コンピュータビジョン

担当: 井尻 敬

特徴点検出

- （復習）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

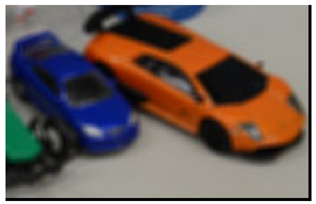
復習

復習：ガウシアンフィルタとその性質

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

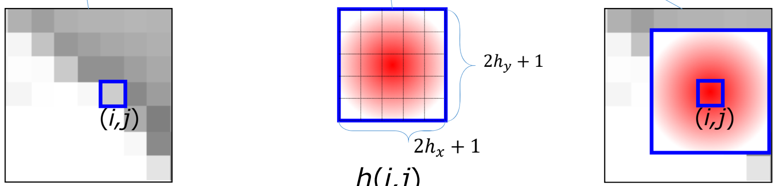
0	-1	0
-1	5	-1
0	-1	0

鮮鋭化

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（convolution）

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: $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$

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

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

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

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

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

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

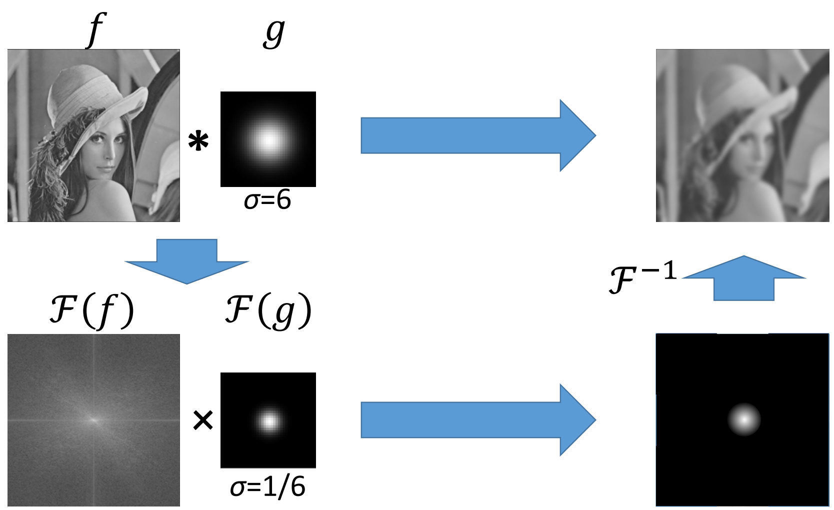
$$= \mathcal{F}[f(x)](\omega) \int_{-\infty}^{\infty} g(t) e^{-it\omega}\, \mathrm{d}t$$

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

つまり，

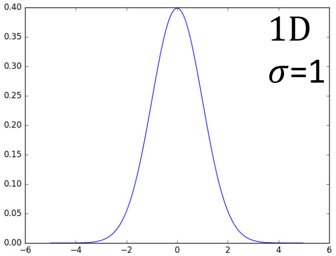
$$\mathcal{F}[f(x) * g(x)](\omega) = \mathcal{F}[f(x)](\omega)\ \mathcal{F}[g(x)](\omega)$$

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

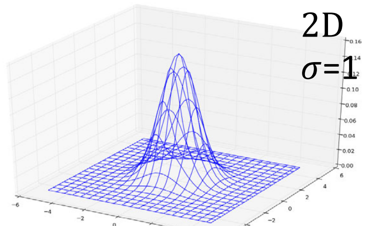


ガウシアンフィルタとは

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



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



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

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

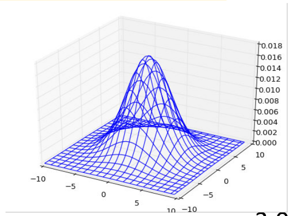
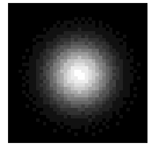
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{x^2}{2\sigma^2}}$$

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

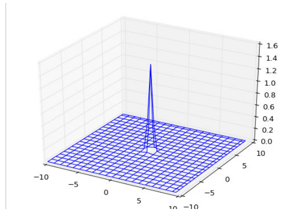
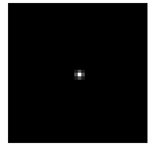
$$\mathcal{F}[g_\sigma(x)](\omega) = \int_{-\infty}^{\infty} g_\sigma(x) e^{-\omega x i} dx = e^{-\frac{\omega^2\sigma^2}{2}}$$

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

$g_a(x)$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

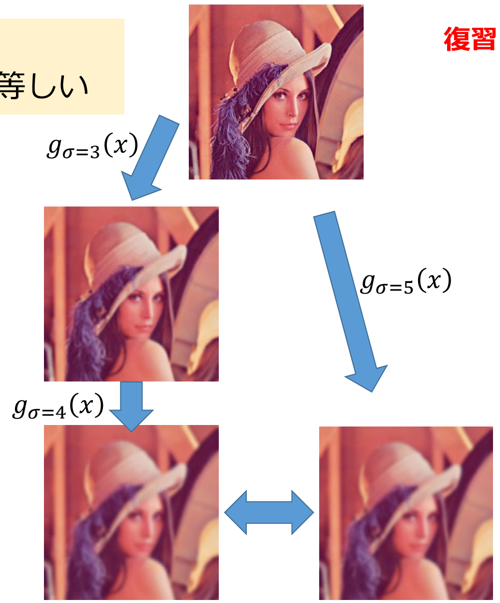
$$g_a(x) = \frac{1}{\sqrt{2\pi a^2}}\, e^{-\frac{x^2}{2a^2}}, \quad g_b(x) = \frac{1}{\sqrt{2\pi b^2}}\, e^{-\frac{x^2}{2b^2}}$$

これらのフーリエ変換は以下の通り

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

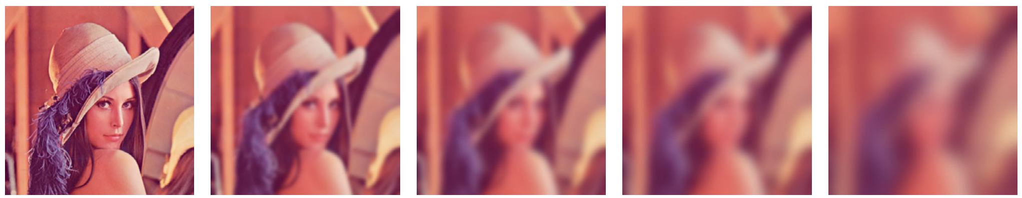
関数$f(x)$に，フィルタを順番に適用する

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



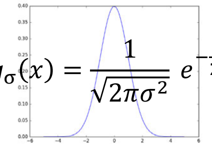
スケールスペース

- 画像の撮影法によって，対象物の大きさは変化する
- 大きさの異なる物体（の特徴量）の比較は結構難しい
- 元画像にσの異なるガウシアンフィルタを適用し，スケールの異なる複数画像を利用して画像処理を行おう，という考え方

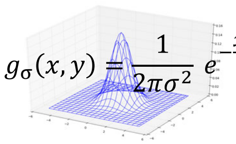


まとめ: ガウシアンフィルタとその性質

- 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



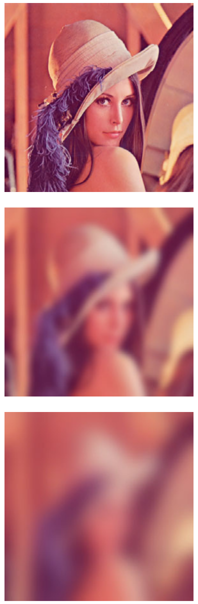
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$



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

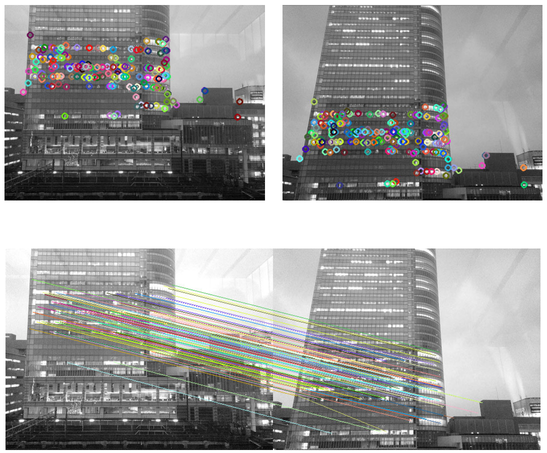
- ガウス関数のフーリエ変換はガウス関数
- 複数のガウシアンフィルタ適用は，一つのガウシアンフィルタで表せる

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



特徴点とは

特徴抽出とマッチング



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

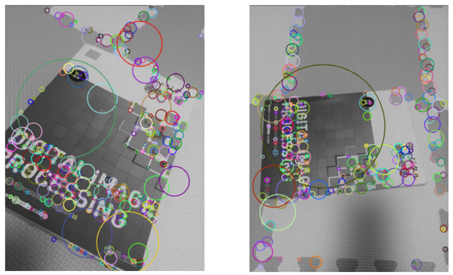
画像内から特徴的な点を検出する

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）

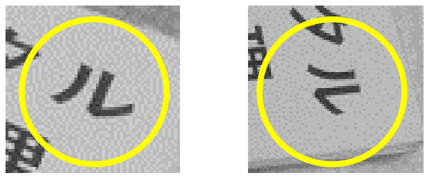
特徴ベクトルとか言われてもしっくりこないという人のために…



- 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

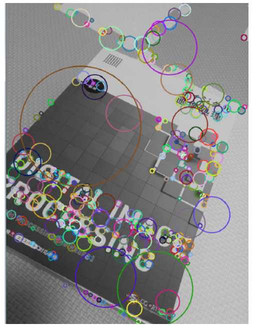
SIFT.py

SIFT特徴
Scale Invariant Feature Transform

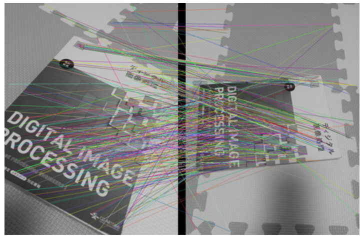
- 有名＆頻繁に利用される特徴量のひとつ
- 周囲の特徴を128次元ベクトルで表現
- 平行移動・回転・拡大縮小に堅固
  - 平行移動・回転・拡大縮小があっても似た特徴ベクトルを出力できる
- 特徴ベクトルにすると局所領域の相違度を計算できる

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

※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

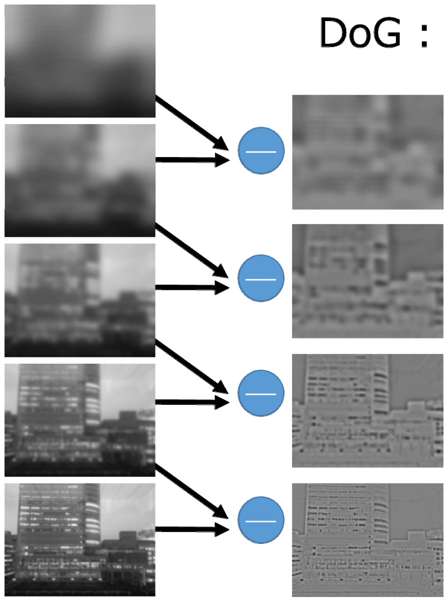
$g_{k^4\sigma} * I$

$g_{k^3\sigma} * I$

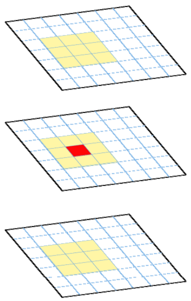
$g_{k^2\sigma} * I$

$g_{k^1\sigma} * I$

$g_{k^0\sigma} * I$



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

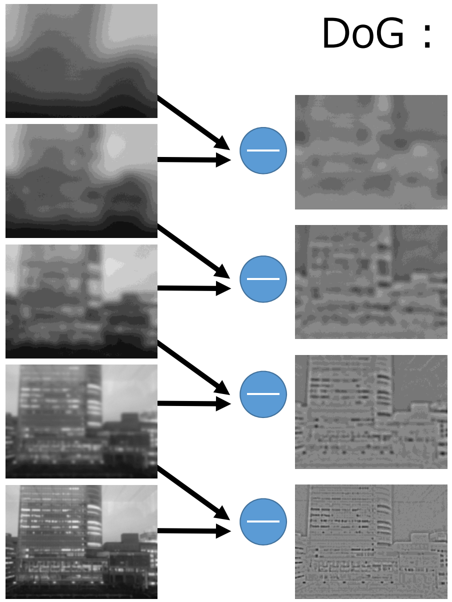
$g_{k^4\sigma} * I$

$g_{k^3\sigma} * I$

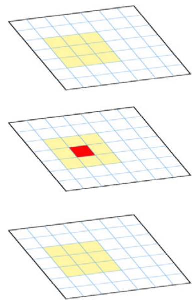
$g_{k^2\sigma} * I$

$g_{k^1\sigma} * I$

$g_{k^0\sigma} * I$



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

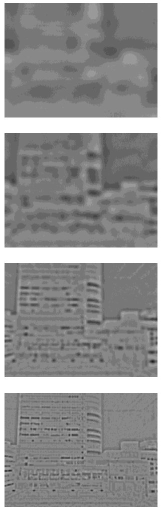


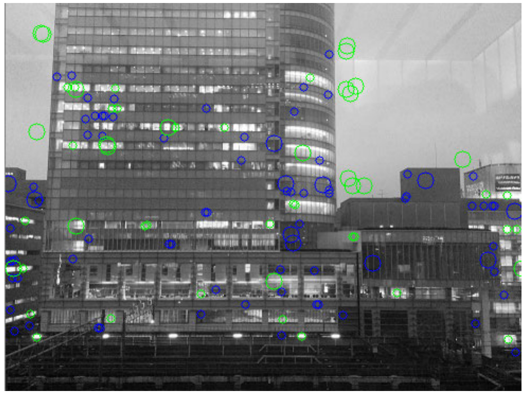
注目画素（赤）の3次元的な隣接画素（黄色）を考慮して局所最大・最小かどうかを判断する

※ σ kはパラメタ
図では σ =2 k=2.0を利用

DoG：Difference of Gaussian

DoG.py



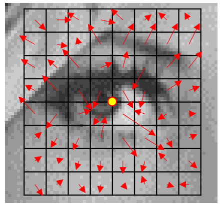


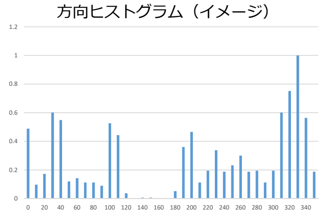
局所的に輝度値が高い・低い点やエッジ，コーナーなどが検出される
その特徴点が現れたスケールも同時に得られる
（どの解像度でその点が特徴的だったかが分かる）

SIFT特徴

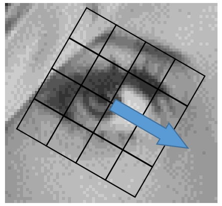
1. 特徴点検出
- DoGの極大・極小を特徴点とする
- Harris 行列を用いてエッジ点は除去
- 閾値処理でノイズ（極大値が小さい点）も除去

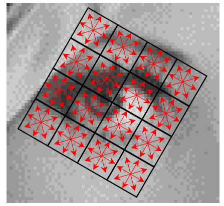
2. 方向検出
- 発見した各特徴点において、DoGの層に対応するガウシアンフィルタのかかった画像を利用し
- 勾配ヒストグラムを生成（方向を36分割し，強度を中心からの距離で重み付け）
- ヒストグラムを正規化し強度が0.8以上の方向を検出(複数検出される→複数の特徴量を生成)





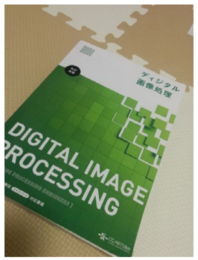
3. 特徴ベクトル計算
- 検出した方向に沿った局所窓を配置
- 領域を4x4分割し，各領域内で勾配ヒストグラムを計算する
- 勾配ヒストグラムを特徴ベクトルとする
  - 勾配は8方向に量子化
  - 4*4*8 = 128次元ベクトルに
- 得られた特徴ベクトルを正規化（ベクトルの総和で割る）

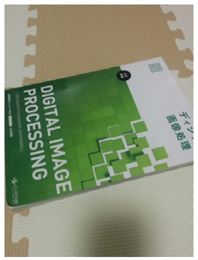


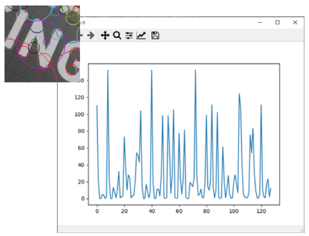


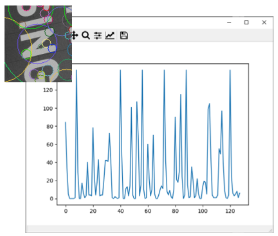
各セルにおいて８方向に量子化した
勾配ヒストグラムを計算

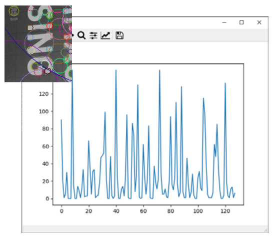
SIFT特徴点の例

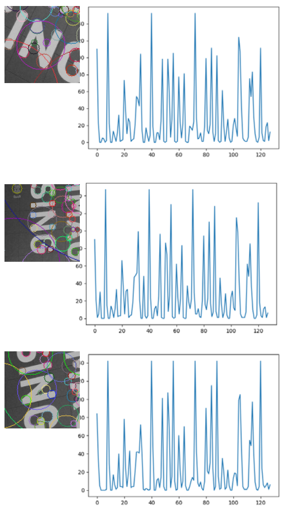


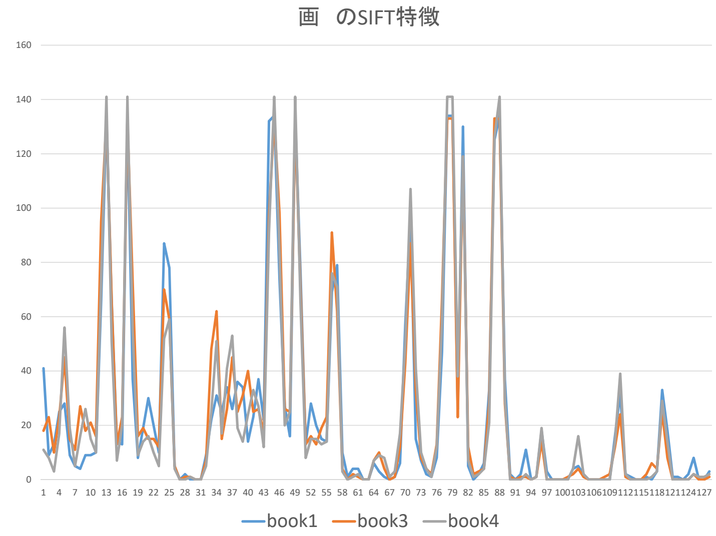


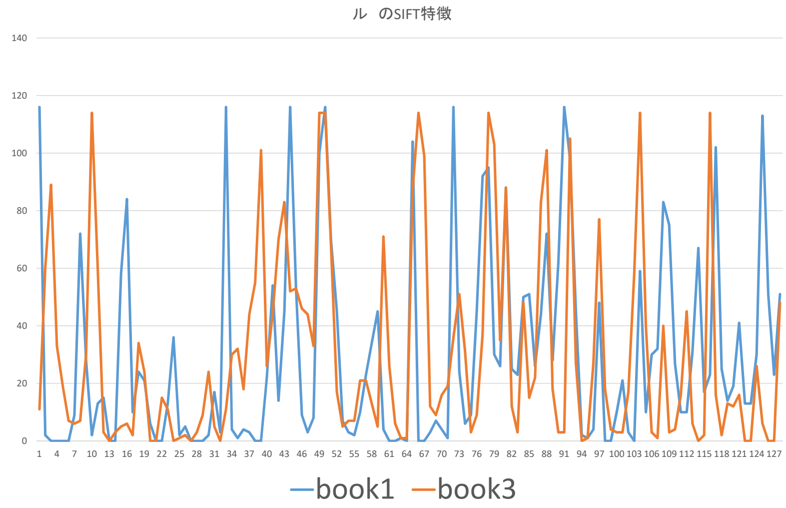












1. 特徴点検出
- DoGの極大・極小を特徴点とする
- Harris 行列を用いてエッジ点は除去
- 閾値処理でノイズ（極大値が小さい点）も除去

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（追記しました）
- 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）

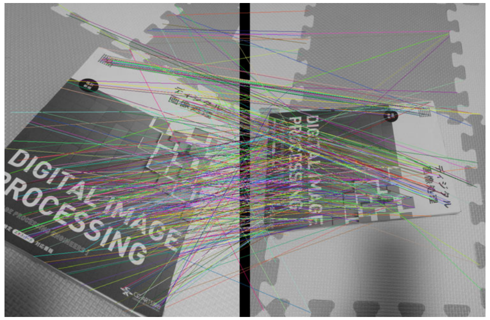
```
# SIFT.py
img1 =  cv2.imread(“画像名.bmp", 0)
sift    = cv2.xfeatures2d.SIFT_create()
key1, des1 = sift.detectAndCompute (img1, None )
```

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。。。

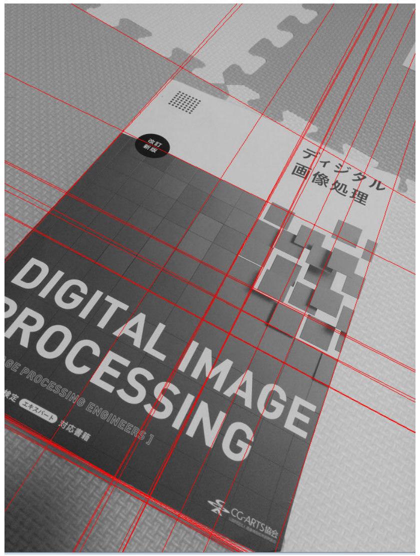
まとめ: SIFT特徴

- 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



Hough変換

Hough.py

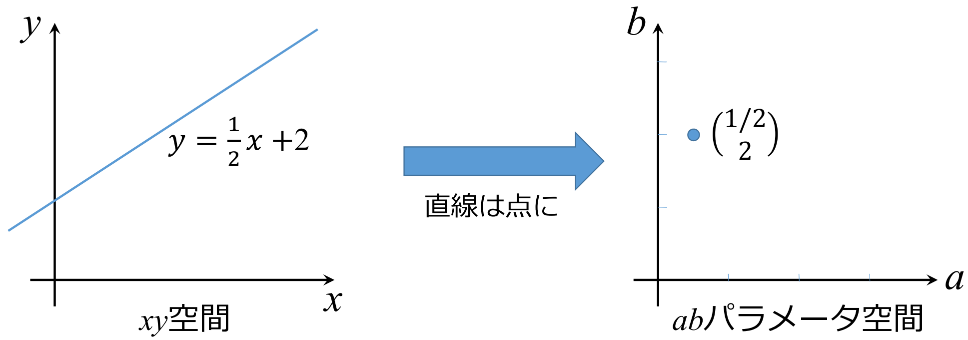


Hough変換とは

- 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

xy空間とab空間

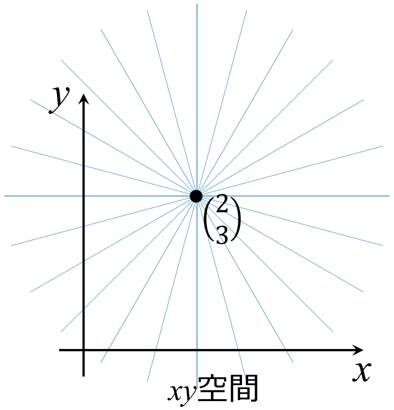
xy空間における直線は『 $y = ax + b$ 』と表せる
直線の傾きaを横軸・y切片bを縦軸にとるab空間を考える



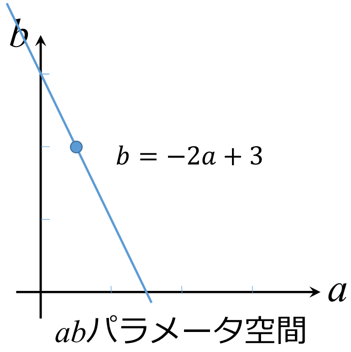
xy空間とab空間

点(x_0, y_0)を通る直線群

$y_0 = ax_0 + b$ より
$b = -ax_0 + y_0$



(2,3)を通る直線群は
パラメータ空間では
直線: $b = -2a + 3$ に

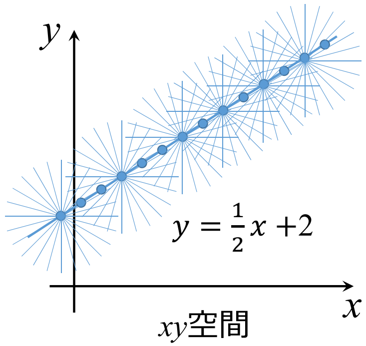


xy空間とab空間

SKIP不要

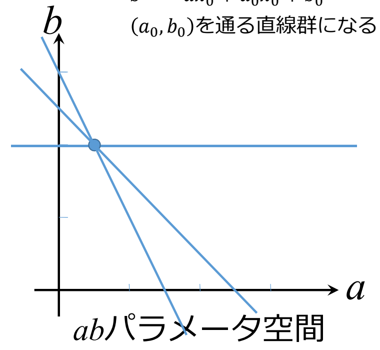
直線$y = a_0x + b_0$ 上の点群
を通る直線群

$y = a_0x + b_0$ を通る点群は
$(x_0, a_0x_0 + b_0)$ と表せる.
この点を通る直線群は
$a_0x_0 + b_0 = ax_0 + b$ より
$b = -ax_0 + a_0x_0 + b_0$
(a_0, b_0)を通る直線群になる



$y = \frac{1}{2}x + 2$ 上の点群
を通る直線群は
$b = -ax_0 + \frac{1}{2}x_0 + 2$
と表せる

$x_0 = 0 \rightarrow b = 2$
$x_0 = 1 \rightarrow b = -a + \frac{5}{2}$
$x_0 = 2 \rightarrow b = -2a + 3$

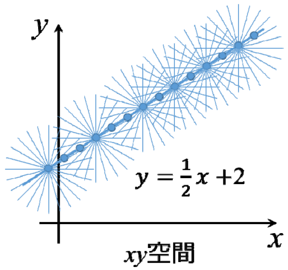


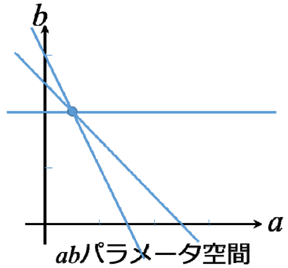
xy空間とab空間

SKIP不要

直線の傾きを横軸，y切片を縦軸にとるab空間を考えると…

- 直線 $y = a_0x + b_0$ → 点(a_0, b_0)に
- 点(x_0, y_0)を通る直線群 → 直線 $b = -x_0a + y_0$に
- 直線 $y = a_0x + b_0$上の点群を通る直線群 →点(a_0, b_0)を通る直線群に



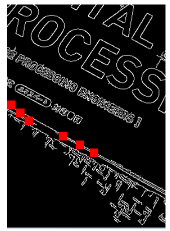


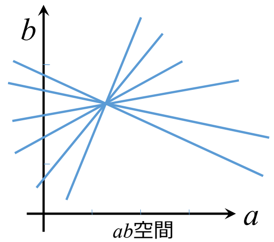
Hough変換

入力：画像

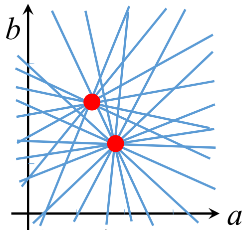
出力：エッジを通る直線群

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

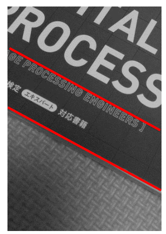




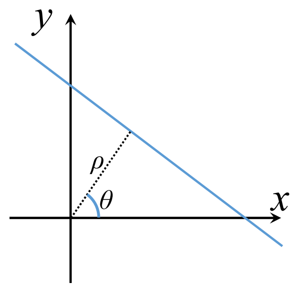
ab空間
実装時はセルの配列で
表現される



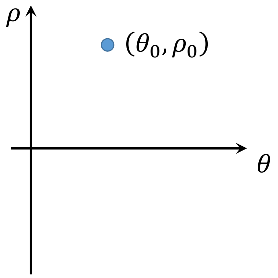
閾値より多くの投票を
受けたセルを検索



- 先のアルゴリズムの問題点
  - 傾きaと切片bのとりうる範囲は[-∞, ∞]である
  - 任意の直線を検出するには，無限に広いab空間に投票する必要が…
- 解決法：直線を 『$\rho = x\cos\theta + y\sin\theta$』 と表す
  - θは直線の傾きに対応，ρは原点から直線の符号付距離を表す
  - θとρの値の範囲は $\theta \in [0, \pi]$，$\rho \in [0, A]$ （Aは画像の対角線長）

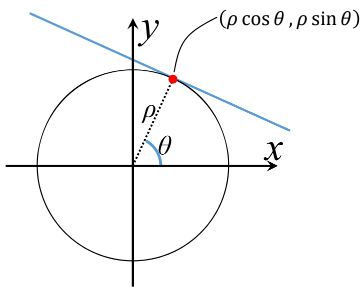


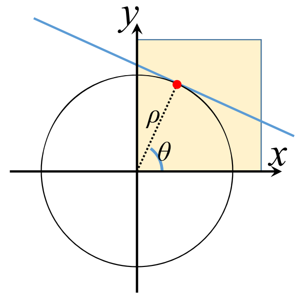
直線 $\rho_0 = x\cos\theta_0 + y\sin\theta_0$
は$\theta\rho$-空間では点になる



捕捉：$\rho = x\cos\theta + y\sin\theta$

- この直線は，点$(\rho\cos\theta, \rho\sin\theta)$を通り，傾き $\frac{-1}{\tan\theta}$ の直線となる
- つまり，半径 ρ の円に接する直線となる



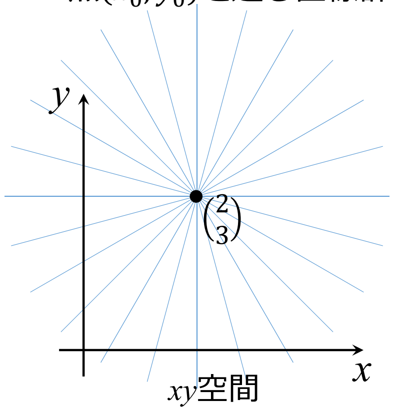


WxHの画像内に
入る範囲で
ρとθを動かす

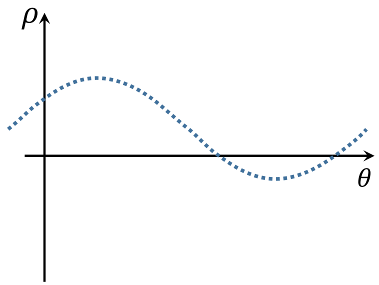
直線を 『$\rho = x\cos\theta + y\sin\theta$』 と表すと…

$$
\begin{aligned}
\rho &= x_0\cos\theta + y_0\sin\theta \\
&= A\sin(\theta + \alpha)
\end{aligned}
$$

点(x_0, y_0)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

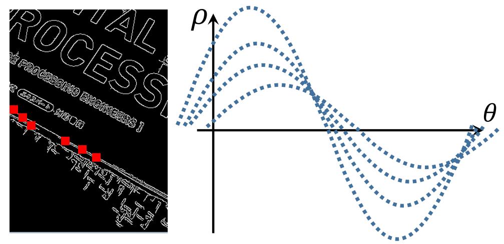


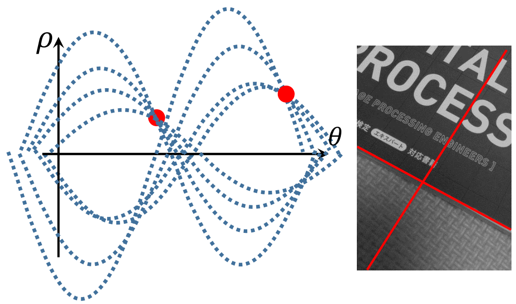
Hough変換

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

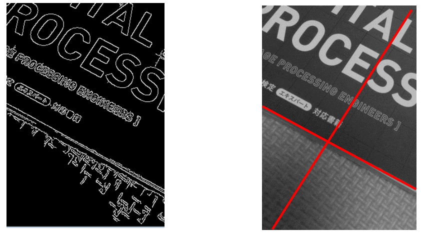




Hough変換で円を検出する

- 直線とほぼ同じ方法で検出可能

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