コンピュータビジョン

担当: 井尻 敬

Contents

01. 序論 ：イントロダクション
02. 特徴検出1 ：テンプレートマッチング、コーナー検出、エッジ検出
03. 特徴検出2 ：ハフ変換、 DoG，SIFT特徴
04. 領域分割 ：領域分割とは， 閾値法， 領域拡張法， グラフカット法，
05. オプティカルフロー：領域分割残り，Lucas-Kanade法
06. パターン認識基礎1 ：パターン認識概論， サポートベクタマシン
07. パターン認識基礎2 ：ニューラルネットワーク、深層学習
08. パターン認識基礎3 ：主成分分析, オートエンコーダ
09. 筆記試験
10. プログラミング演習 1：PC室
11. プログラミング演習 2：PC室
12. プログラミング演習 3：PC室
13. プログラミング演習 4：PC室
14. プログラミング演習 5：PC室

特徴点検出

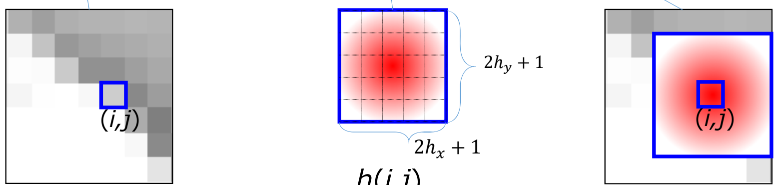
- 復習: 線形フィルタ
  - Sobel filter
  - ガウシアンフィルタとその性質
- 特徴点とは
- SIFT特徴
- Hough変換

復習

線形フィルタとは

出力画素値を周囲画素の重み付和で計算するフィルタ

$$I'(i,j) = \sum_{m=-h_y}^{h_y} \sum_{n=-h_x}^{h_x} h(m,n)\ I(i+m, j+n)$$



$I'(i,j)$ 出力画像

$h(i,j)$ フィルタ

$I(i,j)$ 入力画像

線形フィルタの例

復習

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

ぼかす

0	-1	0
-1	5	-1
0	-1	0

鮮鋭化

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

横方向の微分を計算 $I_x = \frac{\partial I}{\partial x}$
sobel filter

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

縦方向の微分を計算 $I_y = \frac{\partial I}{\partial y}$
sobel filter

線形フィルタ（convolution）

復習

連続 ：

$$(g * f)(x) = \int_{-\infty}^{\infty} g(t)\ f(x-t)\,dt$$

離散 ：

$$(g * f)(x) = \sum_{k=-\infty}^{\infty} g(k)\ f(x-k)$$

『*』を畳み込み積分（Convolution)と呼び，以下の性質が成り立つ

交換 ：$g * f = f * g$

結合 ：$f * (g * h) = (f * g) * h$

分配 ：$f * (g + h) = f * g + f * h$

微分 ：$\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx}$

フーリエ変換: $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$

離散系では、カーネルが十分大きく境界付近で
カーネルの値が０のフィルタ処理の際、この性質
がおおよそ成り立ちます。

畳み込み積分のフーリエ変換: $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$

復習

関数f, gの畳み込み積分は以下の通り定義される，

$$h(x) = \int_{-\infty}^{\infty} f(x-t)g(t)\mathrm{d}t$$

このhのフーリエ変換は以下の通り

$$\mathcal{F}[h(x)](\omega) = \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(x-t)g(t)\mathrm{d}t\right)e^{-ix\omega}\,dx$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-t)g(t)\ e^{-ix\omega}\,\mathrm{d}t\mathrm{d}x$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-t)g(t)\ e^{-i(x-t)\omega}e^{-it\omega}\,\mathrm{d}t\mathrm{d}x$$

$$= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(x-t)e^{-i(x-t)\omega}dx\right)g(t)e^{-it\omega}\,\mathrm{d}t$$

$$= \mathcal{F}[f(x)](\omega)\int_{-\infty}^{\infty} g(t)e^{-it\omega}\,\mathrm{d}t$$

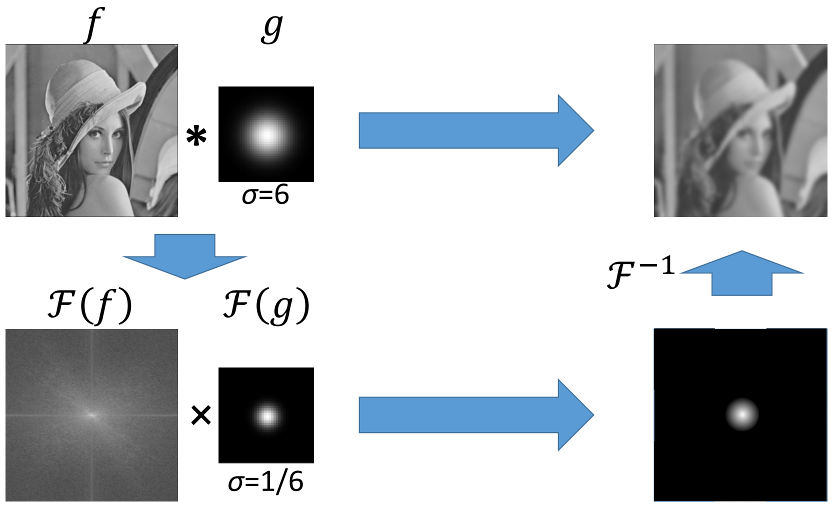
$$= \mathcal{F}[f(x)](\omega)\ \mathcal{F}[g(x)](\omega)$$

つまり，

$$\mathcal{F}[f(x) * g(x)](\omega) = \mathcal{F}[f(x)](\omega)\ \mathcal{F}[g(x)](\omega)$$

畳み込み積分のフーリエ変換: $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$

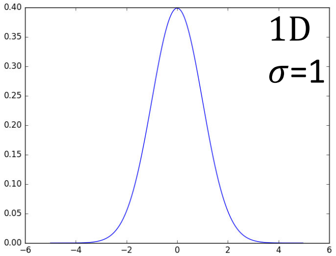
復習



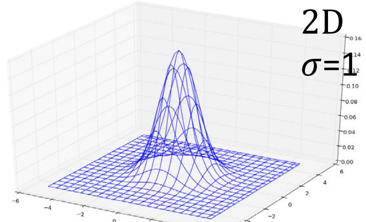
復習

ガウシアンフィルタとは

ガウス関数により畳み込むフィルタのこと
低周波成分のみを通し画像を平滑化する効果がある（ローパスフィルタ）
画像処理において様々な場面で活躍する



$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{x^2}{2\sigma^2}}$$



$$g_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

復習

ガウシアンのフーリエ変換はガウシアン

標準偏差 σのガウス関数

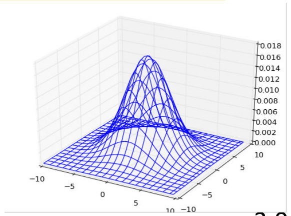
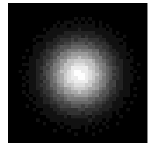
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{x^2}{2\sigma^2}}$$

をフーリエ変換すると標準偏差が逆数のガウシアンになる

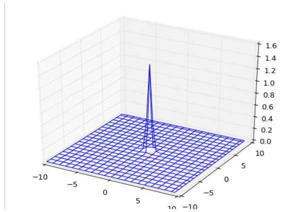
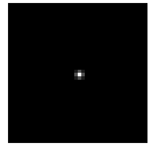
$$\mathcal{F}[g_\sigma(x)](\omega) = \int_{-\infty}^{\infty} g_\sigma(x) e^{-\omega x i} dx$$
$$= e^{-\frac{\omega^2\sigma^2}{2}}$$

または

$$\mathcal{F}[g_\sigma(x)](\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g_\sigma(x) e^{-\omega x i} dx$$
$$= \frac{1}{\sqrt{2\pi}} e^{-\frac{\omega^2\sigma^2}{2}}$$

σ=3.0

σ=1/3.0

復習

$g_a(x)$と$g_b(x)$を連続して畳み込むのは $g_{\sqrt{a^2+b^2}}(x)$を一度だけ畳み込むことと等しい

２つの異なるガウシアンフィルタを用意する

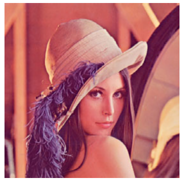
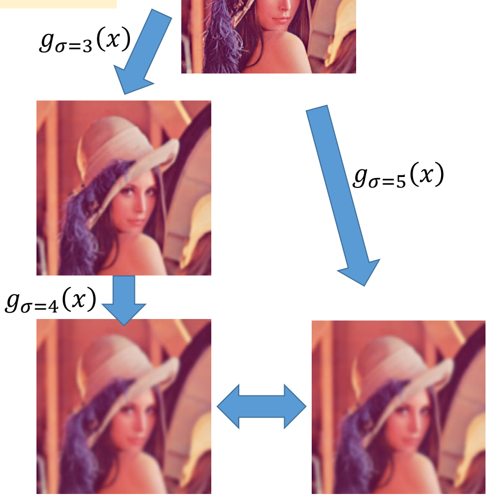
$$g_a(x) = \frac{1}{\sqrt{2\pi a^2}}\, e^{-\frac{x^2}{2a^2}}, \quad g_b(x) = \frac{1}{\sqrt{2\pi b^2}}\, e^{-\frac{x^2}{2b^2}}$$

これらのフーリエ変換は以下の通り

$$G_a(\omega) = e^{-\frac{\omega^2 a^2}{2}}, \quad G_b(\omega) = e^{-\frac{\omega^2 b^2}{2}}$$

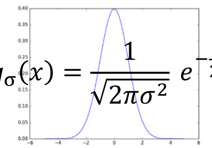
関数$f(x)$に，フィルタを順番に適用する

$$h(x) = g_a(x) * \left(g_b(x) * f(x)\right)$$
$$= \left(g_a(x) * g_b(x)\right) * f(x)$$
$$= \mathcal{F}^{-1}\left(\mathcal{F}\left(g_a(x) * g_b(x)\right)\right) * f(x)$$
$$= \mathcal{F}^{-1}\left(G_a(\omega) G_b(\omega)\right) * f(x)$$
$$= \mathcal{F}^{-1}\left(e^{-\frac{\omega^2(a^2+b^2)}{2}}\right) * f(x)$$
$$= \left(\frac{1}{\sqrt{2\pi(a^2+b^2)}} e^{-\frac{x^2}{2(a^2+b^2)}}\right) * f(x)$$
$$= g_{\sqrt{a^2+b^2}}(x) * f(x)$$

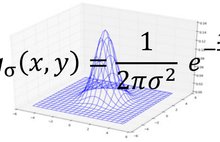



まとめ: ガウシアンフィルタとその性質

- 画像処理において頻繁に利用されるガウシアンフィルタの性質を紹介した



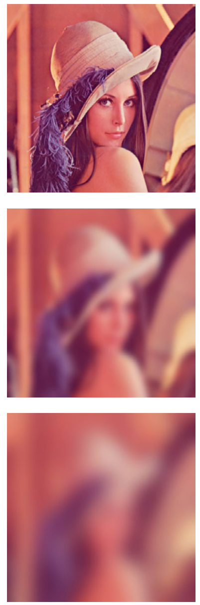
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{x^2}{2\sigma^2}}$$



$$g_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- ガウス関数のフーリエ変換はガウス関数
- 複数のガウシアンフィルタ適用は，一つのガウシアンフィルタで表せる

$$g_a(x) * g_b(x) * f(x) = g_{\sqrt{a^2+b^2}}(x) * f(x)$$



練習問題

(1)右の画像中の赤い画素について、sobel filterを用いて『勾配$(I_x\ I_y)$』『勾配強度の2乗$(I_x^2 + I_y^2)$』『偏角$\theta = atan2\ (I_y, I_x)$』を計算せよ。偏角については計算機等を利用してよい

(2)Gaussian filterの標準偏差 σ を大きくすると、平滑化の強さはどうなるか。次のうち最も適切なものを選べ：

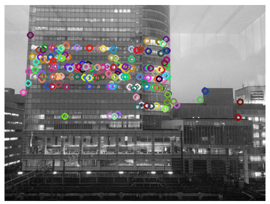
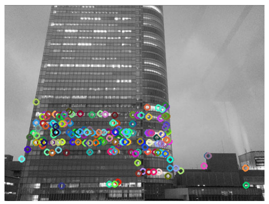
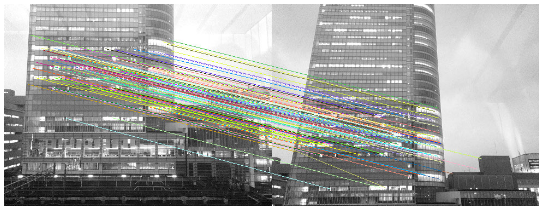
A. 弱くなる（ぼかしが減る）
B. 強くなる（ぼかしが増す）
C. 変わらない

(3)関数 $\frac{1}{\sqrt{2\pi 3^2}} e^{-\frac{x^2}{2\times 3^2}}$ をフーリエ変換せよ

1	1	2	3	4
1	1	2	3	4
6	5	1	7	8
5	6	0	7	8
9	9	10	11	12

特徴点とは

特徴点とは

画像内から特徴的な場所を検出し似た特徴を持つ場所と対応付けしたい

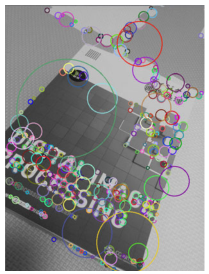
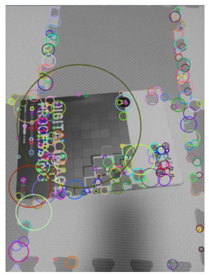
→パノラマ合成，ステレオ視，物体認識，VR（位置あわせ），etc

画像内から特徴的な点を検出する

検出した点の局所的な特徴を計算機が処理できる形で記述したい

- 局所特徴を多次元ベクトルで表現
- 平行移動/拡大/回転に強い記述が理想（平行移動・拡大縮小・回転があっても特徴量が変化しない）

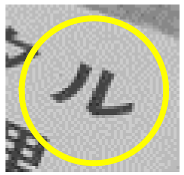
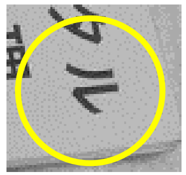
特徴ベクトルとは

- 画像2枚から特徴的な点を沢山抽出できたとしてどれとどれが似ているかを知りたい
- つまり，どれとどれが似た局所画像を持つか知りたい

→検出した特徴点の周囲の情報を，比較できる形（数値データ）に変換したい

!!!特徴ベクトル!!!

- 撮影条件によって対象は回転・拡大縮小・平行移動するので，画像が回転・拡大縮小・平行移動しても似た特徴ベクトルを生成できる手法がほしい
  → この条件を満たすSIFTが良く用いられてきた

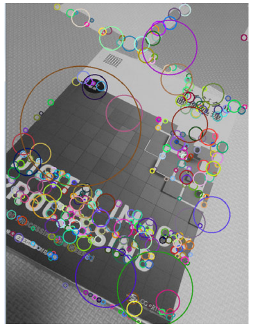
SIFT特徴
Scale Invariant Feature Transform

SIFT.py

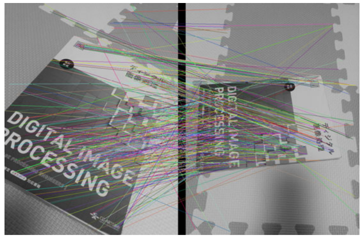
- 有名＆頻繁に利用される特徴量のひとつ
- 周囲の特徴を128次元ベクトルで表現
- 平行移動・回転・拡大縮小に堅固
  - 平行移動・回転・拡大縮小があっても似た特徴ベクトルを出力できる
- 特徴ベクトルにすると局所領域の相違度を計算できる

$$相違度 = \sum_{i=1}^{128} (a_i - b_i)^2$$

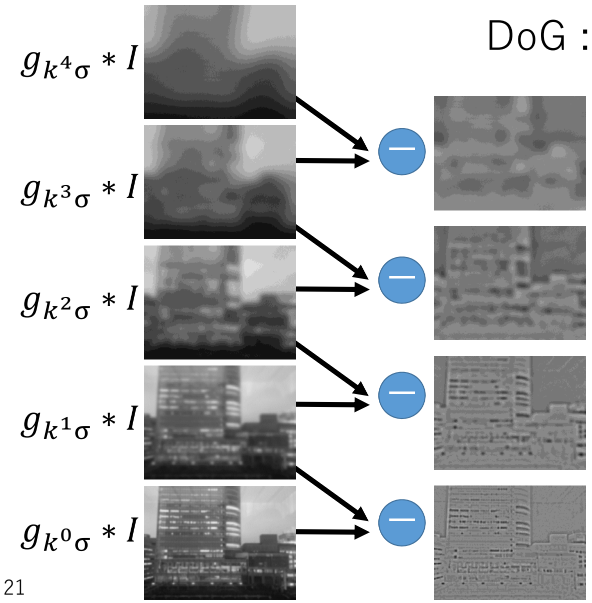
※a_i, b_iは特徴ベクトルの要素
※これは相違度の一例



各点が128次元の特徴ベクトルを持つ

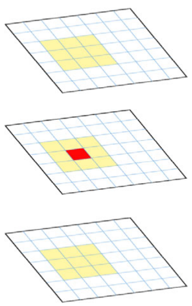


SIFT特徴



DoG : Difference of Gaussian

スケールを考慮して特徴点を検出する
1. 異なるσのガウシアンフィルタをかける
2. ぼかした画像の差分を計算
   これが"Difference Of Gaussian"
3. DoG画像中で局所最大・最小点を発見

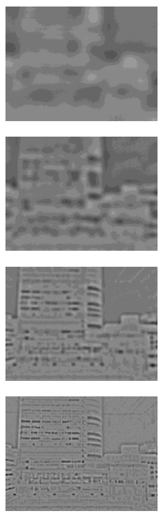


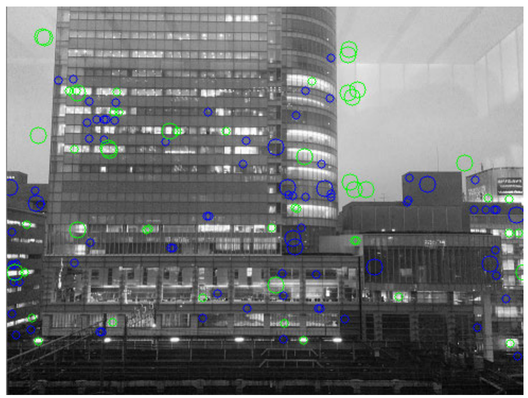
注目画素（赤）の3次元的な隣接画素（黄色）を考慮して局所最大・最小かどうかを判断する

※ σ kはパラメタ
図では σ =2 k=2.0を利用

DoG : Difference of Gaussian

DoG.py





局所的に輝度値が高い・低い点やエッジ，コーナーなどが検出される
その特徴点が現れたスケールも同時に得られる
（どの解像度でその点が特徴的だったかが分かる）

SIFT特徴

1. 特徴点検出

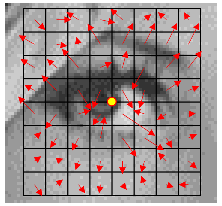
- DoGの極大・極小を特徴点とする
- Harris 行列を用いてエッジ点は除去
- 閾値処理でノイズ（極大値が小さい点）も除去

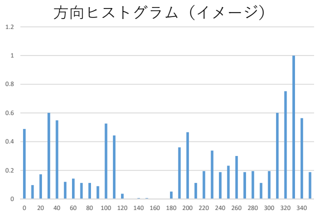
2. 方向検出

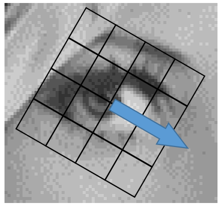
- 発見した特徴点において**そのサイズに合わせた局所領域**を考える
- 勾配ヒストグラムを生成（方向を36分割し，強度を中心からの距離で重み付け）
- ヒストグラムを正規化し強度が0.8以上の方向を検出（複数検出される→複数の特徴量を生成）

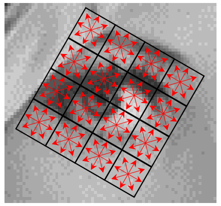
3. 特徴ベクトル計算

- **検出した方向に沿って局所領域を回転**
- 領域を4x4分割し，各領域内で勾配ヒストグラムを計算する
- 勾配ヒストグラムを特徴ベクトルとする
  - 勾配は8方向に量子化
  - 4*4*8 = 128次元ベクトルに
- 得られた特徴ベクトルを正規化（ベクトルの総和で割る）



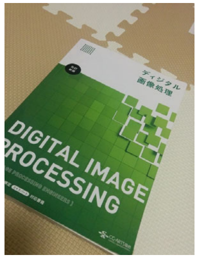


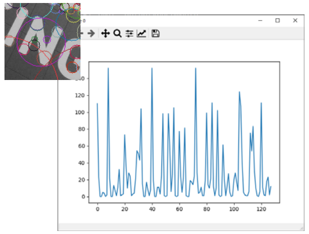


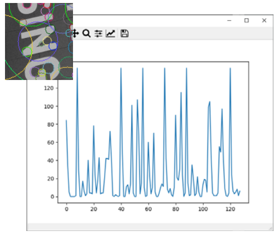


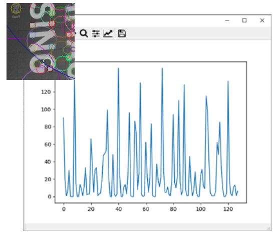
各セルにおいて８方向に量子化した勾配ヒストグラムを計算

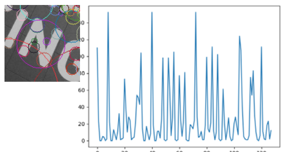
SIFT特徴点の例

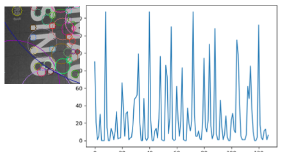


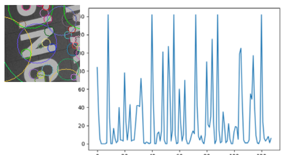


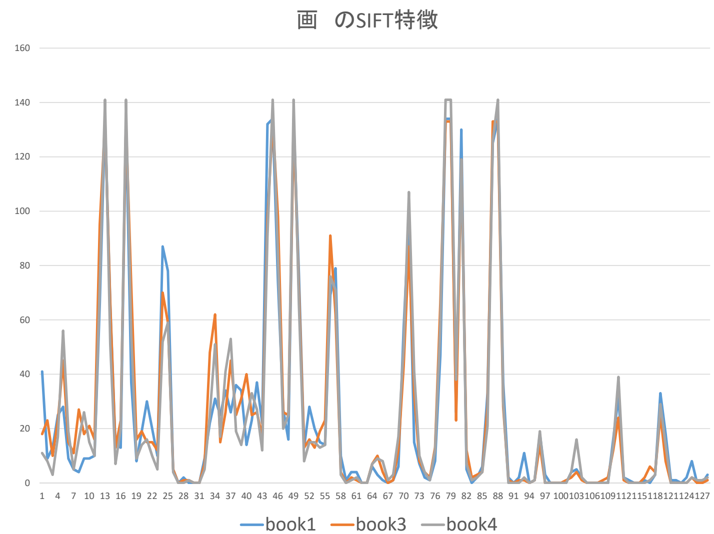


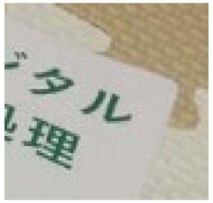


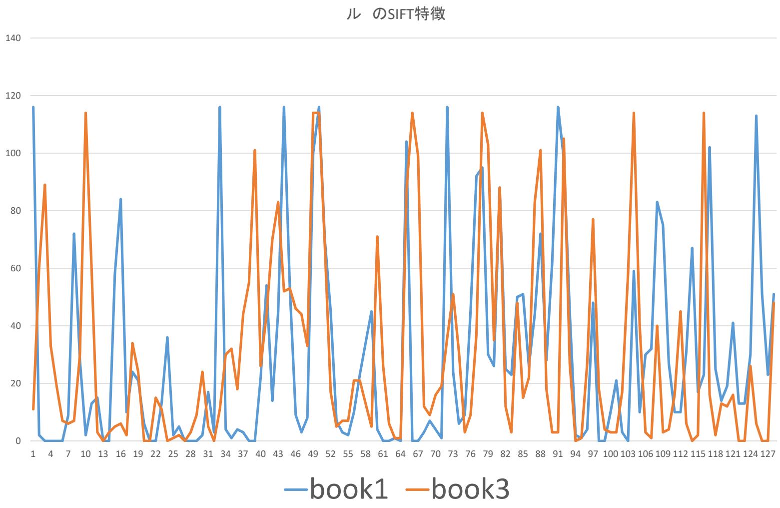












1. **特徴点検出**
- DoGの極大・極小を特徴点とする
- Harris 行列を用いてエッジ点は除去
- 閾値処理でノイズ（極大値が小さい点）も除去

2. **方向検出**
- 発見した特徴点においてそのサイズに合わせた局所領域を考える
- 勾配ヒストグラムを生成（方向を36分割し，強度を中心からの距離で重み付け）
- ヒストグラムを正規化し強度が0.8以上の方向を検出(複数検出される→複数の特徴量を生成)

3. **特徴ベクトル計算**
- 検出した方向に沿って局所領域を回転
- 領域を4x4分割し，各領域内で勾配ヒストグラムを計算する
- 勾配ヒストグラムを特徴ベクトルとする
  - 勾配は8方向に量子化
  - 4*4*8 = 128次元ベクトルに
- 得られた特徴ベクトルを正規化（ベクトルの総和で割る）

質問：
SIFT特徴は
なぜ拡大・回転について不変なのか?

※手順を覚えてほしいわけではなくて、
このように設計した特徴ベクトルが，なぜ拡大縮小と回転に対して不変（変化しにくい）となるかを説明できるようになってほしい

SIFT特徴（実装）

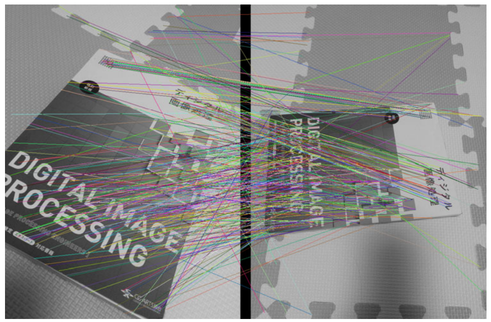
```
# SIFT.py
img1 =  cv2.imread(“画像名.bmp", 0)
sift    = cv2.xfeatures2d.SIFT_create()
key1, des1 = sift.detectAndCompute (img1, None )
```

Python & openCV環境だと上記の3行でSIFT特徴を検出できます
※key1 は特徴点の位置を保持する配列
※des1 は特徴点の特徴ベクトルを保持する配列
※『xfeatures2d.SIFT_create』を書き換えると色々な特徴量を試せます

最近はC++で全部書くのは流行らないみたい.
良い時代ですね。。。

まとめ: SIFT特徴

- 特徴ベクトルとは何かを解説した
  - 検出された特徴点同士を比較するため，特徴点周囲の局所領域をベクトルの形で表すもの.
  - 特徴ベクトルは，SIFT，BRIEF, ORB, SURF, AKAZEなど，沢山の種類がある
  - 特徴ベクトルは目的や対象画像の依存してよいものを選択すべき
- SIFT特徴
  - DoGの極値を特徴点として検出
  - 特徴点のスケールに応じた局所領域を考慮
  - 特徴点周囲の勾配方向に沿って局所窓を回転
  - 局所窓を4分割し，各領域の勾配ヒストグラムを特徴ベクトルとする



練習問題

ある画像の一部領域（3x3）の勾配をsobel filterで計算したところ右図のとおりとなった

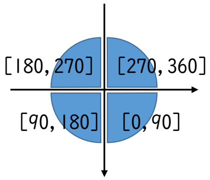
1） 中心画素の勾配強度の2乗を計算せよ

2） 勾配方向を4等分（[0,90] [90,180] [180,270] [270, 360]）し、4つのビンを用意する．中心画素はどのビンに入るか求めよ。

3） 上記の4つのビンを利用し、この3x3画素の勾配ヒストグラムを計算せよ。ただし各画素の勾配強度の2乗を、対応すビンに投票すること

※ SIFTでは、勾配強度を投票しますが、ルートの計算が面倒なのでここでは2乗を投票しています。

$\begin{matrix} I_x \\ I_y \end{matrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$	$\begin{pmatrix} -2 \\ 1 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} -3 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -1 \end{pmatrix}$
$\begin{pmatrix} -2 \\ 2 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -1 \end{pmatrix}$

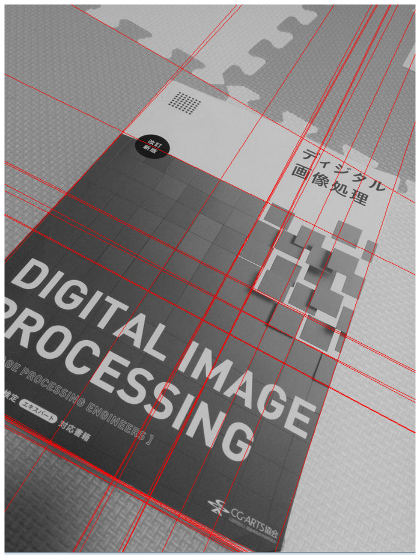


Hough変換

Hough変換とは

- 画像中から直線や円を検出する手法
- 直線や円の一部が破損・劣化していても検出可能

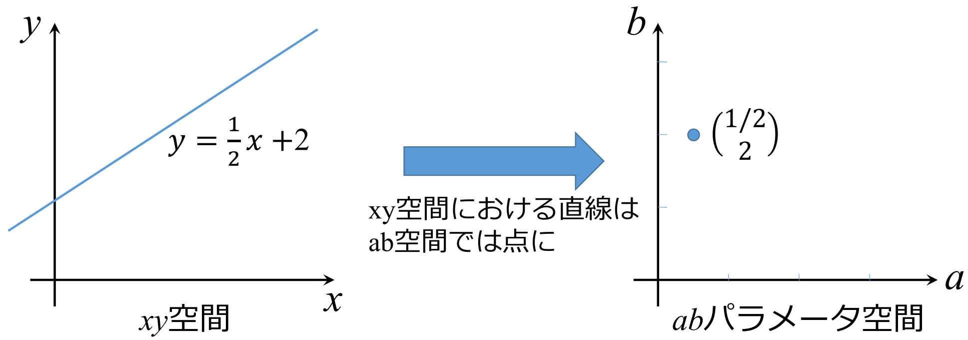
Hough.py



xy空間とab空間

xy空間における直線は『$y = ax + b$』と表せる

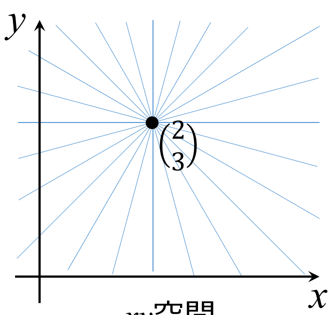
直線の傾きaを横軸・y切片bを縦軸にとるab空間を考える



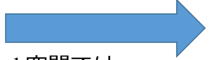
xy空間とab空間

点(2, 3)を通る直線群

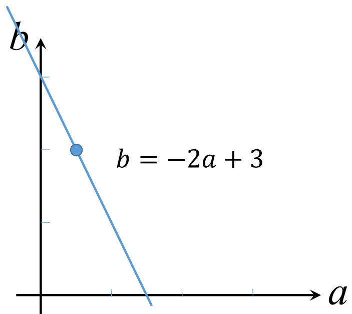
$y = ax + b$ が(2,3) を通るので
$b = -2a + 3$



xy空間における
(2,3)を通る直線群



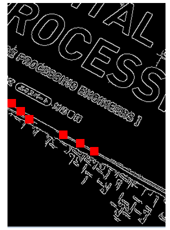
ab空間では
直線: $b = -2a + 3$ に

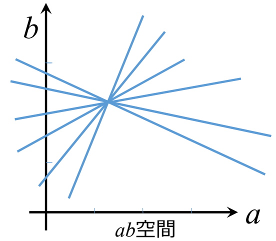


Hough変換

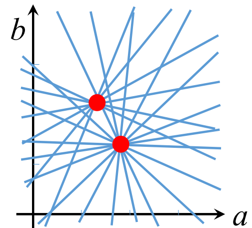
入力：画像
出力：エッジを通る直線群

1. 画像をエッジ画像へ変換
2. 全てのエッジ画素について…
   - エッジ画素を通る直線群はab空間で直線に
   - ab空間を小さなセルに分割し、その直線上のセルの値を1プラスする（**投票**）
3. 閾値より大きなab空間のセルを検索し，そのセルの現す直線を出力
   - 直線は複数発見される

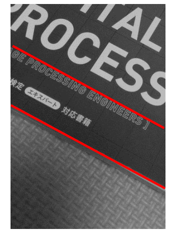




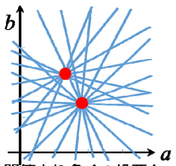
ab空間
実装時はセルの配列で表現される



閾値より多くの投票を受けたセルを検索



- 先のアルゴリズムの問題点
  - $y = ax + b$ として任意の直線を表現する場合、傾きaと切片bのとりうる範囲は[-∞, ∞]
  - 多様な直線を検出するには，無限に広いab空間を用意して投票する必要がある…



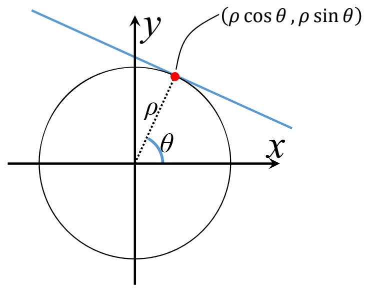
閾値より多くの投票を受けたセルを検索

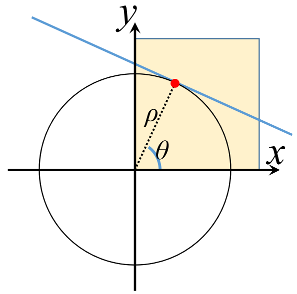
→解決法：直線を 『$\rho = x\cos\theta + y\sin\theta$』 と表す
  - *詳細は次ページ*
  - この表現を使うと θとρの値の範囲を有限にできる

直線：$\rho = x\cos\theta + y\sin\theta$

- この直線は，点$(\rho\cos\theta, \rho\sin\theta)$を通り，傾きは $\frac{-1}{\tan\theta}$
- 半径 ρ の円に接する
- θが$[0, 2\pi]$*の範囲で動くと全ての傾きをカバーできる*（資料修正）
- ρが$[0, A]$*の範囲で動くと画像*（下図の黄色領域）を通る全直線をカバーでき

※ $A = \sqrt{W^2 + H^2}$ は画像の対角の長さ

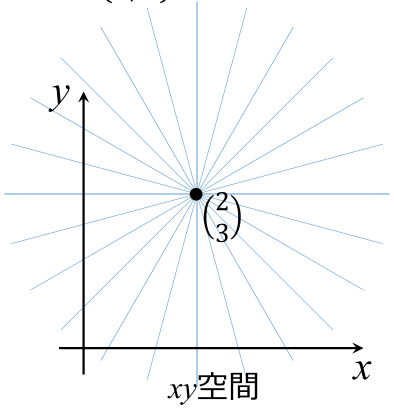




WxHの画像内に入る範囲でρとθを動かす

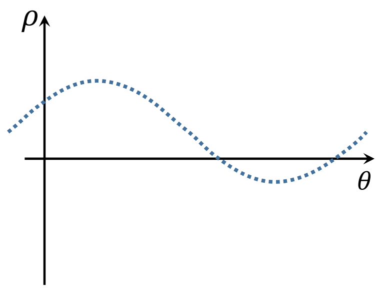
直線を 『$\rho = x\cos\theta + y\sin\theta$』 と表すと…

点$(2, 3)$を通る直線群



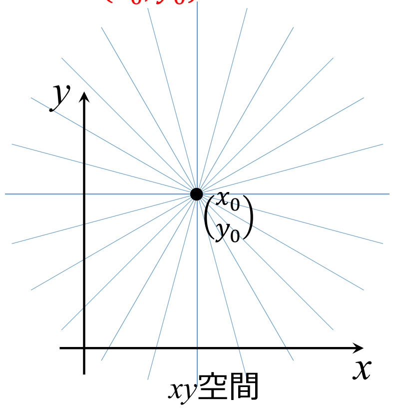
$(2,3)$を通る直線群は
$\rho\theta$ 空間では
$\rho = 2\cos\theta + 3\sin\theta$
という正弦波になる

$$\rho = 2\cos\theta + 3\sin\theta$$
$$\rho = \sqrt{13}\sin(\theta + \alpha)$$



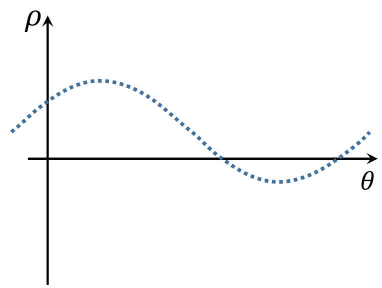
直線を 『$\rho = x\cos\theta + y\sin\theta$』 と表すと…

点(x_0, y_0)を通る直線群



(x_0, y_0)を通る直線群は
$\rho - \theta$ 空間では
$\rho = x_0\cos\theta + y_0\sin\theta$
という正弦波になる

$$\rho = x_0\cos\theta + y_0\sin\theta$$
$$= \sqrt{x_0^2 + y_0^2}\ \sin(\theta + \alpha)$$

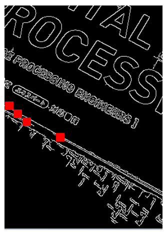


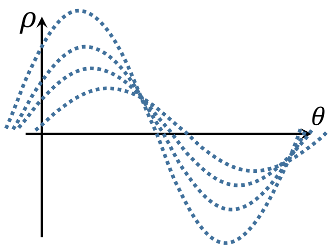
Hough変換

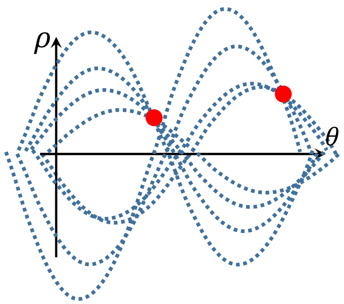
入力：画像

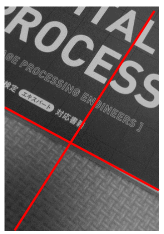
出力：エッジを通る直線群

1. 画像をエッジ画像へ変換
2. 全てのエッジ画素について…
   - エッジ画素を通る直線群は$\rho\ \theta$空間で正弦波に
   - $\rho\ \theta$空間を小さなセルに分割し、その正弦波上のセルの値を1プラスする（**投票**）
3. 閾値より大きな$\rho\ \theta$-空間のセルを検索し，そのセルの現す直線を出力
   - 直線は複数発見される









Hough変換で円を検出する

直線とほぼ同じ方法で検出可能

- 円を $(x - c_x)^2 + (y - c_y)^2 = r^2$ と表現し
- 全景画素 (x, y) を通るすべての円を $c_x c_y r$ の三次元空間に投票
- $c_x c_y r$ 空間にて極大値かつしきい値以上の (c_x, c_y, r)の組を発見する

まとめ : Hough変換

- 画像中の直線や円を検出する手法

0. 直線（または円）を数式で表現する
1. 入力画像からエッジ画像を計算
2. 全てのエッジ画素について…
   パラメータ空間の対応セルの値をプラス1する
   （直線検出ならρθ空間の正弦波を考える）
3. パラメータ空間において値の大きなセルを検索
   そのセルが対応する直線を出力