

情報学群実験第 3C/3i 実験レポート 第 2 回

Title

1250373 溝口 洋熙*

Group 10

May 7th, 2023

概要

目次

第 1 章	画像の理解	1
1.1	実験の背景と目的	1
1.2	実験の方法	1
1.3	実験の結果	3
1.4	考察	3
第 2 章	画像のフィルタ処理	5
2.1	実験の背景と目的	5
2.2	実験の方法	6
2.3	実験の結果	7
2.4	考察	8
参考文献		9
付録		10
A	画像の理解 (April 27th, 2023)	10
B	画像のフィルタ処理 (May 8th, 2023)	13
C	(April 20th, 2023)	17
D	(April 24th, 2023)	17

図目次

1-1	4 ビットシフト	2
1-2	カラーチャネル操作 実験結果	3
1-3	画像の量子化数変換 実験結果	3
1-4	階調反転 実験結果	3
1-5	閾値処理 実験結果	3
1-6	ヒストグラム 実験結果	3
1-7	背景差分 実験結果	3
1-8	背景差分画像の閾値処理	3
2-1	3 × 3 画像フィルタ	6
2-2	カラーチャネル操作 実験結果	7
2-3	平滑化フィルタ	7
2-4	メディアンフィルタ	7
2-5	微分フィルタ適用	7
2-6	ラプラシアンフィルタの適用と処理	7
2-7	色空間変換 実験結果	8

表目次

1-1	実験環境	1
-----	------	---

ソースコード

1-1	グラフ・画像出力	2
1-2	<code>bitshift</code> 関数	2
1-3	判定結果の格納	2
1-4	<code>sum</code> 関数	2
2-1	白色ガウス雑音画像の生成	6
2-2	インパルス雑音画像の生成	6
2-3	メディアンフィルタの適用	6
A-1	カラーチャネル操作	10
A-2	画像の量子化数変換	10
A-3	階調反転	11
A-4	閾値処理	11
A-5	ヒストグラム	12
A-6	背景差分	12
B-1	テスト画像作成	13
B-2	平滑化フィルタ・メディアンフィルタ	13
B-3	微分フィルタ	14
B-4	ラプラシアンフィルタ	15
B-5	色空間変換	16
B-6	色空間変換 RGB 色空間	16
B-7	色空間変換 fucntion	16

第1章

画像の理解

1.1 実験の背景と目的

本章では、画像のカラーチャンネル操作、量子化、階調反転、閾値処理を行う。またグレースケール画像に対してヒストグラムを作成する。

■カラーチャンネル操作 RGB 色空間の画像を、緑チャネルだけを抜き出してグレースケール画像を作成する。赤チャネル、青チャネルについても同様にグレースケール画像を生成する。さらに、RGB 画像の赤チャネルと青チャネルを入れ替えたカラー画像を作成する。

■画像の量子化数変換 グレースケール画像を生成する。緑チャネルは色の濃淡を多く含む。RGB 色空間から色の濃淡を抽出したい場合は、緑 (G) の成分を多く抽出するとよい。具体的な割合を、式 (1.1) に示す。ここでは NTSC 輝度信号を取り出す方法で行う。生成したグレースケール画像に対して、画像の量子化数を変更することによる、画像の変化を確認する。量子化数は 8Bit, 4Bit, 2Bit, 1Bit の 4 種をテストする。量子化数 1Bit の画像を 2 値画像という。

$$\text{Gray scale image} = \text{Red} \times 30\% + \text{Green} \times 59\% + \text{Blue} \times 11\% \quad (1.1)$$

■階調反転 各量子化数の画像に対して、その画像を階調反転させる。階調反転とは、白黒を反転させることである。量子化数による階調変換後の画像を比較する。

■閾値処理 閾値処理とは、ある値（閾値）以上の場合を白、閾値以下を黒とし、2 値画像を作成することである。

■ヒストグラム 量子化数 8Bit のグレースケール画像のヒストグラムを作成する。画素値 $n(n = 0, 1, \dots, 255)$ の画素が何画素含まれているかのヒストグラムを作成する。

■背景差分 自分が写っている写真と、背景だけが写っている写真の差分画像をとる。これを背景差分と呼ぶ。背景差分の後、閾値処理を行う。物体領域を正しく検出するために考慮する点を考察する。

1.2 実験の方法

■実験に用いる装置 このレポート内すべての実験には MathWorks®社の MATLAB®を用いて、表 1-1 の環境下で実験する。

表 1-1: 実験環境

実験機	MacBook Air 2022 (Apple 社) MLY13J/A
プロセッサ	Apple Silicon M2 8 コア CPU, 8 コア GPU
メモリ	8GB
MATLAB®	R2023a - academic use (Update1 9.14.02239454) 64-bit (maci64) March 30, 2023

また、このレポートないすべての実験では MATLAB[®]でプロットしたグラフを出力するための `exportgraphics` 関数、画像を書き出すための `imwrite` 関数を用いる (src.1-1).

src. 1-1: グラフ・画像出力

```
exportgraphics(figurename, 'path/figure_name.pdf', 'ContentType', 'vector');
imwrite(data, "path/figure_name.png");
```

■カラーチャネル操作 `imwrite` 関数を用いて、画像の読み込む。読み込んだ画像は RGB 色空間で保存されており、チャネル 1 には R、チャネル 2 には G、チャネル 3 には B が保存されている。グレイスケール画像を作成するには、式 (1.1) の割合で画像を加算合成する。 m 行 n 列の行列 A に対して、1 行 n 列を取り出したければ、 $A(1,:)$ と記述する。 $:$ は、すべての要素を表す記号である。赤チャネルと青チャネルを入れ替えるためには、赤チャネルの行列と青チャネルの行列を変数に保存し、それお互いのチャネルに代入する。課題（カラー チャネル操作）のスクリプトは、src.A-1.

►p.10

■画像の量子化数変換 量子化数を変更するために、`bitshift` 関数を用いる (src.1-2)。この関数は、`img` を左に n ビットシフトする関数である。右シフトしたい場合は n を負の数で与える。ビットシフトについて、1 ビット右シフトするごとにそのデータは $1/2$ される。これを利用して、量子化数 4Bit の場合は右に 4Bit シフト、量子化数 2Bit の場合は右に 6Bit シフト、量子化数 1Bit の場合は右に 7Bit シフトする。

量子化数 4Bit を例にあげる。仮に画素値が 255 (白) を持つ画素の場合、量子化数を 4Bit にする、つまり 4Bit 右シフトすると、画素値は 15 になる (図 1-1)。このままでは画素値の範囲が 0 から 15 となる。この対策として、全体画素値と $255/15$ の積を取ることで、画素値を 0 から 255 にスケーリングする。課題（画像の量子化数変換）のスクリプトは、src.A-2.

src. 1-2: `bitshift` 関数

```
img = bitshift(img, n);
```

1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1

図 1-1: 4 ビットシフト

■階調反転 各量子化数ごとに階調反転する。階調反転を実現するためには、階調反転した画像を `double` 型に変換したあと、 -1 との積をとり、255 を足した後で `uint8` 型に変換する*。課題（階調反転）のスクリプトは、src.A-3.

►p.11

■閾値処理 MATLAB[®]には、判定結果のブール値を行列に格納する機能がある。src.1-3 より、行列 `mat` の各元が 5 より大きい箇所を 1、5 以下のところを 0 とする行列 `bin` を作成できる。この行列を真理行列と呼ぶ。この機能を用いて、ある閾値に対して、閾値よりも大きければ 1 を戻し、閾値以下であれば 0 を戻す行列を作成する。画素値の範囲を 0 から 255 へするために、行列の各元と 255 の積をとる。今回は、閾値を 64, 128, 192 で実験する。課題（閾値処理）のスクリプトは、src.A-4

src. 1-3: 判定結果の格納

```
mat = [1 2 3; 4 5 6; 7 8 9];
bin = mat > 5;
% -- 結果 --
bin = [0 0 0; 0 0 1; 1 1 1];
```

■ヒストグラム ヒストグラムを作成するために、この関数は行列の元を足し合わせる `sum` 関数を用いる (src.1-4)。各画素値 0 から 255 に対して、その画素値と等しい箇所を 1 とする真理行列を作成し、各元の和を `sum` 関数を用いて算出する。その結果が、ある画素値がいくつ画像に含まれているかを指す。

■背景差分 固定カメラ[†]で撮影した写真を用いる。「背景と被写体が写っている画像 `img_sbj`」「背景のみの画像 `img_bg`」の 2 点を撮影した。背景差分画像は、`img_sbj - img_bg` で生成する。生成した画像に対して、閾値処理する。閾値処理する前後の画像比較、閾値による比較し、考察する。今回、閾値を 32, 64, 128 で実験する。課題（背景差分）のスクリプトは、src.A-6.

►p.12

src. 1-4: `sum` 関数

```
matA = [1 2 3];
s_matA = sum(matA);
% -> 出力:6
matB = [1 2; 1 1; 1 1];
s_matB = sum(matB);
% -> 出力:[3 4]
s_s_matB = sum(sum(matB));
% -> 出力:12
```

* その画像の各画素値が `double` 型であるとき、`imwrite` が、データを自動的にリスケールし書き出すため。

[†] 手での固定は、背景がズレる可能性があるので、カメラを固定して撮影した。

1.3 実験の結果



図 1-2: カラーチャネル操作 実験結果

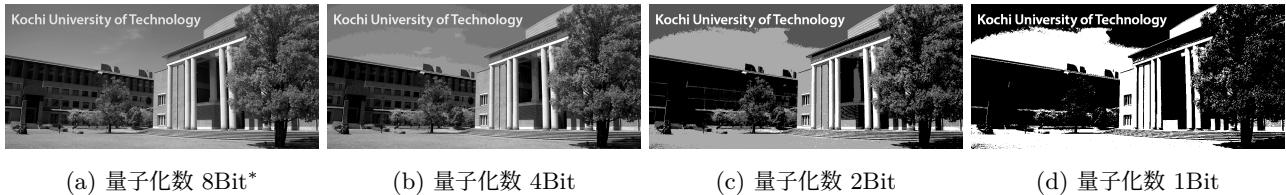


図 1-3: 画像の量子化数変換 実験結果

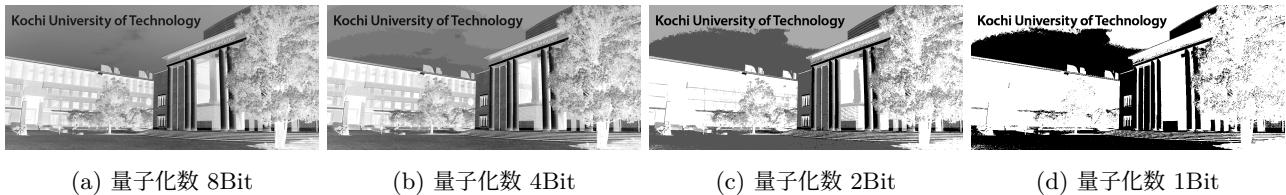


図 1-4: 階調反転 実験結果



図 1-5: 閾値処理 実験結果

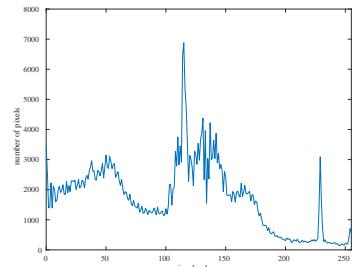


図 1-6: ヒストグラム 実験結果



図 1-7: 背景差分 実験結果



図 1-8: 背景差分画像の閾値処理

1.4 考察

■カラーチャネル操作 実験結果より、原画像に近い画像は、緑チャネルだけを抜き出してグレースケール画像として表示したものである。また、赤チャネルと青チャネルを入れ替えた画像では、文字や建物の概形を、はっきりとらえることがで

*NTSC 輝度信号のグレースケール画像

きる。これらは、式(1.1)でGreenの割合が一番多い理由と考えられる。

■画像の量子化数変換 量子化数を減らすことで、等高線のようなものが見える。これは、元画像でなだらかな画素値の変化箇所が、量子化数を減らすことでとびとびの値になることで生じる。これを擬似輪郭という。擬似輪郭は、量子化数を減らすことにより顕著になる。

■階調反転 量子化数が減ると擬似輪郭が生じる。この擬似輪郭は、階調反転しても変わらない。

■閾値処理 閾値を調節することで、擬似輪郭を境に白と黒に別れることがわかった。

■背景差分 背景差分画像では、被写体がぼんやりと写っている。この画像に対して閾値処理することで、被写体を強調できることが分かった。

第2章

画像のフィルタ処理

2.1 実験の背景と目的

この実験では、画像に対してフィルタの適用や色空間を変換する。

■画像フィルタ 画像に対して、フィルタを適用するとはどのようなことか？ 我々は、携帯電話の写真アプリケーションを用いて、写真を「加工」する。我々は「加工」という行為を「フィルタをかける」と呼ぶが、この「フィルタ」という言葉と、画像処理におけるフィルタは意味が異なる。画像処理におけるフィルタは、画像ないに含まれる雑音を除去したり、特徴を抽出したりすることで欠陥検出をより円滑に行うための基本処理を指す[1]。テスト画像として、以下の画像を用意する。グレースケール元画像を `original_img` とする。

1. 白色ガウス雑音

白色ガウス雑音は、白色性を持つガウス雑音である。今回は、平均 0、標準偏差 10 としてガウス分布の乱数を発生させる。このテスト画像を `wgn_img` とする。

2. インパルス雑音

インパルス雑音とは、超短時間におこる高周波の雑音のことを指す。今回は、画像のランダムな画素を、白または黒で塗り替える。それぞれ全体画素の 1% の割合で作成する。このテスト画像を `in_img` とする。

画像フィルタはいくつかの種類があり、画像雑音の除去やエッジの強調に用いられる。

1. 平滑化フィルタ

- 画像の各画素 p に対して、 n 近傍と中央の画素値の平均や重み付け平均をとり、 p の画素値とするフィルタ。
- 今回の実験では、各画素 p に対して、 3×3 、つまり 8 近傍と p の画素値の平均をとり、中央の画素値として定義する。2つのテスト画像にフィルタを適用し、雑音とフィルタの関係を考察する。

2. メディアンフィルタ

- 画像の各画素 p に対して、 n 近傍と中央の画素値を昇順に整列し、その中央値を p の画素値とする。
- 今回の実験では、各画素 p に対して、 3×3 、つまり 8 近傍と p の画素値を昇順に整列する。その中央値を p の画素値として定義する。2つのテスト画像にフィルタを適用し、雑音とフィルタの関係を考察する。

3. 微分フィルタ

- 微分フィルタは、境界線の強調や局所的な特徴の抽出するフィルタである。しかし、一次微分フィルタ、二次微分フィルタを用いると、画像の雑音も強調される。ここで Prewitt フィルタと Sobel フィルタを用いると、雑音がある画像でもうまく境界線を抽出できる[2, p.87]。
 - **Prewitt Filter**：隣り合う 2 画素の画素値を持ちいて 3 画素ずつをセットにして濃度の変化点を抽出するアルゴリズム[2, p.87]。
 - **Sobel Filter**：画像の各ピクセルの周囲の画素との差を計算して、その差の大きさを使って、エッジを検出するアルゴリズム。
- 今回の実験では、`original_img` に対して、Sobel フィルタを用いて縦微分、横微分、縦微分と横微分の加算合成した画像を作成する。フィルタを適用した画像の特徴と、それぞれの違いを考察する。

4. ラプラシアンフィルタ

- ラプラシアンフィルタは、微分フィルタ同様、境界線を見つけるために使われる方法である。
- 今回の実験では、`original_img`に対して、ラプラシアンフィルタを適用し、閾値処理する。フィルタを適用し閾値処理した画像と特徴と違いを考察する。

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

(a) 平滑化フィルタ

-1	0	1
-2	0	2
-1	0	1

(b) Prewitt フィルタ：横方向

-1	-2	-1
0	0	0
1	2	1

(c) Prewitt フィルタ：縦方向

1	1	1
1	8	1
1	1	1

(d) ラプラシアンフィルタ

図 2-1: 3×3 画像フィルタ

■色空間変換 この実験では、RGB 色空間から、HSV 色空間へ変換する。RGB 色空間は、赤 (Red), 緑 (Green), 青 (Blue) の 3 チャンネルで構成する。HSV は色相 (Hue), 彩度 (Saturation), 明度 (Value) の 3 チャンネルで構成する。色相は、カラー ホイール上の色の位置に対応する、0 から 100 の値、色相の量または中間からの逸脱。特定の色の赤、緑、青成分の中での最大値。いずれも `double` 型で保存される [3]。HSV 色空間の特徴として、人間が色を知覚する方法と類似しており、視覚障害者向けのアクセシビリティ向上に役立つことが挙げられる。たとえば、HSV の「明度」を調節することで、文字が見やすくなる [2, p.97 - p.98]。また、人間が色を知覚する方法と類似していることを踏まえて、HSV 色空間を用いることで、画像の特徴を抽出しやすくなる。今回の実験では、自分の手の写真を RGB 色空間から HSV 色空間へ変換し、肌色領域を抽出する。抽出した肌色領域を白色、そのほかの部分を黒色にして出力する。出力した画像と、RGB 色空間における肌色領域を抽出した場合の精度について考察する。

2.2 実験の方法

■白色ガウス雑音を加えた画像 白色ガウス雑音の作成には `randn` 関数を用い、生成した乱数と、標準偏差の積を取る。生成した乱数を `uint8` 型に変換し、`original_img` と和を取る (src.2-1)。255 を上回る、または 0 を下回る場合、それぞれの値に変換したものを、`wgn_img` とする。

■インパルス雑音を加えた画像 インパルス雑音は、`rand` 関数を用いて作成する。発生率を 1% にするため、乱数の 0.01 未満の画素を黒、乱数の 0.99 より大きい画素を白としてインパルスノイズを設計する (src.2-2)。

■フィルタの適用 平滑化フィルタ、微分フィルタ、ラプラシアンフィルタの適用は、`filter2` 関数、(`filter2(filter, img)`) を用いる。メディアンフィルタは、画像の i 行 j 列に対して、`median` 関数を用いてフィルタ処理する。メディアンフィルタを適用する際に、四方に画素がない画素（1 行 1 列画素、 m 行 1 列画素など）はフィルタ処理できないため、0 パディング処理を行い、メディアンフィルタを適用する (src.2-3)。

src. 2-1: 白色ガウス雑音画像の生成

```
% 画像サイズ : n x m
wgn = 10*randn(n, m);
wgn = uint8(wgn);
wgn_img = wgn + gimg;
```

src. 2-2: インパルス雑音画像の生成

```
% 画像サイズ : n x m
rnd = rand(n, m);
b = (rnd < 0.01);
w = (rnd > 0.99);
in_img(w) = 255;
in_img(b) = 0;
```

src. 2-3: メディアンフィルタの適用

```
for h = 2:img_height % 画像行列 img の高さ
    for w = 2:img_width % 画像行列 img の横幅, median_filter は0パディング後の img
        median_filter(h-1, w-1) = median(img(h-1:h+1,w-1:w+1),"all");
    end
end
```

課題（フィルタ処理）のスクリプトは、src.B-1 - src.B-4.
 ►p.13 ►p.15

■色空間変換 読み込んだ画像はRGB色空間で保存される。この画像をHSV色空間に変換するためには、`rgb2hsv`関数を用いる。出力された値と255の積を取り、HSV色空間で出力された画像を書き出す。色相、彩度、明度それぞれのチャネルを抽出し、MATLAB®のアプリケーションを用いて、肌色要素のHSV成分を出力する(src.B-7)。それぞれの値に合致した画素を、画素値255、ほかの画素値を0とした画像を書き出す。また、RGB色空間における、肌色領域の抽出も行う。肌色部ある点のRGB値と等しい画素を白く塗る。課題（色空間変換）のスクリプトは、src.B-5, src.B-6.
 ►p.16 ►p.16

2.3 実験の結果



図 2-2: カラーチャネル操作 実験結果



図 2-3: 平滑化フィルタ

図 2-4: メディアンフィルタ



図 2-5: 微分フィルタ適用



図 2-6: ラプラシアンフィルタの適用と処理

*横方向微分フィルタ適用後画像と縦方向微分フィルタ適用後画像の和。



(a) 元画像

(b) HSV 色空間への変換後

(c) 肌色領域の抽出 (HSV)

(d) 肌色領域の抽出 (RGB)

図 2-7: 色空間変換 実験結果

2.4 考察

■平滑化フィルタ，メディアンフィルタ `wgn_img` に対して平滑化フィルタを適用すると，雑音部分が目立たなくなった。また，`in_img` に対して平滑化フィルタを適用すると，雑音が取り除かれることなく残った。`wgn_img` と `in_img` に対して メディアンフィルタを適用すると，雑音部分が目立たなくなった。

参考文献

- [1] ヴィスコ・テクノロジーズ株式会社. 画像フィルタ～より容易な血管検出のために(前編). <https://www.visco-tech.com/newspaper/column/detail19/>, Confirmation date: May 9th, 2023.
- [2] 大町真一郎, 陳謙, 大町方子. 画像処理(未来へつなぐデジタルシリーズ = Connection to the future with digital series) . 未来へつなぐデジタルシリーズ = Connection to the future with digital series. 共立出版, 2014.
- [3] MathWorks. `rgb2HSV`. <https://jp.mathworks.com/help/matlab/ref/rgb2HSV.html#d124e1239127>, Confirmation date: May 13th, 2023.

付録

A 画像の理解 (April 27th, 2023)

src. A-1: カラーチャネル操作

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 % color channel 1=red, 2=green, 3=blue
4 red = img(:,:,1);
5 green = img(:,:,2);
6 blue = img(:,:,3);
7 fig0 = figure;
8 imshow(red);
9 imwrite(red,'../Figures/05_11_r.png');
10 fig1 = figure;
11 imshow(green);
12 imwrite(green,'../Figures/05_12_g.png');
13 fig2 = figure;
14 imshow(blue);
15 imwrite(blue,'../Figures/05_13_b.png');
16 fig3 = figure;
17 changeimg(:,:,1) = blue;
18 changeimg(:,:,2) = green;
19 changeimg(:,:,3) = red;
20 imshow(changeimg);
21 imwrite(changeimg,'../Figures/05_14_change.png');

```

src. A-2: 画像の量子化数変換

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 figure;
5 imshow(gimg);
6 imwrite(gimg,"../Figures/05_21_gimg.png");
7 gray_4bit = bitshift(gimg,-4) * (255/15); % XXXX XXXX -> XXXX
8 gray_2bit = bitshift(gimg,-6) * (255/3); % XXXX XXXX -> XX
9 gray_1bit = bitshift(gimg,-7) * (255/1); % XXXX XXXX -> X
10
11 figure;
12 imshow(gray_4bit);
13 imwrite(gray_4bit,"../Figures/05_22_4bit.png");
14 figure;
15 imshow(gray_2bit);
16 imwrite(gray_2bit,"../Figures/05_23_2bit.png");
17 figure;
18 imshow(gray_1bit);

```

```
19 | imwrite(gray_1bit,"../Figures/05_24_1bit.png");
```

src. A-3: 階調反転

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 gimg = double(gimg);
5 gimg_i = -1 * gimg;
6 gimg_i = gimg_i + 255;
7 gimg_i = uint8(gimg_i);
8 figure;
9 imshow(gimg_i);
10 imwrite(gimg_i,"../Figures/05_31_8.png");
11
12 gray_4bit = bitshift(gimg,-4) * (255/15); % XXXX XXXX -> XXXX
13 gray_2bit = bitshift(gimg,-6) * (255/3); % XXXX XXXX -> XX
14 gray_1bit = bitshift(gimg,-7) * (255/1); % XXXX XXXX -> X
15
16 gray_4bit = double(gray_4bit);
17 gimg_i4 = -1 * gray_4bit;
18 gimg_i4 = gimg_i4 + 255;
19 gimg_i4 = uint8(gimg_i4);
20 figure;
21 imshow(gimg_i4);
22 imwrite(gimg_i4,"../Figures/05_32_4.png");
23
24 gray_2bit = double(gray_2bit);
25 gimg_i2 = -1 * gray_2bit;
26 gimg_i2 = gimg_i2 + 255;
27 gimg_i2 = uint8(gimg_i2);
28 figure;
29 imshow(gimg_i2);
30 imwrite(gimg_i2,"../Figures/05_33_2.png");
31
32 gray_1bit = double(gray_1bit);
33 gimg_i1 = -1 * gray_1bit;
34 gimg_i1 = gimg_i1 + 255;
35 gimg_i1 = uint8(gimg_i1);
36 figure;
37 imshow(gimg_i1);
38 imwrite(gimg_i1,"../Figures/05_34_1.png");

```

src. A-4: 閾値処理

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 threshold1 = 64;
5 threshold2 = 128;
6 threshold3 = 192;
7 threshold1_r = gimg > threshold1;
8 threshold1_r = threshold1_r.*255;
9 threshold2_r = gimg > threshold2;
10 threshold2_r = threshold2_r.*255;
11 threshold3_r = gimg > threshold3;

```

付録

```
12 threshold3_r = threshold3_r.*255;
13 figure;
14 imshow(threshold1_r);
15 imwrite(threshold1_r,"../Figures/05_41.png");
16 figure;
17 imshow(threshold2_r);
18 imwrite(threshold2_r,"../Figures/05_42.png");
19 figure;
20 imshow(threshold3_r);
21 imwrite(threshold3_r,"../Figures/05_43.png");
```

src. A-5: ヒストグラム

```
1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 pixel = (0:1:255);
5 nof_pixel = zeros(1,256);
6 for k=1:256
7     lmg = gimg == k - 1;
8     nof_pixel(k) = sum(sum(lmg));
9 end
10 fig0 = figure;
11 plot(pixel,nof_pixel);
12 xlabel('pixcel value');
13 ylabel('number of pixels');
14 axis([0 255 0 8000]);
15 exportgraphics(fig0,'../Figures/05_50_graph.pdf','ContentType','vector');
```

src. A-6: 背景差分

```
1 clear; close all;
2 img1 = imread("fig1.jpg"); % 人あり
3 img2 = imread("fig2.jpg"); % 人なし
4 img1 = 0.3*img1(:,:,1) + 0.59*img1(:,:,2) + 0.11*img1(:,:,3);
5 img2 = 0.3*img2(:,:,1) + 0.59*img2(:,:,2) + 0.11*img2(:,:,3);
6 imwrite(img1,"fig1_g.jpg");
7 imwrite(img2,"fig2_g.jpg");
8 result = int16(img2) - int16(img1);
9 result = abs(result);
10 result = uint8(result);
11 figure;
12 imshow(result);
13 imwrite(result,"../Figures/05_60.png");
14
15 threshold1 = 32;
16 threshold2 = 64;
17 threshold3 = 128;
18 threshold1_r = result > threshold1;
19 threshold1_r = threshold1_r.*255;
20 threshold2_r = result > threshold2;
21 threshold2_r = threshold2_r.*255;
22 threshold3_r = result > threshold3;
23 threshold3_r = threshold3_r.*255;
24 figure;
25 imshow(threshold1_r);
```

```

26 imwrite(threshold1_r,"../Figures/05_61.png");
27 figure;
28 imshow(threshold2_r);
29 imwrite(threshold2_r,"../Figures/05_62.png");
30 figure;
31 imshow(threshold3_r);
32 imwrite(threshold3_r,"../Figures/05_63.png");

```

B 画像のフィルタ処理 (May 8th, 2023)

src. B-1: テスト画像作成

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 gimg_size = size(gimg);
5 gimg_height = gimg_size(1,1);
6 gimg_width = gimg_size(1,2);
7 % white Gaussian noise (wgn)
8 wgn = 10*randn(gimg_height, gimg_width);
9 wgn = uint8(wgn); % cast
10 wgn_img = wgn + gimg;
11 % scaling
12 bin = (wgn_img > 255);
13 wgn_img(bin) = 255;
14 bin = (wgn_img < 0);
15 wgn_img(bin) = 0;
16 % writing
17 imshow(wgn_img);
18 imwrite(wgn_img, './file_white-Gaussian-noise.png');
19
20 % impulse noise (in)
21 in_img = gimg;
22 rnd = rand(gimg_height,gimg_width);
23 in_black_bin = (rnd<0.01); % (1%)
24 in_white_bin = (rnd>0.99); % (1%)
25 in_img(in_white_bin) = 255;
26 in_img(in_black_bin) = 0;
27 % writing
28 figure;
29 imshow(in_img);
30 imwrite(wgn_img, './file_impluse-noise.png');

```

src. B-2: 平滑化フィルタ・メディアンフィルタ

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 gimg_size = size(gimg);
5 gimg_height = gimg_size(1,1);
6 gimg_width = gimg_size(1,2);
7 % white Gaussian noise (wgn)
8 wgn = 10*randn(gimg_height, gimg_width);
9 wgn = uint8(wgn); % cast

```

付録

```
10 wgn_img = wgn + gimg;
11 % scaling
12 bin = (wgn_img > 255);
13 wgn_img(bin) = 255;
14 bin = (wgn_img < 0);
15 wgn_img(bin) = 0;
16 % impulse noise (in)
17 in_img = gimg;
18 rnd = rand(gimg_height,gimg_width);
19 in_black_bin = (rnd<0.01); % (1%)
20 in_white_bin = (rnd>0.99); % (1%)
21 in_img(in_white_bin) = 255;
22 in_img(in_black_bin) = 0;
23
24 % Smoothing filter (sf)
25 filter_sf = ones(3,3) /9;
26 sf_img_wgn = filter2(filter_sf, wgn_img);
27 sf_img_wgn = uint8(sf_img_wgn);
28 sf_img_in = filter2(filter_sf, in_img);
29 sf_img_in = uint8(sf_img_in);
30
31 figure;
32 imshow(sf_img_wgn);
33 imwrite(sf_img_wgn,"../Figures/06_21_sf_img_wgn.png");
34 figure;
35 imshow(sf_img_in);
36 imwrite(sf_img_in,"../Figures/06_22_sf_img_in.png");
37
38 % Median filter (mf)
39 zeroPadding_height = zeros(gimg_height, 1);
40 zeroPadding_width = zeros(1, gimg_width + 2);
41 zeroPadding_img_wgn = [zeroPadding_height wgn_img zeroPadding_height];
42 zeroPadding_img_wgn = [zeroPadding_width ; zeroPadding_img_wgn ; zeroPadding_width];
43 zeroPadding_img_in = [zeroPadding_height wgn_img zeroPadding_height];
44 zeroPadding_img_in = [zeroPadding_width ; zeroPadding_img_in ; zeroPadding_width];
45
46 mf_img_wgn = zeroPadding_img_wgn;
47 mf_img_in = zeroPadding_img_in;
48
49 for h = 2:gimg_height
50     for w = 2:gimg_width
51         mf_img_wgn(h-1,w-1) = median(zeroPadding_img_wgn(h-1:h+1,w-1:w+1),"all");
52         mf_img_in(h-1,w-1) = median(zeroPadding_img_in(h-1:h+1,w-1:w+1),"all");
53     end
54 end
55 figure;
56 imshow(mf_img_wgn);
57 imwrite(mf_img_wgn,"../Figures/06_23_mf_img_wgn.png");
58 figure;
59 imshow(mf_img_in);
60 imwrite(mf_img_in,"../Figures/06_24_mf_img_in.png");
```

src. B-3: 微分フィルタ

```
1 clear; close all;
2 img = imread("../kut.jpg");
```

```

3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 % Differential filter (diff)
5 filter_diff_y = [-1 -2 -1; 0 0 0; 1 2 1];
6 filter_diff_x = filter_diff_y';
7 diff_x_img = filter2(filter_diff_x,gimg);
8 diff_x_img = abs(diff_x_img);
9 diff_x_img = diff_x_img /2; % 最大値で割る
10 diff_x_img = uint8(diff_x_img);
11 diff_y_img = filter2(filter_diff_y,gimg);
12 diff_y_img = abs(diff_y_img);
13 diff_y_img = diff_y_img /2; % 最大値で割る
14 diff_y_img = uint8(diff_y_img);
15 diff_img = diff_y_img + diff_x_img;
16 % writing
17 figure;
18 imshow(diff_x_img);
19 imwrite(diff_x_img,"../Figures/06_31_diff-x-img.png");
20 figure;
21 imshow(diff_y_img);
22 imwrite(diff_y_img,"../Figures/06_32_diff-y-img.png");
23 figure;
24 imshow(diff_img);
25 imwrite(diff_img,"../Figures/06_33_diff-img.png");

```

src. B-4: ラプラシアンフィルタ

```

1 clear; close all;
2 img = imread("../kut.jpg");
3 gimg = 0.3*img(:,:,1) + 0.59*img(:,:,2) + 0.11*img(:,:,3);
4 % Laplacian filter (lf)
5 filter_lf = [1 1 1; 1 -8 1; 1 1 1];
6 lf_img = filter2(filter_lf,gimg);
7 lf_img = abs(lf_img);
8 lf_img = lf_img /8; % 最大値で割る
9 lf_img = uint8(lf_img);
10 figure;
11 imshow(lf_img);
12 imwrite(lf_img,"../Figures/06_41_lf-img.png");
13 % Thresholding
14 num1 = zeros(1, 256);
15 for i = 0:255
16     num1(i+1) = sum(sum(lf_img > i));
17 end
18 x = (0:255);
19 fig0 = figure;
20 plot(x,num1);
21 axis([0 255 0 max(num1)+10000]);
22 xlabel('pixel value');
23 ylabel('number of pixels');
24 exportgraphics(fig0,'../Figures/06_42_Thresholding-graph.pdf','ContentType','vector');
25 bin = lf_img > 30; % Threshold = 30
26 lf_img_thresholding = lf_img;
27 lf_img_thresholding(bin) = 255;
28 figure;
29 imshow(lf_img_thresholding);
30 imwrite(lf_img_thresholding,"../Figures/06_43_lf-img-thresholding.png");

```

付録

src. B-5: 色空間変換

```
1 clear; close all;
2 img = imread("file_hand.png");
3 img_hsv = rgb2hsv(img);
4 img_hsv_255 = img_hsv * 255;
5 img_hsv_255= uint8(img_hsv_255);
6 figure;
7 imshow(img_hsv_255);
8 imwrite(img_hsv,"../Figures/06_51_img-hsv.png");
9 hsv_h = img_hsv(:,:,1); % Hue (色相)
10 hsv_s = img_hsv(:,:,2); % Saturation (彩度)
11 hsv_v = img_hsv(:,:,3); % Value (明度)
12 img_size = size(img);
13 img_height = img_size(1,1);
14 img_width = img_size(1,2);
15
16 % Skin color detection (scd)
17 scd = zeros(img_height, img_width);
18 for h = 1:img_height
19     for w = 1:img_width
20         if((hsv_h(h,w) >= 0.507) || (hsv_h(h,w) <= 0.108)) && (hsv_s(h,w) >= 0.197) && (hsv_v(
21             h,w) >= 0.362) && (hsv_s(h,w) <= 0.622) && (hsv_v(h,w) <= 0.920)
22             scd(h,w) = 255;
23         end
24     end
25 end
26 scd = scd * 255;
27 scd = uint8(scd);
28 figure;
29 imshow(scd);
30 imwrite(scd,"../Figures/06_52_scd.png");
```

src. B-6: 色空間変換 RGB 色空間

```
1 clear; close all;
2 img = imread("file_hand.png");
3 r = img(:,:,1);
4 g = img(:,:,2);
5 b = img(:,:,3);
6 bin_r = (r == 120);
7 bin_g = (g == 98);
8 bin_b = (b == 77);
9 bin_r = bin_r * 255;
10 bin_g = bin_g * 255;
11 bin_b = bin_b * 255;
12 hand = bin_r + bin_g + bin_b;
13 bin = (hand > 255);
14 hand(bin) = 255;
15 imshow(hand);
16 imwrite(hand,"../Figures/06_53_hand.png");
```

src. B-7: 色空間変換 fucntion

```
1 function [BW,maskedRGBImage] = no5_hsvfunc(RGB)
2 %createMask Threshold RGB image using auto-generated code from colorThresholder app.
```

```

3 % [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
4 % auto-generated code from the colorThresholder app. The colorspace and
5 % range for each channel of the colorspace were set within the app. The
6 % segmentation mask is returned in BW, and a composite of the mask and
7 % original RGB images is returned in maskedRGBImage.
8
9 % Auto-generated by colorThresholder app on 08-May-2023
10 %-----
11
12
13 % Convert RGB image to chosen color space
14 I = rgb2hsv(RGB);
15
16 % Define thresholds for channel 1 based on histogram settings
17 channel1Min = 0.507;
18 channel1Max = 0.108;
19
20 % Define thresholds for channel 2 based on histogram settings
21 channel2Min = 0.197;
22 channel2Max = 0.622;
23
24 % Define thresholds for channel 3 based on histogram settings
25 channel3Min = 0.362;
26 channel3Max = 0.920;
27
28 % Create mask based on chosen histogram thresholds
29 sliderBW = ( (I(:,:,1) >= channel1Min) | (I(:,:,1) <= channel1Max) ) & ...
30   (I(:,:,2) >= channel2Min) & (I(:,:,2) <= channel2Max) & ...
31   (I(:,:,3) >= channel3Min) & (I(:,:,3) <= channel3Max);
32 BW = sliderBW;
33
34 % Initialize output masked image based on input image.
35 maskedRGBImage = RGB;
36
37 % Set background pixels where BW is false to zero.
38 maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
39
40 end

```

C (April 20th, 2023)

D (April 24th, 2023)