

情報学群実験第 3C/3i 実験レポート 第 2 回
画像情報と視覚情報の処理に関する実験

1250373 溝口 洋熙*
Group 10

May 28th, 2023

概要

本実験では、画像情報処理、視覚情報処理を扱う。画像情報処理では、カラーチャネル操作や閾値処理、量子化数変換などの基礎的な画像処理を行い、それらの技術を用いて画像フィルタや背景差分画像の作成や肌色領域の抽出や、画像の2次元フーリエ変換をする。パワースペクトルの示す意味や、画像と、画像のパワースペクトルの座標の関係が明らかとなった。また、背景差分画像と色空間変換後の肌色領域抽出で、対象物の抽出における特徴も明らかとなった。視覚情報処理では、方位残効を扱う。MATLAB[®]上で方位残効刺激を作成する。この実験より、順応刺激とする縞模様方位の大きさと方位残効の度合いの関係を、視覚特性を考慮したうえで明らかになった。

目次

| | | |
|-------------|--|----|
| 第1章 | 画像情報処理 | 1 |
| 1.1 | 実験の目的 | 1 |
| 1.2 | 実験の方法と考え方 | 1 |
| 1.3 | 実験の結果 | 5 |
| 1.4 | 考察 | 9 |
| 1.5 | 結論 | 10 |
| 第2章 | 視覚情報処理 方位残効 | 11 |
| 2.1 | 実験の目的 | 11 |
| 2.2 | 実験の方法と考え方 | 11 |
| 2.3 | 実験の結果 | 12 |
| 2.4 | 考察 | 12 |
| 2.5 | 結論 | 12 |
| 第3章 | 関連語句 | 13 |
| 3.1 | ガボールフィルタ | 13 |
| 3.2 | 固有値法 | 13 |
| 3.3 | オプティカルフロー | 13 |
| 3.4 | ストラクチャーフロムモーション (SfM: Structure from Motion) | 14 |
| 参考文献 | | 15 |
| 付録 | | 16 |
| A | 画像の理解 (April 27th, 2023) | 16 |
| B | 画像のフィルタ処理 (May 8th, 2023) | 19 |
| C | 視覚情報処理 方位残効 (May 11th, 2023) | 24 |
| D | 画像のフーリエ変換・逆フーリエ変換 (May 15th, 2023) | 26 |

図目次

| | | |
|------|------------------------------------|----|
| 1-1 | 3 × 3 画像フィルタ | 4 |
| 1-2 | フィルタ適用の実際 | 4 |
| 1-3 | カラーチャネル操作 実験結果 | 5 |
| 1-4 | 画像の量子化数変換 実験結果 | 6 |
| 1-5 | 階調反転 実験結果 | 6 |
| 1-6 | 閾値処理 実験結果 | 6 |
| 1-7 | 画素値ヒストグラム* | 6 |
| 1-8 | 背景差分 実験結果 | 6 |
| 1-9 | 背景差分画像の閾値処理 | 6 |
| 1-10 | カラーチャネル操作 実験結果 | 6 |
| 1-11 | 平滑化フィルタ適用後の画像 | 7 |
| 1-12 | メディアンフィルタ適用後の画像 | 7 |
| 1-13 | Sobel フィルタの適用 | 7 |
| 1-14 | Laplacian フィルタの適用と処理 | 7 |
| 1-15 | 色空間変換 実験結果 | 7 |
| 1-16 | 縦縞・横縞に対するフーリエ変換 | 8 |
| 1-17 | 画像の座標とパワースペクトルの関係 | 8 |
| 1-18 | 高域通過フィルタ | 9 |
| 1-19 | 周波数スペクトル | 10 |
| 2-1 | 生成した画像 | 12 |
| 2-2 | 方位残効のメカニズム（垂直方向に反応する受容野） | 12 |
| 3-1 | エピポーラ幾何 | 14 |

ソースコード

| | | |
|-----|--------------------|----|
| 1-1 | グラフ・画像出力 | 1 |
| 1-2 | bitshift 関数 | 2 |
| 1-3 | 判定結果の格納 | 2 |
| 1-4 | sum 関数 | 2 |
| 1-5 | 白色ガウス雑音画像の生成 | 3 |
| 1-6 | インパルス雑音画像の生成 | 3 |
| 1-7 | メディアンフィルタの適用 | 4 |
| 1-8 | 2次元フーリエ変換と高域通過フィルタ | 5 |
| 2-1 | 矩形と円の作成 | 11 |
| 2-2 | 刺激画像の表示方法 | 11 |
| A-1 | カラーチャネル操作 | 16 |
| A-2 | 画像の量子化数変換 | 16 |
| A-3 | 階調反転 | 17 |
| A-4 | 閾値処理 | 17 |
| A-5 | ヒストグラム | 18 |
| A-6 | 背景差分 | 18 |
| B-1 | テスト画像作成 | 19 |
| B-2 | 平滑化フィルタ・メディアンフィルタ | 19 |
| B-3 | 微分フィルタ | 20 |
| B-4 | Laplacian フィルタ | 21 |
| B-5 | 色空間変換 HSV 色空間 | 22 |
| B-6 | 色空間変換 RGB 色空間 | 22 |
| B-7 | 色空間変換 fucntion_HSV | 22 |
| B-8 | 色空間変換 fucntion_RGB | 23 |
| C-1 | 正弦波縞の作成 | 24 |
| C-2 | 方位残効のデモ作成 | 25 |
| D-1 | 2次元 FFT 1 | 26 |
| D-2 | 2次元 FFT 2 | 28 |
| D-3 | 2次元 FFT 3 | 29 |
| D-4 | 高域通過フィルタ | 30 |

第1章

画像情報処理

1.1 実験の目的

MATLAB[®]を用いて、画像に対して、カラーチャンネルの操作、量子化数変換、階調反転、閾値処理、画素値に対するヒストグラムを作成など、基本的な画像処理を行う。

また、HSV 色空間上で肌色の抽出をする。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いることが挙げられる。視覚障害者向けのアクセシビリティ向上に役立つ [1, p.97 - p.98]。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。

さらに、ノイズ画像に対して、ノイズ除去する。周囲画素値の平均値を中央の画素値にする平滑化フィルタと、周囲画素値の中央値を中央の画素値にするメディアンフィルタを、ノイズの雑音除去具合で比較する。

加えて、画像のエッジを検出するために、微分フィルタを適用する。今回適用する微分フィルタは、Sobel フィルタと Laplacian フィルタである。両フィルタで抽出できるエッジの精度について考察する。

最後に、画像を 2 次元フーリエ変換し、パワースペクトルを出力する。出力したパワースペクトルに対して、高域通過フィルタを適用し、画像座標とパワースペクトル画像座標の関係と、パワースペクトル各値の意味を明らかにする。

1.2 実験の方法と考え方

■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、このレポート内すべての実験では 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 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。`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。グレースケール画像を作成するには、各チャネルの画素値を、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、それぞれ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.16

■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。

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

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。ある数値を 1 ビット右シフトするごとに、その数値は $1/2$ 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。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。
►p.16

src. 1-2: bitshift 関数

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

■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。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.17

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];
```

■閾値処理 閾値処理とは、ある値より大きい場合を白、その値未満を黒とする 2 値画像を作成することである。この値を閾値という。MATLAB®には、判定結果の真理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。
►p.17

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` 関数が、データを自動的にリスケールするため。

■**背景差分** 被写体が写っている画像と、被写体を除いた背景だけが写っている画像の差分画像をとる。これを背景差分と呼ぶ。背景差分の後、閾値処理を行う。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、 $\text{img_sbj} - \text{img_bg}$ で生成する。生成した画像に対して、閾値を 32, 64, 128 で閾値処理する。閾値処理する前後の画像比較、閾値による比較する。課題（背景差分）のスクリプトは、src.A-6。
►p.18

■**画像フィルタ** 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[2]。今回の実験では、白色ガウス雑音とインパルス雑音を含んだ2つのテスト画像を用意する。グレイスケール元画像を original_img とする。

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。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.1-5)。255 を上回る、または 0 を下回る画素値を、0 または 255 へ変換した画像を、wgn_img とする。

インパルス雑音は、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、白と黒で塗り替える。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。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.1-6)。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. 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。

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、1次微分フィルタ、2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 [1, p.87]。
 - Prewitt フィルタ**：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 [1, p.87]。
 - Sobel フィルタ**：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了画像を作成する。

4. Laplacian フィルタ

- Laplacian 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、8 近傍 Laplacian フィルタを適用し、閾値処理する。

src. 1-5: 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. 1-6: 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;
```

[†]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。

| | | |
|-----|-----|-----|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

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

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

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 8 | 1 |
| 1 | 1 | 1 |

(a) 平滑化フィルタ

(b) Sobel フィルタ：横方向

(c) Sobel フィルタ：縦方向

(d) 8 近傍 Laplacian フィルタ

図 1-1: 3×3 画像フィルタ

■フィルタの適用 平滑化フィルタ、微分フィルタ、Laplacian 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、四方すべてに画素がない画素（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.1-7)。

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

(a) Sobel フィルタ：縦方向

| | | | | | |
|-----|-----|-----|-----|-----|-----|
| 127 | 127 | 127 | 127 | 127 | 127 |
| 127 | 127 | 127 | 127 | 127 | 127 |
| 127 | 127 | 127 | 80 | 127 | 127 |
| 127 | 80 | 70 | 20 | 127 | 127 |
| 127 | 60 | 127 | 30 | 127 | 127 |
| 127 | 127 | 127 | 127 | 127 | 127 |

(b) フィルタを適用する画像

$$\begin{aligned} & (-1 \times 127) + (0 \times 127) + (1 \times 80) \\ & + (-2 \times 80) + (0 \times 70) + (2 \times 20) \\ & + (-1 \times 60) + (0 \times 127) + (1 \times 39) \\ & = -57 \end{aligned}$$

| | | | | | |
|-----|-----|-----|-----|-----|-----|
| 127 | 127 | 127 | 127 | 127 | 127 |
| 127 | 127 | 127 | 127 | 127 | 127 |
| 127 | 127 | 127 | 80 | 127 | 127 |
| 127 | 80 | 0 | 20 | 127 | 127 |
| 127 | 60 | 127 | 30 | 127 | 127 |
| 127 | 127 | 127 | 127 | 127 | 127 |

(c) フィルタ処理の演算

(d) フィルタ適用後の画素値

図 1-2: フィルタ適用の実際

src. 1-7: 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（ノイズ画像の生成とフィルタの適用）のスクリプトは、src.B-1 - src.B-4.
 ▶p.19 ▶p.21

■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]。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、`rgb2 hsv` 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。同様な方法で、RGB 色空間における肌色領域の抽出も行う (src.B-8)。課題（色空間変換）のスクリプトは、src.B-5, src.B-6.
 ▶p.23 ▶p.22 ▶p.22

■2 次元フーリエ変換 フーリエ変換は、任意の連続信号に対して各空間周波数成分が、どの程度含まれているかを示す変換である。音声などの 1 次元信号に対しては、離散 1 次元フーリエ変換を適用した。画像は 2 次元信号なので、2 次元離散フーリエ変換し、空間周波数成分を取り出す。出力されるスペクトルは、縦軸に y 方向のスペクトル、横軸には x 方向のスペクトル、輝度には成分の大きさが表現されている。連続データにおいて、2 次元フーリエ変換するためには、式 (1.2) を用

いる。また、離散データ、 N 行 M 列の画像に対して、 x 行 y 列画素の画素値を $f(x, y)$ 、2次元フーリエ変換結果の u 行 v 列画素値を $F(u, v)$ とすると、 $F(u, v)$ は式(1.3)で求められる。

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \exp\left(-j2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)\right) dx dy \quad (j \text{は虚数単位}) \quad (1.2)$$

$$F(u, v) = \sum_{x=1}^N \sum_{y=1}^M f(x, y) \exp\left(-j2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)\right) \quad (1.3)$$

ここで、式(1.4)より、2次元フーリエ変換の直交成分 $(u, v) = (0, 0)$ の値 $F(0, 0)$ は、全画素値の和である。

$$F(0, 0) = \sum_{x=1}^N \sum_{y=1}^M f(x, y) \quad (1.4)$$

MATLAB[®]における高速2次元離散フーリエ変換*は、fft2関数を用いる。出力したデータを整形するために、fftshift関数を用いる。今回は、出力された空間周波数スペクトルを、パワースペクトル†に変換して結果とする。一部の出力結果は、対数‡をとって出力する。出力結果のカラーマップをグレイスケールにする colormap関数、画像データの最大値を白、最小値を黒にスケーリングして出力する imagesc関数を用いる。

また、画像に対して高域通過フィルタを適用するには、パワースペクトルにフィルタを適用し、2次元逆フーリエ変換することでフィルタを適用した画像を得られる。高域通過フィルタは、円形のフィルタを用いる。低周波数領域で、中心から半径50pixel、100pixel以内の値を0にすることで、フィルタを適用する。フィルタは、フーリエ変換後のデータと同じサイズの行列に対して、対象となる円領域を0、ほかの領域を1とし、フィルタ m 行 n 列と、フーリエ変換後データ m 行 n 列との積を取ることで適用する。ここで、fft2後の値は、虚数を含む複素数である。値は、複素数としての値で意味があり、パワースペクトルに対してフィルタを適用してはならない。逆フーリエ変換した複素数値を含むデータを real関数を用いて実数化し、画像として表示する。

src. 1-8: 2次元フーリエ変換と高域通過フィルタ

```
% 画像データを data とする
fft_data = fftshift(fft2(data)); % 2次元フーリエ変換
power = abs(fft_data.^ 2); % 2次元フーリエ変換してパワースペクトルを生成する
log_power = log(1 + power); % 生成したパワースペクトルに対して対数を取る
% 高域通過フィルタを flt とする
fft_img = fft_data.*flt; % フィルタの適用
img = ifftshift(ifft2(fft_img)); % 2次元逆フーリエ変換
img = real(img); % 実数化
```

課題(2次元フーリエ変換)のスクリプトは、src.D-1 - src.D-4。
 ►p.26 ►p.30

1.3 実験の結果



図1-3: カラーチャネル操作 実験結果

*以降、高速2次元離散フーリエ変換を2次元フーリエ変換と記す。

†パワーとは、各値を2乗したもの。

‡パワースペクトルのデータを power とすると、 $\log(1 + \text{power})$ で表現する。

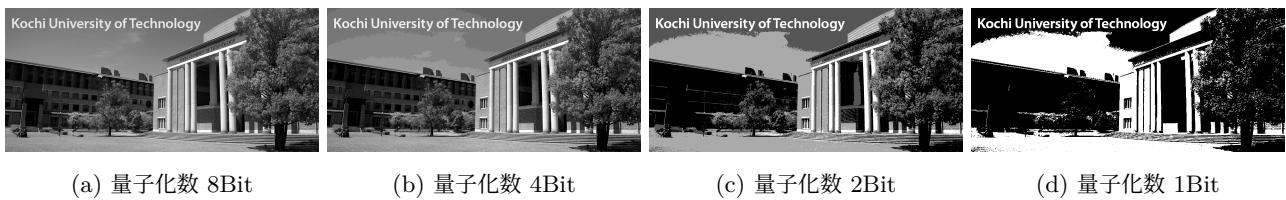


図 1-4: 画像の量子化数変換 実験結果

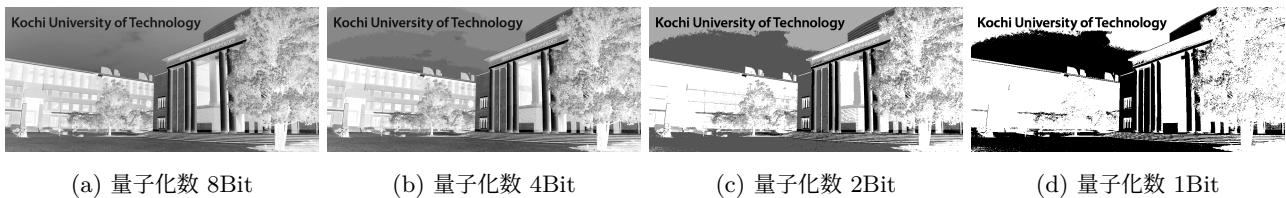


図 1-5: 階調反転 実験結果



図 1-6: 閾値処理 実験結果

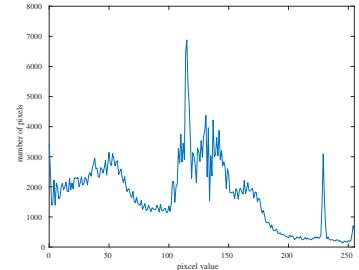


図 1-7: 画素値ヒストグラム*



図 1-8: 背景差分 実験結果



図 1-9: 背景差分画像の閾値処理



図 1-10: カラーチャネル操作 実験結果

■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。

*図 1-4(a) の画素値ヒストグラム



図 1-11: 平滑化フィルタ適用後の画像

図 1-12: メディアンフィルタ適用後の画像

■Sobel フィルタ, Laplacian フィルタ 横微分フィルタを適用すると、縦方向のエッジが強調され、縦方向微分フィルタを適用すると、横方向のエッジが強調された。これらの画像を足し合わせて、255 を上回る値を処理すると、画像全体のエッジが強調された画像を生成できた。Laplacian フィルタを適用すると、画像が暗く出力された。この画像に対して図 1-14(c)を用いて、閾値 30 の閾値処理し、全体画像のエッジが強調された画像を生成できた。



図 1-13: Sobel フィルタの適用

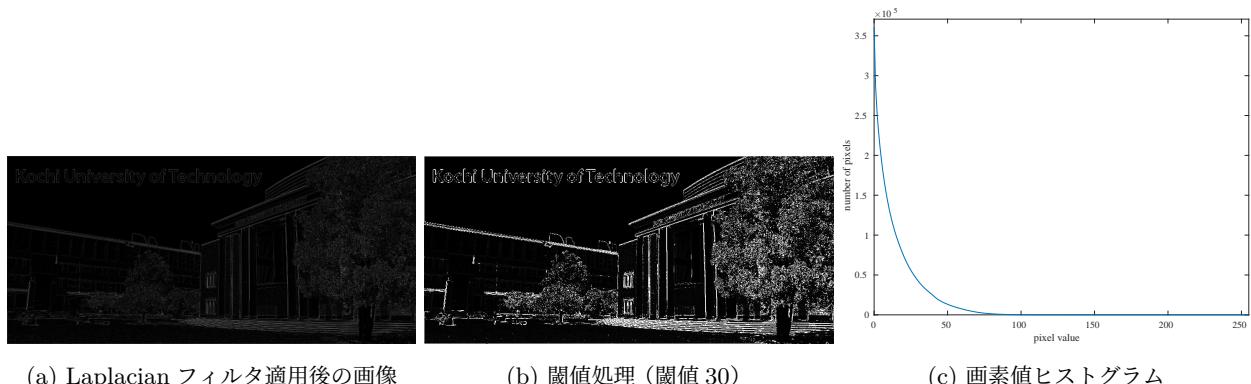


図 1-14: Laplacian フィルタの適用と処理

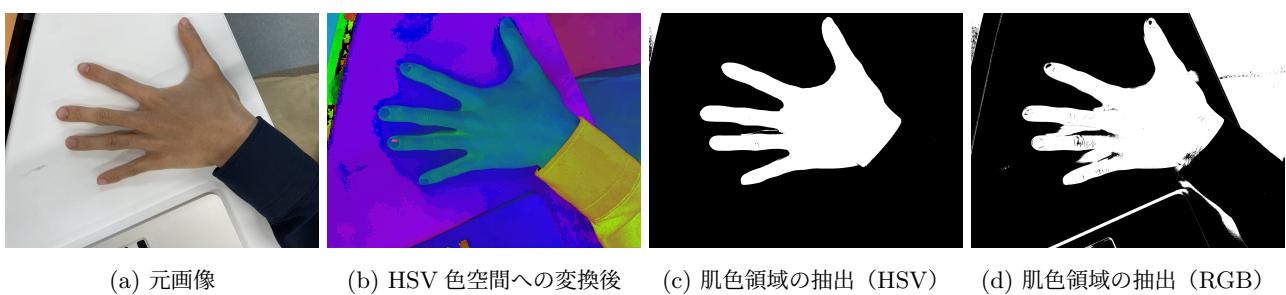


図 1-15: 色空間変換 実験結果

*横方向 Sobel フィルタ適用後画像と縦方向 Sobel フィルタ適用後画像の和。

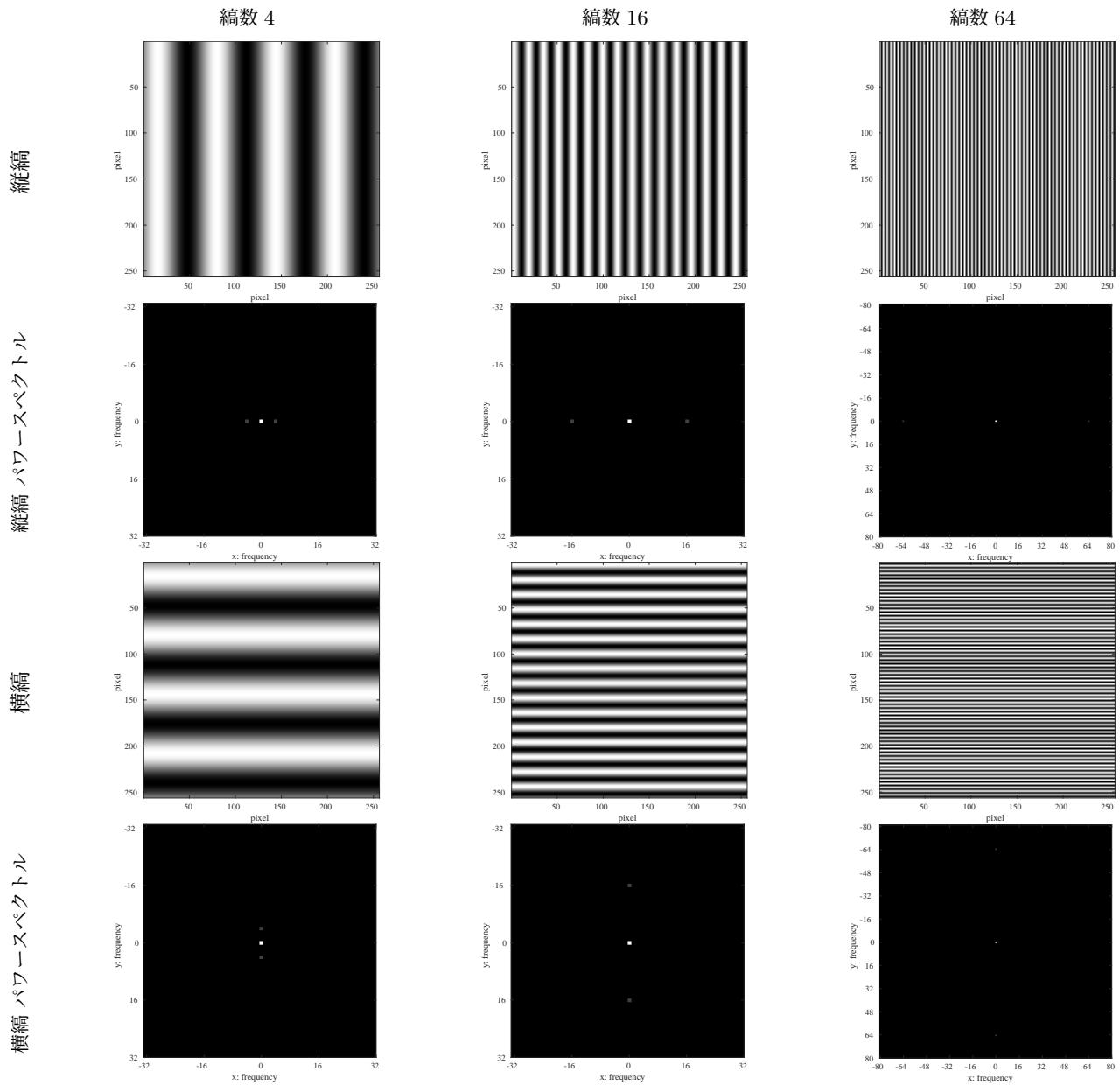


図 1-16: 縦縞・横縞に対するフーリエ変換

■2次元フーリエ変換 長方形の座標による、対数をとったパワースペクトルの違いは、目視で確認できなかった。ただし、両パワースペクトルの差分を取りた行列の最大値、最小値を調べると、いずれも 0 ではなかった。

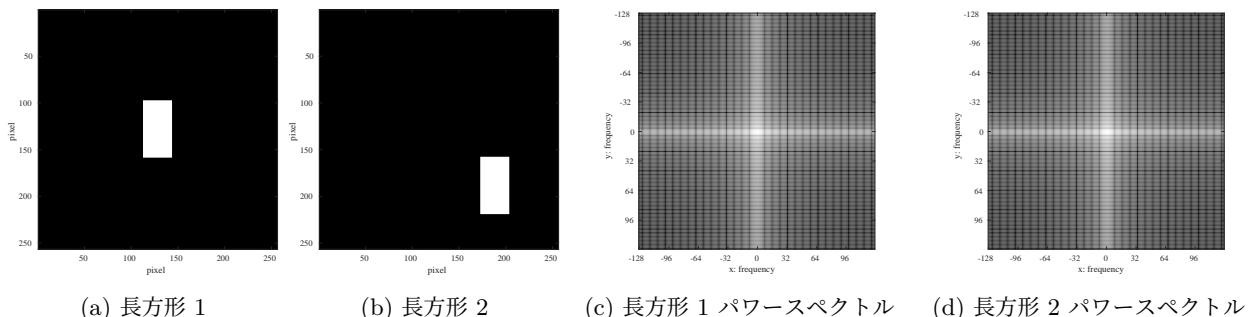


図 1-17: 画像の座標とパワースペクトルの関係

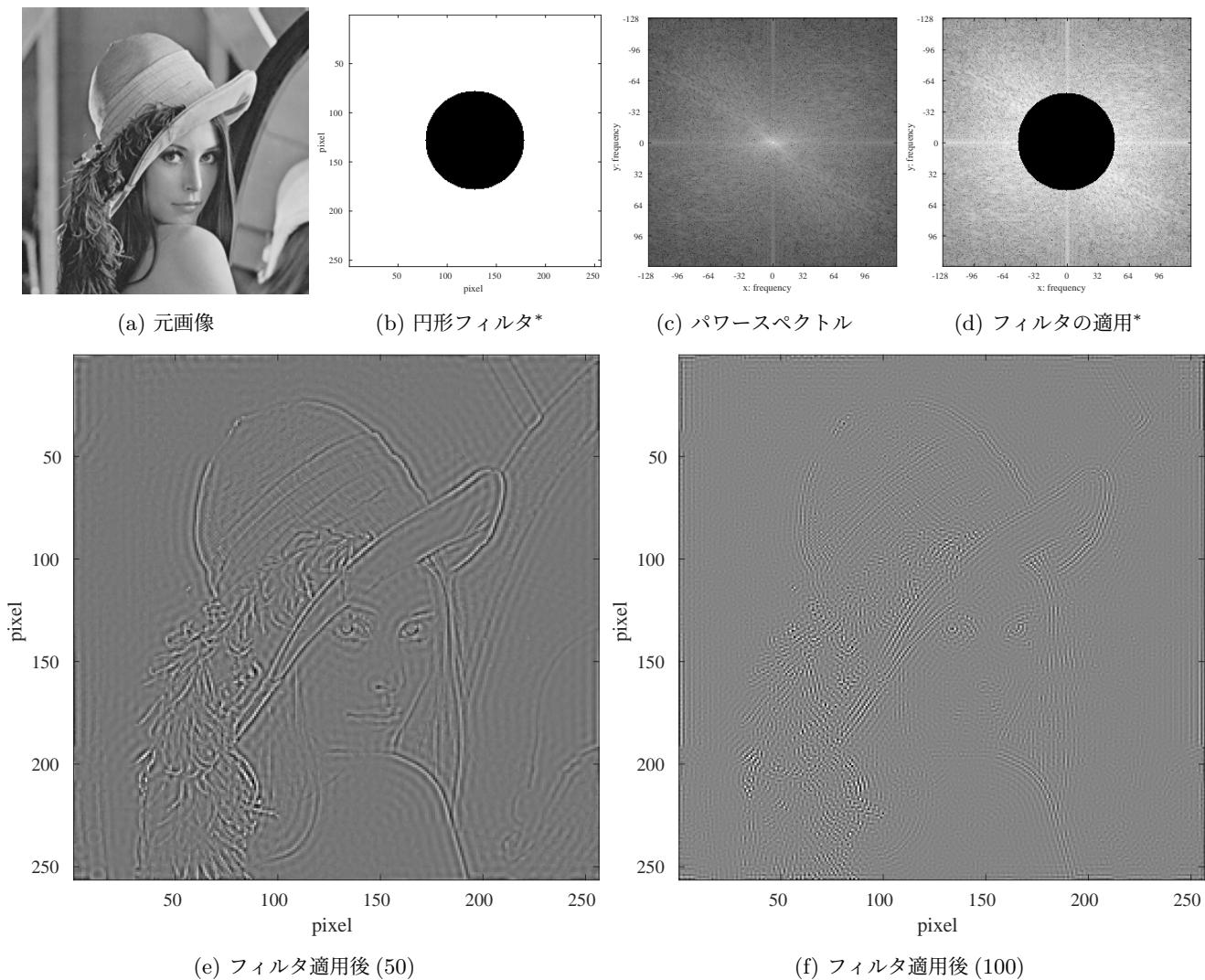


図 1-18: 高域通過フィルタ

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イ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。

■量子化数変換・階調変換・閾値処理 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。量子化数が減ると擬似輪郭が生じる。当然、この擬似輪郭は階調反転しても変わらない。さらに、閾値を調節することで、擬似輪郭を境に明暗がはっきりと分かれた。

■画像処理フィルタ メディアンフィルタは、ノイズに強いことが分かる。これは中央値を算出することにより、ノイズでない値が中央値となるからである。ただし、画素値の中央値を抽出するため、平均値を算出する平滑化フィルタに比べて、計算量が多い。それに対して、平滑化フィルタは、ノイズの画素値を含めた値の平均を算出するので、メディアンフィルタに比べるとノイズに弱い。Sobel フィルタは、画像の隣接画素間の差分を計算する。よって、差が激しい画素の画素値が大きくなり、白く表示される。つまり、ノイズに対してもエッジが強調される。Laplacian フィルタは、画像の2次微分を計算

*例として、半径が 50pixel のフィルタを示す。

することでエッジの位置を抽出している。Laplacian フィルタは高周波成分を增幅するため、ノイズに対してもエッジが強調される。

■背景差分画像・色空間変換 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。今回の実験では、閾値を手探りで探したため、実験結果の閾値が最適か分からなかった。また、実験結果より、強調された部分は、光の当たり方や影の変化で白く光る部分もある。ゆえに、背景差分画像だけでは、被写体の輪郭を正確にとらえることができない。HSV 色空間では、この問題を解決できた。RGB 色空間では、明暗（影や光の状況）で RGB 値が変わるのでに対して、HSV 色空間では、明暗が「明度」というチャネルで保存されている。HSV 色空間での肌色領域抽出は、光の状況や影に左右されなかつたため肌色領域をより正確に抽出できたと考えられる。

■2 次元フーリエ変換 2 次元フーリエ変換により得られるパワースペクトルは、中央付近に低周波数、その周辺に高周波成分が分布する。パワースペクトルの輝度は、含まれている空間周波数成分の量を示す。正弦波縞パワースペクトルの結果より、図 1-19 の結果と一致している。縦縞には横方向に明暗が存在し、縦方向の輝度は変わらないので、パワースペクトルは横方向に存在する。横縞には縦方向の明暗が存在し、横方向の輝度は変わらないので、パワースペクトルは縦方向に存在する。さらに、縞の数が多い、つまり空間周波数が高いと、パワースペクトルは外側が明るくなることも、実験結果と一致する。一方、実験結果より、画像座標とパワースペクトル座標は、依存関係がないことがわかる。これは式 (1.3) からも、点 (x, y) の座標値 x, y が、点 (u, v) の座標値 u, v に影響を与えないことが分かる。同じ大きさの長方形の位置によってパワースペクトルは、完全一致はしないものの、大きく異なることが実験結果から分かる。また、パワースペクトルの中心は、直流成分である。直流成分は、式 (1.4) より、画像全体の画素値を足し合わせたものである。パワースペクトルを確認すると、中心が明るくなっている。これは、縞や長方形の直交成分であろう。

■高域通過フィルタ 低周波領域にフィルタをかけて、高周波領域のみを通過させるフィルタを適用すると、縦横方向のエッジの間隔が狭い領域を確認できた。具体的には、髪の毛や目、帽子や口などを確認できた。円形フィルタの半径を大きくすると、さらにエッジの間隔が狭い部分のみ確認できた。具体的に、フィルタ半径 50pixel の画像で確認できた帽子や口は、フィルタ半径 100pixel の画像では確認できなかった。

1.5 結論

今回の実験では、画像フィルタの用いたノイズの除去や、画像のエッジを抽出した。メディアンフィルタは平滑化フィルタに比べてノイズをよく取り除けることが分かった。

また、被写体の抽出をするために、画像の背景差分をとる手法と、HSV 色空間上の肌色領域を抽出する手法を実験した。結果より、HSV 色空間では「明度」の指標があることから RGB 色空間や背景差分を取る手法に比べて、照明の状態に左右されないことが分かった。

2 次元フーリエ変換では、パワースペクトルと、画像の関係を明らかにした。画像座標と、パワースペクトル座標は関係ないことが明らかになった。また、高域通過フィルタを適用し、低周波数領域を削除した。逆フーリエ変換により、エッジの感覚が狭い部分のみが抽出できた。今回の実験では、フィルタにより直行成分まで削除されている。直行成分が画像上でどのような意味を成すか、実験を通じた考察は今後の課題したい。

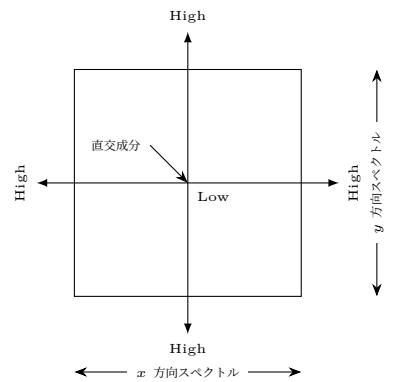


図 1-19: 周波数スペクトル

第2章

視覚情報処理 方位残効

2.1 実験の目的

運動残効は以下のように説明されている。

“ある一定方向への運動をしばらく注視した後に、静止した物体を観測すると、それが逆方向に動いて見える現象。”
[4, p.58]

方位残効とは、ある方向に傾いた線分を眺めて順応した後、テスト刺激として垂直の線分を見ると、順応刺激と反対の方向に傾いて知覚される現象である [5, p.5]. また、方位選択性とは、大脳皮質視覚野細胞は受容野に特定の傾き（方位）線分刺激を提示したときに応答し、それと異なる傾きの刺激には応答しないことをいう [4, p.764].

今回の実験では、方位の大きさと、残効の度合いを比較し、視覚情報処理を考慮したうえで、現象について考察する。

2.2 実験の方法と考え方

■正弦波縞の作成 方位を定義できる正弦波縞を作成する。方位 θ , 空間周波数 f , コントラスト C の正弦波縞は式 (2.1) で表せる。 $L(x, y)$ を点 $L(x, y)$ における輝度, L_0 を全体画像の平均輝度とする。

$$L(x, y) = L_0 \left(1 + C \sin \left(2\pi f(y \sin(\theta) + x \cos(\theta)) \right) \right) \quad (2.1)$$

y 軸方向を基準とし、左に 30° 傾いた空間周波数 0.05cycle/pixel の正弦波縞と、右に 60° 傾いた空間周波数 0.03cycle/pixel の正弦波を作成する。平均輝度は、最大輝度の 0.5 倍、コントラストを 0.5 とする。

■方位残効のデモ作成 方位 $90 \pm 10^\circ$, $90 \pm 45^\circ$ 正弦波縞の順応刺激を上下に提示して、60 秒後に垂直縞 90° のテスト刺激を表示する。全体画像サイズを 縦 900 × 横 400 pixel, 格子縞のサイズを 400 × 400 pixel, 順応時に提示する中央の矩形を 20×100 pixel, 順応後の注視点を 20 pixel とする。正弦波縞の空間周波数を 0.03cycle/pixel, コントラストは 0.5 とする。刺激を観察する際、網膜上の局所的な明るさの順応を避けるために、60 秒の順応時間では矩形を満遍なく見るようにする。

src. 2-1: 矩形と円の作成

```
[w, h] = meshgrid(-199:200, 50:-1:-49);
Y2 = 255*0.95;
cir = (w.^2 + h.^2 >= 10.^2)*Y2; % 円
rct = ones(100,400); % 矩形
rct = rct * Y2;
rct(50-9:50+10,200-49:200+50)=0;
```

src. 2-2: 刺激画像の表示方法

```
fig = figure;
set(fig, 'position', get(0, 'ScreenSize'));
colormap(gray(256));
image(dg_10);
axis off;
axis image;
```

■刺激画像の表示 順応刺激画像を方位残効が生じやすいように、全画面表示にし、60秒後にテスト刺激を提示するためには、MATLAB[®]の `set` 関数を用いる。また、60秒の待機時間を `pause(60)` で設定する。`axis` を設定して、画像に軸や数値ラベルを非表示にする。

2.3 実験の結果

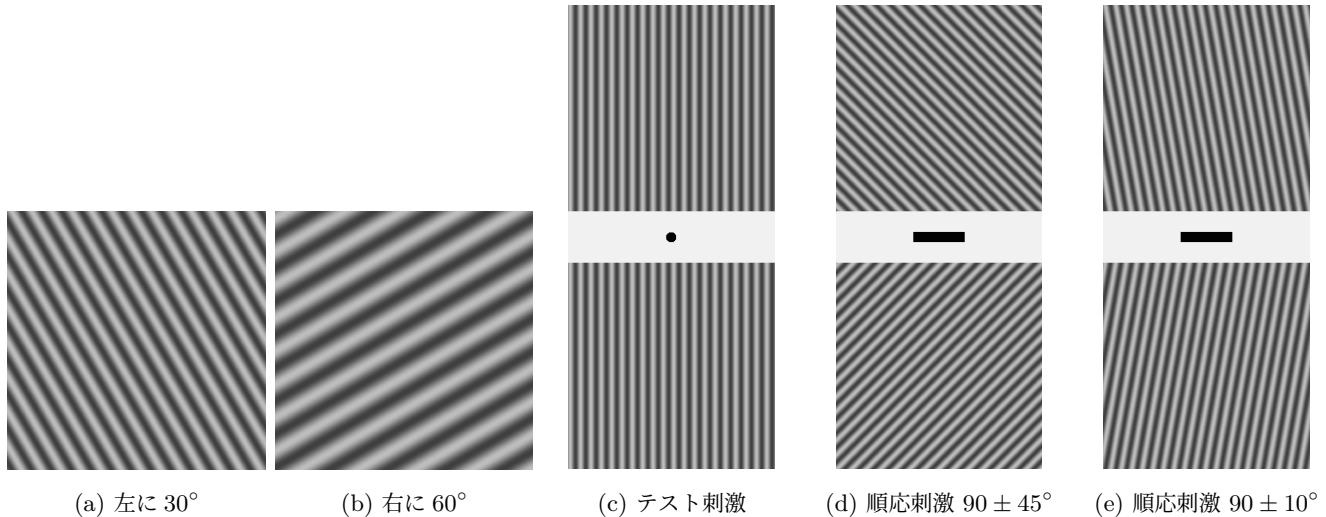


図 2-1: 生成した画像

■方位残効 順応刺激 $90 \pm 10^\circ$ は、方位残効を知覚できた。順応刺激 $90 \pm 45^\circ$ は、方位残効を知覚できなかった。

2.4 考察

運動残効は、運動方位に選択的な細胞のスパイク頻度が高くなることで生じる。運動残効は運動を主に処理する MT 野 (V5) で生じる。方位残効のメカニズムも運動残効と同じメカニズムであるが、運動残効が MT 野で生じるのに対して、方位残効は 1 次視覚野 (V1) で生じる。垂直方向に反応する受容野に対して、順応刺激 $90 - 10^\circ$ の応答と感度を、図 2-2 に示す。垂直方向に反応する受容野について、 $90 \pm n^\circ$ 順応刺激の n が大きい場合、感度が抑制されても応答への影響は少ない。 n が小さい場合、その感度が抑制されると、応答が小さくなるため、方位残効を知覚しやすいと考えられる。

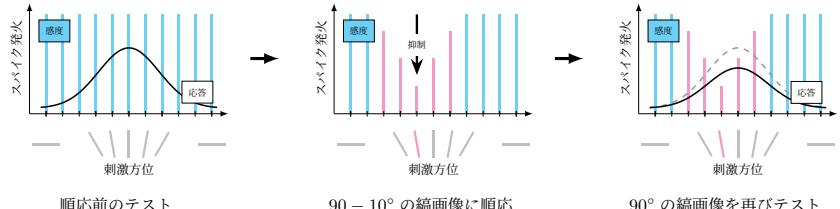


図 2-2: 方位残効のメカニズム（垂直方向に反応する受容野）

2.5 結論

方位残効が、運動残効と同じメカニズムであることを踏まえて、視覚特性を考慮しつつ考察した。順応刺激方位の大小と方位残効の度合いの関係について、視覚特性を根拠とし、明らかとなった。

第3章

関連語句

3.1 ガボールフィルタ

ガボールフィルタとは、画像を周波数領域でテクスチャ解析する方法のひとつ。テクスチャ解析とは、粗い、滑らか、でこぼこなどの直感的な材質を、画素強度の空間的な変動の関数として定量化することである[6]。テクスチャ解析する方法として、フーリエ変換がある。フーリエ変換では画像をいくつかのブロックに区切る必要があり、対象の境界も同時に検出するという問題に対して、結果が粗くなる。周波数の不正確さと、位置の不正確さとの積には下限があり、それを最小にするのは「ガボール関数」である。

ガボール関数は、サルの一次視覚野にある単純細胞の受容野特性によく似ており、ガボール関数の正弦波や余弦波の傾きを変化させることで、この受容野特性をよく再現できる。

[4, p.144]

3.2 固有顔法

固有顔(eigenface)とは、顔画像の認識において最も有名な手法のひとつ。顔認識では、顔を構成する部品(目や鼻、口など)の形状や、配置から特徴点を抽出して認識に利用する。しかし、照明方向や、撮影距離、表情、顔の傾きなどで顔の見え方が変わる。これらを許容したうえで正しく顔認識するには、顔画像をパターンとして扱い、統計的パターン認識手法を適用する方法がある。まず挙げられるのが、パターン間のマッチングを用いた方法である。この方法は最も簡単なパターン認識手法であるが、画像そのものをパターンとして扱った場合、パターンの次元が大きくなる。この問題を解決する方法として提案されたのが、固有顔である。固有顔は主成分分析によりパターンを情報圧縮し、顔画像の識別に利用している。数枚の画像に対して、各画像から平均ベクトルを引いた集合に対して、固有値を求める。この時の固有ベクトルを固有顔と呼ぶ。

[7]

3.3 オプティカルフロー

視覚におけるオプティカルフローとは、前方へ移動するときの、後ろへ流れる背景のことである。このオプティカルフローには2種類の特性があり、身体が対象へ直線的な移動時に起きる「放射状オプティカルフロー」、身体が回るときに起きる「回転性のオプティカルフロー」が存在する[8, p.679]。

また、動画像処理におけるオプティカルフローとは、動画像中の物体の動きを検出して、速度をベクトルで表示する手法を指す。フロー推定方法としてブロックマッチング法を取り上げる。ブロックマッチング法は、画像のある大きさのブロックで分割し、次フレームにおける注目ブロックとの類似度が最も大きいブロックを検出する手法である[9]。

3.4 ストラクチャfromモーション (SfM: Structure from Motion)

ストラクチャfromモーション (SfM) とは、一連の2次元イメージから、3次元シーンの構造を推定するプロセスを指す。具体的には、ドローンによる空撮写真(2次元)から3次元のデータを得るときに用いられる。ここでは、例として2つのビュー(カメラ)によるSfMを取り上げる。カメラ1で読み取った画像と、カメラ2で読み取った画像の対応関係を抽出した後、カメラ1に対するカメラ2の姿勢を求める。このときに、基礎行列を計算し、エピポーラ幾何を記述する(図3-1)。生成したエピポーラ線に対して、基礎行列を計算し、カメラ1座標におけるのカメラ2座標を出力する。最後に、マッチする点の、3次元での位置を決定し、3次元画像として描画する。

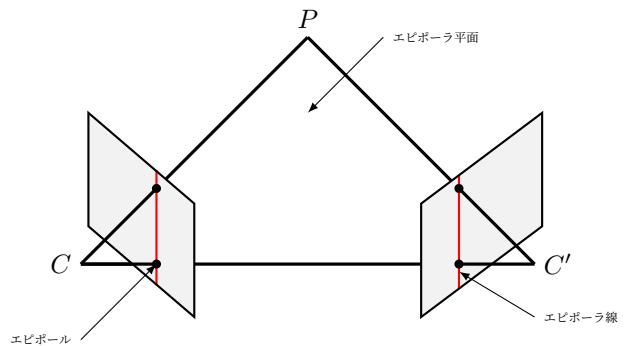


図3-1: エピポーラ幾何

[10]

参考文献

- [1] 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.
- [2] 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.
- [3] MathWorks. rgb2hsv. <https://jp.mathworks.com/help/matlab/ref/rgb2hsv.html#d124e1239127>, Confirmation date: May 13th, 2023.
- [4] 日本認知科学会編. 認知科学辞典. 共立出版, 2002.
- [5] 中島悠介. 方位残効と運動残効のメカニズム. PhD thesis, 早稲田大学, 2017.
- [6] MathWorks. テクスチャ解析. <https://jp.mathworks.com/help/images/texture-analysis-1.html#>, Confirmation date: May 22th, 2023.
- [7] 栗田多喜夫, 長谷川修. 顔画像からの個人識別. 映像情報メディア学会誌, Vol. 51, No. 8, pp. 1132–1135, 1997.
- [8] 宮本省三, 八坂一彦, 平谷 尚大他. 人間の運動学 —ヒューマン・キネシオロジー—. 協同医書出版社, 2016.
- [9] 久保山裕, 三田長久, 吉岡俊英. オプティカルフローを用いた画像中の野鳥検出. In *IEICE Conferences Archives*. The Institute of Electronics, Information and Communication Engineers, 2007.
- [10] MathWorks. structure form motion の概要. <https://jp.mathworks.com/help/vision/ug/structure-from-motion.html>, Confirmation date: May 25th, 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);

```

付録

```
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;
```

```

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);

```

付録

```
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
```

```

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');

```

付録

```
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: Laplacian 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');
```

src. B-5: 色空間変換 HSV 色空間

```

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 >= 113 & r <= 160);
7 bin_g = (g >= 81 & g <= 133);
8 bin_b = (b >= 77 & b <= 98);
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_HSV

```

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

```

付録

```
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
```

src. B-8: 色空間変換 fucntion_RGB

```
1 function [BW,maskedRGBImage] = no5_rgbfunc(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 14-May-2023
10 %-----
11
12
13 % Convert RGB image to chosen color space
14 I = RGB;
15
```

```

16 % Define thresholds for channel 1 based on histogram settings
17 channel1Min = 113.000;
18 channel1Max = 160.000;
19
20 % Define thresholds for channel 2 based on histogram settings
21 channel2Min = 81.000;
22 channel2Max = 133.000;
23
24 % Define thresholds for channel 3 based on histogram settings
25 channel3Min = 33.000;
26 channel3Max = 98.000;
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 視覚情報処理 方位残効 (May 11th, 2023)

src. C-1: 正弦波縞の作成

```

1 clear; close all;
2 C = 0.5; % コントラスト (今回は0.5)
3 f = 0.05; % 空間周波数 (今回は0.05)
4 f1 = 0.03; % 空間周波数 (今回は0.03)
5 d1 = (-1/6)*pi;
6 d2 = (1/3)*pi;
7 L_0 = 128;
8 height = 300;
9 width = 300;
10 L_130 = zeros(height,width);
11 L_r60 = zeros(height,width);
12
13
14 for x = 1:width
15   for y = 1:height
16     L_130(y,x) = L_0 * (1 + C * sin(2 * pi * f*(y*sin(d1)+x*cos(d1))));
17     L_r60(y,x) = L_0 * (1 + C * sin(2 * pi * f1*(y*sin(d2)+x*cos(d2))));
18   end
19 end
20
21
22 fig0 = figure;
23 colormap(gray(256));
24 image(L_130);
25 axis off;

```

付録

```
26 axis image;
27 exportgraphics(fig0 , '../Figures/07_10_l30.pdf','ContentType','vector');
28
29 fig1 = figure;
30 colormap(gray(256));
31 image(L_r60);
32 axis off;
33 axis image;
34 exportgraphics(fig1 , '../Figures/07_11_r60.pdf','ContentType','vector');
```

src. C-2: 方位残効のデモ作成

```
1 clear; close all;
2 C = 0.5; % コントラスト (今回は0.5)
3 f = 0.05; % 空間周波数 (今回は0.05)
4 d1 = (-1/6)*(1/3)*pi; % 10度
5 d2 = (1/4)*pi; % 45度
6 d0 = (0)*pi;
7 L_0 = 128;
8 height = 400;
9 width = 400;
10 L_p10 = zeros(height,width);
11 L_n10 = zeros(height,width);
12 L_p45 = zeros(height,width);
13 L_n45 = zeros(height,width);
14 L = zeros(height,width);
15
16 for x = 1:width
17     for y = 1:height
18         L_p10(y,x) = L_0 * (1 + C * sin(2 * pi * f*(y*sin(d1)+x*cos(d1)))); 
19         L_n10(y,x) = L_0 * (1 + C * sin(2 * pi * f*(y*sin(-1*d1)+x*cos(-1*d1)))); 
20         L_p45(y,x) = L_0 * (1 + C * sin(2 * pi * f*(y*sin(d2)+x*cos(d2)))); 
21         L_n45(y,x) = L_0 * (1 + C * sin(2 * pi * f*(y*sin(-1*d2)+x*cos(-1*d2)))); 
22         L(y,x) = L_0 * (1 + C * sin(2 * pi * f*(y*sin(d0)+x*cos(d0)))); 
23     end
24 end
25
26 [w, h] = meshgrid(-199:200, 50:-1:-49);
27 Y2 = 255*0.95;
28 cir = (w.^2 + h.^2 >= 10.^2)*Y2;
29
30 rct = ones(100,400);
31 rct = rct * Y2;
32 rct(50-9:50+10,200-49:200+50)=0;
33
34 dg_90 = [L; cir; L];
35 dg_10 = [L_p10; rct; L_n10];
36 dg_45 = [L_n45; rct; L_p45];
37
38 fig10 = figure;
39 set(fig10, 'position',get(0,'ScreenSize'));
40 colormap(gray(256));
41 image(dg_10);
42 axis off;
43 axis image;
44 pause(60);
```

```

45 colormap(gray(256));
46 image(dg_90);
47 axis off;
48 axis image;
49
50 pause(5);
51 close();
52 colormap(gray(256));
53 image(dg_45);
54 axis off;
55 axis image;
56 pause(60);
57 colormap(gray(256));
58 image(dg_90);
59 axis off;
60 axis image;
61 pause(5);
62 close all;
63
64 % 保存用
65 fig90 = figure;
66 colormap(gray(256));
67 image(dg_90);
68 axis off;
69 axis image;
70 exportgraphics(fig90 , '../Figures/07_21_dg90.pdf','ContentType','vector');
71
72 fig45 = figure;
73 colormap(gray(256));
74 image(dg_45);
75 axis off;
76 axis image;
77 exportgraphics(fig45 , '../Figures/07_22_dg45.pdf','ContentType','vector');
78
79 fig10 = figure;
80 colormap(gray(256));
81 image(dg_10);
82 axis off;
83 axis image;
84 exportgraphics(fig10 , '../Figures/07_23_dg10.pdf','ContentType','vector');

```

D 画像のフーリエ変換・逆フーリエ変換 (May 15th, 2023)

src. D-1: 2 次元 FFT 1

```

1 clear; close all;
2 H = 256;
3 W = 256;
4 img4 = zeros(H,W);
5 img16 = zeros(H,W);
6 img64 = zeros(H,W);
7 for h = 1:H
8     for w = 1:W
9         img4(h,w) = 1 + sin((2*pi*w) / W*4);
10        img16(h,w) = 1 + sin((2*pi*w) / W*16);

```

付録

```
11     img64(h,w) = 1 + sin((2*pi*w) / W*64);
12 end
13 end
14 img4_fft = abs(fftshift(fft2(img4))).^2;
15 img16_fft = abs(fftshift(fft2(img16))).^2;
16 img64_fft = abs(fftshift(fft2(img64))).^2;
17
18 fig0 = figure;
19 colormap('gray');
20 imagesc(img4);
21 axis image;
22 xlabel('pixel');
23 ylabel('pixel');
24 exportgraphics(fig0,'../Figures/08_11_img4.pdf','ContentType','vector');
25 fig0 = figure;
26 colormap('gray');
27 imagesc(img16);
28 axis image;
29 xlabel('pixel');
30 ylabel('pixel');
31 exportgraphics(fig0,'../Figures/08_12_img16.pdf','ContentType','vector');
32 fig0 = figure;
33 colormap('gray');
34 imagesc(img64);
35 axis image;
36 xlabel('pixel');
37 ylabel('pixel');
38 exportgraphics(fig0,'../Figures/08_13_img64.pdf','ContentType','vector');
39
40 fig0 = figure;
41 colormap('gray');
42 imagesc(img4_fft);
43 axis image;
44 xticks(1:16:256);
45 yticks(1:16:256);
46 xticklabels(-128:16:127);
47 yticklabels(-128:16:127);
48 xlabel('x: frequency');
49 ylabel('y: frequency');
50 axis([128-32 128+32+1 128-32 128+32+1]);
51 exportgraphics(fig0,'../Figures/08_14_img4-fft.pdf','ContentType','vector');
52 fig0 = figure;
53 colormap('gray');
54 imagesc(img16_fft);
55 axis image;
56 xticks(1:16:256);
57 yticks(1:16:256);
58 xticklabels(-128:16:127);
59 yticklabels(-128:16:127);
60 xlabel('x: frequency');
61 ylabel('y: frequency');
62 axis([128-32 128+32+1 128-32 128+32+1]);
63 exportgraphics(fig0,'../Figures/08_15_img16-fft.pdf','ContentType','vector');
64 fig0 = figure;
65 colormap('gray');
66 imagesc(img64_fft);
```

```

67 axis image;
68 xticks(1:16:256);
69 yticks(1:16:256);
70 xticklabels(-128:16:127);
71 yticklabels(-128:16:127);
72 xlabel('x: frequency');
73 ylabel('y: frequency');
74 axis([128-80 128+80+1 128-80 128+80+1]);
75 exportgraphics(fig0,'../Figures/08_16_img64-fft.pdf','ContentType','vector');
76
77 % 直交成分削除
78 img_fft = fftshift(fft2(img4));
79 img_fft(129,129) = 0;
80 img_fft = abs(img_fft).^2;
81 figure;
82 colormap('gray');
83 imagesc(img_fft);
84 axis image;
85 xticks(1:16:256);
86 yticks(1:16:256);
87 xticklabels(-128:16:127);
88 yticklabels(-128:16:127);
89 xlabel('x: frequency');
90 ylabel('y: frequency');
91 axis([128-32 128+32+1 128-32 128+32+1]);
92 fig0 = figure;
93 colormap('gray');
94 imagesc(real(ifft2(ifftshift(img_fft))));
95 axis image;

```

src. D-2: 2 次元 FFT 2

```

1 clear; close all;
2 H = 256;
3 W = 256;
4 img4 = zeros(H,W);
5 img16 = zeros(H,W);
6 img64 = zeros(H,W);
7 for h = 1:H
8     for w = 1:W
9         img4(h,w) = 1 + sin((2*pi*h) / H*4);
10        img16(h,w) = 1 + sin((2*pi*h) / H*16);
11        img64(h,w) = 1 + sin((2*pi*h) / H*64);
12    end
13 end
14 img4_fft = abs(fftshift(fft2(img4))).^2;
15 img16_fft = abs(fftshift(fft2(img16))).^2;
16 img64_fft = abs(fftshift(fft2(img64))).^2;
17
18 fig0 = figure;
19 colormap('gray');
20 imagesc(img4);
21 axis image;
22 xlabel('pixel');
23 ylabel('pixel');
24 exportgraphics(fig0,'../Figures/08_21_img4.pdf','ContentType','vector');

```

付録

```
25 fig0 = figure;
26 colormap('gray');
27 imagesc(img16);
28 axis image;
29 xlabel('pixel');
30 ylabel('pixel');
31 exportgraphics(fig0,'../Figures/08_22_img16.pdf','ContentType','vector');
32 fig0 = figure;
33 colormap('gray');
34 imagesc(img64);
35 axis image;
36 xlabel('pixel');
37 ylabel('pixel');
38 exportgraphics(fig0,'../Figures/08_23_img64.pdf','ContentType','vector');
39
40 fig0 = figure;
41 colormap('gray');
42 imagesc(img4_fft);
43 axis image;
44 xticks(1:16:256);
45 yticks(1:16:256);
46 xticklabels(-128:16:127);
47 yticklabels(-128:16:127);
48 xlabel('x: frequency');
49 ylabel('y: frequency');
50 axis([128-32 128+32+1 128-32 128+32+1]);
51 exportgraphics(fig0,'../Figures/08_24_img4-fft.pdf','ContentType','vector');
52 fig0 = figure;
53 colormap('gray');
54 imagesc(img16_fft);
55 axis image;
56 xticks(1:16:256);
57 yticks(1:16:256);
58 xticklabels(-128:16:127);
59 yticklabels(-128:16:127);
60 xlabel('x: frequency');
61 ylabel('y: frequency');
62 axis([128-32 128+32+1 128-32 128+32+1]);
63 exportgraphics(fig0,'../Figures/08_25_img16-fft.pdf','ContentType','vector');
64 fig0 = figure;
65 colormap('gray');
66 imagesc(img64_fft);
67 axis image;
68 xticks(1:16:256);
69 yticks(1:16:256);
70 xticklabels(-128:16:127);
71 yticklabels(-128:16:127);
72 xlabel('x: frequency');
73 ylabel('y: frequency');
74 axis([128-80 128+80+1 128-80 128+80+1]);
75 exportgraphics(fig0,'../Figures/08_26_img64-fft.pdf','ContentType','vector');
```

src. D-3: 2 次元 FFT 3

```
1 clear; close all;
2 H = 256;
```

```

3 W = 256;
4 img1 = zeros(H,W);
5 img1(H/2-30:H/2+30,W/2-15:W/2+15) = 255;
6 fig0 = figure;
7 imagesc(img1);
8 colormap('gray');
9 axis image;
10 xlabel('pixel');
11 ylabel('pixel');
12 exportgraphics(fig0,'../Figures/08_31_rec1.pdf','ContentType','vector');
13 img2 = zeros(H,W);
14 img2(H/2+60-30:H/2+60+30,W/2+60-15:W/2+60+15) = 255;
15 fig0 = figure;
16 imagesc(img2);
17 colormap('gray');
18 axis image;
19 xlabel('pixel');
20 ylabel('pixel');
21 exportgraphics(fig0,'../Figures/08_32_rec2.pdf','ContentType','vector');
22
23 img1_fft = log(1 + abs(fftshift(fft2(img1))).^2);
24 img2_fft = log(1 + abs(fftshift(fft2(img2))).^2);
25
26 fig0 = figure;
27 colormap('gray');
28 imagesc(img1_fft);
29 axis image;
30 xticks(1:32:256);
31 yticks(1:32:256);
32 xticklabels(-128:32:127);
33 yticklabels(-128:32:127);
34 xlabel('x: frequency');
35 ylabel('y: frequency');
36 exportgraphics(fig0,'../Figures/08_33_rec1-fft.pdf','ContentType','vector');
37 fig0 = figure;
38 colormap('gray');
39 imagesc(img2_fft);
40 axis image;
41 xticks(1:32:256);
42 yticks(1:32:256);
43 xticklabels(-128:32:127);
44 yticklabels(-128:32:127);
45 xlabel('x: frequency');
46 ylabel('y: frequency');
47 exportgraphics(fig0,'../Figures/08_34_rec2-fft.pdf','ContentType','vector');
48
49 diff = img2_fft - img1_fft;
50 diff_max = max(max(diff));
51 diff_min = min(min(diff));

```

src. D-4: 高域通過フィルタ

```

1 clear; close all;
2 img = imread('../LENNA.bmp');
3 img_size = size(img);
4 H = 256;

```

付録

```
5 W = 256;
6 filter_50 = ones(H,W);
7 filter_100 = ones(H,W);
8 c_x = H/2;
9 c_y = W/2;
10 for y = 1:H
11     for x = 1:W
12         d = sqrt((x-c_x)^2 + (y-c_y)^2);
13         if d <= 50
14             filter_50(y,x) = 0;
15         end
16         if d <= 100
17             filter_100(y,x) = 0;
18         end
19     end
20 end
21 fig0 = figure;
22 imagesc(filter_50);
23 colormap('gray');
24 axis image;
25 xlabel('pixel');
26 ylabel('pixel');
27 exportgraphics(fig0,'../Figures/08_41_filter.pdf','ContentType','vector');
28 img_fft = fftshift(fft2(img));
29 fig0 = figure;
30 colormap('gray');
31 imagesc(log(1 + abs(img_fft).^2));
32 axis image;
33 xticks(1:32:256);
34 yticks(1:32:256);
35 xticklabels(-128:32:127);
36 yticklabels(-128:32:127);
37 xlabel('x: frequency');
38 ylabel('y: frequency');
39 exportgraphics(fig0,'../Figures/08_42_fft.pdf','ContentType','vector');
40
41 img_filter50 = img_fft.*filter_50;
42 img_filter100 = img_fft.*filter_100;
43 img_ifilter50 = real(ifft2(ifftshift(img_filter50)));
44 img_ifilter100 = real(ifft2(ifftshift(img_filter100)));
45 fig0 = figure;
46 colormap('gray');
47 imagesc(log(1 + abs(img_filter50).^2));
48 axis image;
49 xticks(1:32:256);
50 yticks(1:32:256);
51 xticklabels(-128:32:127);
52 yticklabels(-128:32:127);
53 xlabel('x: frequency');
54 ylabel('y: frequency');
55 exportgraphics(fig0,'../Figures/08_43_fft-filter.pdf','ContentType','vector');
56 fig0 = figure;
57 colormap('gray');
58 imagesc(img_ifilter50);
59 axis image;
60 xlabel('pixel');
```

```
61 xlabel('pixel');
62 exportgraphics(fig0,'../Figures/08_44_fft-filter-50.pdf','ContentType','vector');
63 fig0 = figure;
64 colormap('gray');
65 imagesc(img_ifilter100);
66 axis image;
67 xlabel('pixel');
68 ylabel('pixel');
69 exportgraphics(fig0,'../Figures/08_45_fft-filter-100.pdf','ContentType','vector');
70
71 img_fft(128,128) = 0;
72 fig0 = figure;
73 colormap('gray');
74 imagesc(real(ifft2(ifftshift(img_fft))));
75 axis image;
```