画像処理１(E1)　後半授業のプログラミング課題

氏名：園山佳典
学籍番号：26002201991

1. はじめに

本課題は以下のとおりである。

Canny エッジ検出アルゴリズムを MATLAB 言語で実装せよ.

【課題①】 画像の平滑化(ガウシアンフィルタ)をせよ

【課題②】 画像のエッジ検出(Sobel フィルタ)をせよ

【課題③】画像のエッジ検出(Canny フィルタ)を設計し勾配画像を作成せよ

【課題④】Canny フィルタの出力勾配画像における非極大値を抑制せよ

【課題⑤】 Canny フィルタの出力勾配画像における勾配極大値に対するヒステリシス閾値処理をせよ

処理内容(実装工夫した点を含む)及び実行結果を記載すること

• MATLAB 言語のプログラム実装部分をレポート(.pdf)に記載すること

• 自分で撮影した画像を用いること

• MATLAB のエッジ検出やフィルタリング関数を利用しない(imfiler, edge 等)こと

• MATLAB の画像読み込み、保存関数は利用してよい

2. 方法

【課題①】

与えられた画像ファイルを imread 関数を使用して読み込む。

読み込んだ画像を im2double 関数を使用して正規化し、RGB 形式の画像データに変換する。

平滑化には、ガウシアンカーネルという重み付き行列を使用する。

meshgrid 関数を使用して、カーネルの各要素に対応する座標値(X, Y)を生成する。次に、ガウシアン関数の定義を使用して、座標値(X, Y)からガウシアンカーネルの各要素を計算する。

作成したガウシアンカーネルをフィルタの合計が 1 になるように正規化する。

conv2 関数を使用してガウシアンカーネルとの畳み込みを行うことで画像の各ピクセルに対してカーネルを適用して平滑化できる。

平滑化後の画像データを保存するための行列(smoothed_img)を用意し、畳み込みを行う。

平滑化後の画像(smoothed_img)を imshow 関数を使用して表示する。

最後に平滑化後の画像を imwrite 関数を使用して保存する。

【課題②】

課題①で保存した平滑化済みの画像ファイル('smoothed_img.jpg')を imread 関数を使用して読み込む。読み込んだ画像を im2gray 関数を使用してグレースケール画像に変換する。

次にエッジ検出に使用する Sobel フィルタを定義する。Sobel フィルタは、勾配を計算するために使用するフィルタで、水平方向の勾配を計算するための filter_x と垂直方向の勾配を計算するための filter_y を定義する。

グレースケール画像に対して、conv2 関数を使用して Sobel フィルタとの畳み込みを行い、各ピクセルの水平方向の勾配を gradient_x、垂直方向の勾配を gradient_y に代入する。atan2d 関数を使用して、勾配の y 成分(gradient_y)と x 成分(gradient_x)からエッジ方向を計算する。そしてエッジ方向を、0 から 3 の整数値に量子化する。エッジ方向を pi を基準に 0 から 3 の範囲に変換し、四捨五入して整数化する。また、エッジ方向が負の値の場合は 4 を加算して 0 から 3 の範囲に変換する。量子化結果を edge_direction_quantized に代入する。最後に(edge_direction_quantized)を imshow 関数を使用して表示する。

【課題③】

与えられた画像を imread 関数を使用して読み込む。読み込んだ画像を rgb2gray 関数を使用してグレースケールに変換する。meshgrid 関数を使用して、カーネルの各要素に対応する座標値(X, Y)を生成する。ガウス関数を 2 次元 x 方向の一次微分したものを `kernel_x` に代入し 2 次元 y 方向の一次微分したものを `kernel_y` に代入する。その後、グレースケール画像と `kernel_x`、`kernel_y` をそれぞれ畳み込みし Gx、Gy に代入する。Gx、Gy から勾配強度を求め `filtered_image` に代入する。勾配強度の画像を正規化します。最後に imshow 関数を使用して表示する。

【課題④】

課題①で保存した平滑化済みの画像ファイル('smoothed_img.jpg')を imread 関数を使用して読み込む。読み込んだ画像を im2gray 関数を使用してグレースケール画像に変換する。ループを使用して、画像の各ピクセルに対して非極大値抑制を行う。各ピクセルの周囲の勾配方向と勾配強度を比較し、勾配方向上での極大値である場合にエッジの強度を残す。

ピクセル(i, j)の勾配方向を gradient_direction(i, j)、勾配強度を gradient_magnitude(i, j)とする。周囲のピクセルの勾配方向と勾配強度は、i+sign(sin(angle))、j+sign(cos(angle))および i-sign(sin(angle))、j-sign(cos(angle))の位置から取得する。

勾配強度が周囲のピクセルよりも大きい場合、ピクセル(i, j)のエッジ強度を edges(i, j)に保存する。最後に imshow 関数を使用して表示する。

【課題⑤】

課題①で保存した平滑化済みの画像ファイル('smoothed_img.jpg')を imread 関数を使用して読み込む。読み込んだ画像を im2gray 関数を使用してグレースケール画像に変換する。課題②で行った Sobel フィルタによる勾配画像の計算部分を使用し、gradient_x と gradient_y の勾配画像が得られる。ヒステリシス閾値処理のために、エッジの強度を閾値に基づいて処理する。高い閾値(Thhigh)と低い閾値(Thlow)を設定する。

gradient_magnitude の平均値を用いて高い閾値(Thhigh)と低い閾値(Thlow)を決定し、gradient_magnitude が高い閾値以上のピクセルをエッジ候補として edges に代入する。ループを使用して、低い閾値以上かつ繋がっているエッジ候補を探す。edges と gradient_magnitude の両方を満たすエッジ候補の座標を取得し、その周囲のピクセルに既にエッジが存在する場合にエッジとする。

これにより、エッジ候補が他のエッジと繋がっている場合にのみエッジとして残すことができる。この処理をエッジが収束するまで繰り返す。最後に imshow 関数を使用して表示する。

3 アルゴリズム

【課題①】

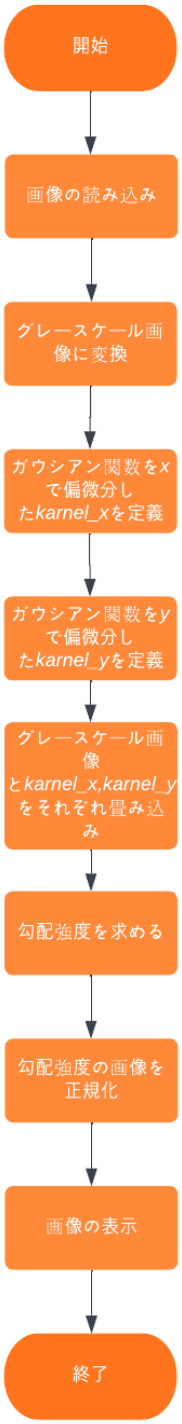
Flowchart 1

k s | July 14, 2023

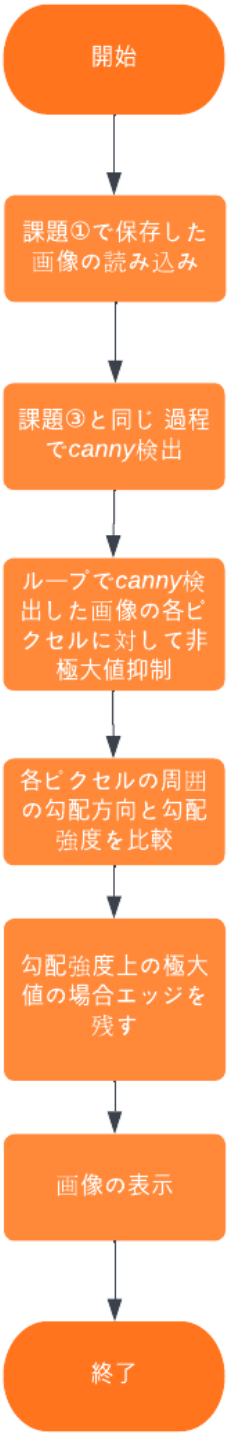
【課題②】


開始
課題①で保存した画像の読み込み
グレースケール画像に変換
Sobelフィルタの定義
勾配方向の計算
エッジ方向の計算
エッジ方向の量子化
画像の表示
終了

【課題③】


開始
画像の読み込み
グレースケール画像に変換
ガウシアン関数をxで偏微分したkarnel_xを定義
ガウシアン関数をyで偏微分したkarnel_yを定義
グレースケール画像とkarnel_x,karnel_yをそれぞれ畳み込み
勾配強度を求める
勾配強度の画像を正規化
画像の表示
終了

【課題④】


開始
課題①で保存した画像の読み込み
課題③と同じ 過程でcanny検出
ループでcanny検出した画像の各ピクセルに対して非極大値抑制
各ピクセルの周囲の勾配方向と勾配強度を比較
勾配強度上の極大値の場合エッジを残す
画像の表示
終了

【課題⑤】

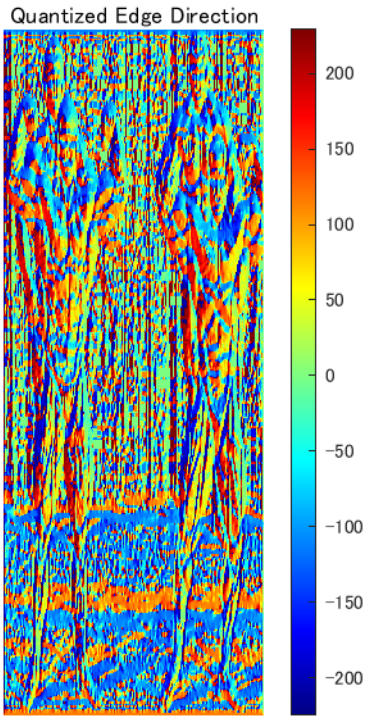

開始
課題①で保存した画像の読み込み
グレースケール画像に変換
課題②と同様にSobelフィルタを適用し勾配を計算
高い閾値(Thhigh)と低い閾値(Thlow)を設定する
勾配の大きさが高い閾値以上のピクセルをエッジ候補とする
画像の表示
終了

3. 結果

入力画像

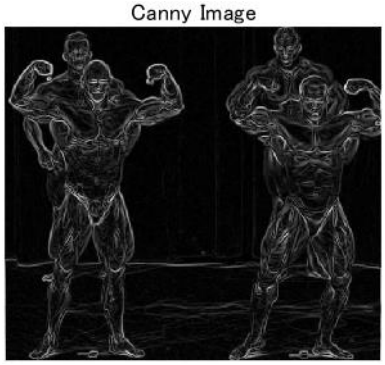
【課題①】出力画像

【課題②】出力画像



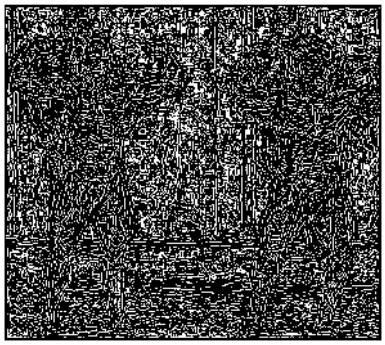
【課題③】出力画像

Canny Image



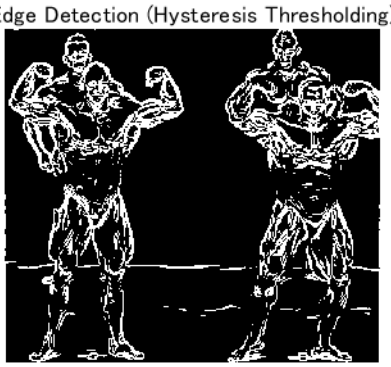
【課題④】出力画像

Gradient Maxima



【課題⑤】出力画像

Edge Detection (Hysteresis Thresholding)



4. 考察

【課題①】

ガウシアンフィルタは、画像のノイズ除去やエッジのぼかし効果など、平滑化ができるフィルタである。出力画像をズームしてみると入力画像と比べ滑らかであることが見て取れる。

【課題②】

Sobel フィルタは、エッジ検出に使用されるフィルタである。Sobel フィルタを適用することで、画像内のエッジを強調し物体の形状や輪郭を抽出できる。出力画像から見てわかる通りエッジが浮かび上がって検出されている。

【課題③】

canny フィルタも Sobel フィルタ同様にエッジ検出に使用されるフィルタである。
Canny フィルタは微分を用いてエッジ検出を行い、出力画像からもエッジが検出されていることがわかる。

【課題④】

非極大値抑制はエッジを細くする効果があり、エッジの鮮明さを向上させるために重要な手法である。勾配の方向を考えることで、エッジの方向に応じてエッジを細くすることができる。よって、画像内の細かな特徴や輪郭がより明らかになる。

【課題⑤】

ヒステリシス閾値処理はエッジ強度に基づいてエッジピクセルを選択し、高い閾値を超えるエッジピクセルにつながるエッジを残すことで、ノイズに対してエッジ検出結果が得られる。また、パラメータを変えて出力すると出力画像のエッジの検出のされ方が変わり、うまく調整することで望みのエッジ検出結果が得られるだろう。

5. 付録：プログラムリスト

【課題①】

```
img = imread('kr.jpg');
img_rgb = im2double(img);

% ガウシアンカーネルのサイズと標準偏差を設定
k_size = [5, 5];
sigma = 1.0;

% ガウシアンカーネルの作成
[X, Y] = meshgrid(-(k_size(2)-1)/2:(k_size(2)-1)/2, -(k_size(1)-
1)/2:(k_size(1)-1)/2);
kernel =  exp(-((X.^2 + Y.^2) / (2 * sigma^2)));
```

```
% カーネルを正規化
kernel = kernel / sum(kernel(:));

% 平滑化後の画像を作成
smoothed_img = zeros(size(img_rgb));
for c = 1:size(img_rgb, 3)
    smoothed_img(:, :, c) = conv2(img_rgb(:, :, c), kernel, 'same');
end

% 元の画像と平滑化後の画像を表示
figure;
subplot(1, 2, 1);
imshow(img_rgb);
title('Original Image');

subplot(1, 2, 2);
imshow(smoothed_img);
title('Smoothed Image');
imwrite(smoothed_img,"smoothed_img.jpg")
```

【課題②】

```
img = imread('smoothed_img.jpg');
img_gray = im2gray(img);

% Sobel フィルタの定義
filter_x = [-1 0 1; -2 0 2; -1 0 1];
filter_y = [-1 -2 -1; 0 0 0; 1 2 1];

% 勾配方向の計算
gradient_x = conv2(double(img_gray), filter_x, 'same');
gradient_y = conv2(double(img_gray), filter_y, 'same');

% エッジ方向の計算
edge_direction = atan2d(gradient_y, gradient_x);
```

```
% エッジ方向の量子化
edge_direction_quantized = round(edge_direction / pi * 4);
edge_direction_quantized(edge_direction_quantized < 0) =
edge_direction_quantized(edge_direction_quantized < 0) + 4;

img_gray = im2gray(img);
% 元の画像とエッジ方向の量子化結果を表示する
subplot(1, 2, 1);
imshow(img_gray);
title('Smoothed Image');
axis off;

subplot(1, 2, 2);
imagesc(edge_direction_quantized);
colormap('jet');
title('Quantized Edge Direction');
colorbar;
axis off;
```

【課題③】

```
image = imread('kr.jpg'); % 画像のパスを適切に指定してください
image_gray = rgb2gray(image); % グレースケールに変換
k_size = [5, 5];
sigma = 1.0;

[X, Y] = meshgrid(-(k_size(2)-1)/2:(k_size(2)-1)/2, -(k_size(1)-
1)/2:(k_size(1)-1)/2);
kernel_x = (-X./(2*pi*sigma^4)).*exp(-(X.^2 + Y.^2)/(2*sigma^2));
kernel_y = (-Y./(2*pi*sigma^4)).*exp(-(X.^2 + Y.^2)/(2*sigma^2));
Gx = conv2(double(image_gray), kernel_x, 'same'); % image_gray を使用する
Gy = conv2(double(image_gray), kernel_y, 'same'); % image_gray を使用する
filtered_image = sqrt(Gx.^2 + Gy.^2);

% 正規化
```

```
filtered_image = filtered_image ./ max(filtered_image(:));

figure;
subplot(1, 2, 1);
imshow(image);
title('Original Image');

subplot(1, 2, 2);
imshow(filtered_image);
title('Canny Image');
```

【課題④】

```
img = imread('smoothed_img.jpg');  % 勾配の極大位置を検出する画像を読み込む
image_gray = rgb2gray(img); % グレースケールに変換
k_size = [5, 5];
sigma = 1.0;

[X, Y] = meshgrid(-(k_size(2)-1)/2:(k_size(2)-1)/2, -(k_size(1)-
1)/2:(k_size(1)-1)/2);
kernel_x = (-X./(2*pi*sigma^4)).*exp(-(X.^2 + Y.^2)/(2*sigma^2));
kernel_y = (-Y./(2*pi*sigma^4)).*exp(-(X.^2 + Y.^2)/(2*sigma^2));
Gx = conv2(double(image_gray), kernel_x, 'same'); % image_gray を使用する
Gy = conv2(double(image_gray), kernel_y, 'same'); % image_gray を使用する
filtered_image = sqrt(Gx.^2 + Gy.^2);

% 正規化
filtered_image = filtered_image ./ max(filtered_image(:));

% 非最大抑制を行って勾配の極大位置を検出する
edges = zeros(size(filtered_image));
[height, width] = size(filtered_image);
for i = 2:height-1
    for j = 2:width-1
        angle = gradient_direction(i, j);
        mag_current = gradient_magnitude(i, j);
```

```
        mag1 = gradient_magnitude(i + sign(sin(angle)), j +
sign(cos(angle)));
        mag2 = gradient_magnitude(i - sign(sin(angle)), j -
sign(cos(angle)));
        if (mag_current >= mag1) && (mag_current >= mag2)
            edges(i, j) = mag_current;
        end
    end
end

% 結果を表示する
subplot(1, 2, 1);
imshow(img_gray);
title('Original Image');
axis off;

subplot(1, 2, 2);
imshow(edges);
title('Gradient Maxima');
axis off;
```

【課題⑤】

```
img = imread('smoothed_img.jpg');  % コーナー検出を行う画像を読み込む

% グレースケールに変換
img_gray = rgb2gray(img);

% Sobel フィルタを適用して画像の勾配を計算
filter_x = [-1 0 1; -2 0 2; -1 0 1];
filter_y = [-1 -2 -1; 0 0 0; 1 2 1];
gradient_x = conv2(double(img_gray), filter_x, 'same');
gradient_y = conv2(double(img_gray), filter_y, 'same');

% 各方向の勾配の大きさを算出
gradient_magnitude = sqrt(gradient_x.^2 + gradient_y.^2);
```

```
% ヒステリシス閾値処理のパラメータ
Thhigh = 2 * mean(gradient_magnitude(:));  % 高い閾値
Thlow = 0.5 * mean(gradient_magnitude(:));  % 低い閾値

% 閾値処理を行う
edges = gradient_magnitude > Thhigh;  % 高い閾値以上のエッジ候補をエッジとする

% ヒステリシス閾値処理
while true
    old_edges = edges;
    [row, col] = find(edges & (gradient_magnitude > Thlow));  % 低い閾値以上かつ繋がっているエッジ候補を探す
    for k = 1:length(row)
        i = row(k);
        j = col(k);
        neighborhood = edges(max(1, i-1):min(size(edges, 1), i+1), max(1, j-1):min(size(edges, 2), j+1));
        if any(neighborhood(:))  % 近傍に既にエッジが存在する場合はエッジとする
            edges(i, j) = true;
        end
    end
    if isequal(old_edges, edges)  % 収束した場合は処理を終了
        break;
    end
end

% 結果を表示する
subplot(1, 2, 1);
imshow(img);
title('Original Image');
axis off;

subplot(1, 2, 2);
imshow(edges);
title('Edge Detection (Hysteresis Thresholding)');
```

```
axis off;
```