

第 11 章

画像処理

画像処理 (image processing) は、画像データの加工、特微量を抽出するための方法を言い、現代社会ではデータサイエンスの重要領域をなす。本章では、OpenCV 3 を用いて、基本的な画像処理を説明する。ただし、使い方に重点を置いており、詳細なアルゴリズムや理論は他書を参照されたい。画像データの表現を説明したのちに、2 値化、エッジ検出、周波数フィルタ、特微量抽出について述べる。この後に、幾つかの認識法についても述べる。

11.1 画像処理の概要

画像処理は、次に示すように、現代社会で数多く実用化されている。

身の回り カメラ、ビデオ、PC、車載カメラ、インターホン、他

公共空間 不審者監視 (ATM、マンション、繁華街)、N システム、河川観測、他

科学・工業・医療 気象予報、環境観測、製品検査、農作物分別、身元照合 (バイオメトリクス)、医療でのレントゲン・CT・MRI 検査、他

これらは、画像データから特微量を抽出し、それを用いて分類、認識を行うことで、価値ある判断が行っている。画像データドリブンの考えは、まさしくデータサイエンスの範疇にあるといえる。この基本を成すのが画像処理である。

画像データの表現には、幾つかの見方があり、これらを説明する。

11.1.1 表色系

色を表現するには様々な方法があり、これを表色系 (color coordinate system) と呼ぶ。それぞれの表色系の色要素で構成される空間を色空間 (color space) と呼ぶ。代表的な表色系を表 11.1 に示す。

本書で扱うカラー画像データは RGB 系とする。ディスプレイは、光の 3 原色 (RGB) に基づいている。すなわち、3 色が同輝度で混じり合うと白色になる。このデータ表現について、

- RGB がそれぞれ 8 ビット (8 ビットは 0 ~ 255 の 256 階調となる) で、その強度を表現するとき、 $256 \times 256 \times 256 = 16,777,216 \simeq 1670$ 万色 の色を表現できる。

表 11.1: 代表的な表色系（チャンネル数は色要素の数をいう）

表色系	チャンネル数	色要素
RGB	3	Red (赤, 700nm), Green (緑, 546.1nm)、Blue (青, 435.8nm), この波長は CIE (国際照明委員会) の定めた値。他の分野では色の波長は幅がある。PC のディスプレイで最も採用されている。
HSV	3	Hue (色相), Saturation (彩度), Value (明度), CG でよく用いられている。デザイン分野でよく用いられる。
XYZ	3	Y (輝度), Z (青み), X (それ以外), 光の陰影変化に比較的強く, ロボットビジョンでよく採用されている。
CMYK	4	Cyan, Magenta, Yellow, Key Plate, 印刷でよく用いられる。

- この場合, 黒は $R = G = B = 0$, 白は $R = G = B = 255$ である。これを各色の 256 階調ともいう。また, 0 ~ 255 を規格化して 0 ~ 1 で表すものもある。
- RGB 系では, 全ての色を表現しにくいことから, XYZ 表色系など他の表色系が定義されている。

ピクセル (pixel)

ピクセルは, コンピュータで画像を扱うときの最小単位を表し, 色情報 (色調や階調)を持つ画素のことと, pix (pic=写真、画像の意の複数形) + element (要素) の造語である。一方, ドット (dot) は単なる点情報を意味し, ピクセルとは異なることに注意されたい。1 ピクセルに 1 ビットの情報しか割り当たなければ、2 色しか表現できない。RGB のチャンネルに各 8 ビット, 計 24 ビットの情報を割り当たれば, 約 1670 万色の色表現が行える。なお, ピクセルの配置は, イメージセンサとディスプレイとでは異なる (図 11.1)。

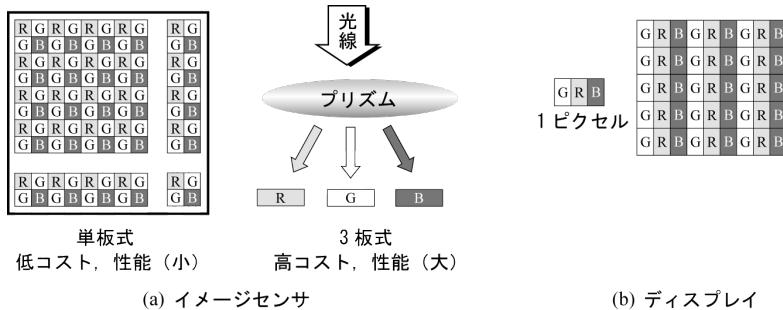


図 11.1: ピクセルの配置

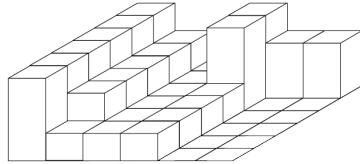
グレースケール (gray scale) とは、黒-灰色-白と明暗が段階的に変わるものという。グレースケールで 8bit 階調というのは黒が 0, 白が 255 で表される。これをプログラム上で規格化して 0~1 という表現もある。

11.1.2 数値としての表現

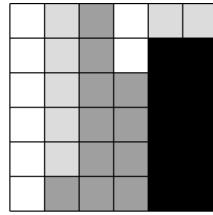
数値として画像データがどのように表現されるかについて、グレースケールを例にとつて説明する。図 11.2 は、縦横 6 分割、各ピクセルが 2 ビット (4 階調, 0 ~ 3) のデータを表している。この例では、黒:0, 灰色 (暗):1, 灰色 (明):2, 白:3 と対応付けられる。また、図 (a) のカッコ内は 2 進数を表している。図 (b) はこの階調を仮想的に高さで表現したイメージ図である。図 (c) はディスプレイの表示であり視認できるものである。

(11)	(00)	(11)	(11)	(10)	(10)
(11)	(10)	(01)	(11)	(00)	(00)
(11)	(10)	(01)	(11)	(00)	(00)
(11)	(10)	(01)	(01)	(00)	(00)
(11)	(10)	(01)	(01)	(00)	(00)
(11)	(01)	(01)	(11)	(00)	(00)

(a) 数値データ



(b) 階調を高さで表現したイメージ



(c) ディスプレイの表示

図 11.2: グレースケールの数値表現

図 (c) で線と見える箇所であっても、図 (b) を見るとその境界部分が急峻なものとながらかなところがあり、どこが境界なのかの判別が難しく、このことがコンピュータ処理における 2 値化、エッジ検出の難しさの一因となっている。

11.1.3 標本化と量子化

画像データも、時系列データと同じように、標本化 (sampling) と量子化 (quantization) がある。

図 11.3 を見て、標本化は定められたサイズの縦横を何分割するかということになる。分割数が大きいほど精密な画像を表現できる。これと類似の尺度に解像度 (resolution) がある。この単位として dpi (dots per inch), ppi (pixel per inch) があり、1 インチあたりの画素数をいう。この数が大きいほど精密な画像を表現できることになる。

量子化は 1 画素あたりの階調を表す。bit がその単位となる。図 (b) は、1 画素あたりの階調を bit で表し、その階調を見やすいように色の数で表している。階調はグレースケールで表現してもよく、8bit 階調ならば、256 段階のグレーで表現される。この図の例では、カラーの色数を階調としている（本書の紙面刷りはグレーにせざるを得ない）。類似の例として、電気電子工学の A/D 変換器で 10[V] フルスケールを N ビット分解能がある。これは、10[V] を 2^N に分解することを意味する。



図 11.3: 標本化と量子化

11.1.4 画像データの入手

各人が考案した画像処理・認識プログラムの性能評価には、共通的な画像データが求められる。この一つに、標準画像データベース **SIDBA**(Standard Image Data-BAsE) があり、世界的な標準画像である。入手しやすいサイトとして次がある。

- 南カリフォルニア大学 SIPI:<http://sipi.usc.edu/database/>
- CIPR Still Images:<http://www.cipr.rpi.edu/resource/stills/>
- 京都大学:http://vision.kuee.kyoto-u.ac.jp/IUE/IMAGE_DATABASE/STD_IMAGES/

他の無料で用いることのできる画像データを掲載しているサイトとして次がある。ただし、2次使用においてはその所属機関を明示するなど、著作権の遵守は必須である。

- 上田市イメージデータベース（上田市にある写真・映像・PDF 資料を提供）:<http://museum.umic.jp/imgdb/>
- 長崎大学電子化コレクション（幕末・明治期の日本の各種写真、ガラパゴス諸島画像など）:<http://www.cipr.rpi.edu/resource/stills/>

11.1.5 OpenCV のドキュメント

OpenCV の各種ドキュメントの在りかは、紙面で述べるのではなく、読者自身が検索して見ることとする。この方法を説明する。まず、“**OpenCVdoc**”は次を意味する。

OpenCVdoc = 公式 HP (<https://opencv.org/>) → “Online documentation” → Doxygen HTM の中で最も新しいバージョン (*n.n.n* 表記) をクリックして現れる Web ページ

例えば、cvtColor（カラー画像の変換を行う関数）のドキュメントの在りかは次のように表す。

OpenCVdoc → “cvtColor”

この意味は、OpenCVdoc のページにある検索入力欄から “cvtColor” で検索し、この適当な検索結果にジャンプして見ることを表す。

これとは別に、OpenCV のドキュメントとして有用なサイトを次にリストアップする。

OpenCV Wiki <https://github.com/opencv/opencv/wiki>

OpenCV-Python Tutorials http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

OpenCV-Python Tutorials' s documentation <http://opencv-python-tutroals.readthedocs.io>

11.1.6 実行方法

本書で用いる OpenCV は ver.3 であるが、Python スクリプトでは次のように “cv2” とする。

```
import cv2
```

本章の画像処理は実時間処理を含むため、スクリプトファイルの拡張子を”.py” として実行する。このため、jupyter notebook を用いない。実行方法は、次の 2 通りを紹介する。

コマンド入力 コマンドプロンプトから次のコマンドを入力する。

```
$ python filename.py
```

Spyder Anaconda が有する統合開発環境であり、この使い方は次を参照されたい。

<https://sites.google.com/site/datasciencetutorial/> → Python 開発環境 → Python スクリプト (.py) の開発方法

11.2 画像処理の例

11.2.1 2 値化

2 値化 (binarization) は、主にグレースケール (例：0 ~ 255) の画像データを対象として、予め定めたしきい値 (threshold) より下ならば黒 (0)，上ならば白 (255) にすることである。[0, 1] に規格化した考え方ならば次の表現となる。ただし、Y は輝度 (グレースケール) である。

$$Y \in [0, 1] = \begin{cases} 0 & \text{if } Y < \text{threshold} \\ 1 & \text{if } Y \geq \text{threshold} \end{cases} \quad (11.1)$$

カラー画像をグレースケールに変換する方法は幾つかあり、OpenCV の標準的な方法は OpenCVdoc → “Color conversions” に次とある。

$$Y = 0.299R + 0.587G + 0.114B \quad (11.2)$$

次のスクリプトは、2 値化処理を行うものである。初めに、オリジナル画像を入力、それをグレースケールに変換、サイズを調整して表示することを示す。

Listing 11.1: IMG_Gray2Bin.py

```
VIEW_SCALE = 1.0          # scale factor for window size
DEFAULT_THRESH_VAL = 128
MAX_VAL = 255             # max value for 8bit image

img_org = cv2.imread('data/lena_std.tif') # original image
img_gry = cv2.cvtColor(img_org, cv2.COLOR_BGR2GRAY) # convert to
                                                gray scale

h = int(img_gry.shape[0]*VIEW_SCALE)
w = int(img_gry.shape[1]*VIEW_SCALE)

cv2.imshow('Input image', cv2.resize(img_org,(h,w)))
cv2.imshow('Grayscale image', cv2.resize(img_gry,(h,w)))
```

画像データの場合、階調を幾つにするか、また画像サイズがまちまちであるから、サイズのスケール調整を行うなどの手続きが必要である。

次に、しきい値を予め設定することは難しいので、視認のもとで行えるように、トラックバー (trackbar) でしきい値を変えられるようにした。

```
def update(threshVal):
    retVal, img_bin = cv2.threshold(img_gry, threshVal, MAX_VAL,
                                    type=cv2.THRESH_BINARY)
    h = int(img_gry.shape[0]*VIEW_SCALE)
    w = int(img_gry.shape[1]*VIEW_SCALE)
    cv2.imshow(WIN_Titile, cv2.resize(img_bin,(h,w)))

cv2.createTrackbar('threshold', WIN_Titile, DEFAULT_THRESH_VAL,
                  MAX_VAL, update)
```

2 値化処理の結果を図 11.4 に示す。



図 11.4: 2 値化

この結果のように、threshold の値を調整したとしても、光の陰影やコントラストが画

像の各領域により異なるため、画像全体に一定のしきい値を用いた2値化では、人物や帽子の抽出は難しいことがわかる。

2値化の有用な用途として、白地に黒の文字が光線の影響でぼけているものをくっきりと抽出する、などがある。この用途ならば、陰影やコントラストを一定にしやすい。しかし、上記の例に見るように、これらが一定でない場合には、しきい値をこれらに合わせて適応的に変化させるという考え方もある。

11.2.2 エッジ検出

エッジ (edge) の意味は、境界、(二つの線の接する) 縁、(刃物の) 刃などとなる。画像では、輝度が大きく変化している箇所 (境界) を指し、人間は画像データの輝度変化を捉えることができる。これは、人間の視覚細胞で特定のエッジ方向に発火するニューロンがあるためである。

画像処理でのエッジ検出 (edge detection) は、この物体の輪郭を求めることがある。ここでは、輝度変化を捉える考え方に基づく方法を説明する。

輝度変化は微分により検出が容易になるという考え方がある。例えば、画像データが連続かつ一次元と想定した場合、この微分を図 11.5 に示す。この微分が示すように、変化

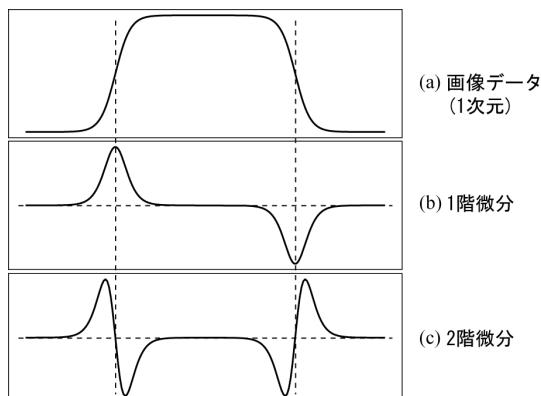


図 11.5: 画像データの微分

しているところがエッジと見なせる。また、2階微分を行い、振れの間のゼロとなる箇所をエッジとして抽出してもよい。画像データは離散データであるから、微分の代わりに差分を用いても同様の結果が得られる。

離散である画像データに、この考え方を適用したときの問題として次が挙げられる。

- 抽出したエッジは、一般に、幅がある。したがって、抽出したエッジをそのまま輪郭線として見なしてよいか否かは、用途に関わる。幅のあるエッジを線と見なすために1ピクセルまでにする細線化処理がある（細線化処理は他書を参照）。
- 微分はノイズに敏感に反応する。すなわち、ノイズを画像の一部と見なすことがある。
- 図に示す微分（差分）は一方向 (x 軸、または、 y 軸のみ) である。一方、画像データは2次元であるため、 x 方向の傾きがあっても y 方向の傾きが無い場合ある。こ

の場合、 y 方向の微分は傾きを検出できない。このため、2 次元に対応できるエッジ抽出器を選ぶ必要がある。

OpenCV は、エッジ検出方法を複数提供している。そのうちの一部を紹介する。

Sobel 法 1 次微分ゆえ、エッジ検出に方向性がある。

Laplacian 法 2 次微分で、2 方向（縦横）に対応できる。

Canny 法 比較的性能が良く、よく用いられる方法。画像中のノイズ影響を低減するため、ガウシアンフィルタを用いた平滑化を行った後に、複数ステップによるエッジ抽出を行う (https://en.wikipedia.org/wiki/Canny_edge_detector)。

これらの OpenCV による実装を次のスクリプトに示す。関数の使い方はスクリプト中に URL で示している。

Listing 11.2: IMG_EdgeDetection.py

```
edge_sob_x = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=5)
edge_lapl = cv2.Laplacian(img, cv2.CV_32F)
edge_cann = cv2.Canny(img, 80, 120)
```

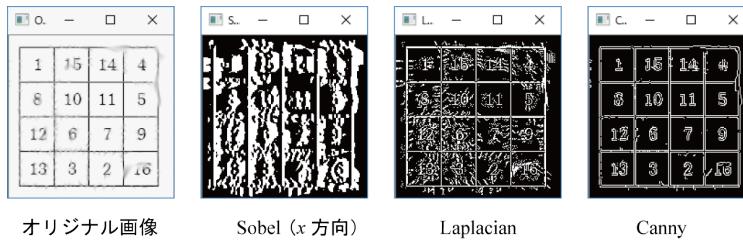


図 11.6: エッジ検出

この結果について述べる。オリジナル画像には、若干のノイズ（正規乱数）をわざと重畳させており、かつ、にじみを与えていた。Sobel 法 (x 方向) の結果は、縦線の検出はできるが、横線の検出はできていないことを示している。Laplacian 法の結果は、縦横の枠線はよく検出できているが、ノイズに敏感に反応して数字の検出が不十分である。Canny 法の結果は、比較的ノイズに頑強で、かつ、縦横方向の検出もできている。ただし、数字を見ると線分の抽出はできておらず、2 つの境界線、すなわち幅のあるエッジとして検出されている。我々が学校で学んだ線は、数学の意味で幅の無いものと習った。しかし、眼に映る以上は幅がある。それも線であると認識する。しかし、画像データの表現で示したように、物体の輪郭とは線ではなく階調の差であり、これをエッジとして抽出すると、幅があるだけでなく 2 つの境界線で表されることとなる。

輪郭線という線は、実は人間の心理的概念で、実在空間には数学の意味での線は存在しない。この概念と画像処理との狭間をさらに埋めることの研究は読者に委ねる。そういうえば、筆者が中学で人物像を描く授業のとき、美術教師から輪郭線で表すのではなく明暗や色彩で人物像を表すように、との言葉を今になって思いだす。

11.2.3 周波数フィルタリング

周波数領域上の表現

時間領域の波形は、フーリエ変換により周波数領域に移され、周波数成分で表現された。2次元画像データは空間領域にあると言える。これに2次元離散フーリエ変換(2D-DFT)を行い周波数領域に移されたものは空間周波数で表現される。空間周波数で表すことも画像の特徴量となる。空間周波数に対するローパスフィルタ、ハイパスフィルタを施すことでも、画像がどのように変化するかを見る。なお、DFT(ここでは、FFTと同じ意味として用いる)は、numpyとOpenCVの両方が提供しており、両方の使用を示す。

2D-DFTおよび、2D-IDFT(離散フーリエ逆変換)は次式で与えられる。

$$F(u, v) = \frac{1}{NM} \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} f(x, y) \exp \left\{ -j \frac{2\pi x u}{N} \right\} \exp \left\{ -j \frac{2\pi y v}{M} \right\} \quad (11.3)$$

$$f(x, y) = \sum_{v=0}^{M-1} \sum_{u=0}^{N-1} F(u, v) \exp \left\{ j \frac{2\pi x u}{N} \right\} \exp \left\{ j \frac{2\pi y v}{M} \right\} \quad (11.4)$$

2次元のDFTの概要は、画像データを横方向に1次元のDFTを施し、次に縦方向に1次元のDFTを施すこととなる。この図的説明を図11.7に示す。

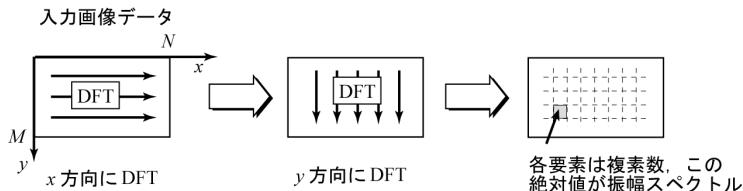


図 11.7: 2D-DFT の概要

時系列データで説明した振幅スペクトルを用いて、画像データの周波数領域での表現を考える。図11.8は、画像データに対して2D-DFTを施し、その絶対値を表現したものである。これは、矩形領域の中心が高周波、離れるほど低周波を表すこととなり、このままでは扱いにくいので、シフト操作により、中心付近が低周波、離れるほど高周波を表現するように、データを組替える。

シフトされた振幅スペクトルに対して、フィルタリング操作(後述)を行い、これに対して逆シフト操作(ISHIFT)を行ったデータに対して2D-IDFTを施すと空間領域での画像データを得ることができる。numpyのFFTを用いた振幅スペクトルを求めるスクリプトを次に示す。

Listing 11.3: IMG_FFT.py

```
img1 = cv2.imread('data/Canon_cornfield.png', 0)
f = np.fft.fft2(img1)
fshift = np.fft.fftshift(f)
mag1 = 20 * np.log(np.abs(fshift))
```

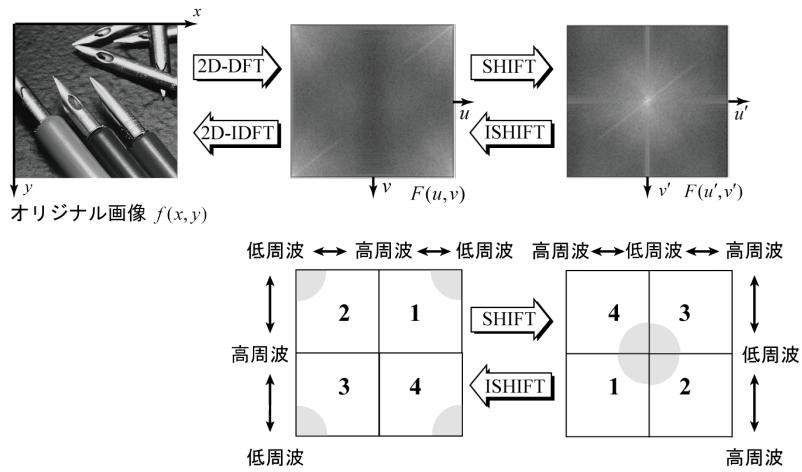


図 11.8: 振幅スペクトルの表現とシフト操作

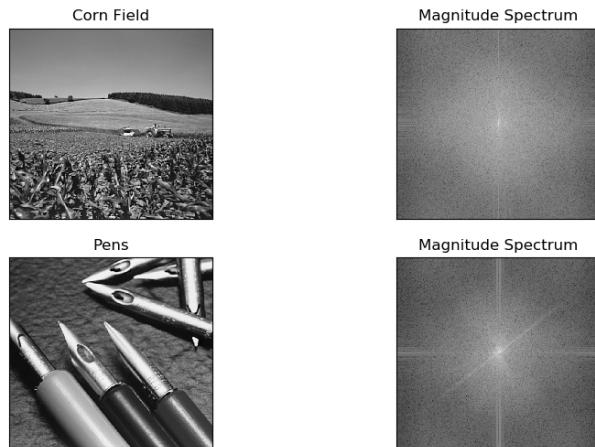


図 11.9: 振幅スペクトル

この結果図 11.9において、明るい方が振幅スペクトルの強度が強いことを表す。Corn Field の結果は、低周波成分から高周波成分までを一様に有している。これは、膨大な数の葉の影響と考えられる。Pens の結果は、比較して低周波成分を多く含んでいる。これは、刃先のエッジ数は Corn の葉の数より非常に少ないため高周波成分は比較的に少ないと考えられる。それよりも、ペン本体の直線形状が低周波成分を多く発生しているものと考えられる。

周波数フィルタリングの例

先の振幅スペクトルを用いた周波数フィルタリングについて、この図的説明を図 11.10 に示す。

図中のローパスフィルタやハイパスフィルタを実現するために mask 操作を行う。この

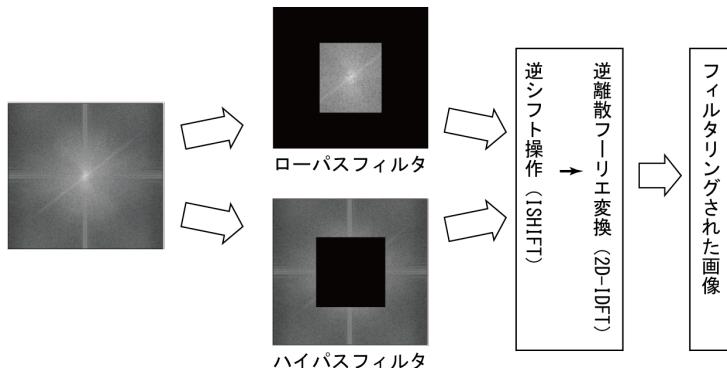


図 11.10: 画像データの周波数フィルタリング手順

mask は、0,1 で表現され、ローパスフィルタは中心部の矩形領域が 1、この外側が 0 である。ハイパスフィルタは、この逆の重みをもつ。この mask と振幅スペクトルを乗じることでフィルタリングが行える。ここでは、mask 形状を矩形としたが、矩形窓は、もともとは無い波がのような画像となるリング現象が生じるので注意を要する。mask には、他に円形や Gaussian 窓が良く用いられる。

OpenCV の DFT を用いたスクリプトを次に示す。

Listing 11.4: IMG_DFT_Filter.py

```

isize=10 # half size of mask
# mask for low pass filter
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-isize:crow+isize , ccol-isize:ccol+isize ] = 1
# apply mask and inverse DFT
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back1 = cv2.idft(f_ishift)
img_back1 = cv2.magnitude(img_back1[:, :, 0], img_back1[:, :, 1])
  
```

図 11.11において、Input image は、512 × 512 ピクセルである。これを OpenCV 提供の DFT で周波数領域に変換し、振幅スペクトルの 2D データの原点中心で 1 辺 20 ピクセル (=isize × 2) の正方形窓をマスクとして設けた。ローパスフィルタの結果は、低周波成分で表される大まかな輪郭が抽出されている。ハイパスフィルタの結果は、毛やひげが抽出され、高周波成分が残っていることが認められる。

ここで、マスクサイズ ($2 \times isize$) の値を画像サイズよりも小さい範囲にしておけば、画像データ量の低減化を図ることが可能である。

11.2.4 特徴点抽出

物体の特徴点 (feature point) の抽出は、物体認識や追跡などに用いられる。例えば、図 11.12 では、人物の顔、服装の柄などで画像処理上の特徴点を抽出し、画像フレームが進行しても、同じ特徴量を有する特徴点を対応付けることで追跡を行っている様子を示している。

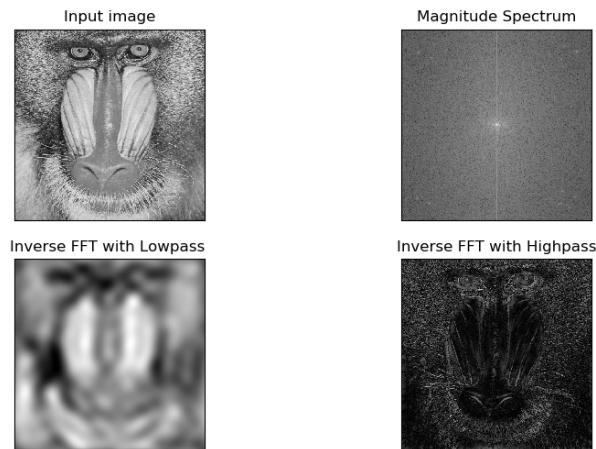


図 11.11: 画像データのフィルタリング結果

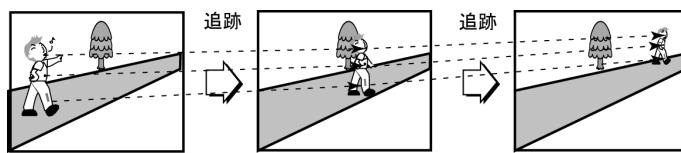


図 11.12: 特徴点を用いた人物追跡

特徴点には、複数の特徴量を含む。特徴量には、画像の色、輝度、輪郭、コーナー、固有値などがあり、これらを特徴ベクトルとして表現し、これを特徴点が有する。したがって、異なる画像フレームで同じ特徴量を有するとは、この特徴ベクトルを見ていることとなる。この概要について

OpenCVdoc → “Feature Detection and Description”

ここに “Understandings Features”，各種コーナー検出、特徴量マッチングなどが説明されている。また、この概要の動画が次にあり、参考になる。

OpenCV GSOC2015:<https://youtu.be/0UbUFn71S4s>

OpenCV GSOC2017:<https://youtu.be/b6nJbE1KK7Y>

OpenCV が提供する特徴点抽出法は、FAST, ORB, BRISK, AKAZE, SHIFT などがある。このうち、AKAZE(Accelerated Kaze) を用いた例を示す。KAZE は SIFT や SURF の欠点を解決したアルゴリズムである。この更なるロバスト性の向上と高速化を図ったのが AKAZE ^{*1} である。OpenCV の説明は次にある。

OpenCVdoc → “AKAZE”

^{*1} P.G.Alcantarilla, J.Nuevo and A.Bartoli: Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces, In British Machine Vision Conference (BMVC), Bristol, UK, September 2013 , <http://www.robosafe.com/personal/pablo.alcantarilla/kaze.html>

特徴点のマッチング

特徴点のマッチングとは、2つの画像から多数の中から類似の特徴ベクトルを見出し、これを対応付ける（マッチング）ことである。マッチングの考え方として次がある。

総当たり探索（Brute-Force）：全探索空間を総当たり探索のため時間がかかるが、確実に最近傍の特徴量を検索することができる。（https://en.wikipedia.org/wiki/Brute-force_search）

高速近似最近傍探索（FLANN: Fast Library for Approximate Nearest Neighbors）：探索対象の特徴点と近い空間のみを探索する。探索空間が減るので高速に検索することができる。ただし、探索のパラメータの指定が適切でないと、探索空間の選択を誤ることとなり、最近傍の特徴点が含まれていない探索空間だけを探索することとなる。（<https://www.cs.ubc.ca/research/flann/>）

ここでは、Brute-Force の kNN マッチングを採用する。すでに述べた kNN は、探索空間から最近傍のラベルを k 個選択し、多数決でクラスラベルを割り当てるアルゴリズムである。 $k=2$ を指定すれば、画像間それぞれ一つずつを選択することとなる。この説明は次にある。

OpenCVdoc → “Feature Matching”

AKAZE + BF(kNN) を用いた例を次のスクリプトにしめす。

Listing 11.5: IMG_FeatureDetectionAKAZE.py

```
# Create A-KAZE detector
# https://docs.opencv.org/3.4.1/d8/d30/classcv_1_1AKAZE.html
akaze = cv2.AKAZE_create()

# Extraction of features and calculate feature vectors
kp1, des1 = akaze.detectAndCompute(img1, None)
kp2, des2 = akaze.detectAndCompute(img2, None)

# Create Brute-Force Matcher
bf = cv2.BFMatcher()

# Matching between feature vectors with Brute-Force+kNN
matches = bf.knnMatch(des1, des2, k=2)

# ratio v.s. connected lines between feature points
ratio = 0.6
good = []
for m, n in matches:
    if m.distance < ratio * n.distance:
        good.append([m])
```

ここに、特徴点は幾つ生成されるかわからないので、ratio を増減させることで特徴点の数を増減できる。すなわち、特徴点を結びつける連結線の数も調整できることになる。

図 11.13 の左図と右図それぞれの特徴点を求め、Brute-Force+kNN でマッチングした結果を示す。この結果を見るように、右図で対象となる本が回転かつ縮小され、他の本の中にあっても特徴点の対応付けができる。この対応付けが正しいものとすれば、物体追跡が可能となる。



図 11.13: 特徴点の対応付けの結果

11.3 その他

動画処理・認識に関する方法を幾つか紹介する。紙面の制約上、紹介だけに留め、詳しい内容は他の成書や論文を参照されたい。なお、スクリプトや概要に関するドキュメントは、次のサイトの検索窓から調べることができる。

- OpenCV Python Tutorial, https://www.tutorialspoint.com/opencv_python/

使い方は Tutorial を参照されたい。この中から、幾つかの基本的な使い方を紹介する。

11.3.1 カメラからの動画取得

初めに、カメラから動画取得の説明は次にある。

OpenCV-Python Tutorials → “Capture Video from Camera”

この説明は、グレースケールである。一方、提供するスクリプト “IMG_Video.py” はカラー画像であり、グレースケールの用い方はスクリプトに書いてあるので、これを参照されたい。

11.3.2 顔認識

ここで顔認識の意味は、人の顔が画像上どこにあるかを認識することで、人の同定を行うわけではない。このアルゴリズムは、Haar cascade を用いている。これは、顔の各部を Harr like 特徴量 (https://en.wikipedia.org/wiki/Haar-like_feature) で表し、検出精度の異なる複数の識別器を連結した Harr cascade で顔か否かの判別を行うものである。この概要是次に記載されている。

OpenCV-Python Tutorials → “Face Detection”

静止画像の顔認識を行うスクリプトは“IMG_FaceDetection.py”，カメラからの動画像を用いた顔認識は“IMG_FaceDetectionFromCamera.py”である。