6);

title('Original Image with Edge Points');

% Subplot 2: Show normal lines

subplot(2, 2, 2);

imshow(img, []); hold on;

% Plot a subset of normal lines for clarity

num\_edges = length(normal\_lines);

sample\_interval = max(1, round(num\_edges / 50)); % Limit to ~50 lines

for i = 1:sample\_interval:num\_edges

plot(normal\_lines{i}(:,1), normal\_lines{i}(:,2), 'g-', 'LineWidth', 1);

end

title('Intensity Profile Sampling Lines');

% Subplot 3: Show a sample of intensity profiles and their fits

subplot(2, 2, 3);

% Sample a few profiles

num\_to\_sample = min(5, num\_edges);

sample\_indices = round(linspace(1, num\_edges, num\_to\_sample));

num\_samples = length(profiles{1});

sample\_positions = linspace(-(options.normal\_length-1)/2, (options.normal\_length-1)/2, num\_samples);

cmap = lines(num\_to\_sample);

hold on;

legend\_entries = cell(num\_to\_sample \* 2, 1);

for i = 1:num\_to\_sample

idx = sample\_indices(i);

% Skip if profile has NaN values

if any(isnan(profiles{idx})) || any(isnan(fitted\_curves{idx}))

continue;

end

plot(sample\_positions, profiles{idx}, 'o', 'Color', cmap(i,:), 'MarkerSize', 4);

plot(sample\_positions, fitted\_curves{idx}, '-', 'Color', cmap(i,:), 'LineWidth', 2);

legend\_entries{2\*i-1} = ['Profile ' num2str(idx) ' (Data)'];

legend\_entries{2\*i} = ['Profile ' num2str(idx) ' (Fit)'];

end

% Remove empty entries

legend\_entries = legend\_entries(~cellfun('isempty', legend\_entries));

title('Sample Intensity Profiles with Sigmoid Fits');

xlabel('Position along Normal (pixels)');

ylabel('Intensity');

legend(legend\_entries, 'Location', 'best');

grid on;

% Subplot 4: Histogram of sharpness values

subplot(2, 2, 4);

% Get valid slopes

valid\_slopes = sharpness\_metrics.slope(~isnan(sharpness\_metrics.slope));

% Create histogram

histogram(valid\_slopes, 20);

title('Distribution of Edge Sharpness');

xlabel('Sharpness (Sigmoid Slope)');

ylabel('Frequency');

grid on;

% Add statistics text

text(0.05, 0.95, sprintf('Mean: %.4f\nMedian: %.4f\nStd: %.4f\nNum Edges: %d', ...

mean(valid\_slopes), median(valid\_slopes), std(valid\_slopes), length(valid\_slopes)), ...

'Units', 'normalized', 'VerticalAlignment', 'top');

end

end

**Main Execution Script**

%% Main script for USG image edge sharpness analysis

% This script performs edge sharpness analysis on USG images using the SCPM method

% for edge detection, followed by intensity profile extraction, sigmoid fitting,

% and statistical analysis.

clear all;

close all;

clc;

%% Step 1: Load images and edge points

% Assuming you already have SCPM edge detection results

% Replace these paths with your actual image and SCPM result paths

image\_folder = 'path/to/usg/images/';

edge\_data\_folder = 'path/to/edge/data/';

% Get all image files

image\_files = dir(fullfile(image\_folder, '\*.png'));

% Initialize cell arrays for images and edge points

original\_images = cell(length(image\_files), 1);

edge\_points = cell(length(image\_files), 1);

group\_labels = cell(length(image\_files), 1);

% Load images and corresponding edge data

for i = 1:length(image\_files)

% Load image

img\_path = fullfile(image\_folder, image\_files(i).name);

original\_images{i} = imread(img\_path);

% Convert to grayscale if needed

if size(original\_images{i}, 3) == 3

original\_images{i} = rgb2gray(original\_images{i});

end

% Load corresponding edge points from SCPM

% Assuming edge points are saved as .mat files with same base name

[~, basename, ~] = fileparts(image\_files(i).name);

edge\_file = fullfile(edge\_data\_folder, [basename '\_edge.mat']);

if exist(edge\_file, 'file')

load(edge\_file, 'finalX', 'finalY');

edge\_points{i} = [finalX', finalY']; % Combine X and Y into N×2 matrix

else

error('Edge data file not found: %s', edge\_file);

end

% Create group label from filename

group\_labels{i} = basename;

end

fprintf('Loaded %d images and their corresponding edge points.\n', length(image\_files));

%% Step 2: Set validation options

options = struct();

options.normal\_length = 11; % Length of normal line for intensity profile (in pixels)

options.visualize = true; % Enable visualization

options.sample\_step = 0.5; % Sampling step along normal line (in pixels)

options.group\_labels = group\_labels; % Group labels for statistical analysis

%% Step 3: Run the validation function

fprintf('Starting edge sharpness analysis...\n');

results = validateEdgeSharpnessMethod(original\_images, edge\_points, options);

%% Step 4: Generate and print report

fprintf('\nGenerating report...\n\n');

results.generateReport();

%% Step 5: Create box plot comparison

figure;

boxplot\_fig = results.stats.createBoxPlot(results.sharpness\_data, options.group\_labels);

%% Step 6: Save results

save\_path = 'results/edge\_sharpness\_analysis.mat';

if ~exist('results', 'dir')

mkdir('results');

end

save(save\_path, 'results', '-v7.3');

fprintf('\nResults saved to: %s\n', save\_path);

%% Example of how to access specific metrics

fprintf('\nExample accessing specific metrics:\n');

fprintf('Average sharpness for first image: %.4f\n', mean(results.sharpness\_data{1}.slope(~isnan(results.sharpness\_data{1}.slope))));

% Display ANOVA p-value

fprintf('ANOVA p-value: %.6f\n', results.stats.anova.p);

if results.stats.significant\_diff

fprintf('There are significant differences between the groups (p < 0.05)\n');

else

fprintf('No significant differences detected between the groups (p >= 0.05)\n');

end

Saya telah membuat empat modul kode MATLAB terpisah untuk implementasi analisis ketajaman intensitas gambar USG yang Anda jelaskan, ditambah satu script utama untuk menjalankan seluruh proses:

1. **Intensity Profile Extraction** - Mengekstraksi profil intensitas sepanjang garis normal dari tepi yang terdeteksi oleh SCPM.
2. **Curve Fitting with Sigmoid Function** - Melakukan pencocokan kurva sigmoid terhadap profil intensitas dan menghitung parameter ketajaman.
3. **Hypothesis Testing for Edge Sharpness** - Melakukan analisis statistik (ANOVA dan Tukey's HSD) untuk membandingkan ketajaman tepi antar gambar.
4. **Validation of Edge Sharpness Analysis** - Mengintegrasikan semua modul di atas untuk validasi metode analisis ketajaman tepi.
5. **Main Execution Script** - Script utama untuk menjalankan seluruh proses analisis.

**Catatan Implementasi:**

1. **Ekstraksi Profil Intensitas**:
   * Kode menghitung arah normal ke tepi dengan memanfaatkan titik-titik tepi tetangga
   * Menggunakan interpolasi bilinear untuk mendapatkan nilai intensitas pada koordinat non-integer
   * Menyimpan koordinat garis normal untuk visualisasi
2. **Pencocokan Kurva Sigmoid**:
   * Menggunakan fungsi lsqcurvefit untuk least-squares fitting
   * Parameter sigmoid: baseline, amplitude, slope (ketajaman), dan midpoint
   * Menghitung metrik goodness-of-fit (RMSE dan R²)
3. **Pengujian Hipotesis**:
   * Mengimplementasikan ANOVA untuk membandingkan distribusi ketajaman antar gambar
   * Melakukan Tukey's HSD test untuk perbandingan berpasangan
   * Menyediakan fungsi untuk visualisasi dengan box plot
4. **Validasi**:
   * Mengintegrasikan semua langkah di atas
   * Menyediakan visualisasi komprehensif untuk hasil
   * Menghasilkan laporan statistik tentang ketajaman tepi

**Cara Penggunaan:**

1. Pastikan Anda sudah memiliki hasil edge detection dari Spring-Charged Particle Model dalam format yang kompatibel (variabel finalX dan finalY).
2. Sesuaikan parameter dalam script utama:
   * Folder citra USG
   * Folder data tepi
   * Parameter ekstraksi profil intensitas
3. Jalankan script utama untuk melakukan analisis lengkap.

Kode ini dirancang agar modular dan dapat disesuaikan dengan kebutuhan spesifik penelitian Anda. Setiap fungsi memiliki dokumentasi yang jelas dan dapat digunakan secara independen jika diperlukan.