Windowsパソコンで始める
短いコードですぐ試せる

画像処理シリーズ

やさしい画像処理入門

PythonとOpenCVで

澤田 英宏，安川 章
共著

CQ出版社

第1章

画像処理の前に…基本操作をマスタしよう

安川 章，澤田 英宏

1-1 画像ファイルを開く　プログラム名：imread.py

安川 章

画像ファイルの読み込みにはimread関数を用います（**リスト1**）．OpenCVではさまざまなファイル形式（bmp，jpg，png，tifなど）に対応しています．imread関数の第2引数（flags）を指定しない場合，グレースケール画像ファイルを指定しても，3チャネルのカラー画像データとして読み込まれます．

戻り値はNumPyのndarray形式の配列に格納された画像データです．グレースケール画像の場合は［y，x］の順の2次元配列，カラー画像の場合は［y，x，チャネル］の3次元配列で，チャネルはB，G，R

リスト1　画像ファイルの読み込み

```
import cv2  #cv2というライブラリを読み込む

# 画像ファイルの読込（戻り値はNumPyの配列（ndarray）に格納された画像データ）
img = cv2.imread("./images/Image.jpg", cv2.IMREAD_UNCHANGED)
                   （第1引数）                （第2引数）
# 画像の幅、高さ、チャンネル数の取得
if img.ndim == 2:
    # NumPy配列が2次元配列のときモノクロ画像
    channel = 1
    height, width = img.shape  #アンパック代入と呼ばれ，1つの箱に入っているデータを別々の変数にバラバラに分ける
elif img.ndim == 3:
    # NumPy配列が3次元配列のときカラー画像
    height, width, channel = img.shape

print(f"Width={width}, Height={height}, Channel={channel}")

# 画像の表示
cv2.imshow("Image", img)  #Imageという名前のウィンドウを作り，変数imgの中身を表示
cv2.waitKey()  #この一行がないとプログラムは画像を一瞬表示した直後に終了しウィンドウが勝手に閉じてしまう
```

（cv2ライブラリの中にあるimreadという関数を使う）

表1　画像の読み込みimreadの引数

	引　数	説明：画像ファイルを開く	
第1引数	filename	画像ファイル名を指定する	
第2引数	flags	画像の読み込みモードを指定する．主な値は以下の通り	
		cv2.IMREAD_UNCHANGED	画像ファイル形式のまま読み込む
		cv2.IMREAD_GRAYSCALE	グレースケール画像として読み込む
		cv2.IMREAD_COLOR	カラー画像として読み込む（初期値）
	戻り値	説　明	
	retval	画像データが格納されたNumPyのndarray配列	

の順となります.

```
retval = imread(filename, flags = cv2.IMREAD_COLOR)
```

引数を**表1**に示します.

imread関数の第1引数で指定する画像ファイル名を，フルパス以外で指定すると，カレント・ディレクトリからの相対パスとなります．カレント・ディレクトリの場所は実行したプログラム（imread.py）の場所とは限らないため，プログラム実行時にエラーが出る場合，imread関数を呼ぶ前にカレント・ディレクトリを自分自身のファイル（__file__）の場所に移動してから実行してください（**リスト2**）.

リスト2　プログラム実行時に画像ファイルが見つからずエラーが出るとき

```
import os
import cv2
# 自分自身のファイル・ディレクトリにカレント・ディレクトリを移動
os.chdir(os.path.dirname(os.path.abspath(__file__)))
# 画像ファイルの読み込み（戻り値はNumPyの配列ndarrayに格納された画像データ）
img = cv2.imread("./images/Image.jpg", cv2.IMREAD_UNCHANGED)
```

1-2　画像ファイルの保存　プログラム名：imwrite.py

安川 章

画像ファイルの保存にはimwrite関数を用います．imread関数と同じようにbmp，jpg，png，tifなどのファイル形式に対応しています.

```
retval = imwrite(filename, img[, params])
```

引数を**表1**に示します.

リスト1の例では全面がブルー（255，0，0）の画像をimagesフォルダに保存します.

表1　imwrite関数の引数

引　数	説明：画像ファイル保存
filename	画像ファイル名を指定する
img	画像データが格納されたNumPyのndarray配列
params	オプション設定（省略可）
戻り値	説　明
retval	True：成功，False：失敗

リスト1　画像ファイルの保存

```
# 画像ファイルの保存
import cv2
import numpy as np

# NumPy形式の画像データ
img = np.full((240, 320, 3), (255, 0, 0), dtype = np.uint8)
# 画像ファイルに保存
cv2.imwrite("./images/BlueImage.jpg", img)
```

1-14 動画ファイルの幅，高さ，フレーム・レートの取得 プログラム名：video_file_property.py

安川 章

動画ファイルに保存されている画像の幅や高さなどを取得するには，VideoCaptureクラスのget関数を用います．

```
<フレーム画像の幅の取得例>
width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
```

主な値は**表1**の通りです．

表1　動画ファイルのフレーム幅や高さ，レートの取得

値	説　明
cv2.CAP_PROP_FRAME_WIDTH	フレーム画像の幅
cv2.CAP_PROP_FRAME_HEIGHT	フレーム画像の高さ
cv2.CAP_PROP_FPS	フレーム・レート（fps，1秒当たりの再生フレーム数）
cv2.CAP_PROP_FRAME_COUNT	全体のフレーム数
cv2.CAP_PROP_POS_FRAMES	再生中の位置のフレーム番号
cv2.CAP_PROP_POS_MSEC	再生中の位置の時間（ミリ秒単位）

1-15 ウェブ・カメラからリアルタイムに映像を取り込む プログラム名：CameraBase.py

澤田 英宏

リアルタイム画像処理を行う場合はPCに内蔵または外付けのウェブ・カメラから映像を取り出すことが必須となりますが，**リスト1**に示すようにOpenCVを使用することで，簡単に映像を取得できます．

次項1-16のリスト1に示すのは撮影済みのビデオ映像を取得するサンプル・コードです．内容は1-15項**リスト1**のカメラ映像取得プログラムとほとんど同じなので，比較しながら確認してください．

▶リスト1：004行…カメラ映像の取得設定

```
cv2.VideoCapture(0, cv2.CAP_V4L)
```

第1引数はUSB接続されたウェブ・カメラの認識順です．基本的に接続カメラが1台の場合は0にすれば動作します．

第2引数はJetson Nano以外の場合は設定する必要はありません．cv2.VideoCapture(0)と記述するだけで動作します．

▶リスト1：005行，006行…取得映像のサイズ設定

```
(cv2.CAP_PROP_FRAME_WIDTH, 640)
(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

取得映像のサイズ設定です．上記の設定では横640ピクセル，縦480ピクセルの映像を取得表示します．例えば1920，1080と設定すれば，フルハイビジョンの映像を取得しますが，各種処理に時間がかかり，実用度は下がります．また，使用するウェブ・カメラがサポートしていないサイズを指定すると，意図

リスト1　カメラ映像取得プログラム（CameraBase.py）
本プログラムはJetsonフォルダおよびPCフォルダとプロジェクトBaseImageに保存されています

```
000: import cv2
001:
002: if __name__ == '__main__':
003:
004:     cap = cv2.VideoCapture(0, cv2.CAP_V4L)
005:     cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
006:     cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
007:
008:     if not cap.isOpened(): # ビデオキャプチャー可能か判断
009:         print("Not Opened Video Camera")
010:         exit()
011:
012:     while True:
013:         ret, img = cap.read()
014:         if not ret:  # キャプチャー画像取得に失敗したら終了
015:             print("Video Capture Err")
016:             break
017:
018:         cv2.imshow("Final result",img)  # 画面表示
019:         if cv2.waitKey(10) > -1:
020:             break
021:
022:     cap.release()
023:     cv2.destroyAllWindows()
```

しないサイズで取得される可能性もあるため，カメラの仕様を確認して必要な解像度に設定してください．

▶リスト1：013行…映像取得と判定

カメラ映像を1フレーム取得して，OpenCVが扱える画像データに変換します．

```
ret,img = cap.read()
```

imgには，1フレーム分のカメラ映像が格納されます．retには取得時のステータスが格納され，カメラ映像の1フレーム分の取得状態が分かり，このフラグを判定することでimgの画像データを処理可能かどうかの判別に使っています．

▶リスト1：018行…映像の表示

OpenCVから利用できるユーザ・インターフェースを使って，実際に画面に映像を表示しています．

```
cv2.imshow("Final result", img)
```

第1引数は**図1**のWindow画面のタイトルを設定します．この名前はWindowの識別に使用する場面があるため，複数の画面を開く場合は同じ名前を付けないようにしてください．

▶リスト1：019行…キー入力待ち

画面表示時にキー入力を待ちます．この記述がない場合は画面が表示されませんので，映像を確認したい場合は必須の記述となります．

```
if cv2.waitKey(10) > -1
```

第1引数は，キー入力の待ち時間を1/1000秒単位で設定します．ここで1000と設定すれば1秒ごとに画面が切り替わる，コマ送りのような表示となります．

ここでは，if文の条件式の「> -1」は，どのキーが押されても条件式が真となります．特定のキーに対応したい場合は直接キー・コードを指定します．例えば「if cv2.waitKey(10) ==27」と設定す

第3章

図形描画

澤田 英宏，安川 章

3-1 直線の描画　プログラム名：Line.py

澤田 英宏

直線や四角，円などを画面に描画することは，さまざまな場面で使用する基本操作となります．**図1**に示す直線の描画から解説します．

●プログラム名：Line.py

Mat画像では左上が原点（$X=0$，$Y=0$）となり，座標はそこから増加します．右方向がX軸正方向，下方向がY軸正方向になります．

リスト1の012行が横線の描画で，013行が左上から右下にかけての斜め線となります．

▶リスト1：012行…横線の描画

```
cv2.line(img=img, pt1=(0, 240), pt2=(640, 240), color=color,
                                                    thickness=5)
```

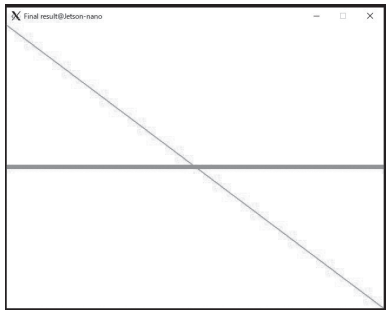


図1　直線を描画した例

リスト1　直線描画プログラム（Line.py）

```
000: import cv2
001: import numpy as np
002:
003: if __name__ == '__main__':
004:     print(cv2.__version__)
005:
006:     size = np.array([480, 640, 3])
                          # 縦480ピクセル 横640ピクセル 3チャンネル
007:     # 白ベースのMat画像を生成
008:     img = np.full(size, (255, 255, 255), dtype=np.uint8)
009:
010:     color = np.array([0., 255., 0.])  # BGR表記
011:     # 直線を描画
012:     cv2.line(img=img, pt1=(0, 240), pt2=(640, 240),
                                    color=color, thickness=5)
013:     cv2.line(img=img, pt1=(0, 0), pt2=(640, 480),
              color=color, thickness=1, lineType=cv2.LINE_AA)
014:
015:     cv2.imshow('Final result', img)
016:     cv2.waitKey(0)
017:
018:     cv2.destroyAllWindows()
```

4-2 周辺2×2画素の平均を取るバイリニア補間 プログラム名：Resize_Linear.py

澤田 英宏

●リサイズすると画像は劣化する

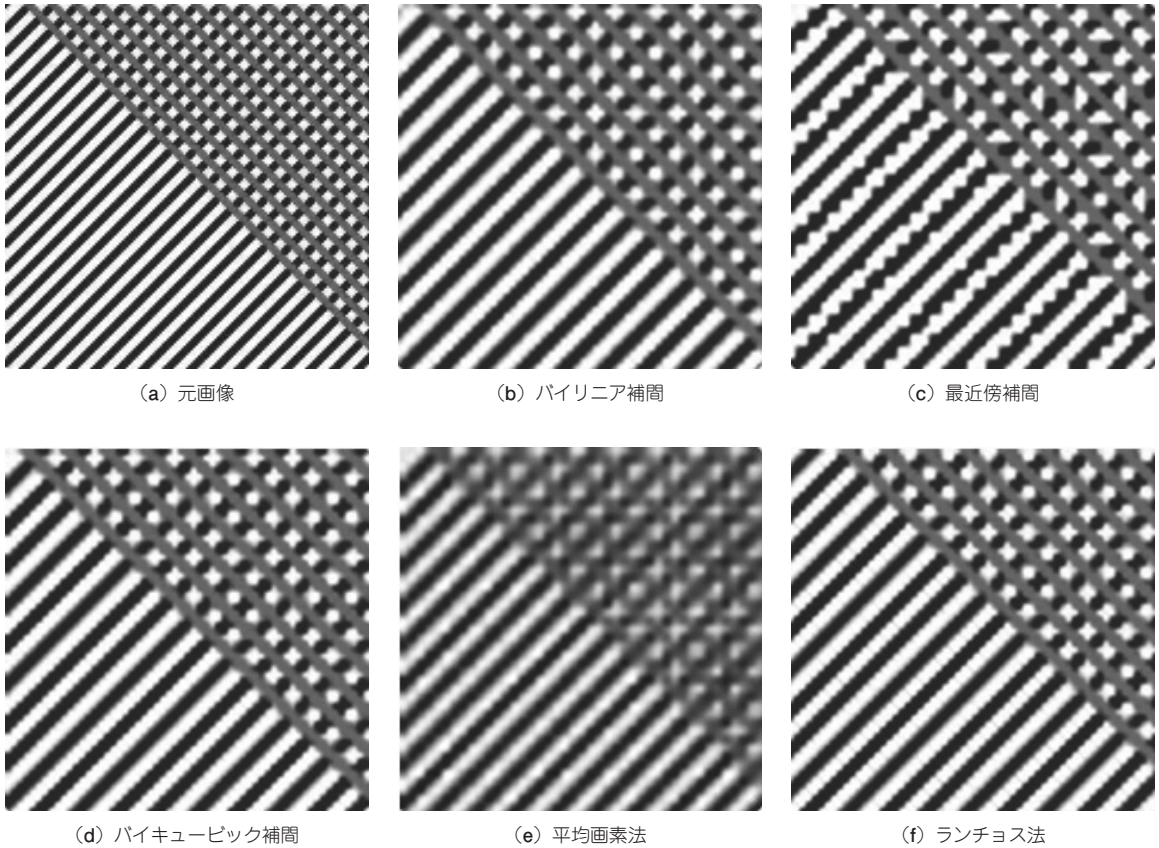
画像の縮小は元の映像から，縮小率に応じて間引きを行い，拡大は広がった部分に周囲の画素値に合わせて，なかった情報を書き足すなどして変換を行っているため，基本的に画像は劣化しています．

例えば縮小時には，元画にはなかった「モアレ」と言われる幾何学模様が発生したり，エッジ部分がギザギザになったりといった劣化が発生します．この劣化を目立たなくするためにさまざまなアルゴリズムが考え出されています．

●OpenCVでサポートされている補間アルゴリズムを使う

OpenCVでも幾つかのアルゴリズムがサポートされていますので使い方を紹介します．

今回は，補間効果を確認するために，コード上で**図1**(a)のような格子状のパターンを1000×1000ピクセルで生成して，この画像に対して640×640ピクセルへの縮小処理を行います［**図1**(b)］．



（a）元画像　（b）バイリニア補間　（c）最近傍補間

（d）バイキュービック補間　（e）平均画素法　（f）ランチョス法

図1　元画像と補間した画像の比較

実際のアルゴリズムごとの補間効果を紙面で確認することは難しいため，サンプル・プログラムを実行して確認してみてください．

動作確認の際は，各補間アルゴリズムの確認と併せて，**リスト1**の016行の`basePixSize = 640`の数字を変更します．

●周囲4つの画素値を参照

注目画素の周辺2×2画素，周囲の4つの画素値を参照して，それぞれの重みの平均値を求めて画素を補間する手法です．計算処理は少し重くなりますが，周辺画素を考慮しているため，単純補間に比べて画像の劣化を抑えることができる方法です．OpenCVでは，デフォルトはこの設定になります．

●プログラム名…Resize_Linear.py

▶リスト1：024行…リサイズ

```
dst = cv2.resize(src, (int(width * resizeRate),
        int(height * resizeRate)), interpolation=cv2.INTER_LINEAR)
```

引数は「4-1 画像の縮小1」で解説した内容と同じで，第3引数に補間のアルゴリズムを指定しています．ここで第3引数に`interpolation=None`を指定，または省略されていた場合は，初期値として`cv2.INTER_LINEAR`が指定されます．

リスト1　バイリニア補間プログラム（Resize_Linear.py）

```
000: import cv2
001: import numpy as np
002:
003: def main():
004:     img = createPatten()
005:     org = img.copy()
006:
007:     img = getResize(img)
008:
009:     cv2.imshow('Original', org)
010:     cv2.imshow('INTER_LINEAR', img)
011:     cv2.waitKey(0)
012:     cv2.destroyAllWindows()
013:
014: def getResize(src):
015:     """CPUを使用"""
016:     basePixSize = 240  # 縦横で大きい辺の変更したいサイズ
017:     height = src.shape[0]
018:     width = src.shape[1]
019:
020:     largeSize = max(height, width)  # 大きい方の辺のサイズ
021:     resizeRate = basePixSize / largeSize  # 変更比率を計算
022:
023:     # ---ここでサイズ変更---
024:     dst = cv2.resize(src, (int(width * resizeRate),int(height * resizeRate)),
                                                        interpolation=cv2.INTER_LINEAR)
025:
026:     return dst
027:
028: # サンプル画像を作成
```

第7章

膨張/収縮

澤田 英宏

7-1 クロージング/オープニング/勾配/トップハット/ブラックハット/収縮/膨張　プログラム名：Morph_Close.pyほか

澤田 英宏

●用途はさまざま…ノイズの除去や線の強調に

白黒画像に対して単純に「膨張/収縮を行うことで得られる結果」を利用する手法を，モルフォロジー変換と呼びます．通常は2値画像に対して行う変換ですが，カラー画像に対して変換を行うと，**図1**のようにモザイク画を生成できます．

ここでは，2値画像についてのみ解説しますが，読後にカラー画像も試してみてください．

●クロージング（プログラム名：Morph_Close.py）

画像の白い部分を指定回数ぶん膨張した後，同じ回数収縮させる処理で，適切な回数を処理すると，黒ノイズ除去に有効な処理となります（**図2**，**図3**）．また，通常のカラー画像に対してクロージング処理した場合は**図1**のようにモザイク画になります．

図1　カラー画像に対してモルフォロジー変換を行うとモザイク画を生成できる

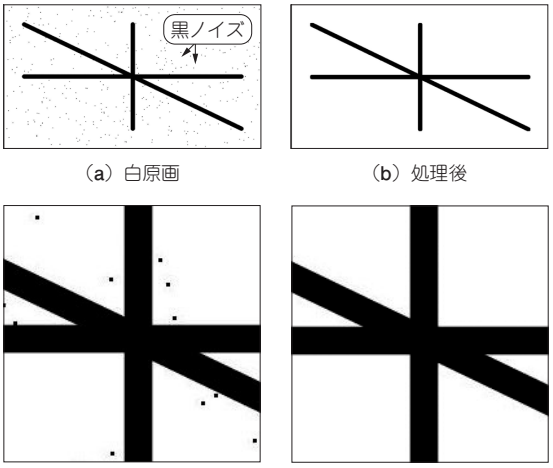
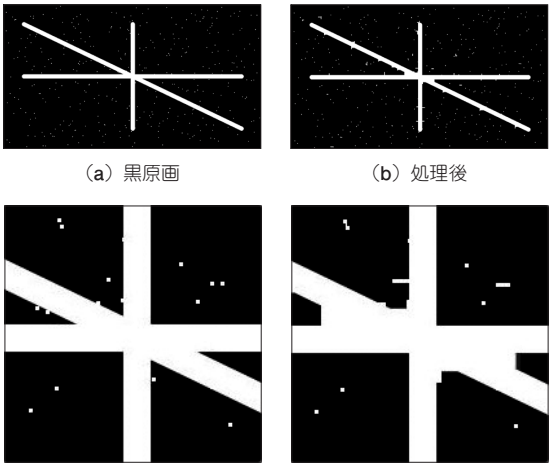


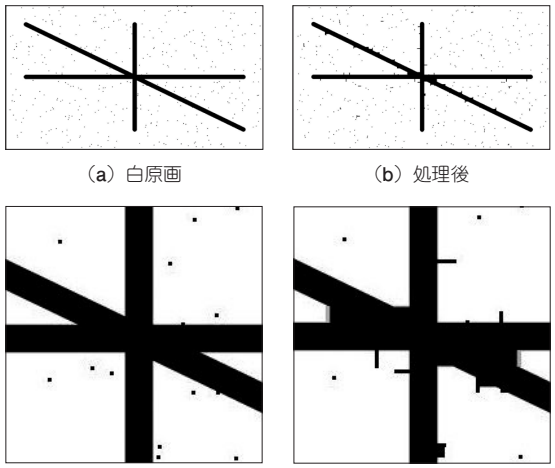
図2　白画像のクロージング処理



（a）黒原画　（b）処理後

（c）黒原画（拡大）　（d）処理後（拡大）

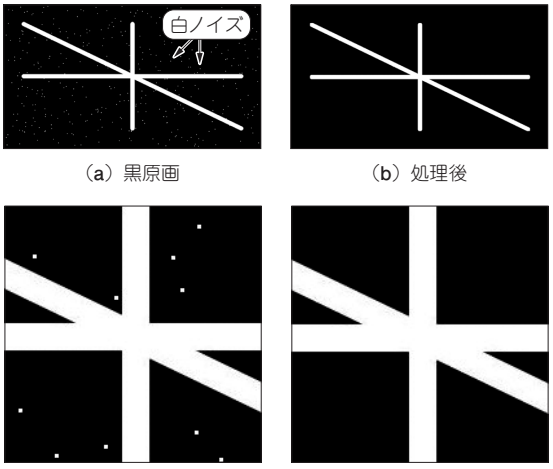
図3　黒画像のクロージング処理



（a）白原画　（b）処理後

（c）白原画（拡大）　（d）処理後（拡大）

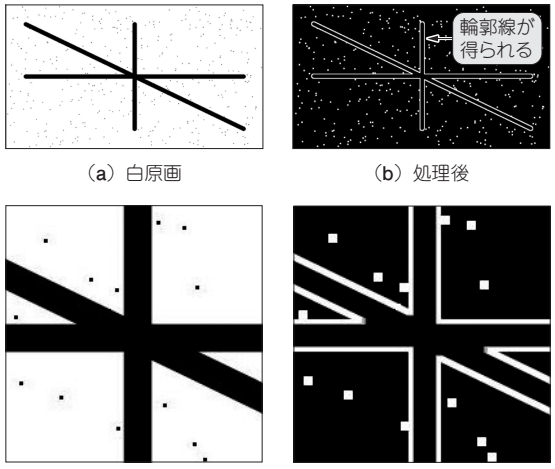
図4　白画像のオープニング処理



（a）黒原画　（b）処理後

（c）黒原画（拡大）　（d）処理後（拡大）

図5　黒画像のオープニング処理



（a）白原画　（b）処理後

（c）白原画（拡大）　（d）処理後（拡大）

図6　白画像の勾配処理

●オープニング（プログラム名：Morph_Open.py）

クロージングとは逆に適切な回数収縮して膨張することで，白ノイズ除去に有効な処理となります（**図4**，**図5**）．

●勾配（プログラム名：Morph_Gradient.py）

白い部分の膨張と収縮を繰り返した画像の差分を求める処理で，輪郭線を得ることができます（**図6**，**図7**）．

第8章

明るさ調整

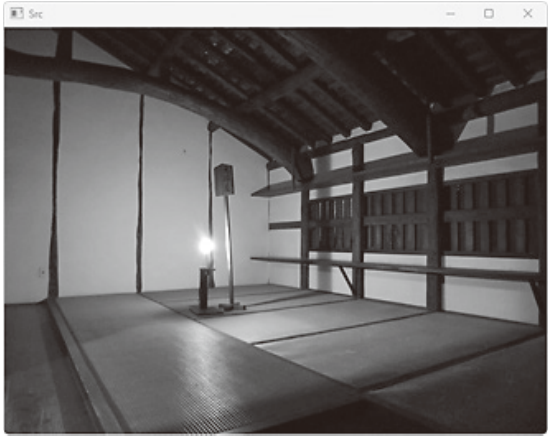
安川 章

8-1 ガンマ補正　プログラム名：gamma_correction.py

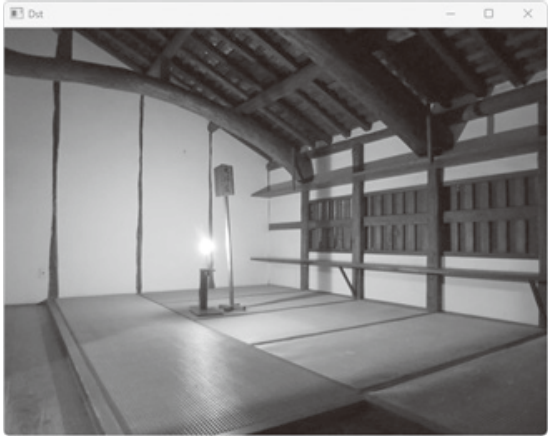
安川 章

●曲線の式を用いて変換する

ガンマ補正は画像の明るさを調整するとき，単に輝度値に指定の倍率を掛けるのではなく，暗い部分や明るい部分をより変化させるように，曲線の式を用いて変換を行います（**図1**）．



（a）処理前



（b）処理後

図1　ガンマ補正

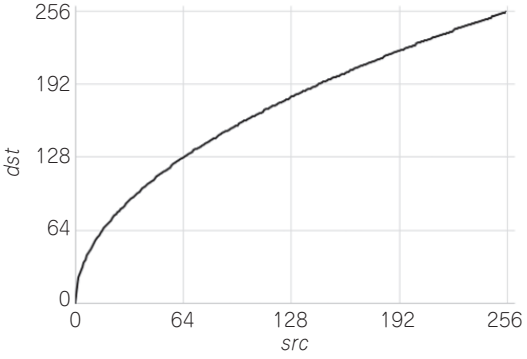


図2　ガンマ補正のカーブ

表1　入力輝度値に対する出力結果を配列に格納しておく

Src	0	1	2	3	…	252	253	254	255
Dst	0	16	23	28	…	253	254	254	255

リスト1　LUT関数を用いてガンマ補正を行う処理の関数

```
def gamma_correction(src: cv2.typing.MatLike, gamma: float) -> cv2.typing.MatLike:
      # ガンマ補正用のLookup tableを作成
      lut = np.zeros((256,1),dtype = 'uint8')
      for i in range(256):
          lut[i][0] = 255 * (i/255) ** (1.0/gamma)
      # ルックアップテーブル変換(ガンマ補正)
      dst = cv2.LUT(src, lut)
      return dst
```

リスト2　リスト1の関数を用いてガンマ補正を行う

```
# 画像の読込
src = cv2.imread("./images/ImageGamma.jpg")
# ガンマ補正
dst = gamma_correction(src, 2)
```

ガンマ補正の式は，入力輝度値を*src*，出力輝度値を*dst*，ガンマ補正値をγとすると，

$$dst = 255 \times \left(\frac{src}{255}\right)^{\frac{1}{\gamma}}$$

の式で表されます．例えば$\gamma = 2$のときは，**図2**のような曲線となります．

ガンマ補正では，画像の全画素の輝度値に対して変換式で計算をするのは非効率であるため，あらかじめ入力輝度値に対する出力結果を配列に格納しておきます（**表1**）．

●プログラム

この配列の名前をLUTとすると，画像に対するガンマ補正は，

```
dst=LUT[src]
```

のように表されます．この配列のことをルックアップ・テーブル（Lookup table）といい，OpenCVではLUT関数を用いて画像の変換を行うことができます．

ルックアップ・テーブルを作成し，LUT関数を用いてガンマ補正を行う処理を関数にまとめると，**リスト1**のようになります．

この関数を用いてガンマ補正を行うには，**リスト2**とすると，dstにガンマ補正された画像データが格納されます．

8-2 ヒストグラム平たん化 プログラム名：equalizeHist.py

安川 章

●ヒストグラム平たん化

ヒストグラム平たん化とは，ヒストグラムが平均的に分布するように輝度値を変換することです．言い換えると特定の輝度値に画素が集中しないようにすることで，画像のコントラストを改善します（**図1**）．**図2**に効果を示します．

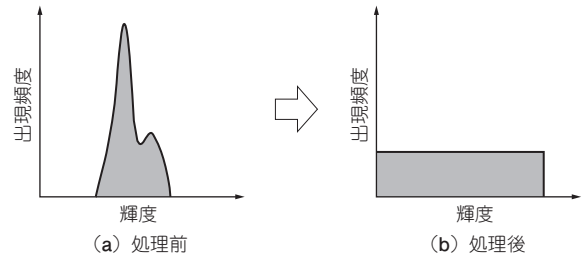
●プログラム

ヒストグラムの全ての頻度を平均の頻度にしようとすると，頻度が平均以上となる輝度値を複数の輝度値へ割り振る必要が出てくるため，実際には，頻度の分布密度が平均的になるように変換を行います．OpenCVではequalizeHist関数を用います．引数を**表1**に示します．

```
dst = cv2.equalizeHist(src)
```

equalizeHist関数で処理できるのは，8ビット1チャネルのグレースケール画像のみのため，カラー画像のヒストグラム平たん化を行うには，cvtColor関数でBGRからHSVへ変換し，split関数でH，S，Vへ分離したのち，明度Vに対してヒストグラム平たん化を行い，H，S，VからHSVへ結合し，HSVからBGRへ変換します（**リスト1**）．

2-4　プレーン結合も参照してください．

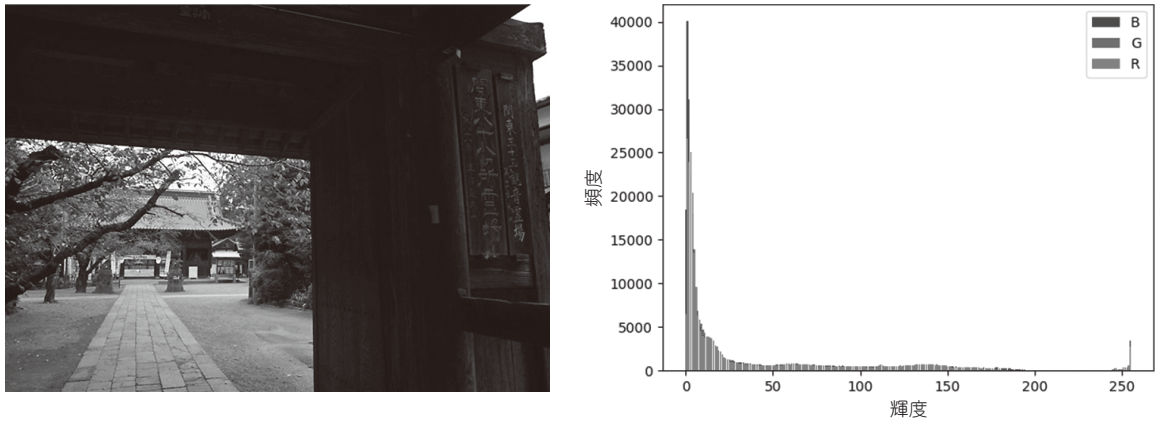


（a）処理前　（b）処理後

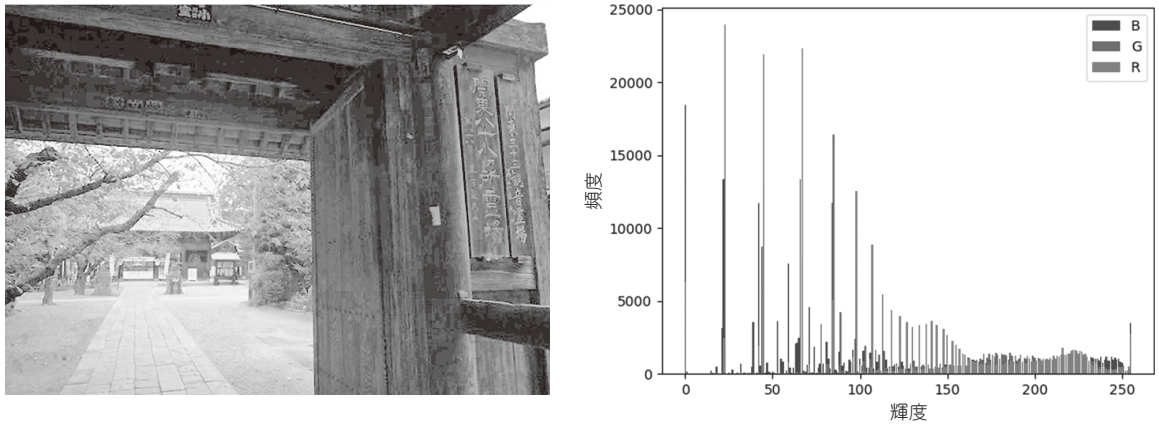
図1
ヒストグラム平たん化のイメージ

表1　equalizeHist関数の引数

引　数	説明：ヒストグラム平たん化
src	ヒストグラム平たん化前の元画像．8ビットのグレースケール画像のみ
戻り値	説　明
dst	ヒストグラム平たん化後の画像



（a）元画像とヒストグラム



（b）処理後の画像とヒストグラム

図2　ヒストグラム平たん化

リスト1　カラー画像のヒストグラム平たん化

```
# 画像ファイルの読込
src = cv2.imread("./images/ImageHist.jpg")
# BGRからHSVへ変換
img_hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV_FULL)
# HSV画像をH, S, Vそれぞれの画像に分割
img_h, img_s, img_v = cv2.split(img_hsv)
# ヒストグラム平たん化(処理が可能なのは8bitグレースケール画像のみ)
img_v_equalize = cv2.equalizeHist(img_v)
# H, S, Vを結合
img_hsv = cv2.merge((img_h, img_s, img_v_equalize))
# HSVからBGRへ変換
img_equalize = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2BGR_FULL)
```

9-5 凸包　プログラム名：convexHull.py

安川 章

凸包（とつほう）とは，点の集まりや輪郭を輪ゴムで留めるようなイメージで構成される凹みのない多角形のことです（**図1**）．主に物体の形状を単純化したり，特徴を抽出したりするために使われます．

OpenCVではconvexHull()関数を用いて求めます（**リスト1**）．

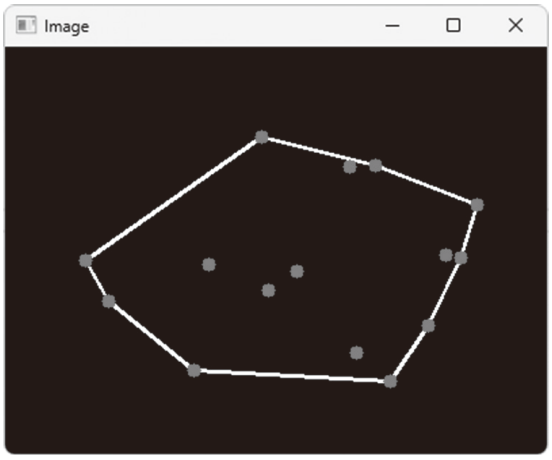


図1　凸包…凹みのない多角形で囲む

リスト1　点座標の集まりを凹みのない多角形で囲むプログラム

```
# ランダムな点
points = []
for i in range(15):
      x = int(random.uniform(50, 350))
      y = int(random.uniform(50, 250))
      points.append((x, y))

# 凸包
hull = cv2.convexHull(np.array(points,dtype='int32'))
# 凸包で囲まれた線を描画
cv2.polylines(img, [hull], True, (255,255,255), 2)
```

11-6 円　プログラム名：FindCircles.py

澤田 英宏

●概要

画像内の円形，または円形に近い物体だけを識別する機能です．**図1**は認識した座標データを使って円

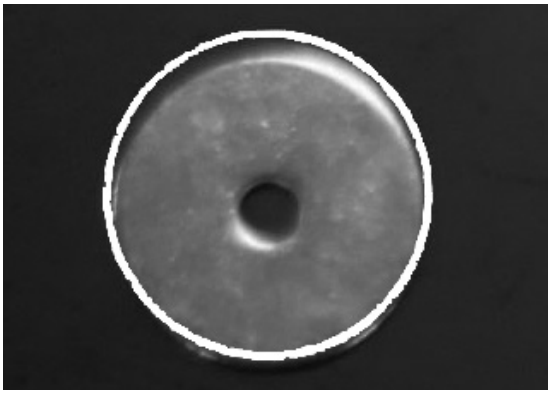


図1　画像中に円形物体があると円を描画

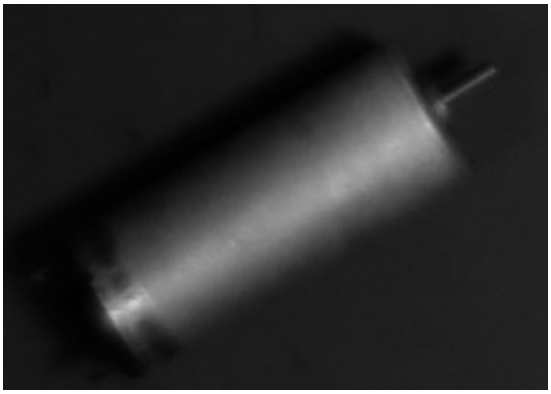
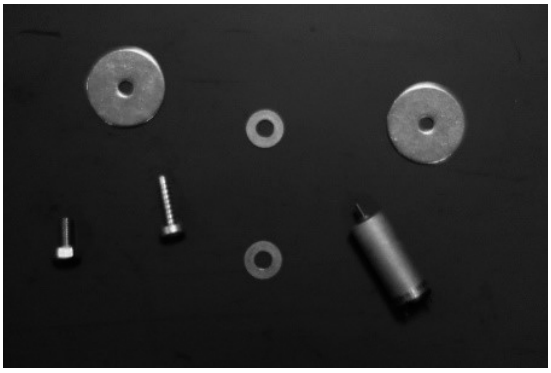
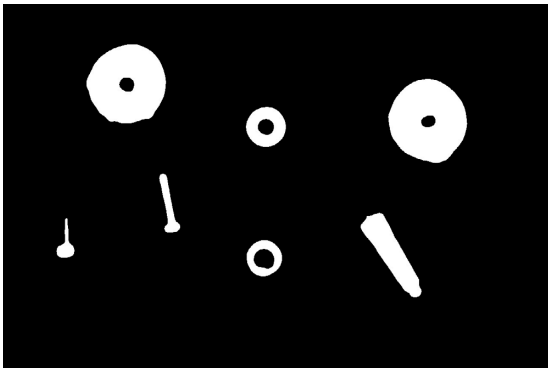


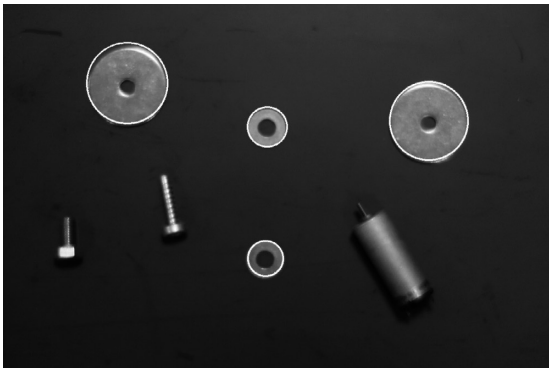
図2　画像中に円形が見つからない場合はなにもしない



（a）原画



（b）2値画像



（c）検出画像

図3
円形検出は複数の処理を組み合わせて行う

を描画しています. **図2**は円形として認識する情報がないため, 円を描画できていません. これらは**図3**(c)の検知画像を拡大したものですが, **図3**(c)の画像を確認すると円形の物体だけ認識できています.

●前処理

円の検出機能を実装するには, 前処理として「2-1 グレースケール変換」と「5-2 ぼかし変換」, 「2-7 2値化」を行っています.

●プログラム…FindCircles.py

▶リスト1：036, 040, 041行…前処理

```
gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, ksize, 0, 0)
ret, gray = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
                                                cv2.THRESH_OTSU)
```

円の検知は, カラー情報で処理するより, グレースケールで実行する方が, 検知精度が変わらず処理も速いので, 通常はグレースケールに変換後, ぼかし機能でノイズを消去した状態で実施します.

cv2.HoughCircles内部でCanny変換しているため, 2値に変換する必要はありませんが, ここでは原画に対する検知の試行錯誤の結果, 2値化してからこの後のcv2.HoughCirclesを実行した方が良い結果を示したので先に2値化で前処理をしています.

検知対象によっては2値化する必要がないこともあります. 対象に合わせて使い分けてください.

▶リスト1：045行…円の検知

変換した画像を利用してcv2.HoughCirclesで円の検知を実行しています.

```
circles = cv2.HoughCircles(image=gray, method=cv2.HOUGH_GRADIENT,
dp=1, minDist=20,  param1=100, param2=20, minRadius=None,
                                                maxRadius=None)
```

第1引数imageは入力画像です(グレースケールまたは2値画像).

第2引数methodは変換方法です. 現時点ではcv2.HOUGH_GRADIENTだけ使えます.

第3引数dpは検知処理する元画像の解像度です. 1で現状の画質で処理します. 1のままでよいと思います.

第4引数minDistは検出される円の中心同士の最小距離であり, 小さすぎると正しい円の周辺に誤検知した円が表示されます.

第5引数pram1は数字が小さいほど, Canny変換において多くのエッジを検出します.

第6引数param2は数字が小さいほど円形判定が緩くなり, 多く検知します.

第7引数minRadiusは検出する円の最小値です.

第8引数maxRadiusは検出する円の最大値です.

リスト1　円を検出するプログラム FindCircles.py

```
000 import cv2
001 import numpy as np
002 import time
003
004
005 def main():
006     img = cv2.imread('../IMG_1610.JPG')
007
008     timeStart = time.time()
009
010     img = getResize(img)
011     dst = getCircles(img)
012
013     timeEnd = time.time()
014     print("{0} = {1}".format('CPU', (timeEnd - timeStart) * 1000) + "/ms")
015
016     cv2.imshow("Final result", dst)
017     cv2.waitKey(0)
018
019     cv2.destroyAllWindows()
020
021
022 def getResize(src):
023     basePixSize = 1280  # 縦横で大きい辺の変更したいサイズ
024     height = src.shape[0]
025     width = src.shape[1]
026
027     largeSize = max(height, width)  # 大きい辺のサイズ
028     resizeRate = basePixSize / largeSize  # 変更比率を計算
029     shotSize = min(height, width) * resizeRate
030     dst = cv2.resize(src, (int(width * resizeRate), int(height * resizeRate)))
031
032     return dst
033
034
035 def getCircles(src):
036     gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
037     # ガウシアンフィルタ
038     ksize = (25, 25)  # 正の奇数で指定する(ここの数字を変えると効果が変更できる)
039     # フィルターの実行
040     gray = cv2.GaussianBlur(gray, ksize, 0, 0)
041     ret, gray = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
042     cv2.imshow('Preimg', gray)
043
044     # 円を検知した座標を取得する
045     circles = cv2.HoughCircles(image=gray, method=cv2.HOUGH_GRADIENT, dp=1, minDist=20,
                                   param1=100, param2=20, minRadius=None, maxRadius=None)
046     if circles is not None:
047         circles = np.uintp(np.around(circles))
048         for circle in circles[0, :]:
049             # 元画像に取得した座標の円を描画する
050             dst = cv2.circle(src, (circle[0], circle[1]), circle[2], (255, 0, 255), 2)
051
052     return dst
    省略
```

●出力結果

図3(c)での検出画像では，円形の物体が認識されてマークされていますが，それ以外の物体は検出されていないことが分かります．

各引数のパラメータを変更して，どのように変化するのか確認してみてください．

▶リスト2：023行…検知した直線の描画

```
for x1, y1, x2, y2 in lines[:, 0]:
```

012行の戻り値は，cv2.HoughLinesとは違い，直線の座標がそのまま得られます．

●出力結果

図1(c)は道路の白線を検出した結果です．
それぞれのパラメータは同じ値を設定しています．

11-8 特定領域の識別「ブロブ解析」プログラム名：Blob_Analysisフォルダ内各種

安川 章

ブロブ(blob)とは，日本語で小塊を意味します．画像処理において，画像を2値化したとき，白の画素がつながったそれぞれの領域をブロブといいます(**図1**)．各ブロブ領域の面積や幅，高さ，形状などの特徴量を取得することをブロブ解析といいます．

●ブロブ解析でできること

製造ラインにおける画像処理には，次に示すようなさまざまなタスクがあります．

- 外観検査: 製品表面の欠陥やキズを検出する
- 寸法測定: 製品寸法を測定する
- 位置合わせ: 製品をロボット・アームなどでピックアップする位置を特定する
- 識別: 製品の種類やロット番号を識別する
- 分類: 製品を種類ごとに分類する

ブロブ解析は，これらのタスクのうち，特に外観検査，位置合わせ，識別，分類に利用されます．
ブロブ解析を行うと，

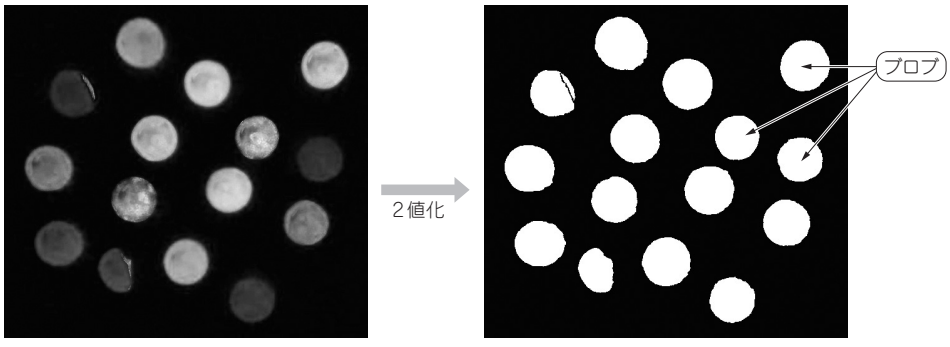


図1　ブロブ…画像を2値化したとき，白の画素がつながったそれぞれの領域
このブロブの面積/幅/高さ/形状などの特徴量を取得することを「ブロブ解析」という

- 部品の個数カウント
- 部品の欠けや割れ，ゴミの付着の検査
- キュウリのような曲がりの形状検査

などが可能になります．

ブロブ解析は，画像中に複数の部品やキズ，割れなどが存在する場合に，それぞれの領域（ブロブ）に対して各種特徴量を求め，各領域のOK/NG判定を行う場合などに用いられます．検査対象が画像中に1つの場合や，検査する場所が特定できる場合（被写体がほぼ動かない場合）は，必ずしも2値化→ラベリング処理を行うとは限らず，ソーベル・フィルタなどの輪郭抽出処理を行い，エッジ間の幅や高さ，位置などの検査を行う場合もあります．

●ブロブ解析に向く3つの関数/クラス

ブロブ解析は「2値化→ラベリング→ブロブ解析」の順で処理を行います．OpenCVでは次の3つの関数/クラスを用いることができます．

1. 2値化された画素の連続領域を抽出するconnectedComponentsWithStats()関数
2. 領域の輪郭を抽出するfindContours()関数
3. ブロブ解析による領域の抽出に特化したSimpleBlobDetectorクラス

▶3は大きめの領域を認識してしまう傾向がある

上記の3のSimpleBlobDetectorクラスは，各ブロブ領域の輝度値，面積，円形度，縦横比，凸性の範囲を指定してブロブを抽出できます．ただし，注意しておくべき点があります．SimpleBlobDetectorクラス内で画像の2値化処理を行う際に，2値化しきい値を徐々に変化させながらブロブ抽出を行い，中心座標が近いブロブを1つのブロブとして扱うというやや特殊な方法を取るので，実際に抽出したい領域よりも大きめの領域を1つのブロブとして認識してしまう傾向があります．

▶1と2は意図した領域に対してブロブ解析を行える

1のconnectedComponentsWithStats()関数，2のfindContours()関数では，事前に自分で画像を2値化する必要があるため，結果的に意図した領域に対してブロブ解析を行うことができます．そのため今回connectedComponentsWithStats()関数とfindContours()関数を使った方法について紹介します．

▶2はより多くの輪郭の特徴量を取得できる

1のconnectedComponentsWithStats()関数では，連続領域のラベル個数，ラベリング画像，矩形領域，面積，重心を取得できますが，2のfindContours()関数では，さらに多くの輪郭の特徴量を取得できます．ただし，connectedComponentsWithStats()関数の面積は，連続領域の画素数であるのに対し，findContours()関数では，輪郭の内側の面積となることに気をつけてください．さらに，輪郭の内側に黒い領域があっても，その面積は考慮されません．例えば，**図2**のように7×6画素の中に6画素の黒い画素がある場合の面積はconnectedComponentsWithStats()関数では36，findContours()関数では30となります．

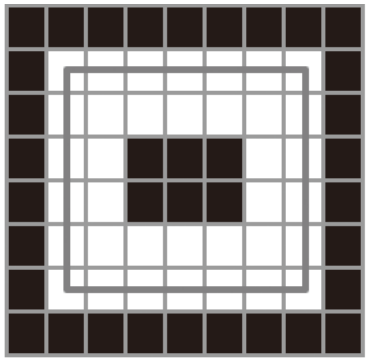


図2　関数によって面積として取得される領域が異なる
7×6画素の中に6個の黒画素がある場合の面積はconnectedComponentsWithStats()関数では36，findContours()関数では30となる

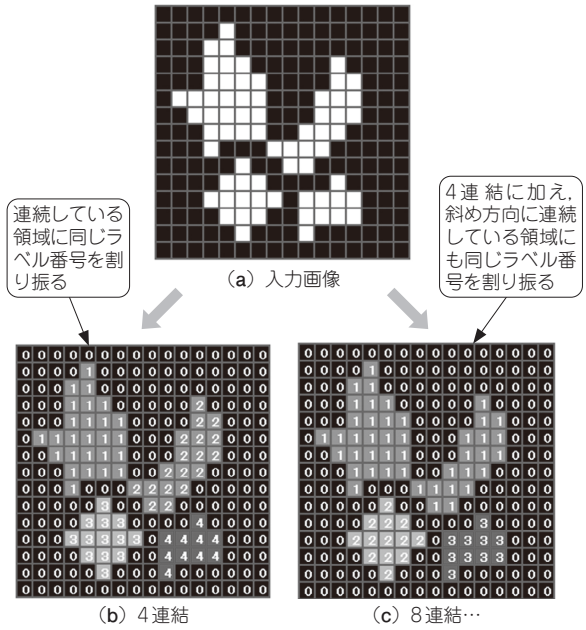


図3　ラベリング処理には4連結と8連結の2種類がある

●利用例1：connectedComponentsWithStats()関数

▶2値化された画素の連続領域を抽出する

2値化した画像の各画素が連結している領域に同じラベル番号を割り振るラベリング処理を行い，各領域（ブロブ）ごとの外接矩形範囲，面積（画素数），重心を取得します．

ラベリングには，2値化された画像の上下/左右方向が連続している領域を同じラベル番号に割り振る4連結と，さらに斜め方向を追加した8連結があります（**図3**）．

リスト1にプログラム例を，**図4**に外接矩形範囲，面積，重心の取得結果を示します．

表1に関数の引数を示します．

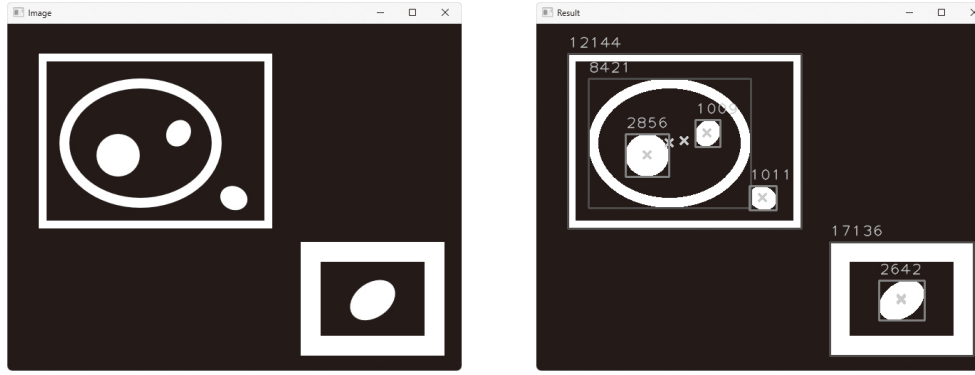
表1　connectedComponentsWithStats()関数

```
connectedComponentsWithStats( image[, labels[, stats[, centroids[, connectivity[,
ltype]]]]] ) -> retval, labels, stats, centroids
```

引　数	説　明
image	2値化された画像（8ビット，1チャネルのグレースケール画像）
labels	ラベリングされた画像を格納するデータ
stats	各ラベルごとに領域の情報が［領域の左上のx座標，領域の左上のy座標，領域の幅，領域の高さ，面積］の配列で格納される
connectivity	8連結の場合は8，4連結の場合は4を指定する
ltype	ラベリング画像（labels）を取得する型をcv2.CV_16UまたはCV2.CV_32S（初期値）で指定する
centroids	ラベルごとの重心座標が［X座標，Y座標］の順で格納される
retval	ラベル個数

リスト1　面積，重心，外接矩形を取得（connectedComponents_sample.py）

```
# ラベリング(8連結)戻り値：ラベル個数，ラベリング画像，矩形領域，重心を取得
retval, labels, stats, centroids = cv2.connectedComponentsWithStats(th, connectivity = 8)
# 結果表示
for i in range(1, retval): # 0番目は背景情報のため，1始まり
      # 矩形領域
      x, y, width, height, area = stats[i] # x座標，y座標，幅，高さ，面積
      # 面積がしきい値以上のときは青，それ以外は赤
      if area > 5000:
          col = (255,0,0) # 青
      else:
          col = (0,0,255) # 赤
      # 矩形の描画
      cv2.rectangle(result,(x,y),(x+width,y+height),col,2)
      # 面積を表示
      cv2.putText(result, f"{area}", (x, y-8), cv2.FONT_HERSHEY_PLAIN,
                  1.5, (255, 255, 0), 1, cv2.LINE_AA)
      # 重心の位置を描画
      cv2.drawMarker(result, (int(centroids[i][0]), int(centroids[i][1])), (255, 255, 0),
                     cv2.MARKER_TILTED_CROSS, 10, 2)
```



（a）入力画像　　（b）面積で分類

図4　外接矩形範囲，面積，重心の取得結果

●利用例2：findContours()関数

▶領域の輪郭を抽出する

findContours()関数は，2値化された画像から連続領域の輪郭を抽出します．輪郭は，ラベリング処理の8連結相当となります．さらに，輪郭の内側にある輪郭の階層情報も取得します．

表2に引数を解説します．**リスト2**が領域の輪郭を階層別に描画するプログラムです．**図5**に実行結果を示します．

表2　findContours()関数

findContours(image, mode, method[, contours[, hierarchy[, offset]]])-> contours, hierarchy

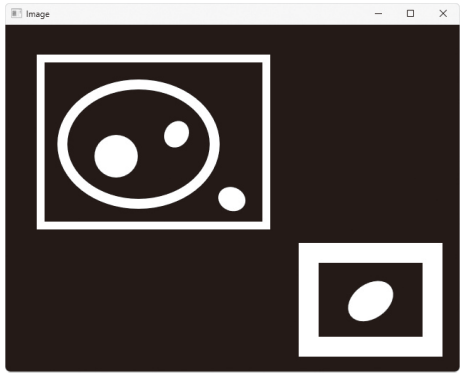
引　数	説　明	
image	2値化された画像（8ビット，1チャネルのグレースケール画像）	
mode	輪郭構造の取得方法を以下の中から設定する	
	cv2.RETR_EXTERNAL	一番外側（白領域の外側）の輪郭情報のみ取得する
	cv2.RETR_LIST	輪郭の内側，外側の区別なく，全ての輪郭情報を取得する
	cv2.RETR_CCOMP	全ての輪郭に関し，外側，内側の2階層の輪郭情報として取得する
	cv2.RETR_TREE	全ての輪郭に関し，外側，内側の全ての階層の輪郭情報を取得する
method	輪郭座標の取得方法	
	cv2.CHAIN_APPROX_NONE	輪郭を構成する全ての座標を取得する
	cv2.CHAIN_APPROX_SIMPLE	縦，横，斜め方向の輪郭の線上の点を除外し，交点のみを取得する
	cv2.CHAIN_APPROX_TC89_L1	Teh-Chin chainアルゴリズムに基づき，輪郭座標を直線近似した交点座標を取得する
	cv2.CHAIN_APPROX_TC89_KCOS	Teh-Chin chainアルゴリズムに基づき，輪郭座標を直線近似した交点座標を取得する
contours	輪郭を構成する座標を以下の配列で取得する contours[輪郭番号][点の番号][0][X座標，Y座標]	
hierarchy	輪郭の階層情報を以下の配列で取得する hierarchy[0][輪郭番号][次の輪郭番号，前の輪郭番号，最初の子供（内側）の輪郭番号，親（外側）の輪郭番号] 次，前，子，親の輪郭が存在しない場合，－1になる	

リスト2　各領域の輪郭を階層別に描画（findContours_sample.py）

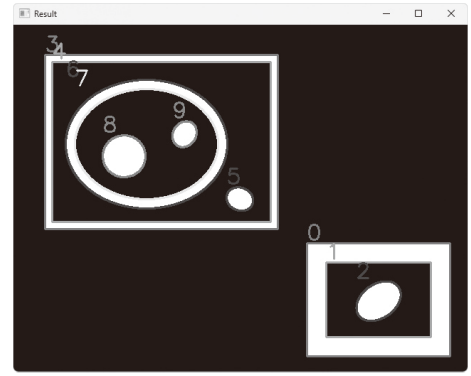
```
# 内、外を区別して階層（level）ごとに色分けして輪郭を描画
def draw_contour(img, contours, hierarchy, index, level):
      # 輪郭線の色の定義
      cols = [(0,0,255),(0,255,0),(255,0,0),(0,255,255),(255,0,255),(255,255,0)]
      # 輪郭の描画
      cv2.polylines(img, [contours[index]], True, cols[level%6], 2)
      # 外接矩形の取得
      x,y,w,h = cv2.boundingRect(contours[index])
      # 輪郭番号の表示
      img = cv2.putText(img, str(index), (x, y-5), cv2.FONT_HERSHEY_SIMPLEX, 1, cols[level%6], 2)
      # 同階層の輪郭を描画
      next_index = hierarchy[0][index][0]
      if (next_index != -1):
          draw_contour(img, contours, hierarchy, next_index, level)
      # 内側の輪郭を描画
      inside_index = hierarchy[0][index][2]
      if (inside_index != -1):
          draw_contour(img, contours, hierarchy, inside_index, level+1)

…

# 輪郭抽出（thは二値化された画像）
contours, hierarchy = cv2.findContours(th, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE )
# 最初の輪郭を描画し、再帰的にすべての輪郭を描画
draw_contour(result, contours, hierarchy, 0, 0)
```



（a）入力



（b）輪郭の抽出結果

図5　領域の輪郭を階層別に描画

●輪郭情報に基づいた各種特徴量

findContours()関数で取得した輪郭情報（contours）を用いたさまざまな特徴量を紹介します．**リスト3**に各種特徴量の取得と表示方法を示します．そのときの出力を**図6**に示します．

▶面積（contourArea）

輪郭の内側の面積を取得します．輪郭の内側に黒い領域があっても無視されます．面積が画素数ではないことに気をつけてください．

▶周囲長（arcLength）

輪郭の外周の長さを取得します．

▶フィレ径，バウンディング・ボックス（boundingRect）

輪郭に外接する，傾いていない矩形領域（左上のX座標，左上のY座標，幅，高さ）を取得します．

▶最小矩形（minAreaRect）

輪郭に外接する，傾いた最小の矩形領域（中心のX座標，中心のY座標），（幅，高さ），回転角度を取得します．

▶重心（moments）

輪郭のモーメントから重心を求めます．重心座標（x，y）は，以下のように求めます．

```
m = cv2.moments(contour)
x,y= m['m10']/m['m00'] , m['m01']/m['m00']
```

▶最小外接円（minEnclosingCircle）

輪郭に外接する最小外接円の中心と半径を求めます．

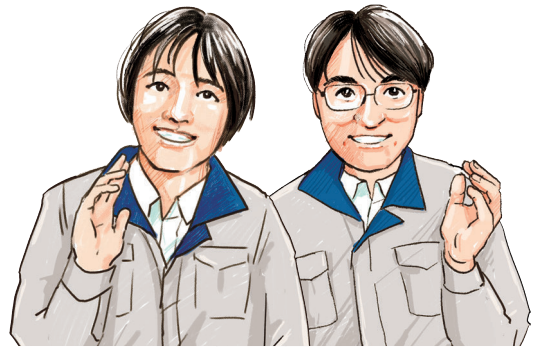
▶楕円近似（fitEllipse）

輪郭を楕円で近似し，近似楕円の中心座標，幅，高さ，傾きを求めます．

▶直線近似（fitLine）

輪郭を直線で近似し，点（x0，y0）を通り，単位ベクトル（vx，vy）の向きの直線を求めます．

ISBN978-4-7898-3149-9
C3055 ￥2600E
CQ出版社
定価 2,860円（本体2,600円）⑩
9784789831499
1923055026001



画像処理シリーズ