コンピュータビジョン
画像の様々な変換

Filtering

- Correlation（相関）

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \quad \text{or} \quad g = f \otimes h$$

- Convolution（畳込み）

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l)$$

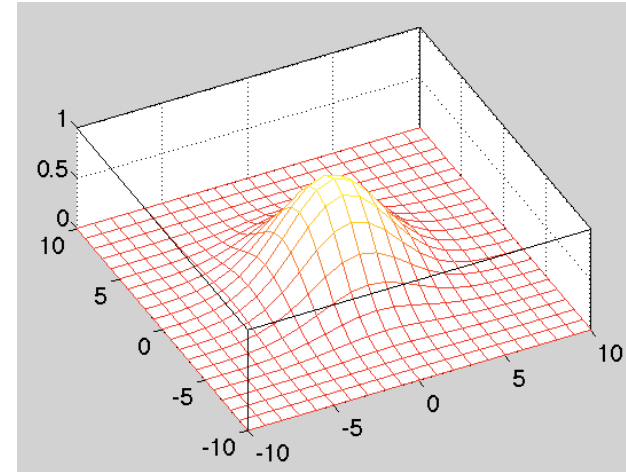or $\quad g = f * h$

f $\qquad\qquad\qquad$ h $\qquad\qquad\qquad$ g

Filtering: Gaussian filter

$$h_{\text{blur}} = \frac{1}{C} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$


```
import numpy

def filter(shape=(3,3),sigma=0.5):
    m,n = shape[0]/2.0, shape[1]/2.0
    y,x = numpy.ogrid[-m:m+1,-n:n+1]
    h = numpy.exp( -(x*x + y*y) / (2.*sigma*sigma) )
    #h[ h < numpy.finfo(h.dtype).eps*h.max() ] = 0
    sumh = h.sum()
    if sumh != 0:
        h /= sumh
    return h
```

gauss2D.py

Filtering: Gaussian filter

$$g = f \otimes h_{\text{blur}}$$


```
import numpy, cv2, gauss2D

image = cv2.imread('lena.png')
sigma = 1.0 # in pixels

# The two lines below can be rewritten as
# blurred = cv2.GaussianBlur(image, (0,0), sigma)
filter = gauss2D.filter((sigma*7, sigma*7), sigma)
blurred = cv2.filter2D(image, -1, filter)

cv2.imwrite('lena_blurred.png', blurred)
```

blur.py

課題

- **Gaussian filterの形状（x,y方向のsigma）を変えてみる**
  - ヒント：

```python
import numpy, cv2, gauss2D                        blur.py

image = cv2.imread('lena.png')
sigma = 1.0 # in pixels

# The two lines below can be rewritten as
blurred = cv2.GaussianBlur(image, (0,0), sigmaX = 1.0, sigmaY =
10.0)
#filter = gauss2D.filter((sigma*7, sigma*7), sigma)
#blurred = cv2.filter2D(image, -1, filter)

cv2.imwrite('lena_blurred.png', blurred)
```

Filtering: 'Unsharp masking'

- 画像をシャープに加工：高周波数成分を増強

$$g_{\mathrm{sharp}} = f + \gamma(f - h_{\mathrm{blur}} \otimes f)$$

入力画像とボカしたものの差
＝シャープな成分

```
import numpy, cv2

f = cv2.imread('lena_blurred.png').astype(numpy.float)
sigma, gamma = 3.0, 0.4

f_blurred = cv2.GaussianBlur(f, (0,0), sigma)
g_sharp = f + gamma*(f-f_blurred)

cv2.imshow('Original', f.astype(numpy.uint8))
cv2.imshow('Result', g_sharp.astype(numpy.uint8))

...
```
unsharp.py

課題

- ## gammaを変えてみる
  - ヒント：numpy.maximum(numpy.minimum(*, 255.0),.0)

- ## sigmaを変えてみる

Fourier transform

Continuous FT(1D)

$$F(\nu) = \int_{-\infty}^{\infty} f(t)e^{-2\pi j\nu t}dt$$

Inverse transform

$$f(t) = \int_{-\infty}^{\infty} F(\nu)e^{2\pi j\nu t}d\nu$$

Discrete FT

$$F_k = \sum_{m=0}^{n-1} f_m e^{-2\pi jmk/n}$$

Inverse transform

$$f_m = \frac{1}{n}\sum_{k=0}^{n-1} F_k e^{2\pi jmk/n}$$

Discrete Fourier Transform(2D)

$$F_{kl} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn}\exp\left\{-2\pi j\left(\frac{mk}{M}+\frac{nl}{N}\right)\right\} \qquad \begin{array}{l} k = 0,\dots,M-1 \\ l = 0,\dots,N-1 \end{array}$$

Fourier transform

```python
...

lowpass_mask = numpy.zeros((480,640))
size = 30
lowpass_mask[240-size:240+size, 320-size:320+size] = 1

while 1:
    stat, image = cap.read()

    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    f = numpy.fft.fft2(image) # Fourier trans.
    f = numpy.fft.fftshift(f) # Centering
    f_amp = numpy.log(numpy.abs(f))/20 # amplitude
    #f = numpy.multiply(f, lowpass_mask) # Low-pass fiter
    #f[240-size:240+size,320-size:320+size] = 0
    f = numpy.fft.ifftshift(f) # De-centering
    invf = numpy.fft.ifft2(f) # Invrese Fourier trans.
    print invf
    image2 = numpy.abs(invf) #(f_amp)
...
```
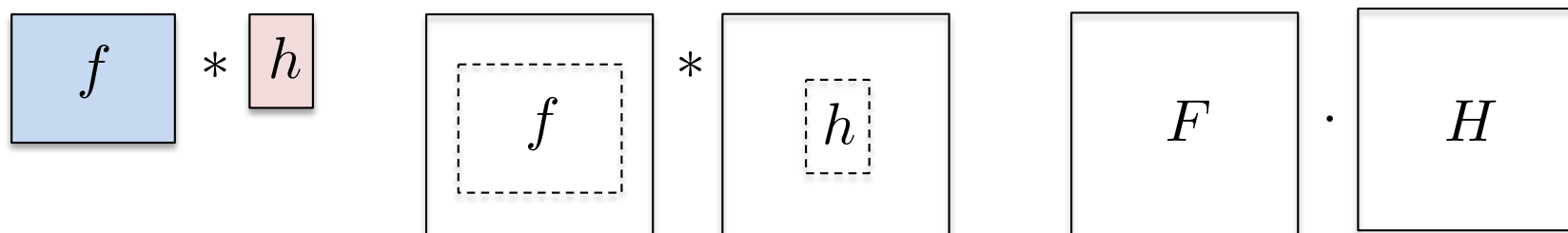
Fourier transform

- 畳込みとフーリエ変換：畳込みは周波数空間では積に

$$g = f * h \quad \longrightarrow \quad G = F \cdot H$$

f * h f * h F · H

（フーリエ変換のシフト不変性にも注）
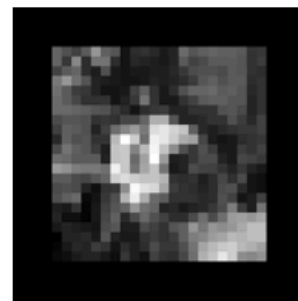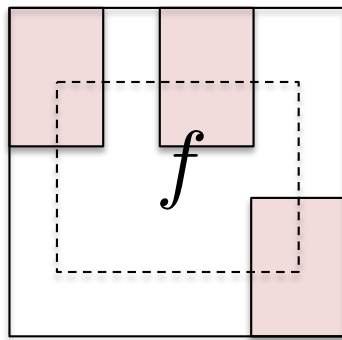
<div>

ffttest2.py

```
lowpass_mask = numpy.zeros((480,640))
size = 30
#lowpass_mask[240-size:240+size, 320-size:320+size] = 1
gauss = gauss2D.filter((size*2-1,size*2-1), size/5)
lowpass_mask[240-size:240+size, 320-size:320+size] =¥
        gauss/numpy.max(gauss)
```
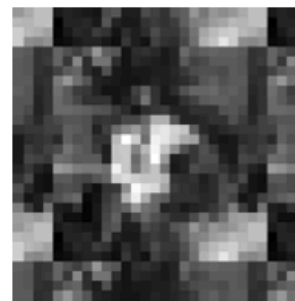
</div>

ヒント：ガウス関数のフーリエ変換はガウス関数になる
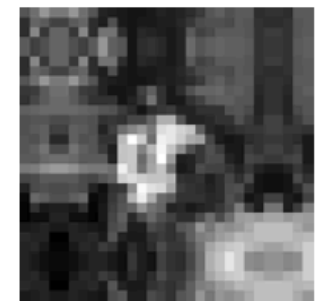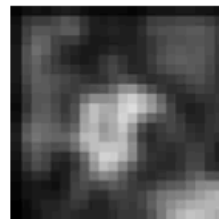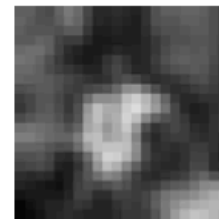
画像の「ふち」の扱い

- 畳込みや相関を計算するとき，画像の「外側」をどう考えるかに自由度がある

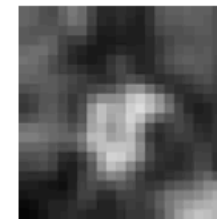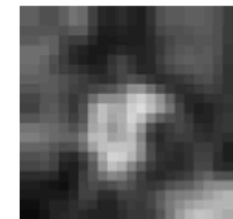Figure 3.13 Border padding (top row) and the results of blurring the padded image (bottom row). The normalized zero image is the result of dividing (normalizing) the blurred zero-padded RGBA image by its corresponding soft alpha value.

[Szeliski10]

Nonlinear filtering: Bilateral

- ## Bilateral filtering (edge-preserving filter)

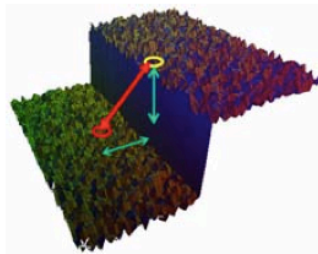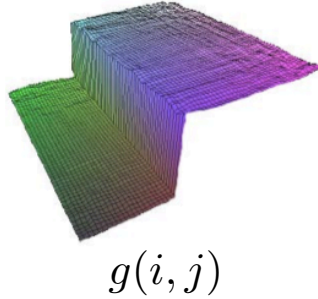$$g(i,j) = \frac{\sum_{k,l} f(i+k, j+l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$
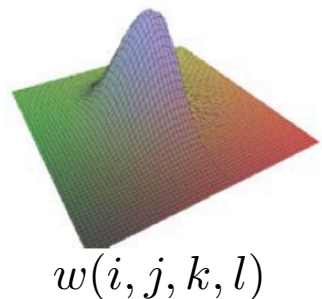
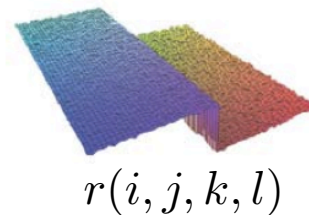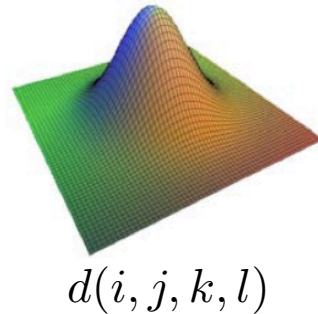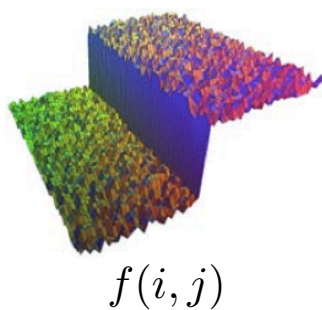$$w(i,j,k,l) = r(i,j,k,l) d(i,j,k,l)$$

Domain filter:

$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right)$$

Range filter:

$$r(i,j,k,l) = \exp\left(-\frac{\|f(i,j) - f(k,l)\|^2}{2\sigma_r^2}\right)$$

$f(i,j)$

$d(i,j,k,l)$

$r(i,j,k,l)$

$w(i,j,k,l)$

$g(i,j)$

[Szeliski10]

Nonlinear filtering: Bilateral

bilateral.py

```python
import numpy, cv2

image = cv2.resize(cv2.imread('1st/lion.jpg', 0), (720, 405))

# Add Gaussian noises to the image
noise = numpy.random.randn(image.shape[0],image.shape[1])
image = image.astype(numpy.float32) + 10*noise
image = numpy.clip(image, 0.0, 255.0)

filtered = cv2.GaussianBlur(image, (0,0), 2)
#filtered = cv2.bilateralFilter(image.astype(numpy.float32),¥
                                               -1, 20, 5)
#filtered = cv2.medianBlur(image.astype(numpy.uint8), 3)

cv2.imshow('Original', image.astype(numpy.uint8))
cv2.imshow('Filtered', filtered.astype(numpy.uint8))
```

Nonlinear filtering: Median

- ## Median filtering
  - 当該ピクセルまわりのフィルタ領域内の中央値を出力とする
  - Shot noiseに有効

```
3 5 6 3
3 2 1 4         sort                    median
2 3 1 2    ────────→   1 1 2 2 2 3 3 4 4  ────────→   2
4 3 4 2
```
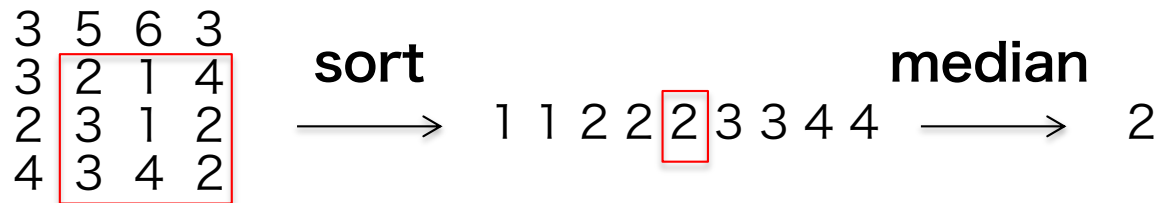
```
# Simulate "shot noise"                        median.py
noise_x = (numpy.random.rand(10000,1)*360).astype(int)
noise_y = (numpy.random.rand(10000,1)*405).astype(int)
image[noise_y, noise_x] = numpy.random.rand(10000,1)*255

#filtered = cv2.GaussianBlur(image, (0,0), 2)
#filtered = cv2.bilateralFilter(image.astype(numpy.float32), -1,
20, 5)
filtered = cv2.medianBlur(image.astype(numpy.uint8), 3)
...
```
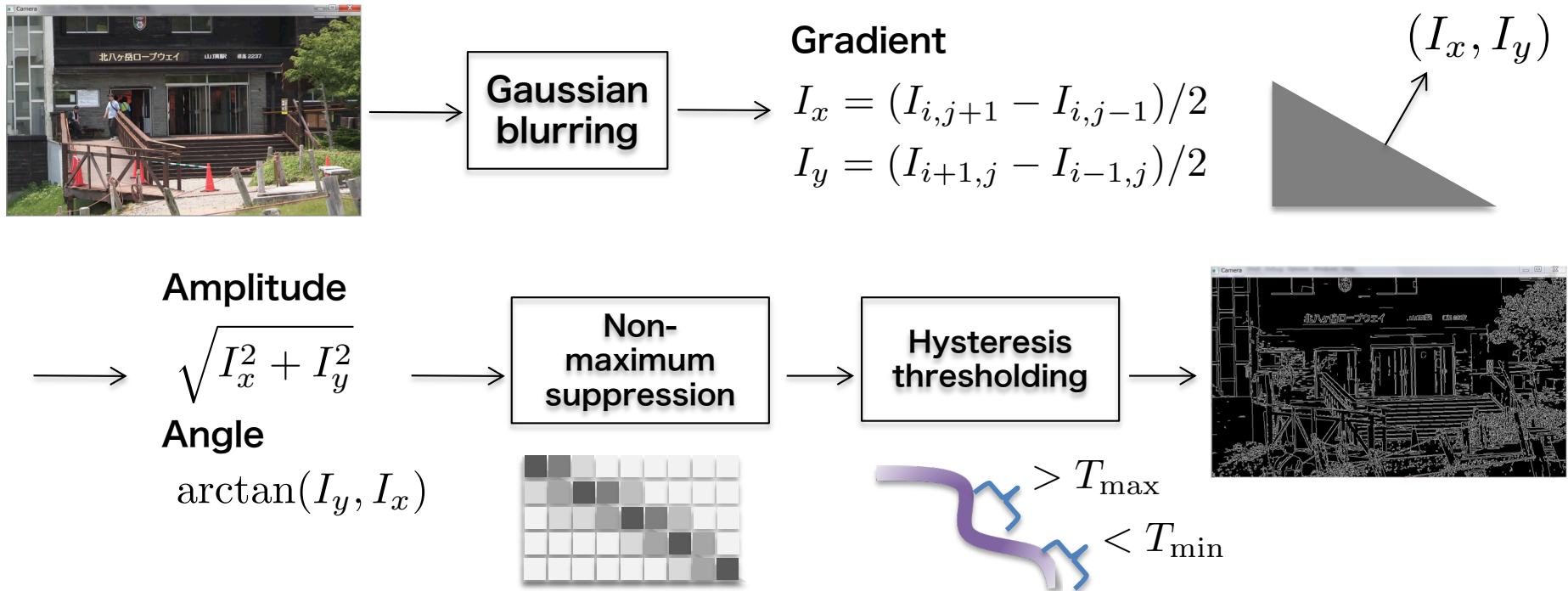
Edge detection

- ## Canny edge detector [Canny1986]

Gradient

$$I_x = (I_{i,j+1} - I_{i,j-1})/2$$
$$I_y = (I_{i+1,j} - I_{i-1,j})/2$$

(I_x, I_y)

Amplitude

$$\sqrt{I_x^2 + I_y^2}$$

Angle

$$\arctan(I_y, I_x)$$

Gaussian blurring

Non-maximum suppression

Hysteresis thresholding

$> T_{\max}$

$< T_{\min}$

```
while 1:
    stat, image = cap.read()
    image = cv2.Canny(image, 100, 200)
    cv2.imshow('Camera', image)
```

edge.py

Distance transform

- 各点から図形までの最短距離を値とするマップ（画像）

- 距離に何を選ぶかで自由度がある
- 応用

- 骨格線(Skelton)の計算
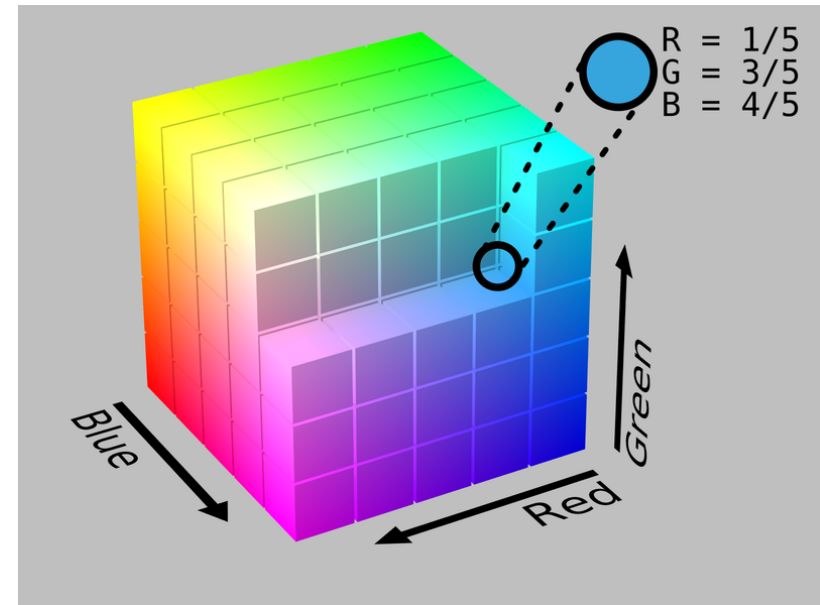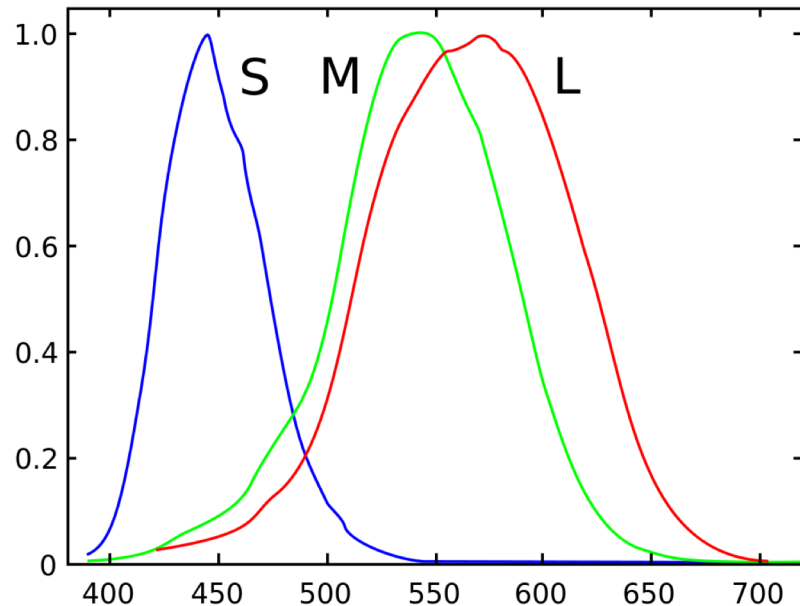- path planning
- 複数方向から計測した形状のマージ

disttrans.py

```python
logo = 255-cv2.imread('tohoku_logo.png', 0)
distmap = cv2.distanceTransform(logo, cv2.DIST_L2, 0)

cv2.imshow('Original', logo)
cv2.imshow('Result', distmap/numpy.max(distmap))
```

色の基礎

- ２種類の視細胞
  - 杆体（rod cells）：輝度の知覚
  - 錐体（cone cells）：S, M, Lの３タイプ → 色の知覚
- RGB表色系
  - R(700nm), G(546.2nm), B(435.8nm)
  - RGB→Grayscale: cvtgray.py

http://en.wikipedia.org/wiki/File:RGB_Cube_Show_lowgamma_cutout_b.png

様々な色の表現：表色系と色空間

- ## XYZ表色系
  - RGB表色系で表せない色を表現
  - Yは輝度を，Zは青の色あいを，Xはそれ以外を表す
  - xy色度図：x=X/(X+Y+Z), y=Y/(X+Y+Z)
- ## L*a*b, L*u*v色空間
  - XYZ空間を非線形変換したもの
- ## HSV
  - Hue（色相＝色味）
  - Saturation（彩度）
  - Value（明度）
- ## この他にHSLやYCrCb
  - OpenCVでサポート

http://ja.wikipedia.org/wiki/ファイル:CIExy1931.png

色検出
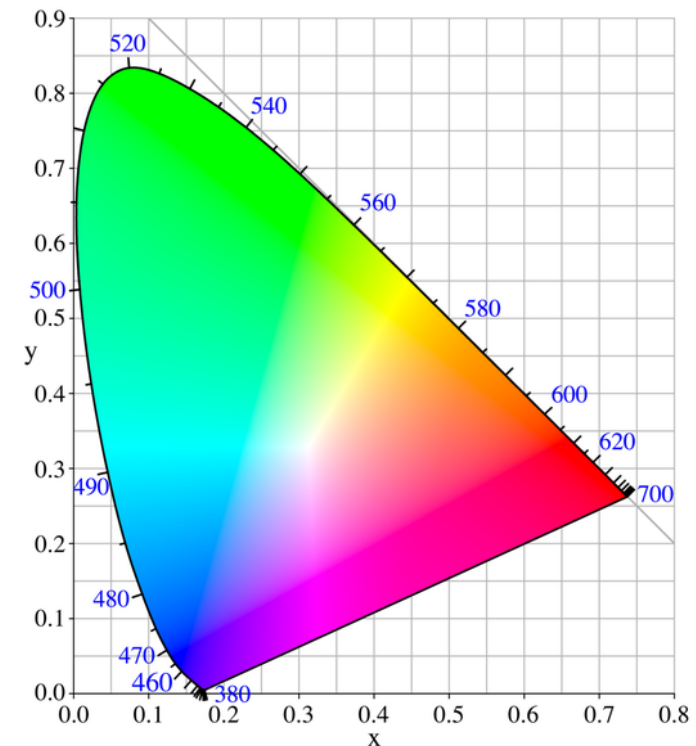
```
...                                              colspace.py
x0, y0, size = 320, 240, 10
mode, hoffset = 0, 0

while 1:
    stat, image = cap.read()

    if mode == 0:
         cv2.rectangle(image, (x0-size,y0-size),¥
                              (x0+size,y0+size), (0,0,255))
    else:
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)¥
                          + numpy.array([hoffset,0,0])
        mask = cv2.inRange(hsv, (hue+hoffset,30,30),¥
                                 (hue+hoffset,255,255))
        image[:,:,1] = numpy.bitwise_or(image[:,:,1], mask)

    cv2.imshow('Camera', image.astype(numpy.uint8))
    key = cv2.waitKey(10)

    (続く)
```
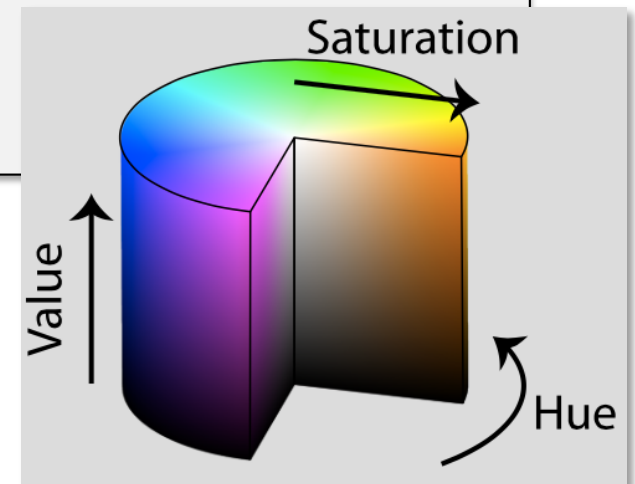
色検出

```
(続き)

if key == 0x1b: # ESC
    mode = mode + 1
    if mode == 2:
        break
    mean_rgb = numpy.mean(numpy.mean(¥
        image[y0-size:y0+size,x0-size:x0+size,:],0),0)¥
                                    .astype(numpy.uint8)
    mean_hsv = cv2.cvtColor(mean_rgb.reshape(1,1,3),¥
                                    cv2.COLOR_BGR2HSV)
    hue, sat = int(mean_hsv[0,0,0]), int(mean_hsv[0,0,1])
    hoffset = 90- hue
    print hue, sat

...
```

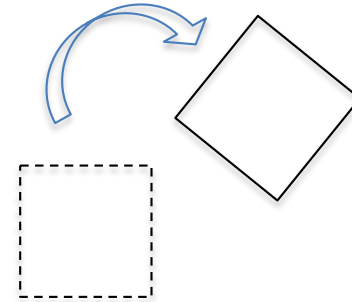注：cvtColorでは，hueは[0:179]にマッピングされる．上では，サンプルした色が90になるようにhue値をオフセットしている．

幾何学変換

- ## Euclidean transformation
  - ### 回転と並進移動

  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

  - ### 回転行列の性質 $\quad \mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
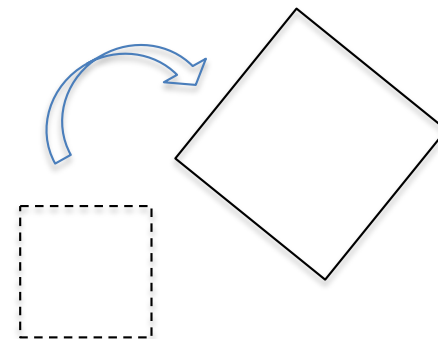
  $$\mathbf{R}\mathbf{R}^\top = \mathbf{R}^\top\mathbf{R} = \mathbf{I} \quad \det \mathbf{R} = 1$$

- ## Similarity transformation
  - ### ＋拡大縮小，相似変換

  $$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$
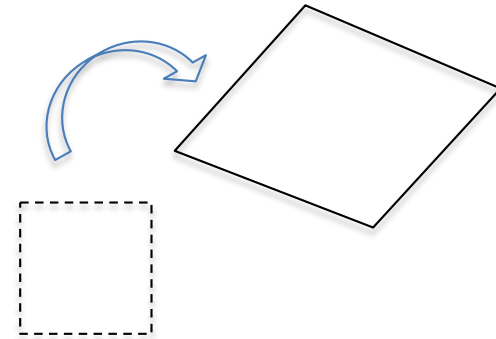
幾何学変換

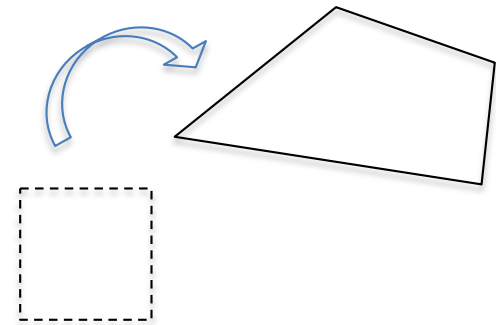- ## Affine transformation
  - ### 平行性の保存

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_{02} \\ a_{12} \end{bmatrix}$$

- ## Projective transformation
  - ### （２次元）射影変換

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}}$$

$$y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$
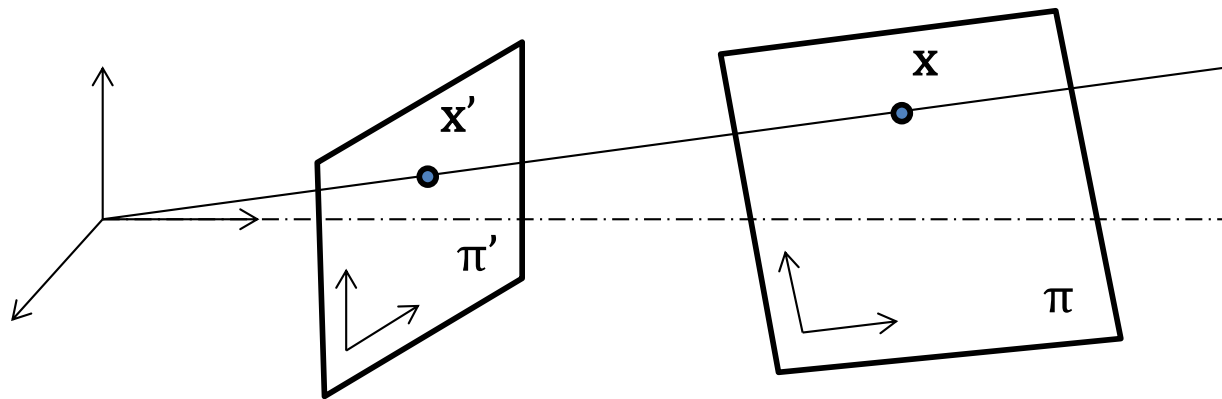
$$\text{or} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \propto \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$\propto$：両辺のベクトルの向きが等しい
（equality up to scale）

2D Projective Trans.

- 中心投影による２平面間の写像が典型的な例
  - 中心投影＝カメラの基本的モデル（pinhole camera）

Affine trans.を含む（projectiveの特殊ケースがaffine）

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \propto \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \propto \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

幾何学変換の階層性

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\, I \mid t \,\right]_{2\times3}$	2	orientation	
rigid (Euclidean)	$\left[\, R \mid t \,\right]_{2\times3}$	3	lengths	
similarity	$\left[\, sR \mid t \,\right]_{2\times3}$	4	angles	
affine	$\left[\, A \,\right]_{2\times3}$	6	parallelism	
projective	$\left[\, \tilde{H} \,\right]_{3\times3}$	8	straight lines	

Table 2.1 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[0^T\ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

[Szeliski10]
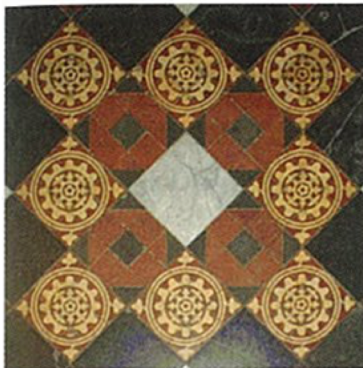
[Hartley-Zisserman03]

画像の幾何学変換

- ## Similarity trans.
  - 画像の中心を回転中心とし，反時計回りに45度回転し，1.5倍

```
h, w, c = image.shape # height, width, # of channels
M = cv2.getRotationMatrix2D((w/2, h/2), 45, 1.5)
image = cv2.warpAffine(image, M, (w, h))
```

- ## Affine trans.
  - 画像の３隅の点の変換先を指定

```
src = numpy.array([(0, 0), (w, 0), (0, h)], numpy.float32)
dst = numpy.array([(0, 0), (w-80, 40), (20, h-50)],¥
                            numpy.float32)
M = cv2.getAffineTransform(src, dst)
image = cv2.warpAffine(image, M, (w, h))
```

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_{02} \\ a_{12} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix}$$

画像の幾何学変換: Projective trans.

- 別の名称：homography, colineation etc.

projective.py

```
image = cv2.imread('floor.png')

src = numpy.array([(93, 58), (51, 196), (308, 190), (265,55)],
numpy.float32)
dst = numpy.array([((w-h)/2,0), ((w-h)/2,h), ((w+h)/2,h),
((w+h)/2,0)],¥                                    numpy.float32)
M = cv2.getPerspecriveTransform(src, dst)
image = cv2.warpPerspective(image, M, (w, h))

cv2.imwrite('floor_square.png', image)
```