デジタルメディア処理2

担当: 井尻 敬

デジタルメディア処理２、2017（前期）

~~4/13 デジタル画像とは ：イントロダクション~~
~~4/20 フィルタ処理1 ：画素ごとの濃淡変換、線形フィルタ，非線形フィルタ~~
~~4/27 フィルタ処理2 ：フーリエ変換，ローパスフィルタ，ハイパスフィルタ~~
~~5/11 画像の幾何変換１ ：アファイン変換~~
~~5/18 画像の幾何変換２ ：画像の補間，イメージモザイキング~~
~~5/25 画像領域分割 ：領域拡張法，動的輪郭モデル，グラフカット法，~~
~~6/01 **前半のまとめ (約30分)と中間試験（約70分）**~~
~~6/08 特徴検出1 ：テンプレートマッチング、コーナー・エッジ検出~~
6/15 特徴検出2 ：DoG、SIFT特徴量、Hough変換
6/22 画像認識1 ：パターン認識概論，サポートベクタマシン
6/29 画像認識2 ：ニューラルネットワーク、深層学習
7/06 画像符号化1 ：圧縮率，エントロピー，ランレングス符号化，MH符号化
7/13 画像符号化2 ：DCT変換，ウエーブレット変換など
7/20 **後半のまとめ (約30分)と期末試験（約70分）**

特徴点検出

- ガウシアンフィルタの性質
- DoGとSIFT特徴量
- Hough変換

ガウシアンフィルタの性質

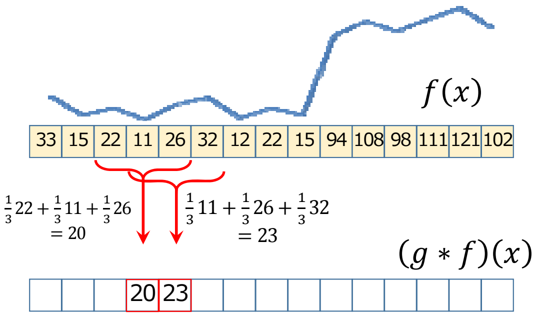
線形フィルタとは（convolution）

連続 ：

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

離散 ：

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



$g(k)$

1/3	1/3	1/3

周囲３ピクセルの平均を取るフィルタ

線形フィルタとは（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

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

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

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

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

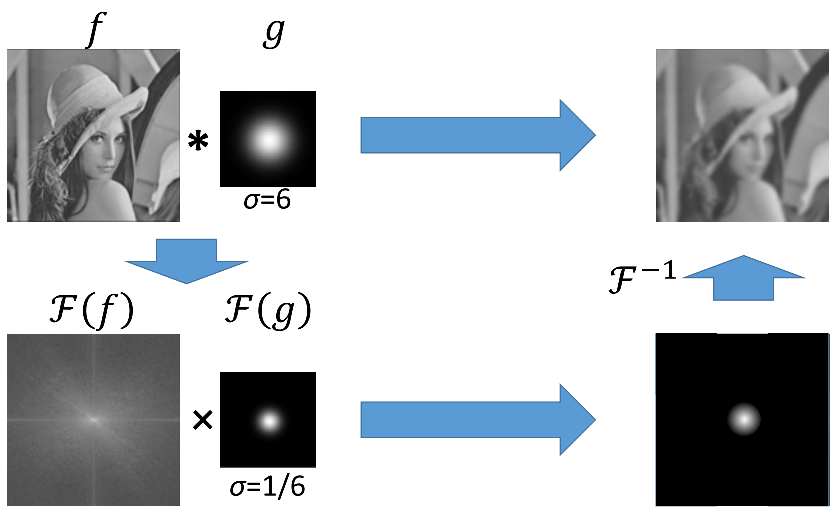
$$= \mathcal{F}[f(x)](\omega)\int_{-\infty}^{\infty} g(t)e^{-it\omega}\,\mathrm{d}t$$

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

つまり，

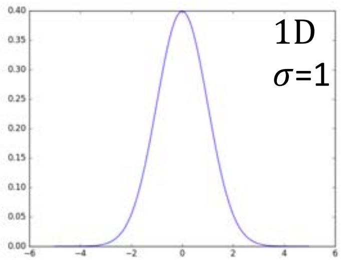
$$\mathcal{F}[f(x) * g(x)](\omega) = \mathcal{F}[f(x)](\omega)\ \mathcal{F}[g(x)](\omega)$$

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

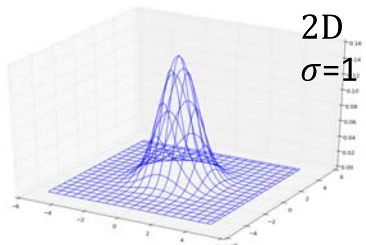


ガウシアンフィルタとは

ガウス関数により畳み込むフィルタのこと
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

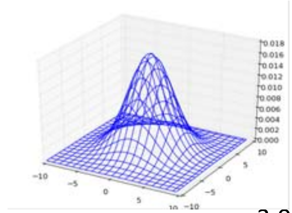
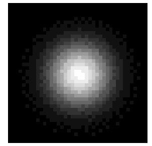
$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{x^2}{2\sigma^2}}$$

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

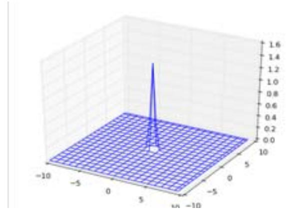
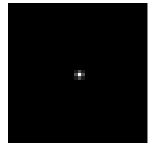
$$\mathcal{F}[g_\sigma(x)](\omega) = \int_{-\infty}^{\infty} g_\sigma(x) e^{-\omega x i} dx$$
$$= e^{-\frac{\omega^2\sigma^2}{2}}$$

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

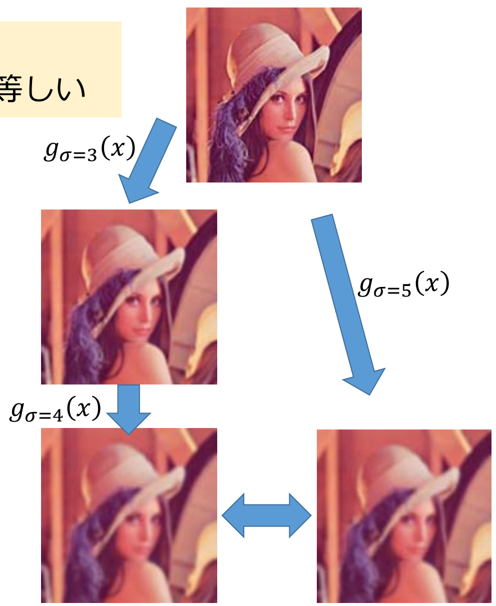
$$g_a(x) = \frac{1}{\sqrt{2\pi a^2}}\, e^{-\frac{x^2}{2a^2}}, \quad g_b(x) = \frac{1}{\sqrt{2\pi b^2}}\, e^{-\frac{x^2}{2b^2}}$$

これらのフーリエ変換は以下の通り

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

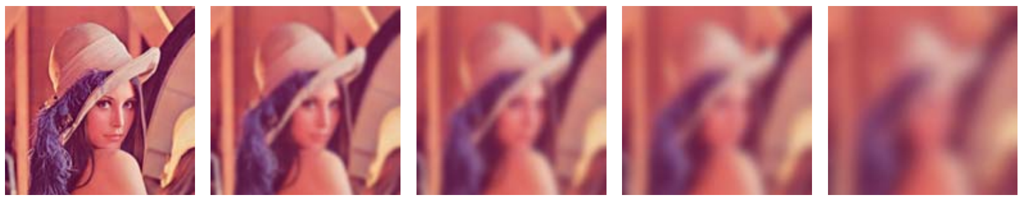
関数$f(x)$に，フィルタを順番に適用する

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



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（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



DoG：Difference of Gaussian

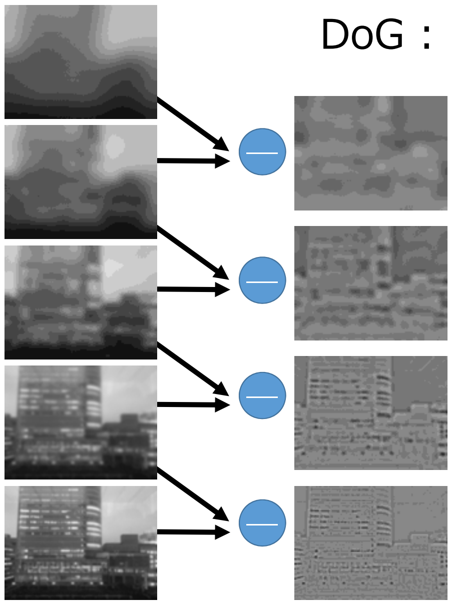
$g_{k^4\sigma} * I$

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

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

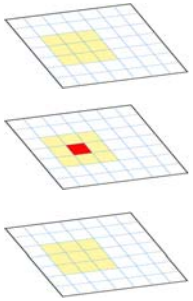
$g_{k^1\sigma} * I$

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



スケールを考慮して特徴点を検出する

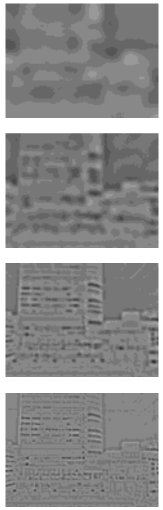
1. 異なるσのガウシアンフィルタをかける
2. ぼかした画像の差分を計算
   これが"Difference Of Gaussian"
3. DoG画像中で局所最大・最小点を発見

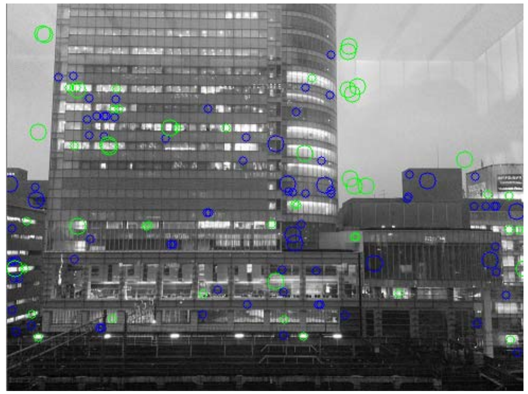


注目画素（赤）の3次元的な隣接画素（黄色）を考慮して局所最大・最小かどうかを判断する

DoG：Difference of Gaussian

DoG.py

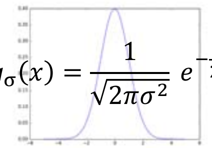




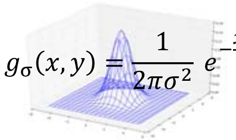
局所的に輝度値が高い・低い点やエッジ，コーナーなどが検出される
その特徴点が現れたスケールも同時に得られる
（どの解像度でその点が特徴的だったかが分かる）

まとめ: ガウシアンフィルタとその応用

- 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

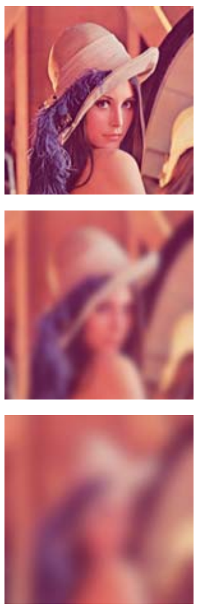


$$g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

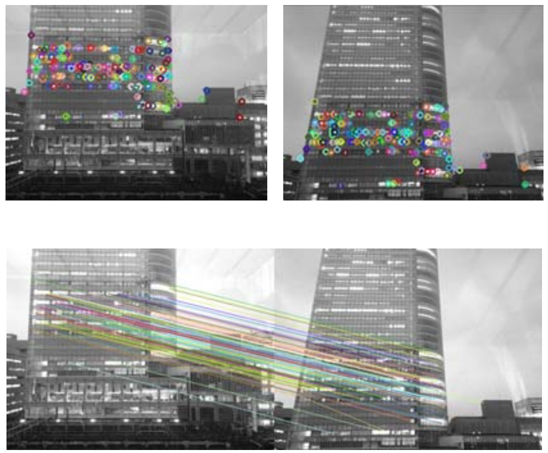


$$g_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- 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
  $$g_a(x) * g_b(x) * f(x) = g_{\sqrt{a^2+b^2}}(x) * f(x)$$
- Difference of Gaussian
- Gaussian Pyramid (今回時間がなく扱えなかった…)



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）

SIFT.py

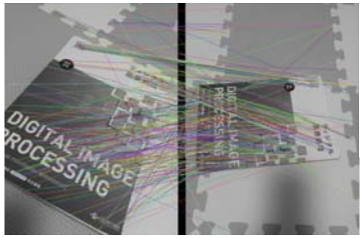
SIFT特徴
Scale Invariant Feature Transform

- 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

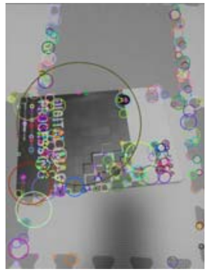
$$\text{相違度} = \sum_{i=1}^{128} (a_i - b_i)^2$$

※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

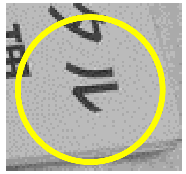


特徴ベクトルとか言われてもしっくりこないという人のために…



- 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!!!



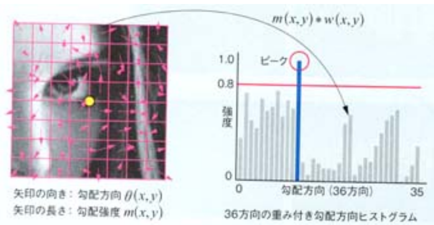
- 撮影条件によって対象は回転・拡大縮小・平行移動するので，画像が回転・拡大縮小・平行移動しても似た特徴ベクトルを生成できる手法がほしい
  → この条件を満たすSIFTが良く用いられてきた

SIFT特徴

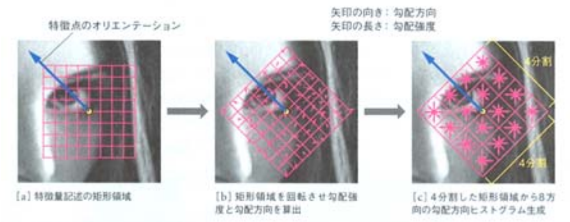
1. 特徴点検出
- DoGの極大・極小を特徴点とする
- Harris 行列を用いてエッジ点は除去
- 閾値処理でノイズ（極大値が小さい点）も除去

2. 方向検出
- 発見した特徴点周囲の勾配ヒストグラムを生成
- 勾配方向は36分割し中心からの距離で重み付け
- ヒストグラムの強度を正規化し強度が0.8以上の方向を検出(複数検出されることも)

3. 特徴ベクトル計算
- 検出した方向に沿って窓を配置
- 窓を4x4分割し，窓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）



教科書図11.17



教科書図11.18

SIFT特徴点の例（教科書図11.19）

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
- 発見した特徴点周囲の勾配ヒストグラムを生成
- 勾配方向は36分割し中心からの距離で重み付け
- ヒストグラムの強度を正規化し強度が0.8以上の方向を検出(複数検出されることも)

3. 特徴ベクトル計算
- 検出した方向に沿って窓を配置
- 窓を4x4分割し，窓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

→ 特徴点周囲の勾配方向分布を特徴量とする

なぜ拡大・回転について堅固なのか

拡大縮小:
スケールを考慮して特徴点を検出し，
そのスケールの窓内でヒストグラムを計算

回転:
局所領域の勾配方向に沿って窓を回転する

※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拡大縮小と回転に対して堅固（変化しにくい）となるかを知ってほしい

SIFT特徴（実装）

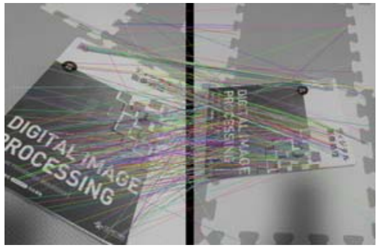
```
# SIFT.py
img1 =  cv2.imread(“画像名.bmp", 0)
sift   = cv2.xfeatures2d.SIFT_create()
key1, des1 = sift.detectAndCompute (img1, None )
```

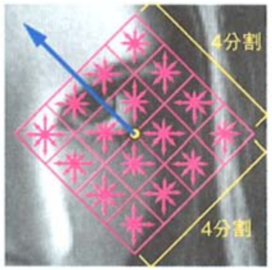
Python & openCV環境だと上記の3行でSIFT特徴を検出できます
※key1 は特徴点の位置を保持する配列
※des1 は特徴点の特徴ベクトルを保持する配列
※『xfeatures2d.SIFT_create』を書き換えると色々な特徴量を試せます

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





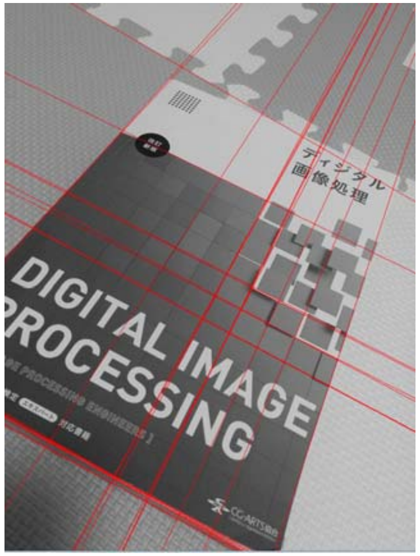
教科書図11.18

Hough変換

Hough変換とは

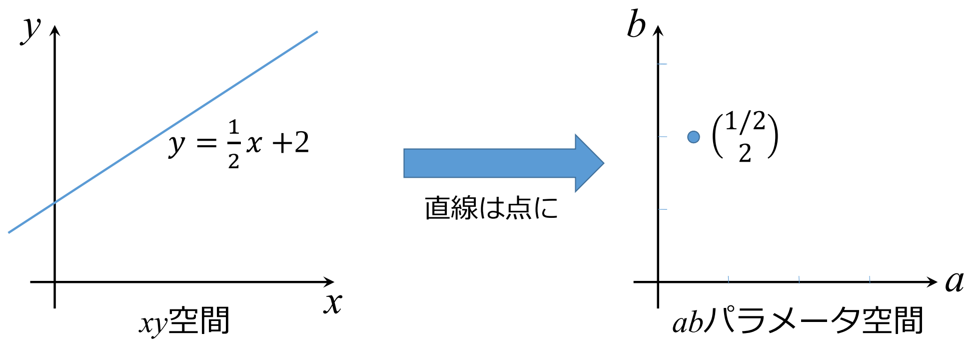
- 画像中から直線や円を検出する手法
- 直線や円の一部が破損・劣化していても検出可能

Hough.py



xy空間とab空間

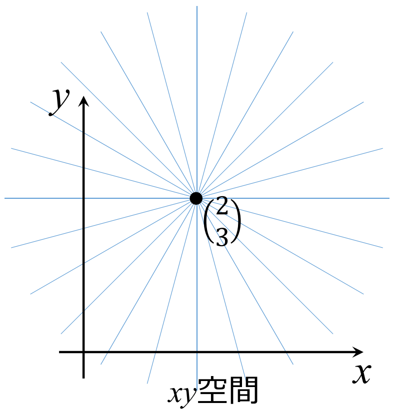
xy空間における直線は『 $y = ax + b$ 』と表せる
直線の傾きaを横軸・y切片bを縦軸にとるab空間を考える



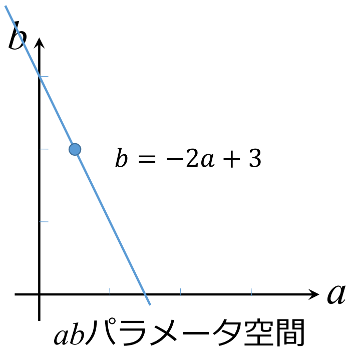
xy空間とab空間

点(x_0, y_0)を通る直線群

$y_0 = ax_0 + b$ より
$b = -ax_0 + y_0$



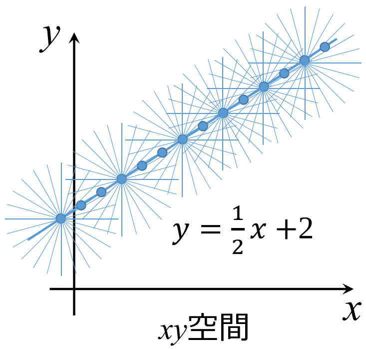
(2,3)を通る直線群は
パラメータ空間では
直線: $b = -2a + 3$ に



xy空間とab空間

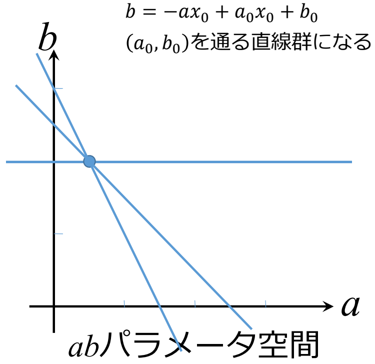
直線$y = a_0x + b_0$ 上の点群
を通る直線群

$y = a_0x + b_0$ 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

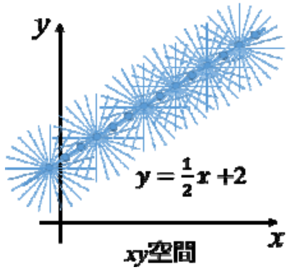
$x_0 = 0 \to b = 2$
$x_0 = 1 \to b = -a + \frac{5}{2}$
$x_0 = 2 \to b = -2a + 3$

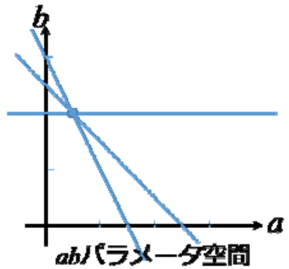


xy空間とab空間

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



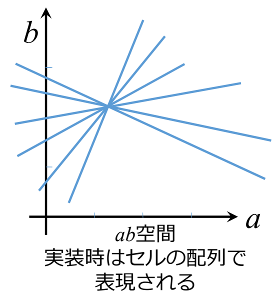


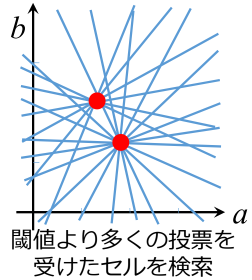
Hough変換

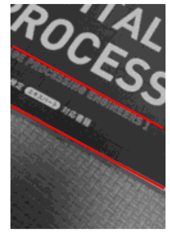
入力：画像

出力：エッジを通る直線群

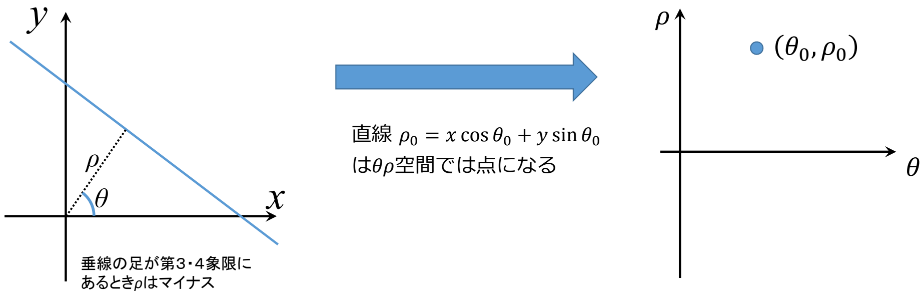
1. 画像をエッジ画像へ変換
2. 全てのエッジ画素について…
   - エッジ画素を通る直線群はab空間で直線に
   - ab空間を小さなセルに分割し、その直線上のセルの値を1プラスする（**投票**）
3. 閾値より大きなab空間のセルを検索し，そのセルの現す直線を出力
   - 直線は複数発見される







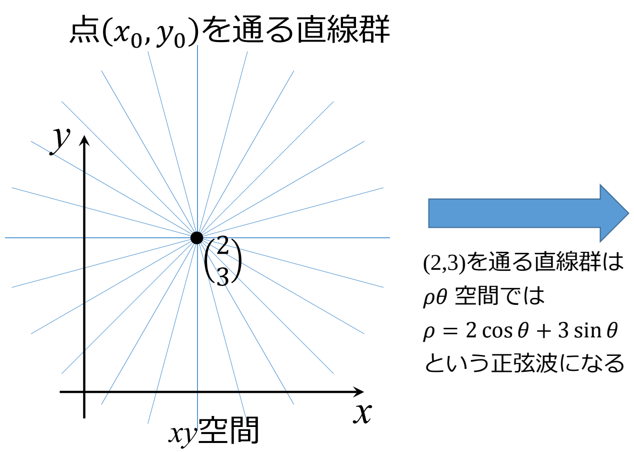
- 先のアルゴリズムの問題点
  - 傾きaと切片bのとりうる範囲は[-∞, ∞]である
  - 任意の直線を検出するには，無限に広いab空間に投票する必要が…
- 解決法：直線を 『$\rho = x\cos\theta + y\sin\theta$』と表す
  - θは直線の傾きに対応，ρは原点から直線の符号付距離を表す
  - θとρの値の範囲は $\theta \in [0, \pi]$，$\rho \in \left[-\frac{A}{2}, A\right]$ （Aは画像の対角線長）

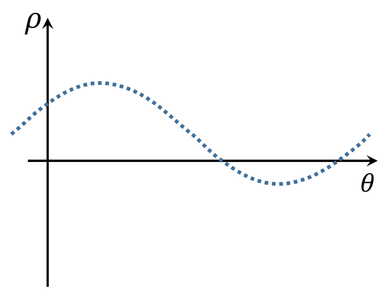


直線を 『$\rho = x\cos\theta + y\sin\theta$』と表すと…

点(x_0, y_0)を通る直線群

$$\rho = x_0\cos\theta + y_0\sin\theta$$
$$= A\sin(\theta + \alpha)$$



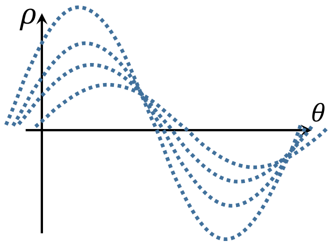


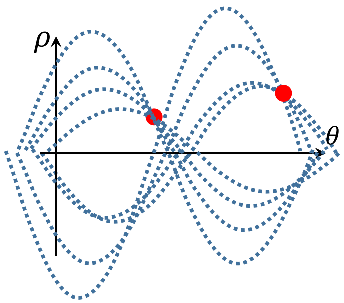
Hough変換

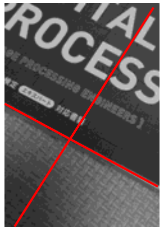
入力：画像

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
- 各自考えてみてください

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