Praktikum Computer Vision

CS5670: Computer Vision

NAMA : **Marco Albert**
NIM : **2155301082**
KELAS : **4 TI A**
DOSEN : **Ananda, S.Kom., M.T., Ph.D.**
ILB : **M. Anwar, S. Tr. Kom.**

PROGRAM STUDI TEKNIK INFORMATIKA
POLITEKNIK CALTEX RIAU
TA 2024 / 2025

I. Intro To Computer Vision (CS5670)

1. Noise Reduction

a. Image Filtering

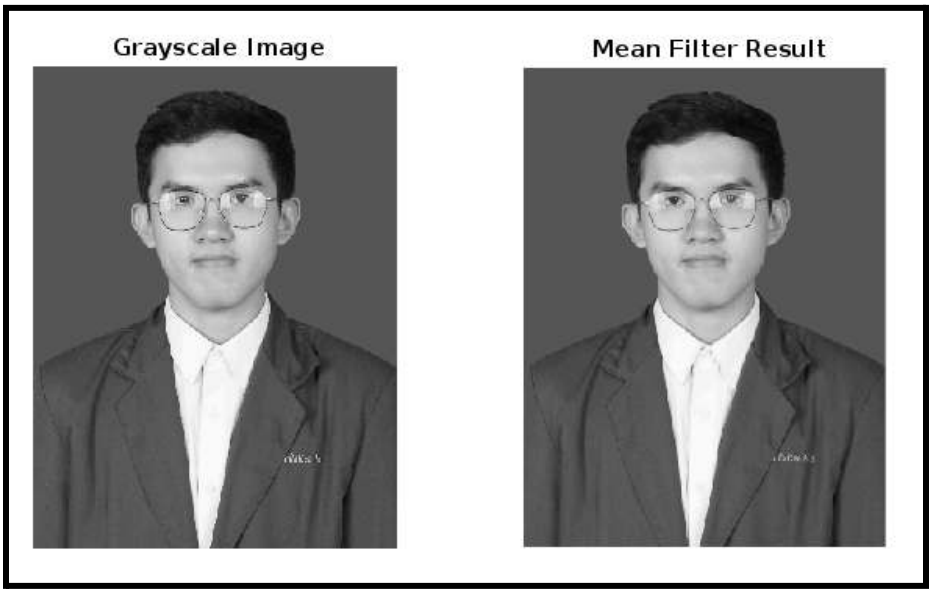
Input :

ImageFiltering.m ×

/MATLAB Drive/ImageFiltering.m

```
gambar1 = imread("ProfilePicture.jpg");

% Konversi ke grayscale
gryGambar1 = rgb2gray(gambar1);

% Definisi kernel mean filter 3x3
kernel = ones(3,3) / 9;

% Proses konvolusi (filtering)
filterImage = conv2(double(gryGambar1), kernel, 'same');

% Tampilkan hasil secara berdampingan
figure;
subplot(1,2,1); % posisi kiri
imshow(gryGambar1);
title('Grayscale Image');

subplot(1,2,2); % posisi kanan
imshow(uint8(filterImage));
title('Mean Filter Result');
```

Output :



Analisis : Dari hasil percobaan di atas, dengan membaca gambar berwarna dan mengonversinya ke dalam format grayscale, diterapkan mean filter menggunakan kernel 3x3 untuk menghaluskan citra (mengurangi noise atau detail halus) melalui operasi konvolusi. Hasilnya ditampilkan berdampingan: citra grayscale asli dan hasil filtering. Keseluruhan proses ini bertujuan untuk menunjukkan efek penyaringan menggunakan rata-rata lokal terhadap kualitas visual citra.

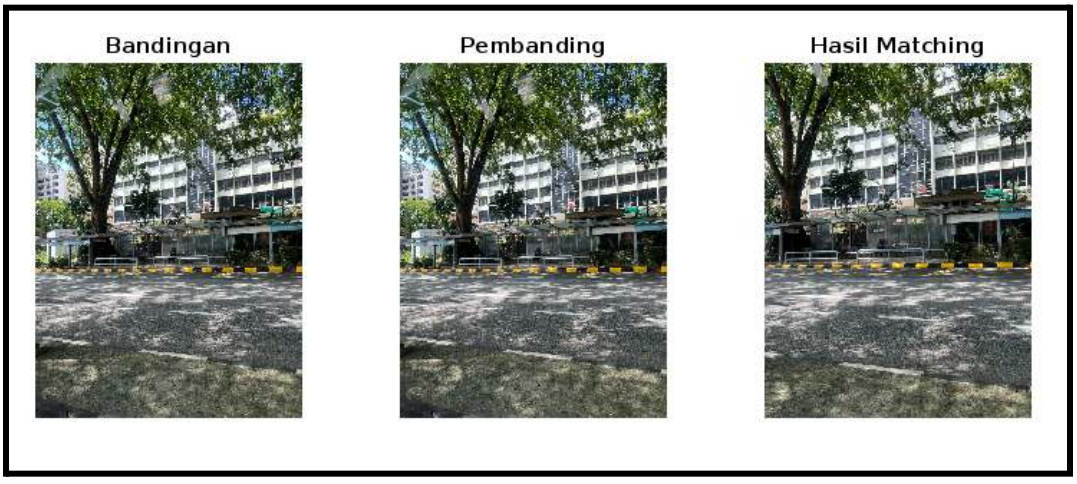
b. Cross Correlation

Input :

CrossCorrelation.m

```
img = imread('Holiday.jpg');
template = imread('Holiday.jpg');
img_gray = rgb2gray(img);
template_gray = rgb2gray(template);

c = normxcorr2(template_gray, img_gray);
[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax);
corr_offset = [xpeak - size(template_gray, 2), ypeak - size(template_gray, 1)];

figure;
subplot(1, 3, 1); imshow(img); title('Bandingan');
subplot(1, 3, 2); imshow(template); title('Pembanding');
subplot(1, 3, 3); imshow(img);
hold on;
rectangle('Position', [corr_offset(1), corr_offset(2), size(template_gray, 2), size(template_gray,1)], ...
    'EdgeColor','g','LineWidth',2);
title('Hasil Matching');
```

Output :



Analisis : Dari hasil percobaan di atas, teknik template matching menggunakan metode normalized cross-correlation pada citra yang sama (yaitu "Holiday.jpg"). Meskipun gambar sumber dan templatenya identik, proses ini tetap dijalankan untuk mendeteksi posisi kecocokan tertinggi antara template dan gambar utama. Setelah menemukan koordinat puncak korelasi, sebuah bounding box digambar pada posisi tersebut untuk menunjukkan area hasil pencocokan. Visualisasi dilakukan dengan membandingkan

gambar utama, template, dan hasil pencocokan yang ditandai. Secara keseluruhan, kode ini menunjukkan cara kerja deteksi kemiripan pola dalam citra menggunakan korelasi ter-normalisasi.

c. Linear Filtering

Input :

LinearFiltering.m

/MATLAB Drive/LinearFiltering.m

```
gambar1 = imread('Holiday.jpg');
dGambar = im2double(gambar1);

kernel = [0     0       0;
          0     0.5     0;
          0     1       0.5];

filtered_img = zeros(size(dGambar));

if size(dGambar, 3) == 3  % Gambar RGB
    for c = 1:3
        filtered_img(:,:,c) = conv2(dGambar(:,:,c), kernel, 'same');
    end
else  % Gambar grayscale
    filtered_img = conv2(dGambar, kernel, 'same');
end

imshow(filtered_img);

```

Output :

Analisis : Dari hasil percobaan di atas, dilakukan filtering pada gambar “Holiday.jpg” menggunakan kernel asimetris yang dirancang secara manual. Gambar dikonversi terlebih dahulu ke format bertipe double untuk memungkinkan perhitungan presisi tinggi dalam konvolusi. Jika gambar berformat RGB, proses filtering dilakukan secara terpisah untuk tiap kanal warna (R, G, B), kemudian digabung kembali. Kernel yang digunakan menghasilkan efek spasial tertentu, kemungkinan menonjolkan arah tertentu dalam citra (seperti diagonal atau kombinasi vertikal-horizontal), tergantung pada distribusi nilai di dalamnya. Hasil akhirnya adalah citra dengan efek visual tertentu tergantung dari karakteristik kernel tersebut, dan ditampilkan langsung.

d. Convolution

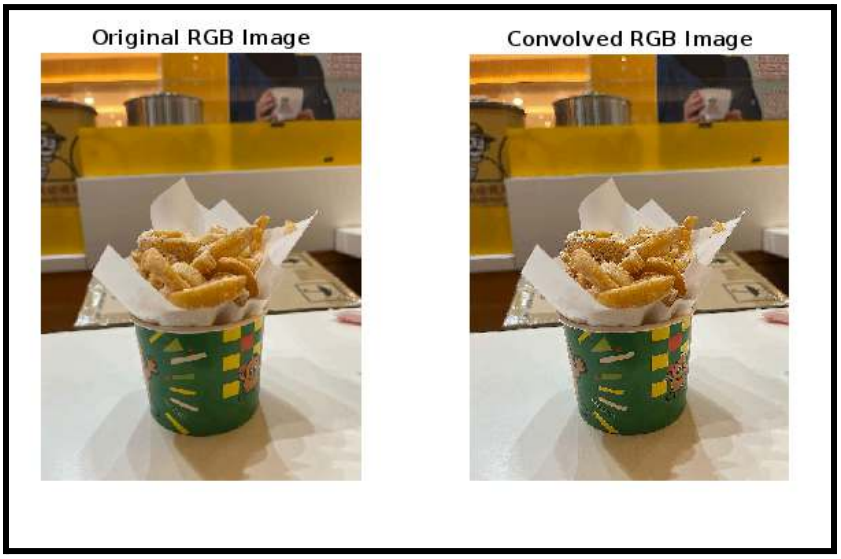
Input :

Convolution.m ×

/MATLAB Drive/Convolution.m

```
img = imread('Potato.jpg');
img = im2double(img);

kernel = [ 0  -1  0;
          -1  5  -1;
           0  -1  0];

kernel_flipped = rot90(kernel, 2);

conv_img = zeros(size(img));

for c = 1:3
    conv_img(:,:,c) = conv2(img(:,:,c), kernel_flipped, 'same');
end

figure;
subplot(1,2,1); imshow(img); title('Original RGB Image');
subplot(1,2,2); imshow(conv_img); title('Convolved RGB Image');
```

Output :



Analisis : Dari hasil percobaan di atas, dilakukan penajaman (sharpening) pada gambar “Potato.jpg” menggunakan kernel konvolusi yang dikenal sebagai sharpen filter. Kernel tersebut dirancang untuk menonjolkan tepi dan detail dengan cara memperkuat perbedaan antara piksel pusat dan sekitarnya. Sebelum proses konvolusi dilakukan, kernel dibalik 180 derajat (dengan rot90(kernel, 2)) sesuai prinsip konvolusi. Filter diterapkan secara terpisah pada ketiga kanal warna (R, G, B) dari gambar RGB yang telah dikonversi ke tipe double untuk akurasi perhitungan. Hasilnya adalah citra yang tampak lebih tajam dan kontras tinggi dibandingkan gambar aslinya, dan keduanya ditampilkan berdampingan untuk perbandingan visual.

e. Mean Filtering

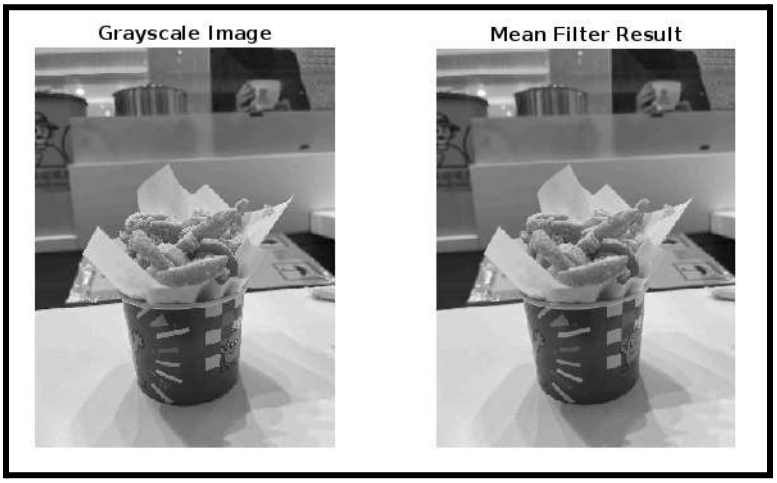
Input :

MeanFiltering.m ×

/MATLAB Drive/MeanFiltering.m

```
gambar1 = imread('Potato.jpg');

% Konversi ke grayscale
gryGambar1 = rgb2gray(gambar1);

% Definisi kernel mean filter
kernel = ones(3,3) / 9;

% Konvolusi/filtering
filterImage = conv2(double(gryGambar1), kernel, 'same');

% Tampilkan hasil secara berdampingan
figure;
subplot(1,2,1); % subplot baris 1, kolom 2, posisi 1
imshow(gryGambar1);
title('Grayscale Image');

subplot(1,2,2); % posisi 2
imshow(uint8(filterImage));
title('Mean Filter Result');

```

Output :



Analisis : Dari hasil percobaan di atas, dilakukan proses smoothing atau perataan citra dengan menerapkan mean filter pada gambar “Potato.jpg” yang telah dikonversi ke grayscale. Mean filter menggunakan kernel 3x3 dengan nilai rata-rata untuk mengurangi noise dan melembutkan detail halus dalam citra. Proses filtering dilakukan dengan operasi konvolusi, menghasilkan gambar yang tampak lebih halus dibandingkan versi grayscale aslinya. Hasil visual dari citra sebelum dan sesudah filter ditampilkan berdampingan untuk mempermudah perbandingan efek penyaringan tersebut.

2. Linear Filter

a. Shifted Left by 1 Pixel

Input :

ShiftedLeftBy1Pixel.m

/MATLAB Drive/ShiftedLeftBy1Pixel.m

```
% Baca gambar
Monument = imread('Monument.jpg');

% Kernel untuk shifting (misalnya ke kiri)
kernel = [0 1 0];

% Siapkan array kosong untuk hasil shifting
shifted_img = zeros(size(Monument), 'like', Monument);

% Lakukan filtering per channel warna (R, G, B)
for c = 1:3
    shifted_img(:,:,c) = imfilter(Monument(:,:,c), kernel, 'replicate');
end

% Tampilkan perbandingan antara gambar asli dan hasil shifting
figure;
subplot(1,2,1);
imshow(Monument);
title('Original Image');

subplot(1,2,2);
imshow(shifted_img);
title('Shifted Image (Left by 1 Pixel)');
```

Output :

Analisis : Dari hasil percobaan di atas, dilakukan efek pergeseran (shifting) horizontal ke kiri pada gambar “Monument.jpg” dengan menerapkan kernel [0 1 0] menggunakan fungsi imfilter. Kernel tersebut secara efektif memindahkan nilai piksel ke posisi sebelah kiri, sehingga menghasilkan efek visual seolah gambar bergeser satu piksel ke kiri. Proses dilakukan secara terpisah pada setiap kanal warna (R, G, B) dan menggunakan metode 'replicate' untuk menangani batas gambar agar piksel tepi tetap terisi. Hasil akhirnya ditampilkan berdampingan untuk membandingkan gambar asli dan hasil pergeseran, memperlihatkan dampak sederhana namun nyata dari operasi spasial tersebut.

b. Blur (Mean Filter)

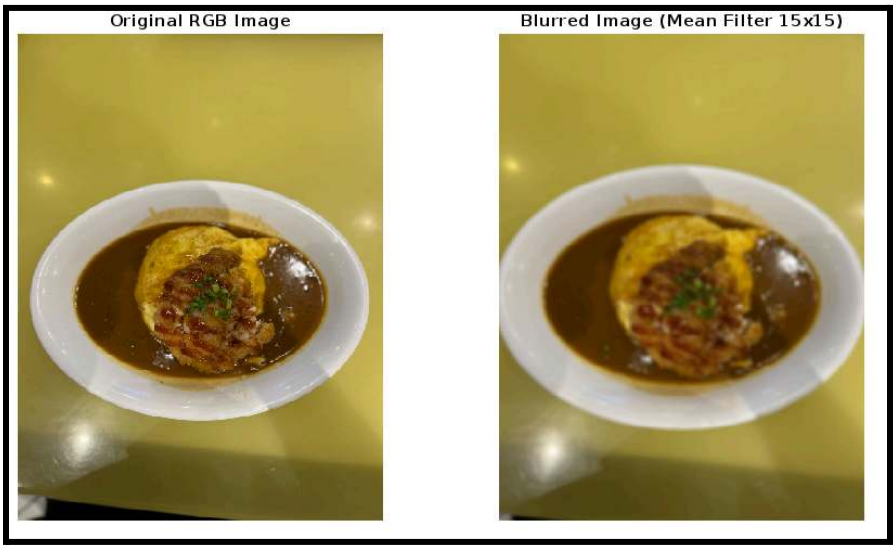
Input :

Blur.m ×

/MATLAB Drive/Blur.m

```
% Baca gambar
Curry = imread('ChickenCurry.jpg');

% Buat kernel blur (mean filter 15x15 untuk efek blur lebih jelas)
kernel = ones(15,15) / (15*15);

% Siapkan array kosong untuk gambar hasil blur
blurred_img = zeros(size(Curry), 'like', Curry);

% Lakukan blur per channel warna (RGB)
for c = 1:3
    blurred_img(:,:,c) = imfilter(Curry(:,:,c), kernel, 'replicate');
end

% Tampilkan perbandingan antara gambar asli dan hasil blur
figure;
subplot(1,2,1);
imshow(Curry);
title('Original RGB Image');

subplot(1,2,2);
imshow(blurred_img);
title('Blurred Image (Mean Filter 15x15)');
```

Output :



Analisis : Dari hasil percobaan di atas, dengan menerapkan efek blur pada gambar “ChickenCurry.jpg” dengan menggunakan mean filter berukuran besar (15x15), yang menghasilkan perataan intensitas piksel secara menyeluruh dan menghilangkan banyak detail halus dalam gambar. Proses dilakukan secara terpisah untuk tiap kanal warna (R, G, B), dan fungsi imfilter digunakan dengan metode 'replicate' untuk menjaga nilai piksel di tepi gambar. Hasil blur yang cukup signifikan ditampilkan berdampingan dengan gambar asli, memperlihatkan perbedaan mencolok dalam ketajaman dan kejelasan visual. Teknik ini umum digunakan untuk mengurangi noise atau menciptakan efek artistik pada citra.

c. Sharpening Filter

Input :

SharpeningFilter.m

MATLAB Drive/SharpeningFilter.m

```
img = imread('Selfie.jpg');
img = im2double(img);

blur_kernel = ones(3,3)/9;
identity_kernel = zeros(3,3);
identity_kernel(2,2)=2;

sharpen_kernel = identity_kernel - blur_kernel;
sharpened_img = zeros(size(img));

for c = 1:3
    sharpened_img(:,:,c) = imfilter(img(:,:,c), sharpen_kernel, 'replicate');
end

figure;
subplot(1,2,1); imshow(img); title('Original Image')
subplot(1,2,2); imshow(sharpened_img); title('Sharpened Image (Unsharp Masking');
```

Output :

Analisis : Dari hasil percobaan di atas, dengan menerapkan teknik penajaman citra menggunakan metode unsharp masking pada gambar “Selfie.jpg”. Citra awal dikonversi ke format double untuk presisi perhitungan, lalu dibentuk kernel penajaman dengan cara mengurangi kernel blur dari kernel identitas yang diperkuat. Hasilnya adalah filter yang mempertegas tepi dan detail citra tanpa mengubah struktur global secara drastis. Proses ini dilakukan per kanal warna (RGB), dan hasil akhirnya ditampilkan berdampingan dengan citra asli, memperlihatkan peningkatan ketajaman visual yang membuat kontur dan fitur wajah tampak lebih jelas.

3. Gaussian

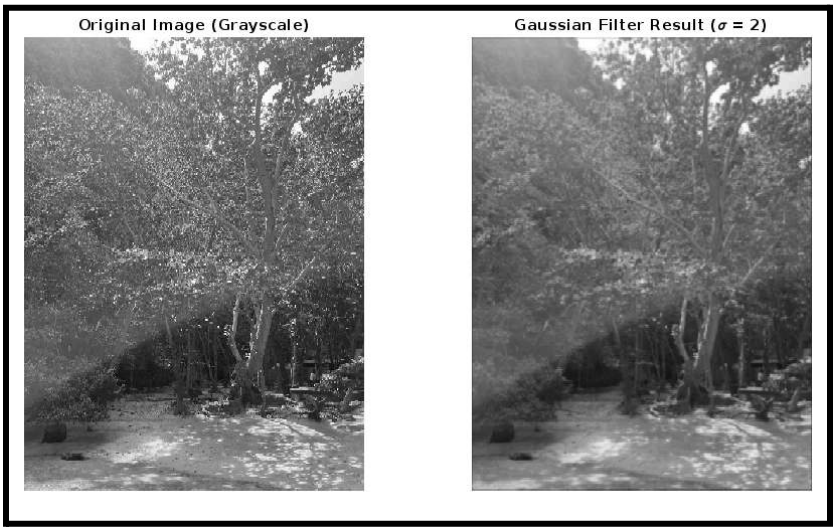
Input :

Gaussian.m ×

/MATLAB Drive/Gaussian.m

```
% Baca dan konversi gambar ke grayscale
gambar = imread('Tree.jpg');
abu_abu = rgb2gray(gambar);

% Parameter Gaussian
sigma_val = 2;
ukuran_filter = 2 * ceil(3 * sigma_val) + 1;

% Buat filter Gaussian
filter_gaussian = fspecial('gaussian', ukuran_filter, sigma_val);

% Proses filtering
hasil_filter = imfilter(abu_abu, filter_gaussian, 'same');

% Tampilkan hasil perbandingan
figure;
subplot(1,2,1);
imshow(abu_abu);
title('Original Image (Grayscale)');

subplot(1,2,2);
imshow(hasil_filter);
title(['Gaussian Filter Result (\sigma = ', num2str(sigma_val), ')']);

```

Output :



Analisis : Dari hasil percobaan di atas, menerapkan filter Gaussian pada gambar "Tree.jpg" yang telah dikonversi ke grayscale. Dengan menggunakan parameter sigma dan ukuran filter yang disesuaikan, filter Gaussian dibuat untuk melakukan smoothing atau penghalusan citra dengan cara mengurangi noise dan detail halus. Hasil filtering diterapkan pada citra grayscale menggunakan fungsi imfilter. Perbandingan antara gambar asli (grayscale) dan hasil filter ditampilkan berdampingan, memperlihatkan efek penghalusan yang menonjolkan bagian-bagian gambar yang lebih besar atau lebih halus sambil mengurangi noise. Teknik ini sering digunakan dalam berbagai aplikasi pengolahan citra untuk mempersiapkan gambar sebelum proses lebih lanjut, seperti deteksi tepi.

4. Filter : Thresholding

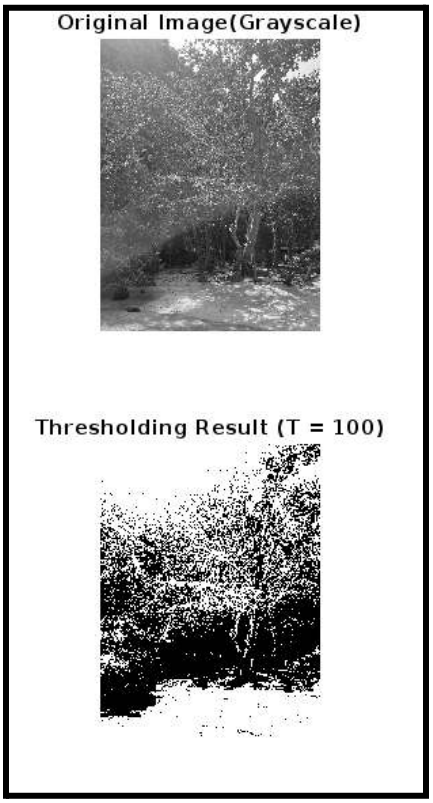
Input :

Thresholding.m

/MATLAB Drive/Thresholding.m

```
% Baca gambar dan konversi ke grayscale
pohon = imread('Tree.jpg');
abu2 = rgb2gray(pohon);

% Tetapkan nilai threshold
batas_threshold = 100;

% Proses thresholding
biner_mask = abu2 > batas_threshold;
gambar_biner = uint8(biner_mask) * 255;

% Tampilkan gambar asli dan hasil thresholding
figure;

subplot(2,1,1);
imshow(abu2);
title('Original Image(Grayscale)');

subplot(2,1,2);
imshow(gambar_biner);
judul = sprintf('Thresholding Result (T = %d)', batas_threshold);
title(judul);

```

Output :



Analisis : Dari hasil percobaan di atas, dengan menerapkan teknik thresholding pada gambar "Tree.jpg" yang telah dikonversi ke grayscale. Dengan menggunakan nilai threshold yang ditetapkan (100), citra diproses untuk menghasilkan gambar biner, di mana piksel dengan nilai intensitas lebih besar dari

threshold diubah menjadi putih (255) dan yang lebih kecil menjadi hitam (0). Hasilnya adalah citra biner yang memisahkan area terang dan gelap berdasarkan nilai threshold. Kedua gambar — asli (grayscale) dan hasil thresholding — ditampilkan berdampingan untuk memperlihatkan perbedaan yang jelas antara keduanya, dengan gambar biner yang menampilkan pola yang lebih jelas sesuai ambang batas yang ditentukan. Teknik ini sering digunakan dalam segmentasi citra untuk memisahkan objek dari latar belakang.

5. Sharpening Revisited

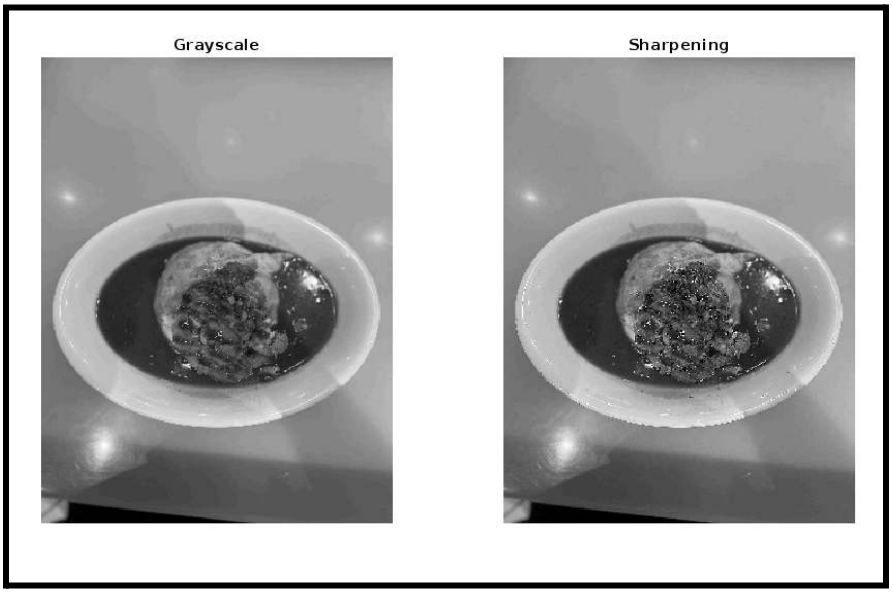
Input :

SharpeningRevisited.m

/MATLAB Drive/SharpeningRevisited.m

```
% Membaca gambar dan mengubah ke skala abu-abu
curry = imread('ChickenCurry.jpg');
abu_abu = rgb2gray(curry);

% Membuat filter penajaman (unsharp)
penajam = fspecial('unsharp');

% Terapkan filter ke gambar grayscale
hasil_tajam = imfilter(abu_abu, penajam, 'same');

% Menampilkan hasil sebelum dan sesudah sharpening
figure;

subplot(1,2,1);
imshow(abu_abu);
title('Grayscale');

subplot(1,2,2);
imshow(hasil_tajam);
title('Sharpening');

```

Output :



Analisis : Dari hasil percobaan di atas, dengan menerapkan teknik penajaman (sharpening) pada gambar "ChickenCurry.jpg" yang telah dikonversi ke grayscale. Filter penajaman yang digunakan adalah unsharp mask, yang difungsikan untuk mempertegas detail dan tepi citra dengan mengurangi komponen halus dan meningkatkan kontras. Setelah penerapan filter, gambar yang telah ditajamkan ditampilkan berdampingan

dengan gambar grayscale asli. Perbandingan ini memperlihatkan peningkatan ketajaman pada citra yang mempermudah deteksi fitur atau objek yang lebih jelas dalam gambar. Teknik ini berguna untuk memperbaiki ketajaman visual atau menonjolkan detail dalam citra.

6. Smoothing with box filter revisited

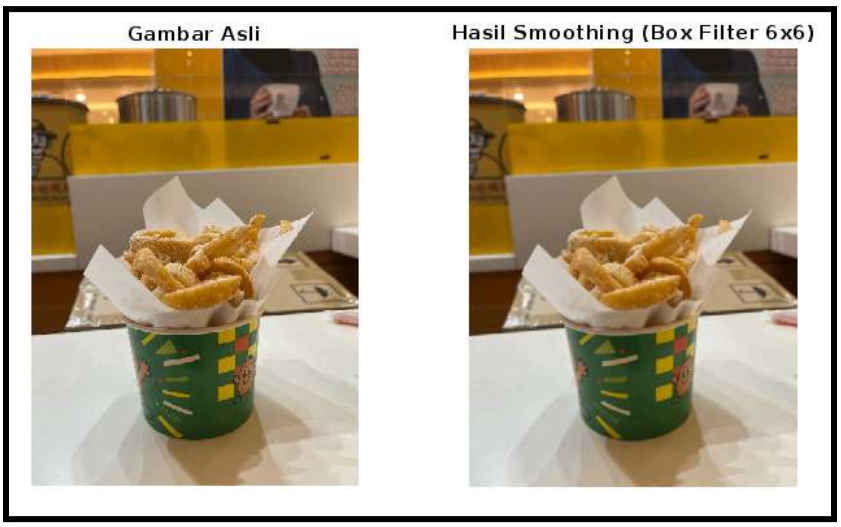
Input :

SmoothingWithFilter.m

/MATLAB Drive/SmoothingWithFilter.m

```
% Membaca dan mengubah gambar ke tipe double
gambar = imread('Potato.jpg');
gambar = im2double(gambar);

% Membuat box filter 6x6
filter_box = ones(6,6) / (6*6);

% Siapkan array untuk menyimpan hasil smoothing
hasil_smooth = zeros(size(gambar));

% Terapkan smoothing ke masing-masing channel warna
for channel = 1:3
    hasil_smooth(:,:,channel) = imfilter(gambar(:,:,channel), filter_box, 'replicate');
end

% Menampilkan gambar asli dan hasil smoothing
figure;
subplot(1,2,1);
imshow(gambar);
title('Gambar Asli');

subplot(1,2,2);
imshow(hasil_smooth);
title('Hasil Smoothing (Box Filter 6x6)');

```

Output :



Analisis : Dari hasil percobaan di atas, dengan menerapkan efek smoothing pada gambar “Potato.jpg” menggunakan box filter berukuran 6x6, yang berfungsi untuk meratakan intensitas piksel di sekitar tiap piksel pusat, menghasilkan efek penghalusan yang mengurangi detail halus dan noise. Proses smoothing dilakukan untuk masing-masing kanal warna (RGB) secara terpisah menggunakan fungsi imfilter dengan metode 'replicate' untuk menangani tepi gambar. Hasilnya adalah gambar yang tampak lebih halus dibandingkan gambar asli, dan perbandingan keduanya ditampilkan berdampingan, memperlihatkan efek smoothing yang lebih jelas pada citra dengan ukuran filter yang cukup besar. Teknik ini berguna untuk mengurangi noise atau mempersiapkan gambar untuk pemrosesan lebih lanjut.

7. Image Transformation

a. Add Pixel

Input :

AddPixel.m

/MATLAB Drive/AddPixel.m

```
% Membaca gambar
potato = imread('Potato.jpg');

% Menambahkan nilai 20 ke setiap piksel
potato_plus20 = uint8(double(potato) + 20);

% Menampilkan hasil
imshow(potato_plus20);
title('Potato + 20 Pixel Value');

```

Output :

Analisis : Dari hasil percobaan di atas, menambahkan nilai konstan (20) ke setiap piksel dalam gambar "Potato.jpg", Proses ini dilakukan dengan mengonversi gambar ke tipe double terlebih dahulu agar dapat menambahkan nilai lebih dari rentang piksel aslinya (0-255) secara aman, kemudian mengembalikannya ke tipe uint8 setelah penambahan. Hasilnya adalah gambar yang lebih terang, karena setiap piksel mengalami peningkatan nilai intensitas sebesar 20. Meskipun demikian, jika penambahan melebihi nilai maksimum (255), itu akan dibatasi oleh batasan tipe data uint8. Hasilnya ditampilkan untuk menunjukkan perubahan visual pada gambar akibat penambahan tersebut.

b. Rotation

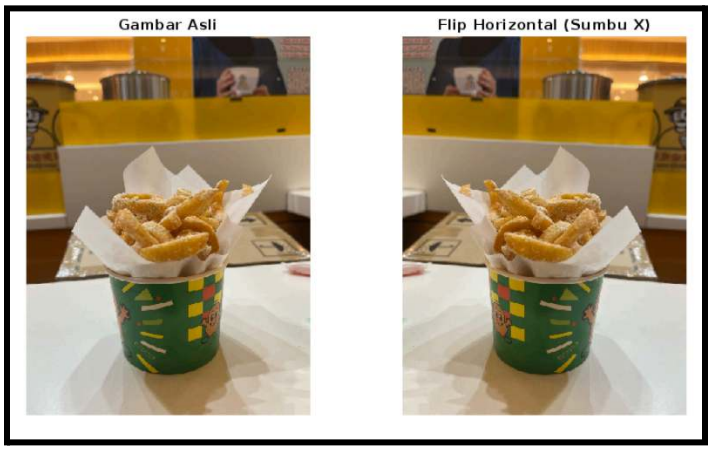
Input :

Rotation.m

/MATLAB Drive/Rotation.m

```
% Membaca gambar
potato = imread('Potato.jpg');

% Melakukan rotasi horizontal (flip sumbu X)
Xrotation = flip(potato, 2);

% Menampilkan gambar asli dan hasil rotasi
figure;
subplot(1,2,1);
imshow(potato);
title('Gambar Asli');

subplot(1,2,2);
imshow(Xrotation);
title('Flip Horizontal (Sumbu X)');

```

Output :



Analisis : Dari hasil percobaan di atas, melakukan rotasi horizontal pada gambar "Potato.jpg" dengan menggunakan fungsi flip pada sumbu X. Hasilnya adalah gambar yang dibalik secara horizontal, sehingga objek dalam gambar tampak terbalik kiri-kanannya. Proses ini tidak mengubah ukuran atau orientasi vertikal gambar, hanya menghasilkan cermin horizontal. Gambar asli dan hasil flip horizontal ditampilkan berdampingan, memungkinkan pengguna untuk melihat perbedaan secara jelas antara keduanya. Teknik ini sering digunakan untuk manipulasi citra dasar atau augmentasi data.

II. Computer Vision (CS5670)

1. Edge Detection

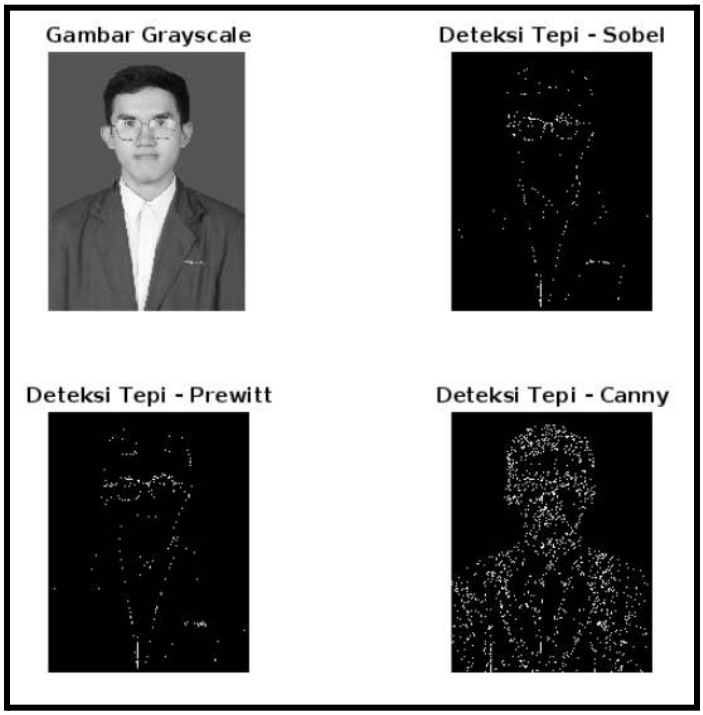
Input :

EdgeDetection.m ×

/MATLAB Drive/Semester_8/ComputerVision/EdgeDetection.m

```
% Membaca gambar dan ubah ke grayscale
foto = imread('ProfilePicture.jpg');
abu_abu = rgb2gray(foto);  % atau gunakan im2gray jika MATLAB versi terbaru

% Terapkan tiga metode edge detection
tepi_sobel = edge(abu_abu, 'sobel');
tepi_prewitt = edge(abu_abu, 'prewitt');
tepi_canny = edge(abu_abu, 'canny');

% Tampilkan hasil perbandingan ke dalam satu figure
figure('Name', 'Perbandingan Deteksi Tepi - Sobel, Prewitt, Canny');

subplot(2,2,1);
imshow(abu_abu);
title('Gambar Grayscale');

subplot(2,2,2);
imshow(tepi_sobel);
title('Deteksi Tepi - Sobel');

subplot(2,2,3);
imshow(tepi_prewitt);
title('Deteksi Tepi - Prewitt');

subplot(2,2,4);
imshow(tepi_canny);
title('Deteksi Tepi - Canny');

```

Output :



Analisis : Dari hasil percobaan di atas, menerapkan tiga metode deteksi tepi yang berbeda pada gambar "ProfilePicture.jpg" yang telah dikonversi ke grayscale. Ketiga metode yang digunakan adalah Sobel, Prewitt, dan Canny, yang semuanya berfokus pada identifikasi perubahan intensitas yang tajam di dalam citra, yang menunjukkan tepi objek. Hasil deteksi tepi untuk masing-masing metode ditampilkan dalam

satu figure dengan empat subplot, membandingkan gambar grayscale asli dengan hasil deteksi tepi dari setiap metode. Perbandingan ini memberikan gambaran yang jelas tentang bagaimana setiap metode menangani dan menonjolkan tepi dalam citra, dengan Canny biasanya memberikan hasil yang lebih halus dan lebih terperinci dibandingkan Sobel dan Prewitt.

2. Characterizing Edges

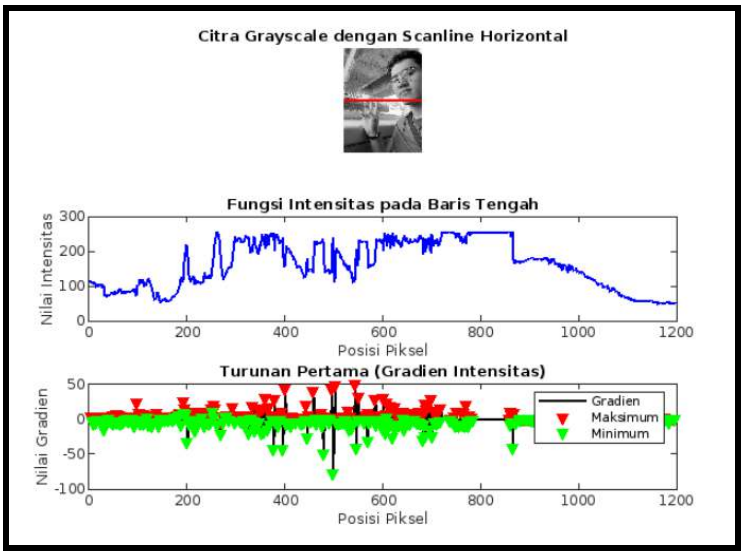
Input :

CharacterizingEdges.m

/MATLAB Drive/Semester_8/ComputerVision/CharacterizingEdges.m

```m
% Membaca dan mengonversi gambar ke grayscale
gambar = imread('Selfie.jpg');
abu_abu = rgb2gray(gambar);

% Ambil baris tengah dari gambar
baris_tengah = round(size(abu_abu, 1) / 2);
nilai_intensitas = double(abu_abu(baris_tengah, :));

% Hitung turunan pertama (gradien)
gradien = diff(nilai_intensitas);

% Tampilkan hasil dalam figure
figure;

% Tampilan gambar dengan garis horizontal
subplot(3,1,1);
imshow(abu_abu);
hold on;
line([1 size(abu_abu,2)], [baris_tengah baris_tengah], 'Color', 'r', 'LineWidth', 1);
title('Citra Grayscale dengan Scanline Horizontal');

% Grafik fungsi intensitas
subplot(3,1,2);
plot(nilai_intensitas, 'b', 'LineWidth', 1.5);
title('Fungsi Intensitas pada Baris Tengah');
xlabel('Posisi Piksel');
ylabel('Nilai Intensitas');

% Grafik turunan pertama (gradien)
subplot(3,1,3);
plot(gradien, 'k', 'LineWidth', 1.5);
hold on;
[~, puncak] = findpeaks(gradien);
[~, lembah] = findpeaks(-gradien);
plot(puncak, gradien(puncak), 'rv', 'MarkerFaceColor', 'r');
plot(lembah, gradien(lembah), 'gv', 'MarkerFaceColor', 'g');
title('Turunan Pertama (Gradien Intensitas)');
xlabel('Posisi Piksel');
ylabel('Nilai Gradien');
legend('Gradien', 'Maksimum', 'Minimum');

```

Output :



Analisis : Dari hasil percobaan di atas, terjadi perubahan intensitas piksel pada baris tengah gambar "Selfie.jpg" yang telah dikonversi ke grayscale. Pertama, gambar dibaca, dan baris tengahnya diambil untuk dianalisis. Kemudian, nilai intensitas pada baris tersebut dihitung dan turunan pertama (gradien) dari intensitas tersebut diekstrak untuk mendeteksi perubahan tajam, yang menunjukkan tepi atau perubahan signifikan dalam citra. Hasilnya ditampilkan dalam tiga grafik: gambar dengan garis horizontal yang menunjukkan posisi baris tengah, grafik intensitas pada baris tengah, dan grafik gradien yang menunjukkan perubahan intensitas. Pada grafik gradien, puncak dan lembah diidentifikasi untuk menandakan posisi tepi atau peralihan tajam. Teknik ini digunakan untuk menganalisis tepi dan struktur dalam citra berdasarkan perubahan intensitas.

3. Image Derivatives

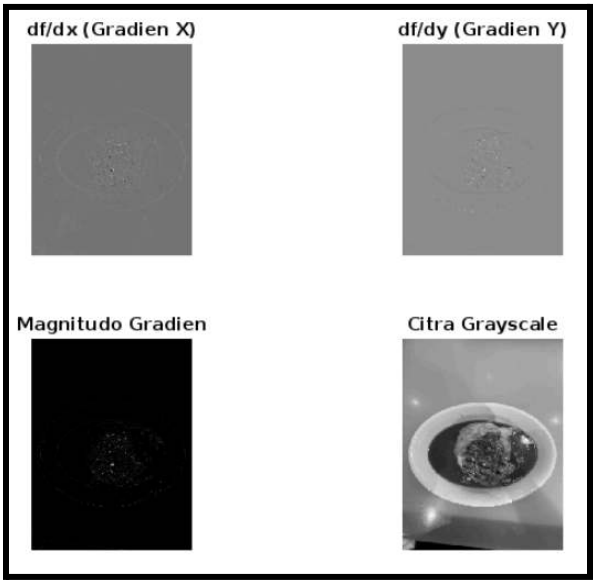
Input :

ImageDerivatives.m

/MATLAB Drive/Semester_8/ComputerVision/ImageDerivatives.m

```
% Membaca gambar dan mengubah ke grayscale
gambar = imread('ChickenCurry.jpg');
abu_abu = rgb2gray(gambar);
abu_abu = im2double(abu_abu);

% Kernel untuk arah horizontal dan vertikal
kernelX = [-1 1];
kernelY = [-1; 1];

% Konvolusi untuk arah x dan y
grad_x = conv2(abu_abu, kernelX, 'same');
grad_y = conv2(abu_abu, kernelY, 'same');

% Menghitung besar gradien
magnitude = sqrt(grad_x.^2 + grad_y.^2);

% Tampilkan hasil dalam 4 subplot
figure;

subplot(2,2,1);
imshow(grad_x, []);
title('df/dx (Gradien X)');

subplot(2,2,2);
imshow(grad_y, []);
title('df/dy (Gradien Y)');

subplot(2,2,3);
imshow(magnitude, []);
title('Magnitudo Gradien');

subplot(2,2,4);
imshow(abu_abu, []);
title('Citra Grayscale');

```

Output :



Analisis : Dari hasil percobaan di atas, menerapkan teknik deteksi gradien pada gambar "ChickenCurry.jpg" yang telah dikonversi ke grayscale dan dinormalisasi menjadi tipe double. Dua kernel konvolusi, satu untuk arah horizontal (gradien X) dan satu lagi untuk arah vertikal (gradien Y), digunakan

untuk menghitung perubahan intensitas citra dalam kedua arah tersebut. Hasil konvolusi ini menunjukkan perubahan intensitas pada sumbu X dan Y, yang kemudian digabungkan untuk menghitung magnitudo gradien, yang menggambarkan kekuatan perubahan intensitas secara keseluruhan. Hasilnya ditampilkan dalam empat subplot: gradien dalam arah X, Y, magnitudo gradien, dan gambar grayscale asli. Teknik ini digunakan untuk mendeteksi tepi dan perubahan struktur dalam citra, yang merupakan langkah awal dalam banyak algoritma pemrosesan citra, seperti deteksi tepi atau analisis bentuk.

4. Image Gradient

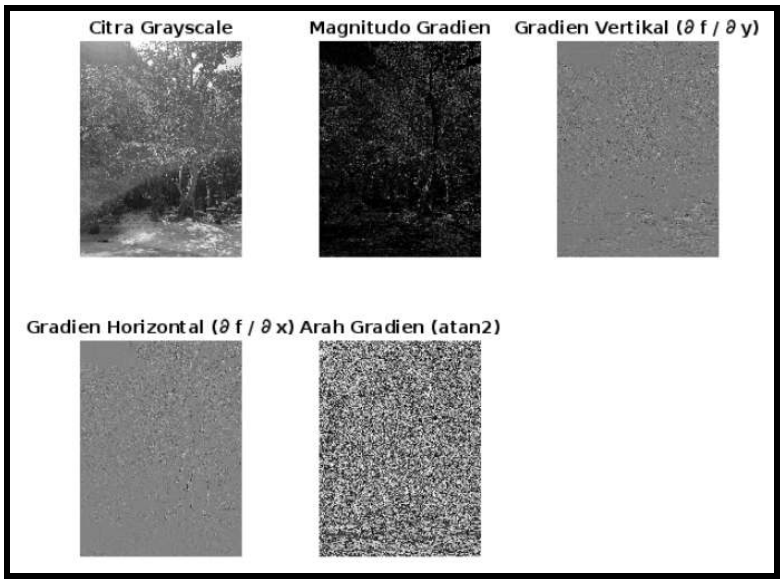
Input :

ImageGradient.m

/MATLAB Drive/Semester_8/ComputerVision/ImageGradient.m

```
% Membaca gambar dan ubah ke grayscale
gambar = imread('Tree.jpg');
abu = rgb2gray(gambar);

% Konversi ke tipe double untuk pemrosesan
abu_double = double(abu);

% Filter Sobel untuk arah x dan y
gradX = imfilter(abu_double, fspecial('sobel')', 'replicate'); % Transpose untuk arah X
gradY = imfilter(abu_double, fspecial('sobel'), 'replicate');  % Normal untuk arah Y

% Hitung magnitudo dan arah gradien
besar_gradien = sqrt(gradX.^2 + gradY.^2);
arah_gradien = atan2(gradY, gradX);

% Tampilkan hasil ke dalam subplot
figure;

subplot(2,3,1);
imshow(abu);
title('Citra Grayscale');

subplot(2,3,2);
imshow(besar_gradien, []);
title('Magnitudo Gradien');

subplot(2,3,3);
imshow(gradY, []);
title('Gradien Vertikal (\partial f / \partial y)');

subplot(2,3,4);
imshow(gradX, []);
title('Gradien Horizontal (\partial f / \partial x)');

subplot(2,3,5);
imshow(arah_gradien, []);
title('Arah Gradien (atan2)');

```

Output :



Analisis : Dari hasil percobaan di atas, menerapkan deteksi gradien menggunakan filter Sobel pada gambar "Tree.jpg" yang telah dikonversi ke grayscale dan tipe *double*. Dua arah gradien dihitung: horizontal (X) dan vertikal (Y) dengan menggunakan filter Sobel yang diterapkan terpisah pada kedua arah tersebut. Kemudian, magnitudo gradien dihitung sebagai akar kuadrat dari jumlah kuadrat kedua gradien, yang menunjukkan kekuatan perubahan intensitas pada setiap piksel. Arah gradien dihitung menggunakan fungsi *atan2*, yang memberikan informasi tentang orientasi perubahan intensitas. Hasilnya ditampilkan dalam lima subplot, yang memperlihatkan citra grayscale asli, magnitudo gradien, gradien dalam arah X dan Y, serta arah gradien. Teknik ini berguna dalam analisis citra untuk mendeteksi tepi dan orientasi struktur dalam gambar.

5. Smoothing (Gaussian Filter)

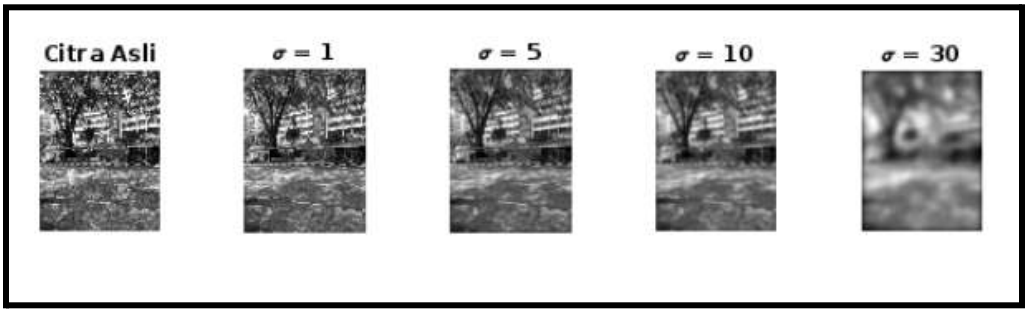
Input :

Smoothing.m

/MATLAB Drive/Semester_8/ComputerVision/Smoothing.m

```
% Baca citra dan konversi ke grayscale jika berwarna
gambar = imread('Holiday.jpg');
if ndims(gambar) == 3
    gambar = rgb2gray(gambar);
end

% Daftar nilai sigma untuk Gaussian filter
daftar_sigma = [1, 5, 10, 30];

% Tampilkan gambar asli dan hasil smoothing
figure;
subplot(1, numel(daftar_sigma)+1, 1);
imshow(gambar, []);
title('Citra Asli');

% Proses Gaussian smoothing untuk setiap nilai sigma
for idx = 1:numel(daftar_sigma)
    s = daftar_sigma(idx);
    ukuran_kernel = 2 * ceil(3 * s) + 1;
    filter_gauss = fspecial('gaussian', ukuran_kernel, s);
    hasil_smooth = imfilter(gambar, filter_gauss, 'same');

    subplot(1, numel(daftar_sigma)+1, idx+1);
    imshow(hasil_smooth, []);
    title(['\sigma = ', num2str(s)]);
end

```

Output :



Analisis : Dari hasil percobaan di atas, melakukan pemrosesan Gaussian smoothing pada gambar "Holiday.jpg" dengan menggunakan beberapa nilai sigma untuk filter Gaussian. Gambar pertama kali dikonversi ke grayscale jika berupa gambar berwarna. Selanjutnya, untuk setiap nilai sigma yang ditentukan dalam daftar ([1, 5, 10, 30]), kernel Gaussian dihitung dengan ukuran yang sesuai, dan gambar disaring menggunakan filter Gaussian tersebut. Hasilnya ditampilkan dalam satu figure dengan beberapa subplot: gambar asli diikuti oleh hasil smoothing untuk masing-masing nilai sigma. Teknik ini berguna untuk menghaluskan gambar dan mengurangi noise, dengan efek smoothing yang lebih kuat pada nilai sigma yang lebih besar.

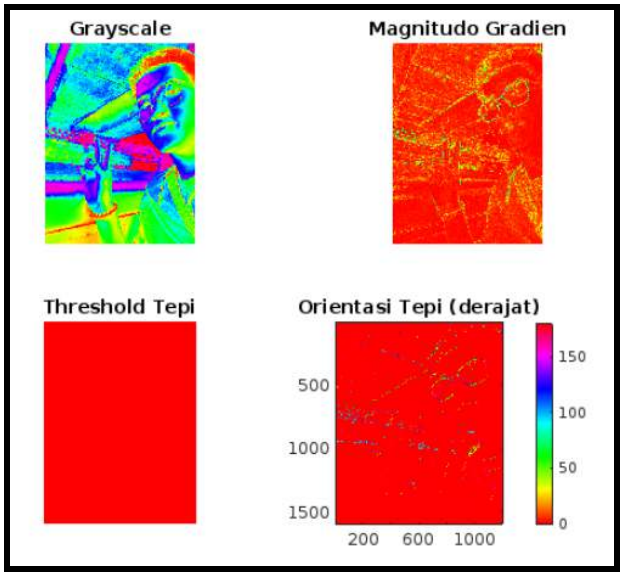
6. Derivative Of Gaussian

Input :

DerivativeOfGaussian.m

```
% Load image dan konversi ke grayscale
img = imread('Selfie.jpg');
gray_img = rgb2gray(img);

% Konversi ke tipe double
gray_dbl = im2double(gray_img);

% Kernel Sobel X dan Y
sobel_x = fspecial('sobel')';
sobel_y = fspecial('sobel');

% Hitung turunan arah X dan Y
Ix = imfilter(gray_dbl, sobel_x, 'replicate');
Iy = imfilter(gray_dbl, sobel_y, 'replicate');

% Hitung besaran gradien dan normalisasi
grad_magnitude = sqrt(Ix.^2 + Iy.^2);
grad_norm = mat2gray(grad_magnitude);  % Normalisasi ke [0,1]

% Deteksi tepi berdasarkan threshold
threshold_val = 0.2;
binary_edges = grad_norm > threshold_val;

% Hitung arah orientasi gradien
orientasi = atan2(Iy, Ix);
orientasi_deg = mod(rad2deg(orientasi), 180);  % Batasi antara 0-179 derajat

% Tampilkan hasil
figure('Name', 'Perbandingan Hasil Deteksi Tepi');

subplot(2,2,1);
imshow(gray_img);
title('Grayscale');

subplot(2,2,2);
imshow(grad_norm);
title('Magnitudo Gradien');

subplot(2,2,3);
imshow(binary_edges);
title('Threshold Tepi');

subplot(2,2,4);
orientasi_only_edges = NaN(size(orientasi_deg));
orientasi_only_edges(binary_edges) = orientasi_deg(binary_edges);
imagesc(orientasi_only_edges);
colormap(hsv);
colorbar;
title('Orientasi Tepi (derajat)');
```

Output :



Analisis : Dari hasil percobaan di atas, melakukan deteksi tepi pada gambar "Selfie.jpg" menggunakan operator Sobel untuk menghitung turunan gradien pada arah horizontal (X) dan vertikal (Y). Gambar dikonversi ke grayscale dan kemudian ke tipe *double* untuk pemrosesan lebih lanjut. Dari turunan gradien, besaran gradien dihitung dan dinormalisasi ke rentang [0, 1]. Deteksi tepi dilakukan dengan menetapkan threshold pada besaran gradien, menghasilkan citra biner yang menunjukkan lokasi tepi. Selain itu, arah orientasi gradien dihitung menggunakan *atan2* dan dikonversi ke derajat dengan batas antara 0-179 derajat. Hasilnya ditampilkan dalam empat subplot: gambar grayscale asli, magnitudo gradien, hasil deteksi tepi setelah thresholding, dan orientasi tepi dalam bentuk peta warna. Teknik ini berguna untuk ekstraksi fitur tepi dalam citra, yang dapat digunakan dalam berbagai aplikasi pengolahan citra seperti pengenalan bentuk dan deteksi objek.

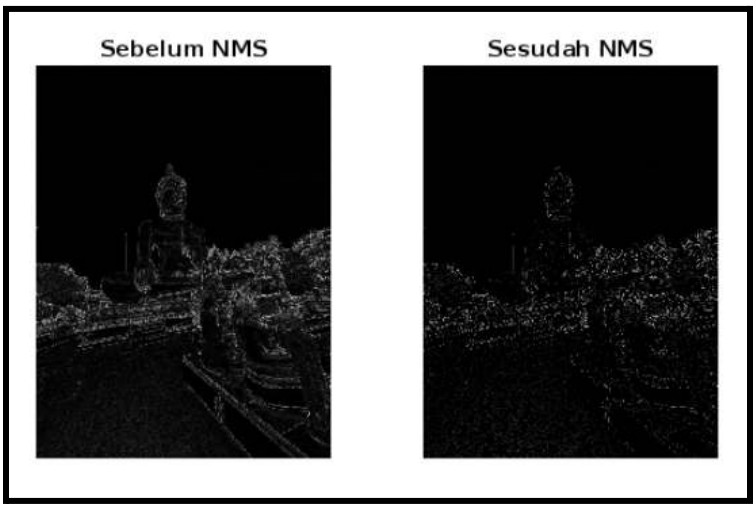
7. Non-Maximum Supression

Input :

NonMaximumSupression.m

```
% Hitung gradien dan arah
grad_x = imfilter(gray_img, sobel_x, 'replicate');
grad_y = imfilter(gray_img, sobel_y, 'replicate');
magnitude = sqrt(grad_x.^2 + grad_y.^2);
angle = atan2(grad_y, grad_x) * (180 / pi);
angle(angle < 0) = angle(angle < 0) + 180;

% Inisialisasi hasil Non-Maximum Suppression
[nRows, nCols] = size(gray_img);
nms_result = zeros(nRows, nCols);

% Lakukan NMS (Non-Maximum Suppression)
for i = 2 : nRows-1
    for j = 2 : nCols-1
        direction = angle(i, j);
        curr_mag = magnitude(i, j);

        if ((direction >= 0 && direction < 22.5) || (direction >= 157.5 && direction <= 180))
            prev = magnitude(i, j-1);
            next = magnitude(i, j+1);
        elseif (direction >= 22.5 && direction < 67.5)
            prev = magnitude(i-1, j+1);
            next = magnitude(i+1, j-1);
        elseif (direction >= 67.5 && direction < 112.5)
            prev = magnitude(i-1, j);
            next = magnitude(i+1, j);
        elseif (direction >= 112.5 && direction < 157.5)
            prev = magnitude(i-1, j-1);
            next = magnitude(i+1, j+1);
        end

        if (curr_mag >= prev) && (curr_mag >= next)
            nms_result(i, j) = curr_mag;
        else
            nms_result(i, j) = 0;
        end
    end
end

% Tampilkan hasil
figure('Name', 'Hasil Deteksi Tepi: Sebelum & Sesudah NMS');
subplot(1,2,1);
imshow(magnitude, []);
title('Sebelum NMS');

subplot(1,2,2);
imshow(nms_result, []);
title('Sesudah NMS');
```

Output :



Analisis : Pada percobaan di atas, menerapkan deteksi tepi menggunakan operator Sobel untuk menghitung gradien arah horizontal (X) dan vertikal (Y) pada gambar "Monument.jpg" yang telah dikonversi ke grayscale dan diubah menjadi tipe *double*. Selanjutnya, besaran gradien dihitung dengan

menggabungkan gradien X dan Y, dan arah gradien dihitung dalam derajat. Nilai-nilai negatif pada arah gradien diperbaiki untuk berada dalam rentang [0, 180] derajat.

Proses utama dalam kode ini adalah penerapan teknik *Non-Maximum Suppression* (NMS), yang bertujuan untuk menghaluskan hasil deteksi tepi dengan menghilangkan nilai gradien yang tidak maksimal di sepanjang arah gradien. NMS dilakukan dengan memeriksa tiga nilai dalam arah gradien tertentu: nilai saat ini dan dua nilai tetangganya (sebelumnya dan sesudahnya). Jika nilai saat ini lebih besar dari dua nilai tetangganya, maka nilai tersebut dipertahankan; jika tidak, diatur menjadi 0.

Hasil deteksi tepi sebelum dan sesudah NMS ditampilkan dalam dua subplot untuk membandingkan efek dari NMS yang membuat tepi menjadi lebih jelas dan terdefinisi. Teknik ini efektif dalam memperbaiki hasil deteksi tepi dan mengurangi noise yang dapat mengganggu analisis lebih lanjut.

8. Hysteresis

Input :

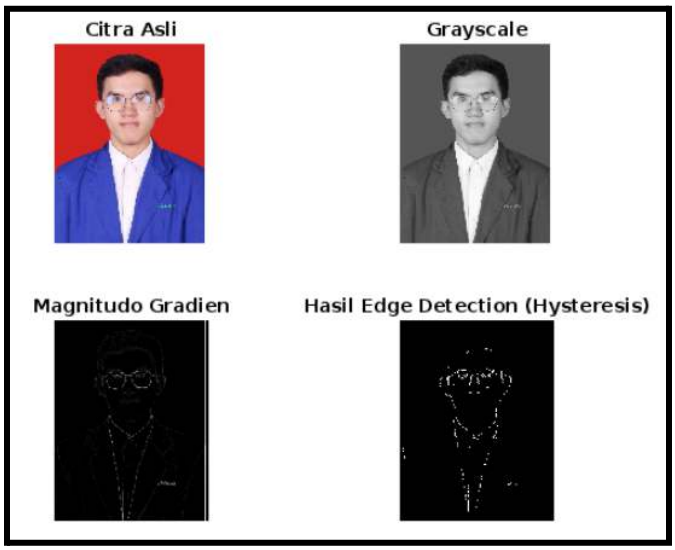
Hysteresis.m

/MATLAB Drive/Semester_8/ComputerVision/Hysteresis.m

```
% Baca gambar dan konversi ke grayscale
gambar = imread('ProfilePicture.jpg');
citra_gray = rgb2gray(gambar);

% Definisikan filter Sobel untuk arah X dan Y
filter_x = fspecial('sobel');
filter_y = filter_x';

% Hitung gradien arah X dan Y
grad_x = imfilter(double(citra_gray), filter_x, 'replicate');
grad_y = imfilter(double(citra_gray), filter_y, 'replicate');

% Hitung magnitudo gradien dan normalisasi
mag_gradien = sqrt(grad_x.^2 + grad_y.^2);
mag_gradien = mag_gradien / max(mag_gradien(:));

% Threshold bawah dan atas
batas_bawah = 0.1;
batas_atas = 0.3;

% Identifikasi tepi kuat dan lemah
tepi_kuat = mag_gradien > batas_atas;
tepi_lemah = (mag_gradien >= batas_bawah) & (mag_gradien <= batas_atas);

% Label komponen terhubung dari tepi lemah
label_tepi = bwlabel(tepi_lemah, 8);
info_area = regionprops(label_tepi, 'PixelIdxList');

% Inisialisasi citra hasil
hasil_edge = false(size(citra_gray));

```

```
% Hubungkan tepi lemah dengan tepi kuat jika berdekatan
for idx = 1:length(info_area)
    indeks_pixel = info_area(idx).PixelIdxList;
    [baris, kolom] = ind2sub(size(citra_gray), indeks_pixel);
    ditemukan = false;

    for k = 1:length(indeks_pixel)
        r = baris(k); c = kolom(k);
        baris_min = max(1, r - 1); baris_max = min(size(citra_gray,1), r + 1);
        kolom_min = max(1, c - 1); kolom_max = min(size(citra_gray,2), c + 1);

        % Periksa apakah area sekitarnya terdapat tepi kuat
        area_cek = tepi_kuat(baris_min:baris_max, kolom_min:kolom_max);
        if any(area_cek(:))
            ditemukan = true;
            break;
        end
    end

    % Tandai area jika terhubung dengan tepi kuat
    if ditemukan
        hasil_edge(indeks_pixel) = true;
    end
end

% Tampilkan hasil
figure('Name', 'Deteksi Tepi dengan Hysteresis Thresholding');

subplot(2,2,1);
imshow(gambar);
title('Citra Asli');

subplot(2,2,2);
imshow(citra_gray);
title('Grayscale');

subplot(2,2,3);
imshow(mag_gradien, []);
title('Magnitudo Gradien');

subplot(2,2,4);
imshow(hasil_edge);
title('Hasil Edge Detection (Hysteresis)');
```

Output :



Analisis : Pada percobaan di atas, diterapkan teknik deteksi tepi menggunakan filter Sobel untuk menghitung gradien arah horizontal dan vertikal pada gambar "ProfilePicture.jpg" yang telah dikonversi ke grayscale. Gradien dihitung dengan menggunakan filter Sobel untuk mendeteksi perubahan intensitas gambar, yang kemudian diubah menjadi magnitudo gradien dan dinormalisasi.

Tahapan selanjutnya adalah melakukan thresholding untuk mengidentifikasi tepi yang kuat dan lemah, dengan batasan bawah dan atas yang ditetapkan pada nilai 0.1 dan 0.3, berturut-turut. Tepi yang kuat

langsung ditandai sebagai bagian dari tepi, sementara tepi yang lemah diidentifikasi sebagai kandidat yang mungkin terhubung ke tepi kuat.

Proses "hysteresis" digunakan untuk menghubungkan tepi lemah dengan tepi kuat. Jika piksel tepi lemah berdekatan dengan piksel tepi kuat, maka piksel tersebut akan dianggap sebagai bagian dari tepi, sehingga memperjelas tepi yang terdeteksi dan mengurangi noise.

Hasil akhirnya menunjukkan citra asli, citra grayscale, magnitudo gradien, dan hasil deteksi tepi menggunakan thresholding hysteresis. Teknik ini efektif untuk mendeteksi tepi yang lebih jelas dan mengurangi pengaruh noise dalam gambar, memastikan bahwa hanya tepi yang signifikan yang terdeteksi.

9. Canny Edge Detector

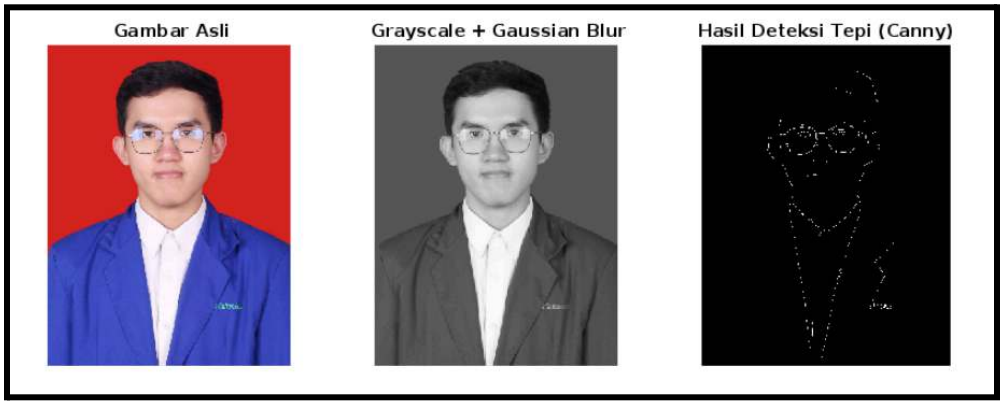
Input :

CannyEdgeDetector.m

/MATLAB Drive/Semester_8/ComputerVision/CannyEdgeDetector.m

```
% Membaca gambar dan mengubah ke grayscale
gambar_asli = imread('ProfilePicture.jpg');
citra_gray = rgb2gray(gambar_asli);

% Terapkan Gaussian blur untuk mengurangi noise
ukuran_filter = [5 5];
sigma = 1.4;
gauss_filter = fspecial('gaussian', ukuran_filter, sigma);
citra_blur = imfilter(citra_gray, gauss_filter, 'replicate');

% Lakukan deteksi tepi menggunakan operator Canny
ambang = [0.1 0.3]; % Threshold bawah dan atas
tepi_canny = edge(citra_blur, 'canny', ambang);

% Tampilkan hasil dalam satu figure
figure('Name', 'Tahapan Deteksi Tepi Canny', 'NumberTitle', 'off');

subplot(1,3,1);
imshow(gambar_asli);
title('Gambar Asli');

subplot(1,3,2);
imshow(citra_blur);
title('Grayscale + Gaussian Blur');

subplot(1,3,3);
imshow(tepi_canny);
title('Hasil Deteksi Tepi (Canny)');

```

Output :



Analisis : Pada percobaan di atas, mengimplementasikan deteksi tepi menggunakan metode Canny dengan langkah-langkah pra-pemrosesan. Dimulai dengan mengonversi gambar berwarna menjadi grayscale menggunakan rgb2gray. Setelah itu, Gaussian blur diterapkan untuk mengurangi noise yang dapat mengganggu proses deteksi tepi, dengan filter berukuran 5x5 dan sigma 1.4. Proses blur ini bertujuan untuk menekan detail halus dan fokus pada tepi yang lebih signifikan.

Setelah citra diblur, operator Canny diterapkan untuk mendeteksi tepi dengan menggunakan dua ambang batas: 0.1 untuk ambang bawah dan 0.3 untuk ambang atas. Canny deteksi tepi adalah metode yang efisien untuk menemukan tepi yang tajam dan mengurangi pengaruh noise.

Hasil akhir ditampilkan dalam tiga bagian: gambar asli, citra yang telah diblur dan diubah menjadi grayscale, serta hasil deteksi tepi menggunakan algoritma Canny. Secara keseluruhan, metode ini menggabungkan pemrosesan pra-pemrosesan dan deteksi tepi untuk mendapatkan hasil yang lebih bersih dan jelas, terutama pada gambar dengan noise.

10. Scale Space

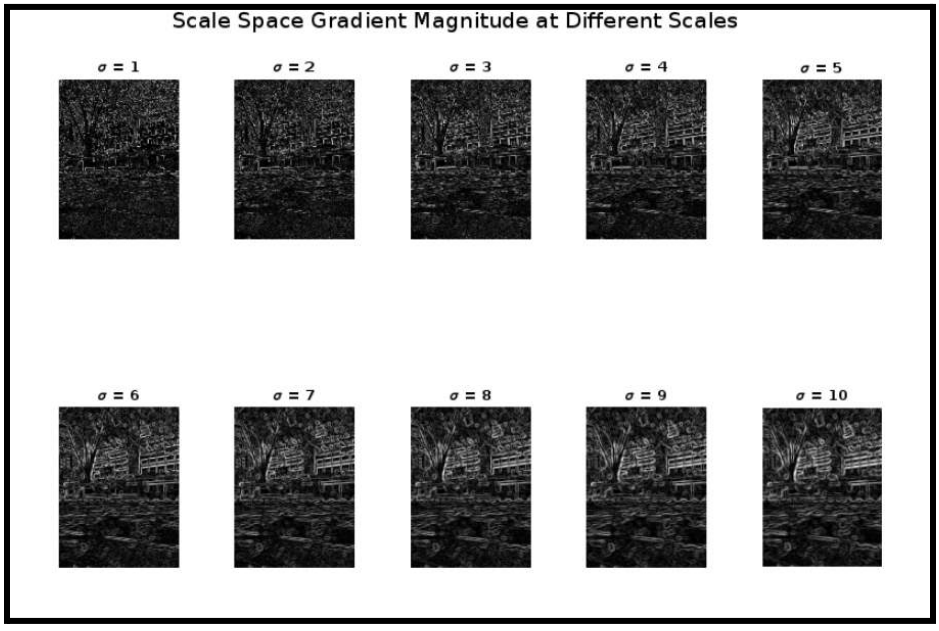
Input :

ScaleSpace.m

/MATLAB Drive/Semester_8/ComputerVision/ScaleSpace.m

```
% Membaca gambar dan mengonversi ke grayscale
gambar_asli = imread('Holiday.jpg');
if size(gambar_asli, 3) == 3
    gambar_gray = rgb2gray(gambar_asli);
else
    gambar_gray = gambar_asli; % Jika sudah grayscale, gunakan gambar asli
end

% Mengubah ke tipe double
gambar_double = im2double(gambar_gray);

% Mendefinisikan skala sigma
scales = linspace(1, 10, 10);

% Menampilkan gambar dengan berbagai skala
figure('Name', 'Scale Space', 'NumberTitle', 'off');
colormap gray;

% Loop untuk menghitung gradien pada berbagai skala
for i = 1:length(scales)
    sigma = scales(i);

    % Terapkan Gaussian filter
    gambar_smoothed = imgaussfilt(gambar_double, sigma);

    % Hitung gradien pada arah x dan y
    [Gx, Gy] = imgradientxy(gambar_smoothed);

    % Hitung magnitude gradien
    magnitude = sqrt(Gx.^2 + Gy.^2);

    % Menampilkan hasil untuk setiap skala
    subplot(2, ceil(length(scales)/2), i);
    imagesc(magnitude);
    axis off;
    axis image;
    title(['\sigma = ', num2str(sigma)]);
end

% Menambahkan judul utama
sgtitle('Scale Space Gradient Magnitude at Different Scales');
```

Output :



Analisis : Dari percobaan di atas, mengimplementasikan konsep scale space untuk memvisualisasikan perubahan magnitudo gradien pada gambar dengan berbagai skala Gaussian. Pertama, gambar diubah menjadi grayscale jika diperlukan, lalu dikonversi menjadi tipe double untuk pemrosesan lebih lanjut. Kemudian, beberapa nilai sigma ditetapkan untuk filter Gaussian yang akan digunakan pada gambar.

Proses utama dilakukan dalam sebuah loop di mana filter Gaussian dengan sigma yang bervariasi diterapkan pada gambar. Setiap gambar yang telah diblur dihitung gradiennya pada arah x dan y menggunakan fungsi imgradientxy. Magnitudo gradien dihitung dengan rumus Pythagoras dari gradien x dan y, yang memberikan gambaran sejauh mana intensitas gambar berubah di setiap titik.

Hasil dari setiap skala ditampilkan dalam subplot dengan setiap gambar magnitudo gradien pada skala tertentu. Hal ini menunjukkan bagaimana fitur dalam gambar menjadi lebih atau kurang jelas tergantung pada skala blurring yang digunakan. Secara keseluruhan, kode ini memungkinkan untuk mengamati bagaimana deteksi tepi dan fitur pada gambar berubah seiring dengan variasi skala.