実践 OpenCV 4 for Python

画像映像情報処理と機械学習

永田雅人／豊沢 聡◉共著

CUTT
カットシステム

■**サンプルファイルのダウンロードについて**

本書掲載のサンプルファイルは、一部を除いてインターネット上のダウンロードサービスからダウンロードすることができます。詳しい手順については、本書の巻末にある袋とじの内容をご覧ください。

なお、ダウンロードサービスのご利用にはユーザー登録と袋とじ内に記されている番号が必要です。そのため、本書を中古書店から購入されたり、他者から貸与、譲渡された場合にはサービスをご利用いただけないことがあります。あらかじめご承知おきください。

・本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送またはE-mail）でお送りください。
・電話によるお問い合わせはお受けできません。
・本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
・匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

はじめに

近年、コンピュータに画像や映像を認識させる研究は著しい進展を見せており、成果を応用した製品も続々と実用化されています。たとえば、腕や手を振るジェスチャからのコンピュータ操作、自動車やロボットの自律運転、CT スキャン画像から脳出血の部位を検出する医用画像診断、部品や製品の傷や変形をチェックする外観検査などです。身近なものでは、スマートフォンカメラの顔検出があります。

このような技術をコンピュータビジョンといいます。

OpenCV（Open Source Computer Vision Library）は、コンピュータビジョン技術を扱いやすいプログラミングライブラリとして提供するオープンソースソフトウェアです。シンプルなので、初学者でも容易にプログラムを作成できます。高水準な API により、拡張や保守が楽になる簡潔なコードが書けます。しかも、おなじ機能を自分で実装するよりたいていは高速です。先進技術をつねに追っているので、ディープラーニングのような旬な技術も利用できます。画像認識の研究開発者なら、基本的な処理は OpenCV に任せ、問題解決に本質的なアルゴリズムに専念できます。

OpenCV には C++ インタフェースもありますが、本書のターゲットは Python です。処理速度は C++ にかないませんが、C++ でできることはほとんど Python でもできます。そして、スクリプト言語である Python のほうが気楽に始められます。

本書では、Python 版 OpenCV をサンプルプログラムに沿って学びます。各節は、冒頭で設定した課題を達成することを目指したシンプルなサンプルを中心に構成しています。コードはいずれも全行を掲載してあるので、わからなくてもとりあえず動かして楽しめます。どんなに長くても 100 行もありませんから、読むのもそうむずかしくはありません。そのあとは、プログラムの調整や改良をつうじて、OpenCV の扱いかた、各種アルゴリズムの利用方法、リファレンスマニュアルに慣れ親しんでください。本書を読み終えれば、ユニークなコンピュータビジョンアプリケーションが作成できるようになっているでしょう。

コンピュータビジョンの対象は何気なく接している身近なシーンが主なので、結果がわかりやすく、趣味のプログラミングとしても、より深い研究テーマとしても楽しめるテーマだと思います。本書がコンピュータビジョンやプログラミングの入り口となれば幸いです。

2020 年 10 月　永田 雅人

本書の構成

本書は第 1 章から第 5 章でよく使われる関数や手法を、第 6 章から第 8 章で応用的な処理をそれぞれ説明します。OpenCV を初めて使われる、あるいは OpenCV の開発環境が準備できていない方は、付録 A または B のインストールからスタートしてください。

次に各章の構成を示します。

第 1 章　OpenCV について

本書で扱う技術を中心に、OpenCV の応用場面を紹介します。また、プログラミングで必要な画像や映像の形式、ピクセル、カラーモデルといった基本構造を説明します。まずは実践という読者は飛ばして第 2 章に進み、必要におうじて参照してください。

第 2 章　画像・映像の入出力

画像、ビデオファイル、カメラ映像の入出力および操作方法を、簡単な画像処理とともに説明します。

第 3 章　ユーザインタフェース

図形や文字などのグラフィックスの描画、キー入力、トラックバーやマウスの操作といったユーザインタフェースの処理方法を紹介します。また、性能評価に必要な処理時間の測定方法も説明します。

第 4 章　チャンネルとマスクの処理

画像をチャンネルや部分領域に分けて処理する方法を説明します。

第 5 章　画像の演算

ピクセル単位での画像の操作、整数型画像データの浮動小数点数型への変換、畳み込み演算変換を説明します。

第 6 章　画像情報の取得

映像からさまざまな情報を抽出する方法を紹介します。具体的には、ヒストグラム、周波数フィルタリング、DCT 情報圧縮、オプティカルフローを扱います。

第 7 章　画像情報による物体認識

画像から抽出した情報をもとに物体を認識する方法を説明します。扱うトピックはテンプレートマッチング、ヒストグラムを用いた色ベースの画像判別、特徴点抽出と特徴点を用いた物体検出です。

第 8 章　ディープラーニングによる物体認識

Python のディープラーニング用モジュール cv2.dnn を用いてクラス分類、顔領域の判定、一般物体の認識を扱います。

付録

OpenCV の開発環境の構築方法を説明します。対象は Python、OpenCV for Python、Visual Studio Code です。ディープラーニングについては本書で利用するモデルファイルの入手先と作成方法を示しました。加えて、参考文献をまとめました。

第 2 章からの各節では、それぞれ 1 つのトピックを扱います。まず、節冒頭（たとえば 2.1 節）でターゲットとなる課題を定義し、その課題の達成に必要な基礎知識と初出の OpenCV 関数を示します。最初の項（たとえば 2.1.1）では、完成したプログラムの実行状況を示します。次の項（2.1.2）にはプログラム（コード）を全文提示します。次項で説明するコードは網掛けになっています。最後の項（2.1.3）では、初出の関数の定義を示しつつ、コードを基本的には番号順に解説します。

サンプルプログラムの画面例はカラーのものが多いですが、紙面ではモノクロです。色合いは実行することで確認してください。

やや高度な話題は、節末の Note に加えました。必須の知識ではないので、空いたときに読んでいただければ幸いです。

■開発環境

開発環境の構築方法は Windows と macOS を対象に付録 A および B で説明しました。

Python も Python 用の OpenCV ライブラリも、プラットフォーム依存性のほとんどないプログラミング環境なのでさまざまな OS で開発、実行できます。しかし、機能によっては一部記述どおりには動作しないことがあります。本書掲載のサンプルプログラムで不具合が生じたときは、次の動作確認済みのプラットフォームとバージョンを用いてください。

- プラットフォーム：Windows 10 64 ビット版または macOS 10.15
- Python：3.7.8
- OpenCV：OpenCV 4.4.0

NumPy は OpenCV をインストールしたときに（依存関係から）自動で導入されたもの（1.19）を想定しています。

バージョンの確認

Python のバージョンは sys.version から調べられます。各種の外部モジュールのバージョンは __version__ プロパティです。

以下は Windows での実行例ですが、他の OS でも要領はおなじです。

```
C:\temp>python
Python 3.7.8 (tags/v3.7.8:4b47a5b6ba, Jun 28 2020, 08:53:46) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

# Pythonのバージョン（3.7.8）
>>> import sys
>>> sys.version
'3.7.8 (tags/v3.7.8:4b47a5b6ba, Jun 28 2020, 08:53:46) [MSC v.1916 64 bit (AMD64)]'

# OpenCVのバージョン
>>> import cv2
>>> cv2.__version__
'4.4.0'

# NumPyのバージョン
>>> import numpy as np
>>> np.__version__
'1.19.2'
```

カメラ

本書のサンプルプログラムのいくつかは、ビデオ入力にカメラを用います。OpenCV は自動的にカメラを認識しますが、認識しなければ、カメラ付属のユーティリティやカメラを利用するその他のアプリケーション（Microsoft Teams や Zoom など）で、カメラの動作を確認してください。

それでも動作しなければ、OS のプライバシー設定でカメラアクセスが制限されているかもしれません。Windows 10 では、［Windows の設定］→［プライバシー］→［カメラ］から確認します。macOS では、［システム環境設定］→［セキュリティとプライバシー］→［カメラ］でターミナルにカメラアクセスを許可します。macOS の VS Code には現時点でカメラアクセスが許可できないようなので、カメラを利用するサンプルプログラムはターミナルから実行してください。

文字エンコーディング（Windows）

Windows のコンソールであるコマンドプロンプトのデフォルトエンコーディングは Shift JIS（SJIS）です。そのため、UTF-8 で記述したサンプルプログラム（後述）を type コマンドなどで表示すると次のように文字化けします。

```
C:¥opcv4>type 2_1.py
import sys
import cv2

# 逕ｻ蜒上ヵ繧ｩ繧､繝ｫ縺ｮ隱ｭ縺ｿ霎ｼ縺ｿ
img = cv2.imread('data/fruits.png')
... 略
```

コードで日本語文字のある箇所はコメント行だけなので、実行に支障はありません。

エンコーディングは chcp コマンドから変更できます。コマンドを単体で用いれば現在のエンコーディング方式が表示されます。

```
C:¥opcv4>chcp
現在のコード ページ: 932
```

出力にある数値をコードページといい、Windows 固有の文字エンコーディング方式識別方法です。932 は Shift JIS に相当します。エンコーディングの名称とコードページの対応は次の Microsoft のサイトを参照してください。

```
https://docs.microsoft.com/ja-jp/windows/win32/intl/code-page-
identifiers
```

エンコーディングをサンプルプログラムと同じ UTF-8 に変更するには、chcp コマンドの引数に 65001 を指定します。以降、UTF-8 文字が正常に表示されます。

```
C:¥opcv4>chcp 65001
Active code page: 65001

C:¥opcv4>type 2_1.py
import sys
```

```
import cv2

# 画像ファイルの読み込み
img = cv2.imread('data/fruits.png')
... 略
```

日本語を含んだパス名は認識されません（たとえば C:/temp/ テスト /file.png）。すべて半角英数にしてください。

サンプルプログラム

本書に掲載したサンプルプログラムは、巻末の「袋とじ」にあるダウンロードサービスからダウンロードできます。ダウンロードしたファイルはどこに置いてもかまいません。本文では説明の都合上、サンプルの所在を C:/opcv4 と記述しています。保存した場所で読み替えてください。

プログラムファイル

ダウンロードしたフォルダ直下にはサンプルプログラムが収容されています。

プログラムファイルは、2.1 節なら 2_1.py のように節単位で名づけられています。

節によっては参考用のコードもあり、それらは 3_1n.py のように節番号に n が加わっています。本文には掲載していませんが、ダウンロードサービスには含まれています。

付録記載のもの（ディープラーニングモデル構築）は複数のプログラムファイルで 1 つのトピックを構成しているので、E_1_1.py のように 3 つ目の番号で連番になっています。

付録を除くすべてのプログラムファイルは all.py で順に実行できます。

サンプルの画像とビデオ

サンプルプログラムが参照する画像あるいはビデオファイルは、エッセンシャルでない箇所でコードが長くならないよう、おおむねハードコードしています。これらのファイルは data サブフォルダに同梱してあり、コードから data/fruits.png のように相対パスでアクセスできると想定しています。画像あるいはビデオファイルを他所に移動したり、他のファイルを参照するときは、コード内のパスを変更してください。

ビデオの属性情報を必要とするプログラムの中には、簡略化のために属性値（たとえばビデオのフレームサイズ）をハードコードしているものもあります。こちらも、必要におうじて変

更してください。

ビデオファイルのフォーマットは、いずれもコンテナは MP4、コーデックは H.264 です。OpenCV 公式サイト、あるいは下記のロイヤリティフリーサイトから入手したものについては、扱いやすいようにオリジナルから形式を変換しています。

```
https://www.pexels.com
```

画像やビデオを書き込むプログラムは、この data サブフォルダにファイルを保存します。

モデルファイル

第 8 章のディープラーニングのモデルあるいはデータファイルは model サブフォルダに収容してあります。画像やビデオ同様ハードコードされているので、フォルダを移動したときは、コード内のパスを変更してください。

これらのファイルのオリジナルの入手元 URL は、付録 D にまとめました。また、該当するコード下部にもコメントで記述してあります（本書では未掲載）。

付録 E の参考サンプルプログラムは生成したモデルファイルをこのフォルダに保存します。

インポート時の別名指定

公式リファレンスでは、OpenCV モジュールの名称は cv2 ではなく cv です。これは、プログラムでインポートするモジュールに次のように別名をつけることを前提としています。

```
import cv2 as cv
```

本書では import cv2 でインポートし、そのまま cv2.imread で参照しています。本書記載の関数定義も同様です。

NumPy は、慣習にしたがって別名に np を指定します。

```
import numpy as np
```

パス区切り記号

パス区切り記号は、本文でもコードでも Unix スタイルのフォワードスラッシュ（/）を用います（ただし、本書掲載の Windows のコンソール画面では ¥ のままです）。円記号（¥）を用いる Windows でも、Python/OpenCV ではフォワードスラッシュで問題なく解釈されます。

文字コード

コード内の文字列は出力メッセージも含めて半角英数です。ただし、可読性を考慮して、コメントは日本語で記述しています。文字エンコーディングは UTF-8（BOM なし）、改行コードは LF（0x0A）です。

ライセンス

本書で使用するソフトウェアのライセンスは次の URL で規定されています。使用前に確認してください。

- OpenCV － BSD ライセンス
  `https://opencv.org/license/`
- Python － Python Software Foundation（PSF）
  `https://docs.python.org/ja/3/license.html`
- NumPy －独自ライセンス
  `https://numpy.org/doc/stable/license.html`

リファレンス

以下、利用するソフトウェアの公式リファレンスおよび本書の関数定義フォーマットを示します

OpenCV 公式リファレンス

OpenCV の関数や定数は次の OpenCV 公式リファレンス（以下、公式リファレンス）から参照します。

```
https://docs.opencv.org/
```

公式リファレンス以外では、次の OpenCV 公式の Q&A サイトが参考になります。現在、このフォーラムには約 31,000 件以上の質問や回答が登録されています。

```
https://answers.opencv.org/questions/
```

次に公式リファレンスの読みかたを説明します。

トップページには各バージョンへのリンクが列挙されているので、利用しているバージョンをクリックします（本書では 4.4.0）。

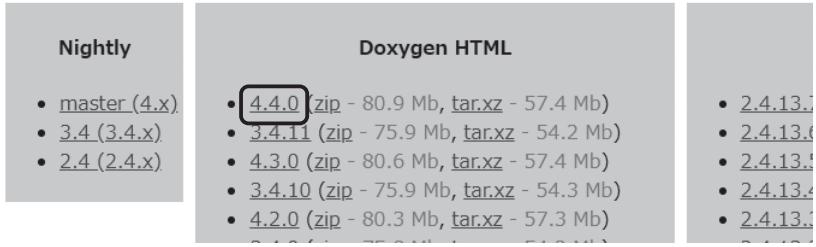


図 0.1 ● OpenCV ドキュメントのメインページ

OpenCV のモジュール（core や imgproc など）別に分けられているので、目的の関数や定数をピンポイントに見つけるのはむずかしいですが、右上のサーチフィールドが予測入力をサポートしています。

図 0.2 ●公式リファレンストップページ

「imread」で検索したところを次に示します。このキーワードを含む関数および定数がドロップダウンリストとして表示されます。「imread cv」をクリックすると目的の関数にジャンプします。

図 0.3 ●右上のサーチフィールドから検索

検索キーワードは C/C++ での名称をベースにしているため、Python 版が異なる名前だと予測が失敗します。たとえば、Python 版の `ORB_create`（7.3 節）は C/C++ では `ORB` クラスとそのメンバメソッド `create` なので No Matches になります。キーワードを変えてためしてください（この場合なら「ORB」だけ）。

使用している OpenCV のバージョンの確認方法は前述の「開発環境」で説明したとおりです。

OpenCV の関数定義

関数定義は C++ と Python とで共通です。最初の定義が C++ の、その下が Python のものです。詳細説明には、ところどころ C++ 特有の記法（`#include` ～や `Mat::` など）が出てきますが、気にしなくてかまいません。

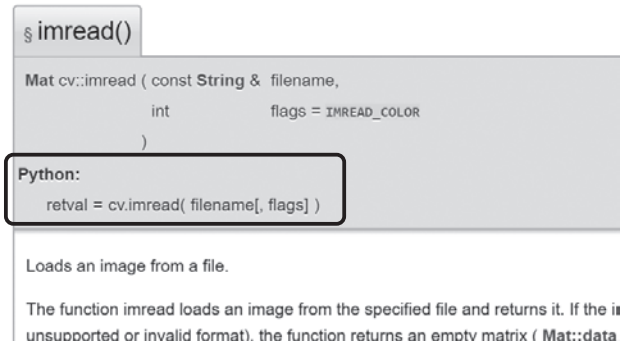


図 0.4 ●公式リファレンスの imread 関数の定義（下が Python のもの）

角カッコ（[]）で囲まれている引数はオプションなので省けます。指定がなければ、C++ 欄にある値がデフォルトで用いられます。上記の例では `flags` で、そのデフォルト値は `IMREAD_COLOR` です。

引数の指定順序は、定義のとおりでなければなりません。`cv2.imread` 関数では `filename` が先で、（指定するなら）`flags` があとです。このように、位置順序で意味が定まる指定方式を位置引数といい、本書のサンプルプログラムは基本的にこのスタイルを用いています（キーワード引数については 2.1 節の Note 参照）。

Python リファレンス

Python の公式ドキュメントの邦訳版は次の URL からアクセスできます。

```
https://docs.python.org/ja
```

デフォルトでは最新のバージョンのドキュメントが表示されます。ここからチュートリアルやライブラリリファレンスにアクセスできます。特定のバージョンのドキュメントは、ページ上端のプルダウンメニューから指定できます。リリース番号（たとえば 3.7.8 の末尾の 8）まで細かい指定はできませんが、マイナー番号（真ん中の 7）まで一致していればよほどのことがなければ問題はありません。

使用している Python のバージョンの確認方法は前述の「開発環境」で説明したとおりです。

NumPy リファレンス

NumPy のマニュアルは次の URL からアクセスできます。

```
https://numpy.org/doc/
```

トップページにはバージョンごとにマニュアルがリストされています。使用している NumPy のバージョンを選択します。マニュアルページには NumPy API Reference セクションがあり、ここがリファレンスです。

OpenCV は画像データを収容する `np.ndarray` を多用します。このオブジェクトについては「Array Objects」章の「The N-dimensional array (ndarray)」節から参照できます。本書でも折に触れて説明します（とくに 1.3 節および 2.3 節）。

使用している NumPy のバージョンの確認方法は前述の「開発環境」で説明したとおりです。

本書の関数定義

本書では、クラスや関数は初出の節でその定義を示します。対象は OpenCV と NumPy で、標準的な Python のものは取り上げません。

関数定義は公式リファレンスと似た形式で記述していますが、わかりやすくするために追加あるいは省略した部分もあります。次に、本書の関数定義の形式を `cv2.Canny` 関数（2.2 節）から示します。

Cannyアルゴリズムによるエッジ検出	
edges = cv2.**Canny**(**image**, **threshold1**, **threshold2**[, **edges**[, **apertureSize**=3[, **L2gradient**=False]]])	
image	入力画像
threshold1	Canny関数で使用する第1の閾値
threshold2	Canny関数で使用する第2の閾値
edges（引数）	エッジ検出結果の画像を返す
apertureSize	カーネルサイズ。3、5、7のいずれか。
L2gradient	L2ノルムの利用設定。FalseはL1ノルムとなる。
edges（戻り値）	エッジ検出結果の画像を返す

1 行目が関数の機能の概略と定義です。関数に戻り値があれば、「=」で左辺（ここでは edges）に代入するように示しています。

関数名に続くカッコの中が引数です。示された名称はキーワード引数のキーワードとして利用できます（2.1 節の Note ）。角カッコ（[]）で囲まれている引数はオプションです。オプション未指定時のデフォルト値は = で示しています。たとえば、apertureSize のデフォルトは 3 です。公式リファレンスの Python 関数欄にはオプション引数のデフォルト値は示されていませんが、ここでは C++ 欄の記述を転記しています。

以下がそれぞれの引数の簡単な説明です。

最後の網掛けのある行は（あれば）戻り値（edges）の説明です。

4 番目の引数にも edges があり、戻り値同様に結果画像を返すとあります。ここには、戻り値とまったくおなじデータが返されます。第 4 引数があるのは、C++ 関数の戻り値が引数渡しだからです。Python では戻り値で返ってくるので不要ですが、統一のために残されています。本書では利用しませんが、使うのなら、あらかじめ指定のデータ（ここでは NumPy 配列）を収容できる変数を用意してから指定します。両方指定すると、おなじ画像がそれぞれに格納されたようにみえますが、実体はおなじものです（参照がおなじ）。

本書では引数を順に説明しますが、この引数渡しの戻り値はスキップします。したがってこの場合、説明が第 1、第 2、第 3 ときたところで第 5 に飛びます。

目　次

1 OpenCV について

本章では、OpenCV でなにができるかをかいつまんで説明します。また、画像処理プログラミングで知っておくべきカラーモデルやデータ型といった画像や映像の表現方法を簡単に示します。

1.1 OpenCV とは

OpenCV（Open Source Computer Vision Library）は画像映像処理ライブラリです。一般的な意味での画像処理だけでなく、ゲーム、AR（拡張現実）、映像アートまでさまざまな分野で活用されています。また、扱う対象も普通の写真だけでなく、医療用 CT 画像、衛星画像、宇宙観測画像、ヒートマップといった特殊な画像に広がっています。

OpenCV の標準ライブラリには 2 値化、フィルタ処理、テンプレートマッチングなど古典的な画像処理メカニズムから、物体や顔の認識、映像解析、特徴の検出と記述、物体追跡、オプティカルフローなど高度なアルゴリズムまで用意されています。最近注目を集めているディープラーニング（深層学習）も利用できます。本書では扱いませんが、最先端のアルゴリズムを含んだ OpenCV Contrib と呼ばれる拡張モジュールも利用可能です。

サポートしている開発環境も多岐にわたっています。言語では Python、C++、Java、MATLAB 用のインタフェースが用意されています。プラットフォームも Windows、macOS、Linux、Android、iOS などで動作します。つまり、一般的な PC 環境があれば、誰でもコンピュータビジョンプログラミングを始めることができます。

オープンソースで、商用目的でも無償で利用できます。Google、Yahoo、Microsoft、Intel、IBM、Sony、Honda、Toyota、NASA などさまざまな企業や機関も OpenCV を利用しています。たとえば Google はストリートビューの全方位画像に、NASA は火星探査ロボットビジョンに OpenCV を活用しています。

OpenCV は今年（2020 年）で 20 歳の誕生日を迎えました。執筆時点での最新バージョンは 2020 年 7 月リリースの 4.4 です。

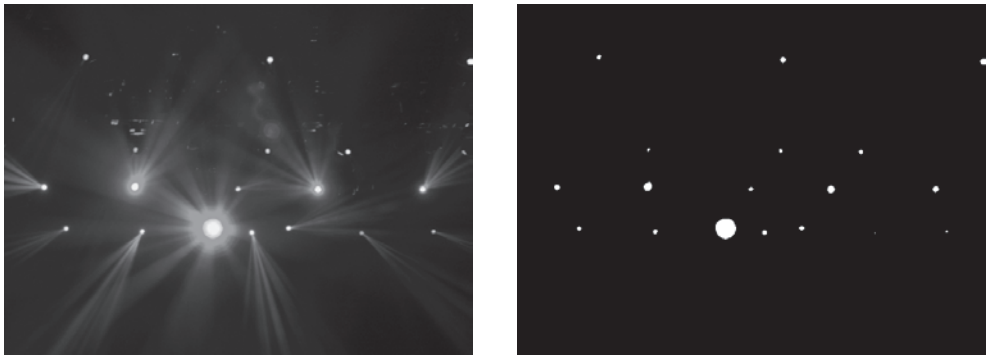
1.2 OpenCV の応用例

本書で扱う画像処理アルゴリズムをいくつかかいつまんで紹介します。これでもたくさんあるように思えますが、OpenCV には 2,500 種類以上のアルゴリズムが実装されているので、まだほんの入り口でしかありません。

1.2.1 2 値化

2 値化は、画像のピクセル値を 0 か 255 などどちらかの値に変換する操作を指します。

たとえば、白黒の濃淡が 0 ～ 255 の値で描かれている白黒（グレースケール）画像のピクセル値を、所定の値を境に 0（真っ黒）と 255（真っ白）のどちらかにします。次の例は、閾値を 128 として 2 値化した結果を示しています。左図が元画像、右図が変換後です。結果から、微妙な濃淡が消えて、白と黒に鮮やかに分かれることがわかります。



入力画像　　2 値化画像

図 1.1 ●グレースケール画像と 2 値画像

閾値の決め方には工夫が必要です。0 と 255 の中間のように決め打ちの値を用いた方法は 2.5 節で、ユーザ操作でインタラクティブに変化させる方法は 3.3 節で説明します。アルゴリズムによる自動判定は 2.5 節の Note で取り上げます。

1.2.2　エッジ検出

画像からモノの輪郭だけを抽出することで、ヒトが線で描くような画像を生成する処理をエッジ（輪郭）検出といいます。

輪郭は、最も素朴には隣り合うピクセルの値が極端に異なる箇所を強調することで検出できます。もっとも、実際にはライティングなどの影響のために輪郭ははっきりと出ません。そこで、各種のエッジ検出アルゴリズムが考案されています。その中でも性能が高いといわれているのは Canny（キャニー）アルゴリズムです。

左図の道路標識のシーンに Canny アルゴリズムを適用すると、右図になります。

入力画像　　エッジ画像

図 1.2 ●エッジ検出例

Canny アルゴリズムを用いたエッジ検出は 2.2 節で説明します。

1.2.3　モルフォロジー演算

モルフォロジー（morphology）演算は、画像中の幾何学的な構造を解析あるいは処理するときに用いる数学的な方法の 1 つです。ノイズ除去、平滑化、テクスチャ解析、エッジ検出など各種の画像処理に用いられます。

モルフォロジー演算にはいくつかの方法があります。簡単にいうと、背景が黒、その中の物体が白の白黒画像で、白に隣接するピクセルを白にする操作を dilate（膨張）といいます。こうすると、白い物体が膨れ上がったり、物体内の黒い箇所が白く埋められたり、離れている複数の白い物体がくっついたりします。反対に、黒に隣接するピクセルを黒にする操作が erode（収縮）で、白い物体が縮こまったり、白い細線が消えたりします。Open（開放）は erode に

続いて dilate を行う操作で、close（閉鎖）は反対に dilate に続いて erode を行う操作です。

次の例は、左の入力画像に対し、それぞれ dilate 処理、erode 処理、close 処理を施したものです。バックネットからわかるように、dilate や close 処理は黒い細線などの微小領域を消去し、erode は反対に強調します。

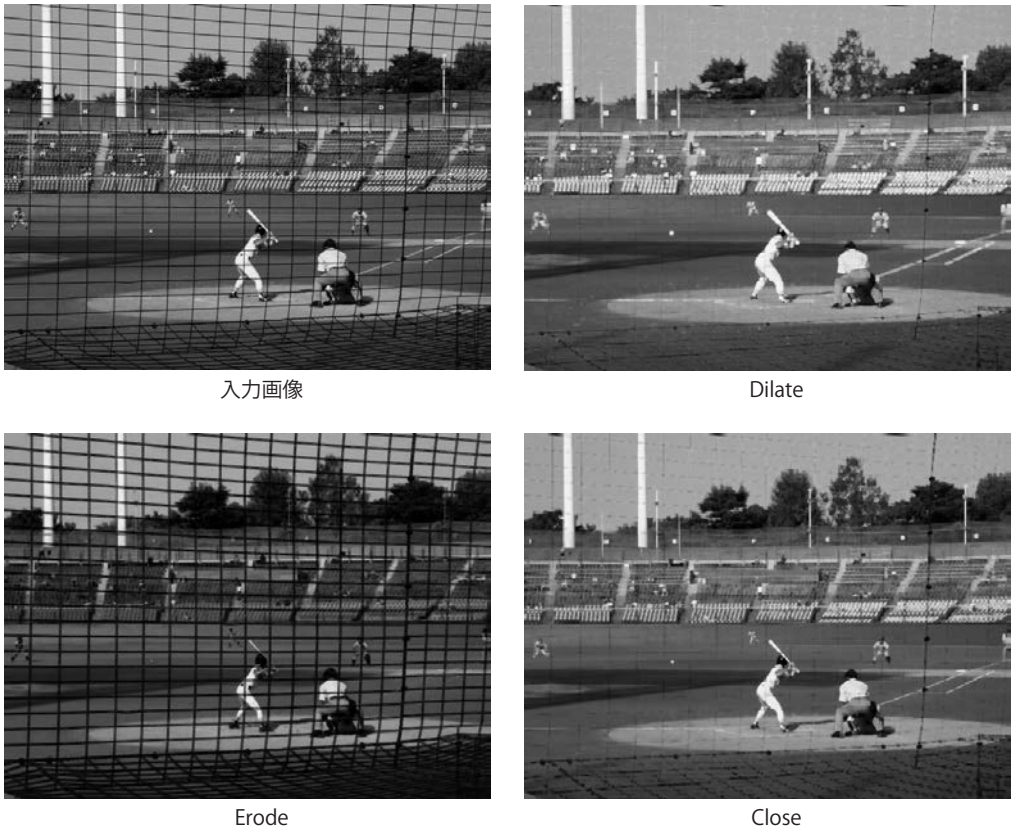


図 1.3 ●モルフォロジー演算処理

モルフォロジー演算は 3.7 節で、その応用例を 4.3 節でそれぞれ説明します。

1.2.4　マスク処理とクロマキー合成

ある画像に別の画像の 1 物体を重ねた合成写真を作成したいとします。たとえば、次の左図に示す本が重なった机上にリンゴの絵を重畳します。

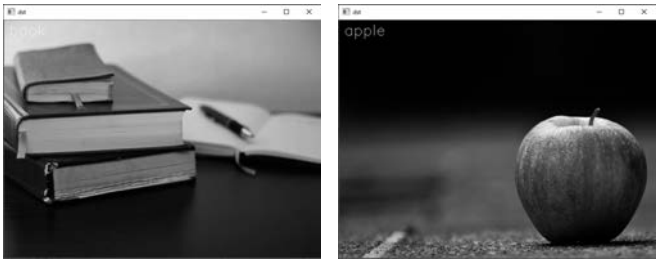


図 1.4 ●マスク処理（元画像）

印画した紙の写真なら、物体をはさみで切り取って、対象の上に貼りつけるでしょう。いわゆるコラージュです。画像処理では次のように処理します。まず、物体とおなじかたちをしたマスクを用意し、これをもとにリンゴ（前景）だけを切り抜きます（左から 2 番目の図）。机（背景）でもおなじようにリンゴの入るところだけをくりぬきます（左から 3 番目の図）。最後に、2 番目と 3 番目を重ねればできあがりです（右図）。

図 1.5 ●マスク処理（処理手順）

もっとも、背景が込み入っているとマスクは簡単に用意できません。そこで、スタジオで撮影した人物と別の背景映像を重ね合わせるクロマキー合成とおなじテクニックを用います。たとえば、次のようにアヒルを緑の背景で撮影します。これで、プログラムから「緑が背景、それ以外が前景としてマスクを生成せよ」と指示できるようになります。

図 1.6 ●クロマキー合成用前景（背景は緑）と自動生成されたマスク

クロマキー合成では、赤－緑－青ベースの RGB 画像より、色相－彩度－明度の HSV 画像のほうが処理が容易です。RGB では 3 つの要素を複雑に組み合わせないと適切な「緑」が指定できませんが、HSV なら「緑」の値（あるいはその範囲）をじかに指定できるからです。

RGB から HSV などの色変換は 4.1 節、HSV の扱いは 4.2 節、マスク処理とクロマキー合成は 4.3 節で説明します。

1.2.5 ヒストグラム

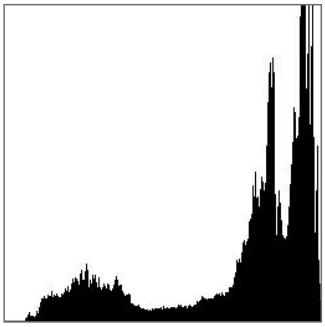
画像のヒストグラム（度数分布図）は、所定の値を持つピクセルの数をグラフにしたものです。2 つの画像のヒストグラムを比較することで、互いにどれだけ似ているかが判断できます。

ヒストグラムは 1 次元のものと多次元のものがあります。1 次元のものは、一般にピクセルの輝度を対象にします。カラー画像のときは、モノクロに変換することで輝度だけを抽出します。1 次元ヒストグラムは横軸がピクセル値、縦軸がピクセル数なので、単純な棒グラフで表現できます。

多次元には、RGB のカラー要素それぞれの値をカウントした 3 次元、その中の 2 つのカラーだけに着目した 2 次元があります。たとえば、HSV（色相－彩度－明度）カラーモデルでは、色相 H を横軸、彩度 S を縦軸とし、(h, s) の組の値を持つピクセルの数を輝度で示します。横軸 H は左端が赤で、順に黄、緑、水色、青、紫と経て、右端でまた赤に戻る色相環の位置を示します。縦軸 S は下にいくほど色がくすみ、上にいくにしたがい鮮やかになります

次に入力画像（左図）から得た輝度ベースの 1 次元ヒストグラム（中央図）と H と S の 2 次元ヒストグラムを示します。


入力画像


1 次元ヒストグラム

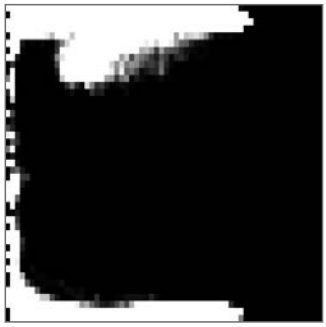

2 次元ヒストグラム

図 1.7 ●ヒストグラム

1 次元ヒストグラムでは右寄りにピークがあるので、明るいピクセルが多い、つまり明るい画像だということが読み取れます。2 次元ヒストグラムでは左上が明るいので、赤〜黄の鮮やかな色が比較的多いことがわかります。

1 次元ヒストグラムは 6.1 節、2 次元ヒストグラムを用いた類似度の判定は 7.2 節でそれぞれ説明します。

1.2.6　オプティカルフロー

映像内の物体の動きの早さと方向を判定するには、オプティカルフローを用います。

オプティカルフローは端的には、前後のフレームに写っているおなじ物体のおなじ箇所どうしを線分で結ぶことで得られます。線分が長いほど物体は高速で移動しており、点ならば移動はしていないことになります。

次の左図は走行車両の前方映像で、右が得られたオプティカルフローを視覚化したものです。線分から左脇を前方（消失点）へとすり抜けていく車両の移動量（速さ）が大きいことがわかります。


入力画像

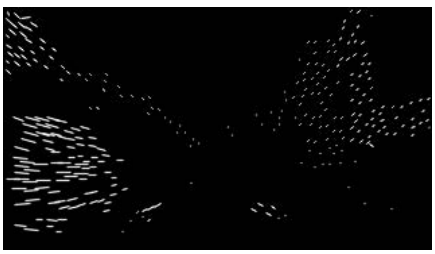

オプティカルフロー

図 1.8 ●オプティカルフロー検出

オプティカルフローの検出アルゴリズムはいろいろと提案されていますが、本書では 6.4 節で Farneback アルゴリズムという手法を紹介します。

1.2.7 テンプレートマッチング

テンプレートマッチングは、画像からたとえばコップや椅子、標識といった所定の物体を検出する手法です。

テンプレートマッチングでは、あらかじめ指定のテンプレート画像を用意し、このテンプレートを画像内のテンプレート画像とおなじサイズの部分領域に順次照らし合わせながら探索します。テンプレートと最も近しいと判断された部分領域が、そのテンプレートの画像が検出された箇所です。

次の例では左図が探索したいテンプレート画像で、右図がその検出結果です（四角枠）。

テンプレート画像

検索結果（検出箇所に枠）

図 1.9 ●テンプレートマッチング

テンプレートマッチングは、7.1 節で説明します。

1.2.8 特徴点検出

テンプレートマッチングは画像に対象とまったくおなじものがあるかをチェックする方法なので、たとえばテンプレートの撮影方向が異なっているだけで、その対象が存在していてもマッチしません。そこで、対象そのものではなく、対象の特徴的な点がどれだけ似ているかだけに注目して検出する方法が考案されました。

このテクニックを用いるには、まず特徴的な点を検出しなければなりません。この技術を特

徴点検出といい、OpenCV には特徴点抽出クラスがいくつか実装されています。具体的には、次のように画像中の線の端点、角などを主とした特徴点を抽出します。

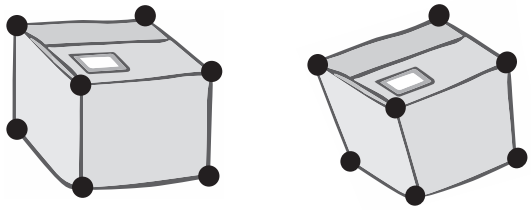
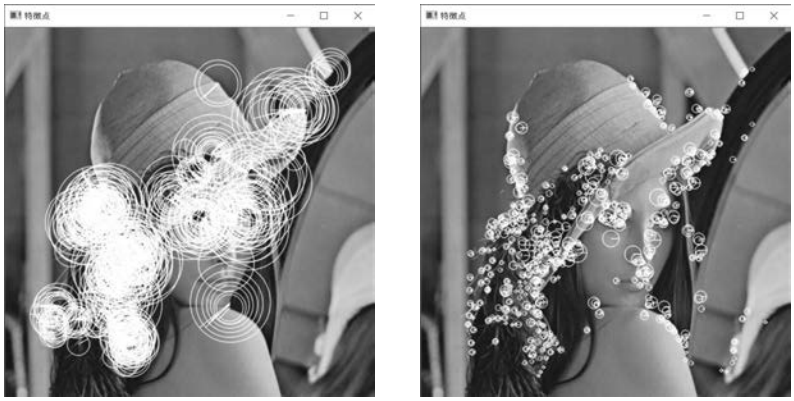


図 1.10 ●特徴点

次の図は、ORB および AKAZE と呼ばれる方式で得た特徴点の位置を描画した例です。円の半径はその特徴点の強度を表しています。円の半径の大きさが両者で大きく異なるのは、左のほうが強度が強いというわけではなく、円のサイズのパラメータ設定が異なるからです。どちらの方法でも、目、鼻、髪の毛の輪郭部分などが検出されているのがわかります。



ORB 特徴点　　AKAZE 特徴点

図 1.11 ●特徴点抽出手法

特徴点を用いて物体を検出する方法は 7.3 節で取り上げます。

1.2.9　ディープラーニングによる物体認識

画像の「どこに」「なにが」あるかを答えるためにいろいろな手法が考案されてきましたが、昨今では認識性能の高さからディープラーニング（深層学習）がよく用いられます。

無作為にあたえられた画像内になにがあるかを答える問題を一般物体認識といいます。画像になにが写しだされているかを訊くことで相手がヒトか機械かをチェックするウェブのアクセス方式があることからわかるように、機械にはかなりむずかしい問題です。世界には物体の種類が無数にあり、その向きや色などのバリエーションも無限にあるからです。しかし、種類を絞ったうえでモデルを構築し、画像を大量に学習させれば、かなりの精度で物体を認識できるようになりました。

次にディープラーニングによる物体認識の例を示します。左図では歩行者の顔を、右図では道路上の一般的な物体（信号機と車）をそれぞれ認識しています。

図 1.12 ●顔と一般物体の認識（右図はトリミングされている）

ディープラーニングは第 8 章で取り上げます。

1.3 画像の構造

OpenCV は、基本的に 1 枚の画像を対象に各種の操作をします。ここで画像は、写真、映像の 1 フレーム、あるいはグラフィックスが描画されたキャンバスのような静止画像全般を指します。Python 版 OpenCV では、画像を数値計算用モジュール NumPy の `ndarray` 多次元配列（n-dimensional array）のオブジェクトに収容して管理します。

本書では、この `np.ndarray` で表現された画像を NumPy 配列と呼びます。

オブジェクト（変数）が NumPy 配列かは `type` 関数から確認できます。次の例では、NumPy オブジェクト `mono` を確認しています。

```
>>> type(mono)
<class 'numpy.ndarray'>
```

1.3.1　画像サイズ

1 枚の画像を構成している 1 つ 1 つの点をピクセルといいます。画像の素なので、画素ともいいます。ピクセルは、次のように直交座標領域に縦横に整然と並んでいます。

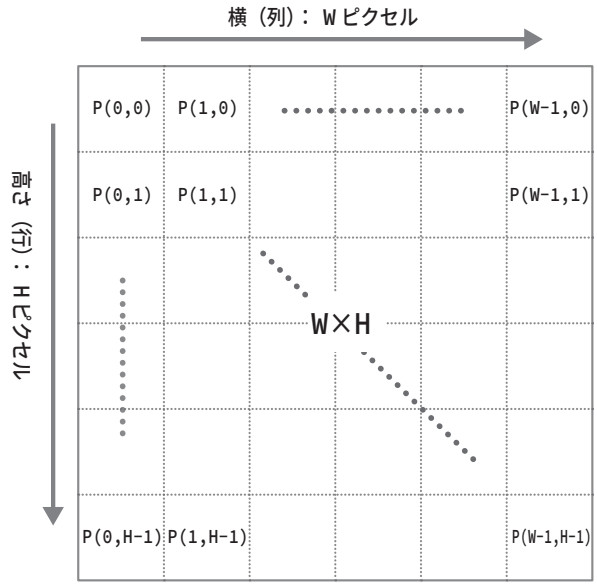


図 1.13 ●画像の構成

座標系の横と縦のピクセルの数の組を画像のサイズといい、640 × 400 あるいは (640, 400) のように表記します。直交座標の範囲は、左上を起点とし、横（x）は左から右方向に 0 から幅−1 まで、縦は上から下方向におなじく 0 から高さ−1 までです。上図ではそれぞれ W − 1、H − 1 で示しています。

NumPy 配列では、幅と高さの値を `np.ndarray.shape`（タプル）から取得できます。ただし、行列計算用に開発されている NumPy 配列では、高さ（行）、幅（列）の順に値が並んでいるので注意が必要です。