**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

1. 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。

1. アルゴリズム

【課題①】

ダイアグラム

中程度の精度で自動的に生成された説明

【課題②】

ダイアグラム

自動的に生成された説明

【課題③】

ダイアグラム

自動的に生成された説明

【課題④】

ゲーム画面のスクリーンショット

中程度の精度で自動的に生成された説明

【課題⑤】

ゲーム画面のスクリーンショット

中程度の精度で自動的に生成された説明

1. 結果

入力画像

屋外, 道路, 建物, フロント が含まれている画像

自動的に生成された説明

【課題①】出力画像

犬, フロント, 建物, 写真 が含まれている画像

自動的に生成された説明

【課題②】出力画像

文字と数字と文字の加工写真

中程度の精度で自動的に生成された説明

【課題③】出力画像

白黒の写真に写ってる男性の顔の絵

低い精度で自動的に生成された説明

【課題④】出力画像

背景パターン が含まれている画像

自動的に生成された説明

【課題⑤】出力画像

白黒の写真に写ってる男性の顔の絵

中程度の精度で自動的に生成された説明

1. 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。

1. 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;