

空間フィルタ (3)

複数の画像処理

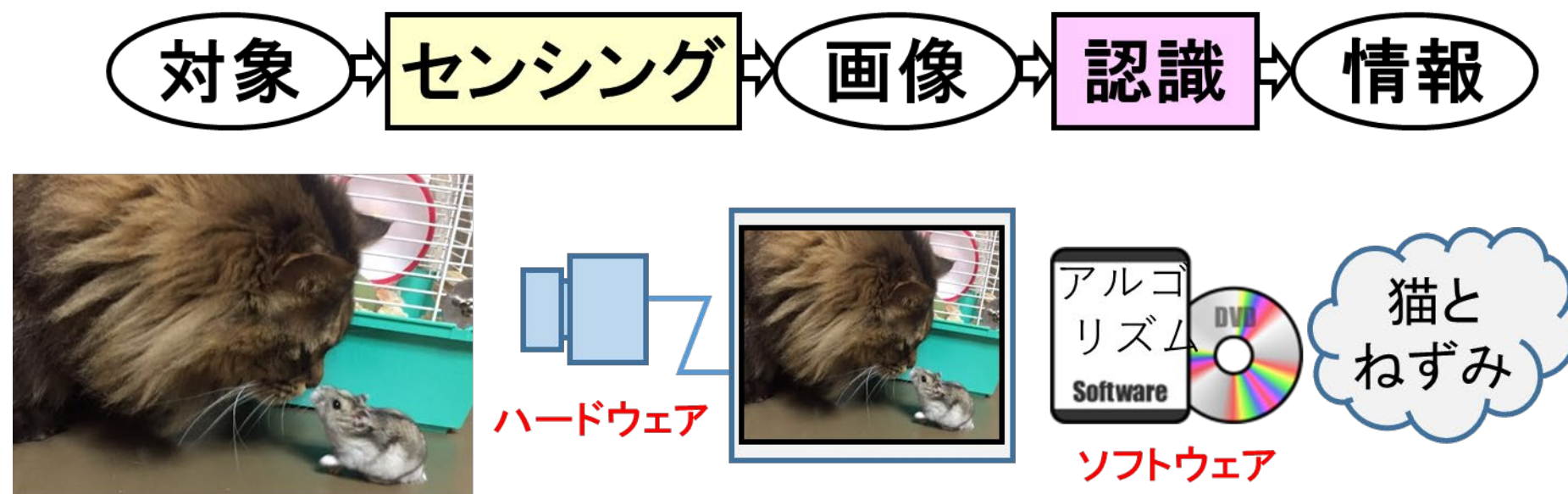
森本、澤野、塚田

本日の予定

- 第7回：空間フィルタ(3)
 - 講義：ラプラシアンフィルタ，メディアンフィルタ
複数の画像の利用
 - 演習：各フィルタの実装，間違い探し

画像処理

- 撮像が大切 きれいな画像を撮る
(ピントのあったコントラストのはっきりした)



- 前処理 (空間フィルタの活用)
⇒ 目的 (検査内容) に適した画像へと加工

空間フィルタ 選定と設定が重要

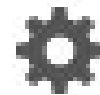
空間フィルタ

- 1.まずは、きれいな元画像（＝ピントの合ったコントラストのはっきりした）を撮像する。
その上で強調したい部分があれば、画像加工＝空間フィルタを使用する。
- 2.一般的な前処理とは、 3×3 の中心画素を様々な係数を適用した周辺濃度値を使って強調したい結果に置き換えることである。
- 3.それぞれの原理を知って用途ごとに効果のある前処理フィルタを使う、さらに組み合わせて使用すれば検査が安定する。

復習

- | | |
|----------|--------------------------------|
| ② 画素単位 | 画素へのアクセス、スキャン |
| ③ // | 階調変換 |
| ④ // | ヒストグラム |
| ⑤ 空間フィルタ | 平均化（平均値、ガウシアン） |
| ⑥ // | エッジ検出
（プリューウィット、ソーベル） |
| ⑦ // | ラプラシアン ⇒ エッジ検出
メジアン ⇒ ノイズ除去 |

ピエール=シモン・ラプラス は、フランスの数学者・物理学者・天文学者。「天体力学概論」(traité intitulé Mécanique Céleste)と「確率論の解析理論」という名著を残した。1789年にロンドン王立協会フェローに選出された。



ラプラシアンフィルタ (p. 55)

- 二次微分フィルタ (微分=差分を2回繰り返す)

cf : ソーベルフィルタ

(一次) 微分フィルタ

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

横方向

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

縦方向

- 関数 $f(x, y)$ のラプラシアンの定義

$$\frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$$

1次微分

X-1	X	X+1
-----	---	-----

$$I_x(x, y) = I(x + 1, y) - I(x, y)$$

$$I_y(x, y) = I(x, y + 1) - I(x, y)$$

Y-1
Y
Y+1

2次微分

$$\begin{aligned} I_{xx}(x, y) &= \{I(x + 1, y) - I(x, y)\} \\ &\quad - \{I(x, y) - I(x - 1, y)\} \\ &= I(x - 1, y) - 2I(x, y) + I(x + 1, y) \end{aligned}$$

$$\begin{aligned} I_{yy}(x, y) &= \{I(x, y + 1) - I(x, y)\} \\ &\quad - \{I(x, y) - I(x, y - 1)\} \\ &= I(x, y - 1) - 2I(x, y) + I(x, y + 1) \end{aligned}$$

ラプラシアンフィルタ (p. 55)

$$\frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$$

1次微分

X-1	X	X+1
-----	---	-----

$$I_x(x, y) = I(x + 1, y) - I(x, y)$$

$$I_y(x, y) = I(x, y + 1) - I(x, y)$$

Y-1
Y
Y+1

2次微分

$$\begin{aligned} I_{xx}(x, y) &= \{I(x + 1, y) - I(x, y)\} \\ &\quad - \{I(x, y) - I(x - 1, y)\} \\ &= I(x - 1, y) - 2I(x, y) + I(x + 1, y) \end{aligned}$$

$$\begin{aligned} I_{yy}(x, y) &= \{I(x, y + 1) - I(x, y)\} \\ &\quad - \{I(x, y) - I(x, y - 1)\} \\ &= I(x, y - 1) - 2I(x, y) + I(x, y + 1) \end{aligned}$$

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

=

0	1	0
1	-4	1
0	1	0

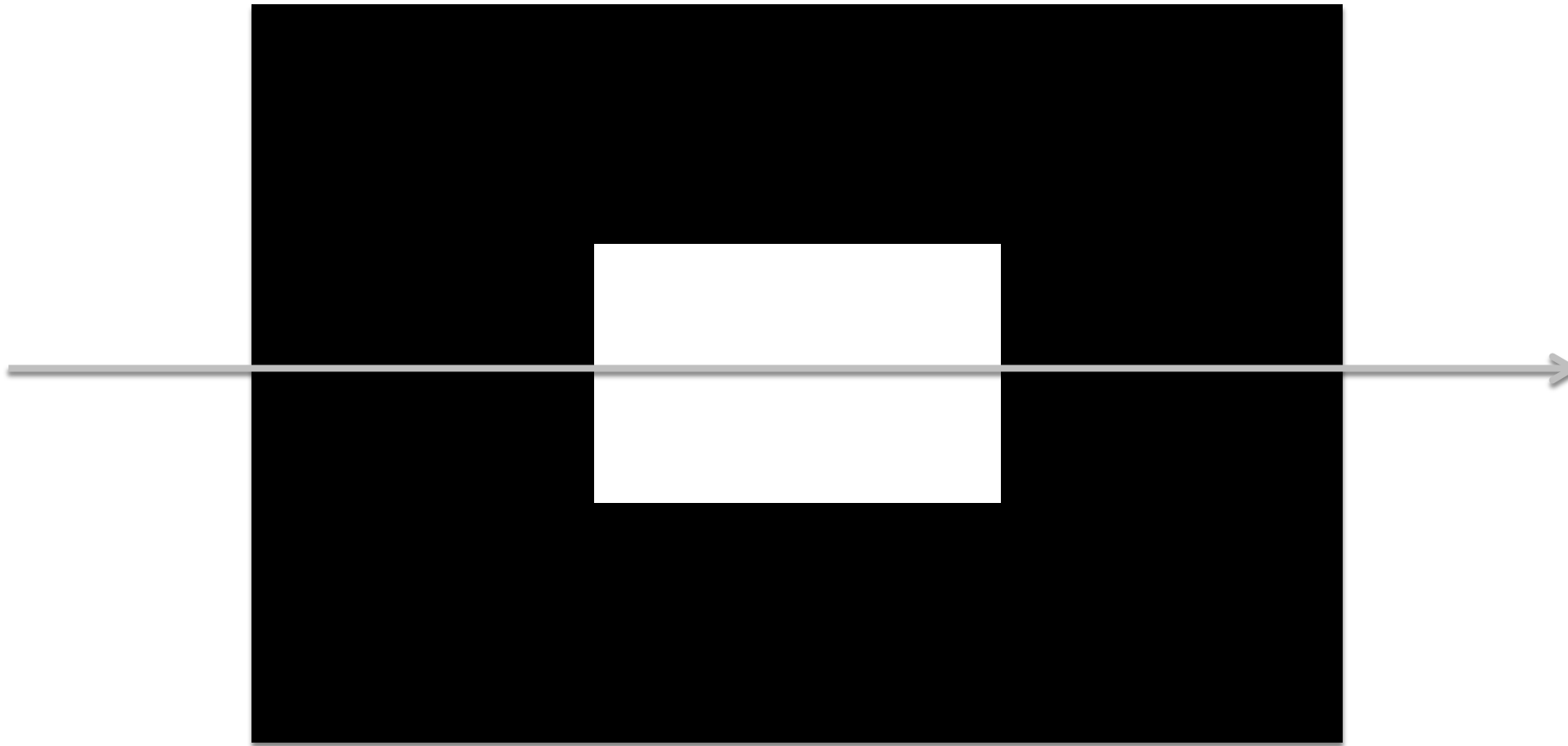
横方向

縦方向

4近傍

微分（差分）の概念

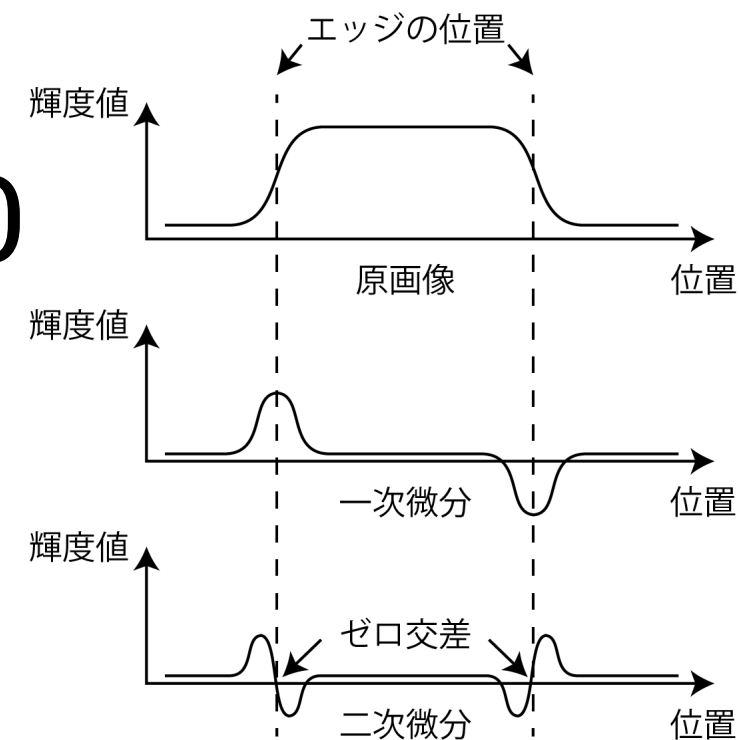
変化を見る



画素値 0 0 0 1 1 1 1 1 0 0 0

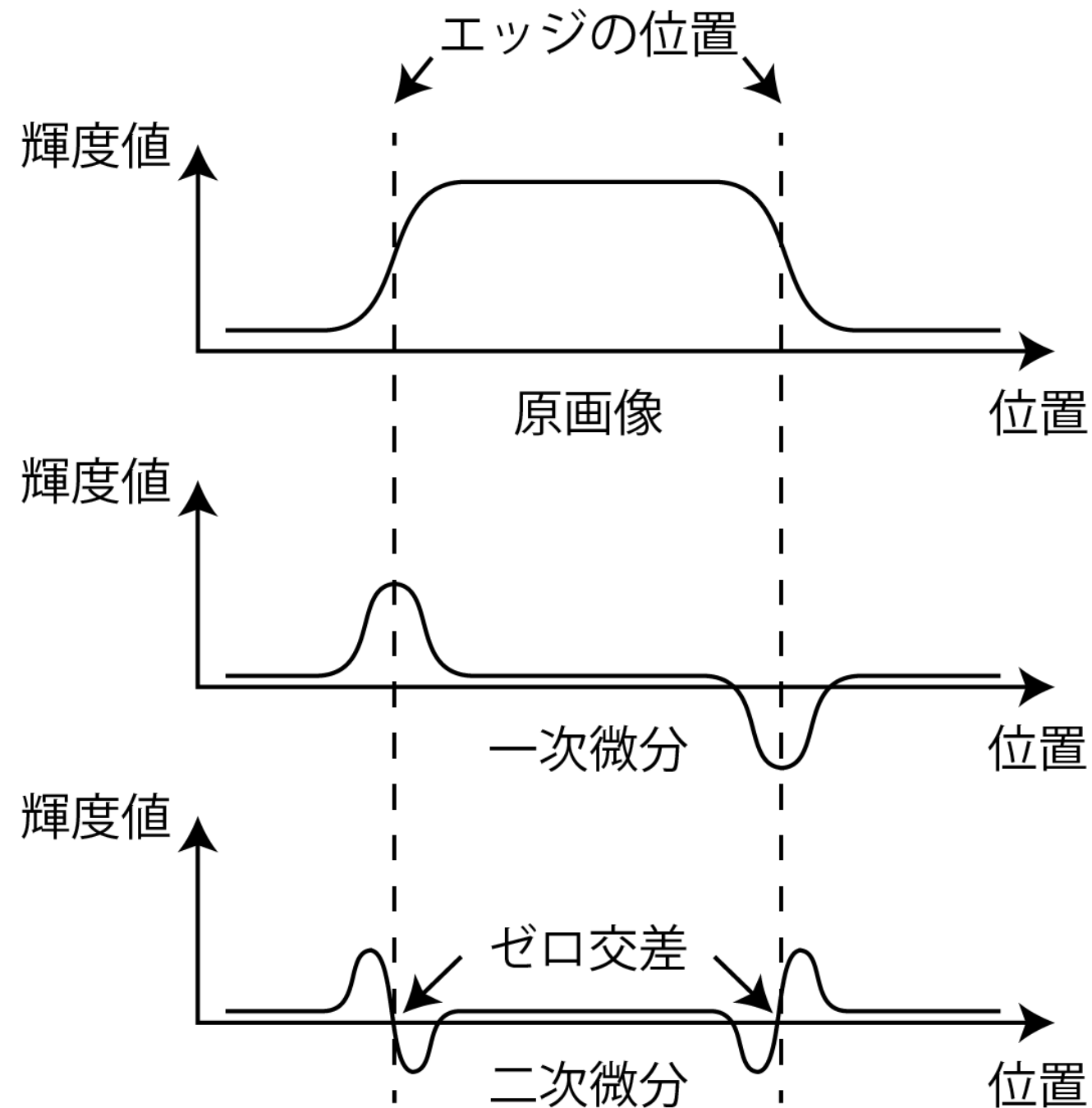
1次微分 0 0 1 0 0 0 0 -1 0 0

2次微分 0 1 -1 0 0 0 -1 1 0



ゼロ交差

- 二次微分によるエッジ強調



ラプラシアンフィルタ

- 二次微分を利用して画像から輪郭を抽出

0	0	0
1	-2	1
0	0	0

横方向

+

0	1	0
0	-2	0
0	1	0

縦方向

=

0	1	0
1	-4	1
0	1	0

4近傍

1	1	1
1	-8	1
1	1	1

8近傍

ラプラシアンフィルタ

0	1	0
1	-4	1
0	1	0

0	0	0	0	0	0
0	0	10	0	0	0
0	0	10	0	0	0
0	0	10	10	10	0
0	0	10	0	0	0
0	0	0	0	0	0

0	0	0	0	0	0
0					0
0					0
0					0
0					0
0	0	0	0	0	0

ラプラシアンフィルタ

0	1	0
1	-4	1
0	1	0

0	0	0	0	0	0
0	0	10	0	0	0
0	0	10	0	0	0
0	0	10	10	10	0
0	0	10	0	0	0
0	0	0	0	0	0

0	0	0	0	0	0
0	10	-30	10	0	0
0	10	-20	20	10	0
0	10	-10	-20	-30	0
0	10	-30	20	10	0
0	0	0	0	0	0

ラプラシアンフィルタ

0	1	0
1	-4	1
0	1	0

絶対値をとる

0	0	0	0	0	0
0	0	10	0	0	0
0	0	10	0	0	0
0	0	10	10	10	0
0	0	10	0	0	0
0	0	0	0	0	0

0	0	0	0	0	0
0	10	30	10	0	0
0	10	20	20	10	0
0	10	10	20	30	0
0	10	30	20	10	0
0	0	0	0	0	0

ラプラシアンとソーベル




ラプラシアンフィルタ










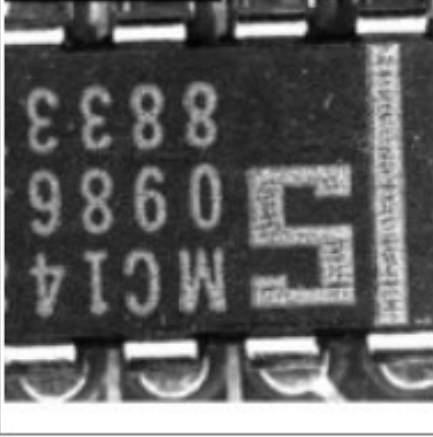




ソーベルフィルタ

標準画像

Color Image			
			
Lenna	Mandrill	Parrots	Girl
			
Aerial	Earth	Sailboat	Pepper
			
Airplane	Couple	Balloon	Milkdrop

標準画像

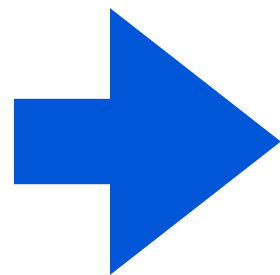
Monochrome Image			
			
Lenna	Lighthouse	Boat	Girl
			
Barbara	Building	Bridge	Cameraman
			
Woman	Text	Airplane	Lax

メディアンフィルタ (p. 42)

- 非線型フィルタの一つ
- 中央値による計算
- ノイズ除去に有効

C f : 平均化フィルタ
ぼかしてノイズ低減

8	9	1
2	3	8
7	6	1



8 9 1 2 3 8 7 6 1
小さい順に ↓ 並べかえ
1 1 2 3 6 7 8 8 9
中央(5番目) の値

演習: メディアンフィルタの実装

リンゴの木の画像を使ってフィルタの効果を確認

- プロジェクト名: median
- OpenCVの関数を用いて
メディアンフィルタの実装

```
//メディアンフィルタ (入力, 出力, フィルタサイズ)
```

```
cv::medianBlur(src_img, median_img, FILTER_SIZE);
```

- フィルタサイズ: 3以上の奇数

```
#include <iostream>
#include <opencv2/opencv.hpp>
#define FILE_NAME "apple_grayscale.jpg"
//ウィンドウ名
#define WINDOW_NAME_INPUT "input"
#define WINDOW_NAME_OUTPUT "output"
#define FILTER_SIZE (3) //フィルタサイズ (3以上の奇数)
```

```
int main(int argc, const char * argv[]) {
    //画像をグレースケールで入力
    cv::Mat src_img = cv::imread(FILE_NAME, 0);
    if (src_img.empty()) { //入力失敗の場合
        fprintf(stderr, "File is not opened.\n");
        return (-1);
    }
    cv::Mat median_img; //出力画像の宣言
    //メディアンフィルタ (入力, 出力, フィルタサイズ)
    cv::medianBlur(src_img, median_img, FILTER_SIZE);
    //表示
    cv::imshow(WINDOW_NAME_INPUT, src_img);
    cv::imshow(WINDOW_NAME_OUTPUT, median_img);
    cv::waitKey(); //キー入力待ち (止める)
    return 0;
}
```

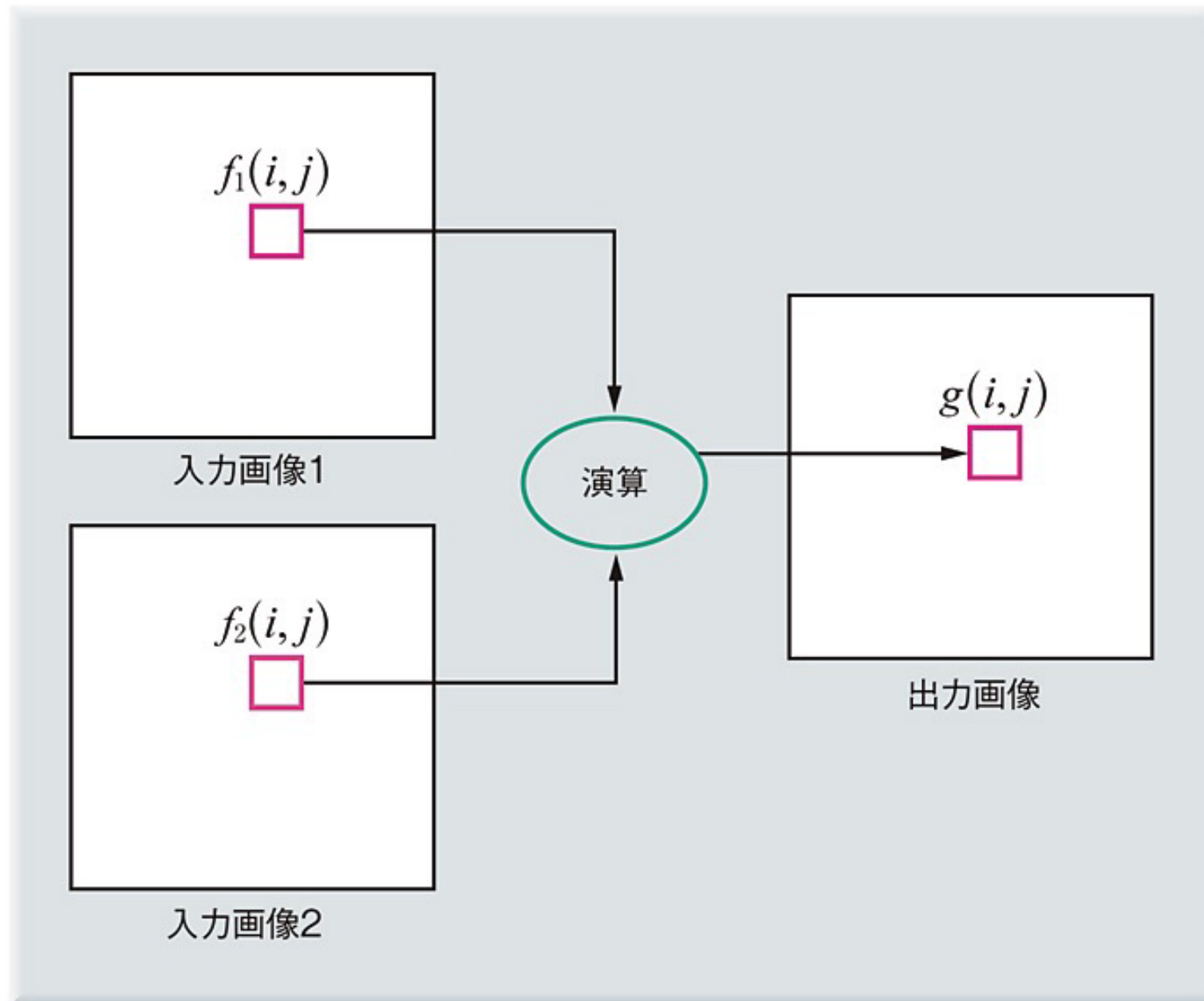
フィルタのサイズを変えて、結果の違いを確認してみてください

複数の画像の利用

- 画像間演算: 2枚またはそれ以上の枚数の入力画像を用いて同一画素に対しての演算
- 例: 2入力画像の画像値 f_1, f_2 における平均画素値 g の例

$$g = \frac{f_1 + f_2}{2}$$

■図 4.20——画像間演算



アルファブレンディング

■図 4.21——画像間演算の例（画素値の平均）



[a] 入力画像1



[b] 入力画像2



[c] 出力画像

アルファブレンディングの表現方法

- 重み付き平均の利用

$$g = \alpha f_1 + (1 - \alpha) f_2$$

$$(0 \leq \alpha \leq 1)$$



$\alpha=0.5$

演習

アルファブレンディングの実装

- プロジェクト名: alpha
- 二つの画像をグレースケール化して入力し、透過度($\alpha=0.3$)で合成せよ
- 注意点: 画素値取得の際はdouble型を用いる



入力画像1



入力画像2

合成結果 ($\alpha=0.3$)



```
#include <iostream>
#include <opencv2/opencv.hpp>
#define FILE_NAME1 "input1.jpg"
#define FILE_NAME2 "input2.jpg"
#define ALPHA (0.3) //透過度
//ウィンドウ名
#define WINDOW_NAME_INPUT1 "input1"
#define WINDOW_NAME_INPUT2 "input2"
#define WINDOW_NAME_OUTPUT "output"

int main(int argc, const char * argv[]) {
    //画像をグレースケールで入力
    cv::Mat src_img1 = cv::imread(FILE_NAME1, 0); //画像1
    cv::Mat src_img2 = cv::imread(FILE_NAME2, 0); //画像2
    if (src_img1.empty() || src_img2.empty()) { //入力失敗の場合
        fprintf(stderr, "File is not opened.\n");
        return (-1);
    }
}
```

```
cv::Mat result_img = cv::Mat(src_img1.size(), CV_8UC1); //合成画像
int x, y;
```

```
//合成画像の出力
```

```
for (y=0; y<src_img1.rows; y++) {
    for (x=0; x<src_img1.cols; x++) {
        //画素値の取得 (double型)
        double s1 = (double)src_img1.at<unsigned char>(y, x);
        double s2 = (double)src_img2.at<unsigned char>(y, x);
        int s_result = ALPHA * s1 + (1.0 - ALPHA) * s2;
        result_img.at<unsigned char>(y, x) = (unsigned char)s_result;
    }
}
```

```
//表示(略)
```


演習

複数画像間の演算

- プロジェクト名: grayscaleSub
- グレースケール画像を差分して, 絶対値を取り(abs関数), 画像を出力せよ.
- 閾値20で二値化した画像を出力せよ.



差分画像



二値画像

```
#include <iostream>
#include <opencv2/opencv.hpp>
#define FILE_NAME1 "input1.jpg"
#define FILE_NAME2 "input2.jpg"
#define THRESHOLD (20) //閾値

//ウィンドウ名
#define WINDOW_NAME_INPUT1 "input1"
#define WINDOW_NAME_INPUT2 "input2"
#define WINDOW_NAME_SUB "sub"
#define WINDOW_NAME_OUTPUT "output"

int main(int argc, const char * argv[]) {
    //画像をグレースケールで入力
    cv::Mat src_img1 = cv::imread(FILE_NAME1, 0); //画像1
    cv::Mat src_img2 = cv::imread(FILE_NAME2, 0); //画像2
    if (src_img1.empty() || src_img2.empty()) { //入力失敗の場合
        fprintf(stderr, "File is not opened.\n");
        return (-1);
    }
}
```

一つ前と
ほぼ同じ

```
cv::Mat sub_img = cv::Mat(src_img1.size(), CV_8UC1); //差分画像
cv::Mat result_img = cv::Mat(src_img1.size(), CV_8UC1); //結果画像(二値)
int x, y;
```

//差分の出力

```
for (y=0; y<src_img1.rows; y++) {
    for (x=0; x<src_img1.cols; x++) {
        //画素値の取得 (int型)
        int s1 = (int)src_img1.at<unsigned char>(y, x);
        int s2 = (int)src_img2.at<unsigned char>(y, x);
        int s_result = abs(s1 - s2); //差分 (int型)
        sub_img.at<unsigned char>(y, x) = s_result; //差分画像に出力

        if (s_result > THRESHOLD) { //閾値処理
            s_result = 255;
        } else {
            s_result = 0;
        }
        result_img.at<unsigned char>(y, x) = s_result;
    }
}
//表示 (略)
```