|  |
| --- |
| 博士課程教育リーディングプログラム  実世界データ循環学リーダー人材育成プログラム |
| データツールファースト |
| OpenCV入門 |

|  |
| --- |
| 名古屋大学  2014/03/02 |

* Linuxは、Linux Torvaldsの米国およびその他の国における登録商標あるいは商標です。
* その他の、記載されている会社名、製品名は、各社の登録商標または商標です。

目次

[1. はじめに 1](#_Toc381528658)

[1.1 背景 1](#_Toc381528659)

[1.2 本書について 1](#_Toc381528660)

[2. OpenCV概略 2](#_Toc381528661)

[2.1 OpenCVとは 2](#_Toc381528662)

[2.2. モジュール構成と機能 3](#_Toc381528663)

[3. 基本 5](#_Toc381528664)

[3.1 基本構造体 5](#_Toc381528665)

[3.2 配列操作 9](#_Toc381528666)

[3.3 描画 13](#_Toc381528667)

[4. 画像処理 18](#_Toc381528668)

[4.1 画像変換 18](#_Toc381528669)

[4.2 幾何学変換 20](#_Toc381528670)

[4.3 フィルタリング 23](#_Toc381528671)

[4.4 構造や形状の解析 27](#_Toc381528672)

[4.5 特徴検出 31](#_Toc381528673)

[4.6 物体検出 36](#_Toc381528674)

[5. ユーザインタフェースとメディア入出力 38](#_Toc381528675)

[5.1 ユーザインタフェース 38](#_Toc381528676)

[5.2 メディア入出力 41](#_Toc381528677)

[6. オブジェクト検出と機械学習 46](#_Toc381528678)

[6.1 オブジェクト検出 46](#_Toc381528679)

[6.2. 機械学習 49](#_Toc381528680)

[7. 付録 53](#_Toc381528681)

[7.1 OpenCVのインストール 53](#_Toc381528682)

[7.2 サンプルコード 57](#_Toc381528683)

[7.3演習問題 61](#_Toc381528684)

[8. 参考文献 62](#_Toc381528685)

[9. 変更履歴 63](#_Toc381528686)

# はじめに

## 1.1 背景

近年、USB接続のウェブカメラが安価になり、タブレットPCや携帯電話などはカメラを塔載したものが標準となってきました。インターネットのブロードバンド化や、無線LANサービスの拡大などにより、撮影した画像をいつでも手軽に公開できるようになりました。公開する際には、飾りをつけたり、プライバシなど問題になりそうなところを見せないようにしたり、画像を加工して公開している人は多いと思われます。

デジタルカメラなどの機能として、全員が笑顔で、ちゃんと目を開けている瞬間を選んで撮影できたり、撮影した画像の目だけが大きく強調されたりするようなものもあります。監視カメラでは、暴れている人や不審な行動をしている人を自動的に識別するような取り組みもされています。また、最近普及してきた自動車の自動ブレーキシステムも、カメラ画像が最も重要な情報として使用されています。

このように、意識するしないに関わらず、画像データを扱う場面が多くなっています。静止画を加工するだけでなく、その画像から何らかの情報を得たり、動画からリアルタイムに情報を得たり、何らかの情報を加えた動画を生成するなど、これらをまとめて、コンピュータビジョンと呼びます。

## 1.2 本書について

本書は、OpenCVを使用してコンピュータビジョン処理を行うための入門書です。OpenCVの基本的な使い方を習得することを目的としています。実践的に代表的な使用例を挙げて説明しますが、各関数や機能の網羅的な説明はしません。必要に応じて、参考文献に挙げた資料を参照してください。

読者としては、Linux、C++言語について習得していること、行列計算など理解していることを仮定しています。

まず2章ではOpenCVの概略について説明します。3章では以降の章で使用する基本的な機能について説明します。4章ではOpenCVでできる画像処理を、5章ではユーザインタフェースとメディア入出力について、6章ではオブジェクト検出と機械学習について、使用例を示して説明します。7章は付録として、OpenCVのインストール方法、サンプルコードとそのビルド手順、演習問題を用意しています。

# 2. OpenCV概略

## 2.1 OpenCVとは

OpenCV (Open Source Computer Vision) は、コンピュータビジョンを扱うための画像処理ライブラリです。1999年頃から Intel が開発し、2008年からは Willow Garage が引き継いで開発をしています。BSDライセンスのオープンソースとして公開されていますので、無償で、自由に改変でき、再配布も可能です。

<http://opencv.org/>

<http://opencv.jp/>



図2.1 OpenCVロゴ

以下のような豊富な機能を持っています。

* さまざまな画像形式に対応
* 簡易なユーザインタフェース
* 画像の基本構造の保持と操作
* 画像処理 (エッジ抽出、フィルタリングなど)
* 構造解析 (輪郭抽出、特徴点抽出、矩形領域抽出など)
* モーション解析と物体追跡 (オプティカルフローなど)
* パターン認識 (機械学習によるオブジェクト検出)
* カメラキャリブレーションと3次元再構築

動画に対する処理もでき、カメラからの入力画像をリアルタイムに処理することもできます。

動作するプラットフォームとして、Windows, Linux, MacOS, iOS, Android をサポートしています。本書では、Linux (CentOS 6.5) を用います。

言語インタフェースとして、C++, C, Python, Java があり、これらの言語で使用することができます。現在、ウェブページや本には、C 言語インタフェースで解説されているものが多いのですが、今後は、C 言語インタフェースは非推奨となるとのことです。本書では、C++言語インタフェースを対象にします。

OpenCV 2.4.8 が、2013年12月31日にリリースされました。次のリリースとして、OpenCV 3.0 が予定されており、2.4系列は 2.4.8 が最後になるそうです。本書では、OpenCV 2.4.8 を対象にします。

## 2.2. モジュール構成と機能

主なモジュールとして、以下のようなものがあります。

* [core](http://docs.opencv.org/modules/core/doc/core.html)

基本(配列操作、構造体、関数など)を提供します。

* [imgproc](http://docs.opencv.org/modules/imgproc/doc/imgproc.html)

画像処理の機能を提供します。

coreモジュールに依存しています。

* [highgui](http://docs.opencv.org/modules/highgui/doc/highgui.html)

ユーザインタフェースやメディア入出力のための機能を提供します。

core, imgproc モジュールに依存しています。

* [objdetect](http://docs.opencv.org/modules/objdetect/doc/objdetect.html)

オブジェクト検出の機能を提供します。

core, imgproc, highguiモジュールに依存しています。

* [ml](http://docs.opencv.org/modules/ml/doc/ml.html)

機械学習の機能を提供します。

coreモジュールに依存しています。

* [video](http://docs.opencv.org/modules/video/doc/video.html)

ビデオ解析の機能を提供します。

core, imgprocモジュールに依存しています。

* [flann](http://docs.opencv.org/modules/flann/doc/flann.html)

クラスタリングと多次元空間探索の機能を提供します。

coreモジュールに依存しています。

* [features2d](http://docs.opencv.org/modules/features2d/doc/features2d.html)

特徴検出とディスクリプタ抽出などのの機能を提供します。

core, imgproc, highgui, flannモジュールに依存しています。

* [calib3d](http://docs.opencv.org/modules/calib3d/doc/calib3d.html)

カメラキャリブレーション，姿勢推定，立体画像の機能を提供します。

core, imgproc, highgui, flann, features2dモジュールに依存しています。

* [gpu](http://docs.opencv.org/modules/gpu/doc/gpu.html)

GPUを利用する機能を提供します。

core, imgproc, highgui, objdetect, ml, video, flann, features2d, calib3d, legacy, photoモジュールに依存しています。

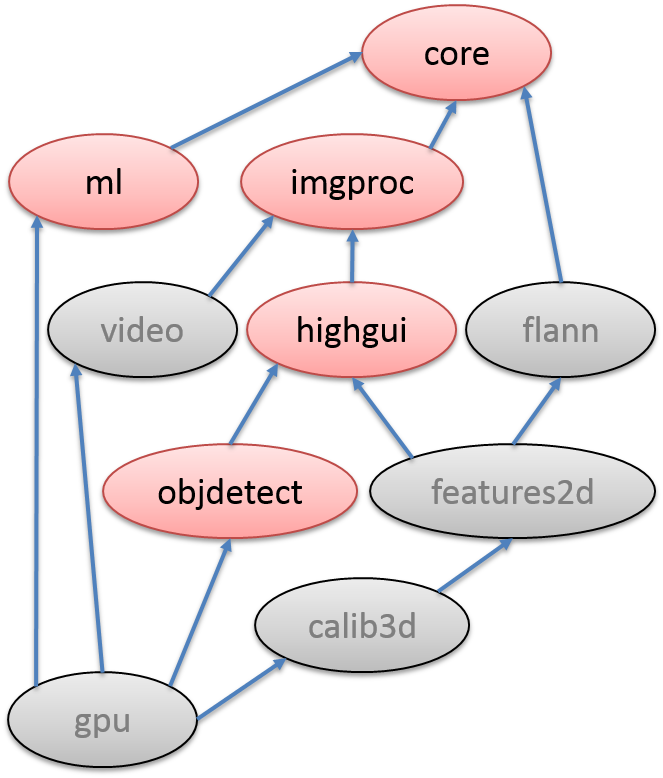


図2.2 モジュール依存関係

以降の章では、OpenCVの基本的な機能、画像処理、ユーザインタフェースとメディア入出力、オブジェクト検出、機械学習について、C++ 言語インタフェースの代表的な使用例を挙げて紹介していきます。その他については参考文献等を参照してください。

OpenCVのC++ 言語インタフェースのクラスや関数などの大部分は、cv 名前空間内に存在します。cv:: を前置するか、"using namespace cv;" と宣言してください。以降のコード片では、"using namespace cv;" と宣言しているものとして、cv:: を省略します。

# 3. 基本

基本となる機能(配列操作、構造体、関数など)は、主にcoreモジュールによって提供されています。

## 3.1 基本構造体

基本となる構造体の主なものを挙げます。

* Mat

n次元の密な配列クラス。(最大CV\_MAX\_DIMまで)

主に、画像データを保持するために、2次元配列として使用します。

画像データは、ピクセル(画素)の集まりです。画像の左上が原点となる座標系になっています。

グレースケールの画像データは、1色(1チャンネル) を保持します。配列の型がCV\_8UC1の場合、符号なし8ビット整数型(0〜255)です。配列の型がCV\_32FC1の場合は、32ビット浮動小数点型で、0.0から1.0までの範囲を使用します。範囲外の値のまま、例えば画像として表示すると負の数は0.0に相当し、1.0より大きい数は1.0に相当する階調で表示されることになります。

カラーの画像データは、青、緑、赤の3色(3チャンネル)を保持します。配列の型がCV\_8UC3 の場合、各チャンネルが符号なし8ビット整数型(0〜255)です。配列の型がCV\_32FC3 の場合は、各チャンネルが32ビット浮動小数点型で、0.0から1.0までの範囲を使用します。範囲外の値のまま、例えば画像として表示すると負の数は0.0に相当し、1.0より大きい数は1.0に相当する階調で表示されることになります。

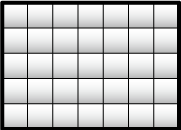
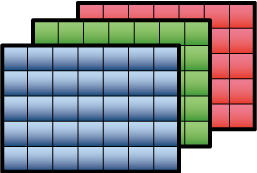
 

図3.1白黒とカラーの画像データ

(<http://opencv.jp/cookbook/opencv_mat.html>より抜粋)

OpenCVではいわゆる RGB でなく、BGR (Blue, Green, Redの順)で扱います。

* + 画像データのコンストラクト (Sizeについては後述)

|  |
| --- |
| Mat gray\_img(Size(320,240), CV\_32FC1);  Mat color\_img(Size(640,480), CV\_8UC3); |

* + 画像データの完全なコピー

|  |
| --- |
| Mat dst\_img = color\_img.clone(); |

* + 画像データが空かどうか

|  |
| --- |
| if (dst\_img.empty()) {  return -1;  } |

* + 画像データの大きさ取得 (Sizeについては後述)

|  |
| --- |
| Size sz = dst\_img.size(); /\* サイズ \*/  int width = dst\_img.cols; /\* 幅 \*/  int height = dst\_img.rows; /\* 高さ \*/ |

* Size\_

画像や矩形の幅(width)と高さ(height)を表現するためのクラステンプレート。

利便性のために，以下のように型の別名が定義されています。

|  |
| --- |
| typedef Size\_<int> Size2i;  typedef Size2i Size;  typedef Size\_<float> Size2f; |

* + サイズのコンストラクト

|  |
| --- |
| Size sz(320,240);  Size2f sz2(320.0, 240.0); |

* + 幅と高さの取得

|  |
| --- |
| int w = sz.width;  int h = sz.height;  float w2 = sz2.width;  float h2 = sz2.height; |

* Point\_

2次元座標上の点(x,y)のためのクラステンプレート。

利便性のために，以下のように型の別名が定義されています。

|  |
| --- |
| typedef Point\_<int> Point2i;  typedef Point2i Point;  typedef Point\_<float> Point2f;  typedef Point\_<double> Point2d; |

* + 2次元座標のコンストラクト

|  |
| --- |
| Point pt(100,200);  Point2f pt2(100.0, 200.0); |

* + 座標の取得

|  |
| --- |
| int x = pt.x;  int y = pt.y;  float x2 = pt2.x;  float y2 = pt2.y; |

* Rect\_

2次元の矩形を表現するためのクラステンプレート。

左上の座標(x,y)と、矩形の幅(width)と高さ(height)によって表現されます。

矩形の上側と左側の境界線は矩形に含まれますが、右側と下側の境界線は含まれません。

利便性のために，以下のように型の別名が定義されています。

|  |
| --- |
| typedef Rect\_<int> Rect; |

* + 矩形のコンストラクト

|  |
| --- |
| Rect rect(100,200,320,240);  Rect rect2(Point(100,200),Size(320,240));  Rect rect3(Point(100,200),Point(420,440)); |

* + 左上座標の取得

|  |
| --- |
| Point pt = rect.tl();  int x = rect.x;  int y = rect.y; |

* + 幅と高さの取得

|  |
| --- |
| Size sz = rect.size();  int w = rect.width;  int h = rect.height; |

* Vec

短い数値ベクトル用のクラステンプレート。

例えば、特徴検出した結果などを保持するのに使用されています。

利便性のために，以下のように型の別名が定義されています。

|  |
| --- |
| typedef Vec<uchar, 2> Vec2b;  typedef Vec<uchar, 3> Vec3b;  typedef Vec<uchar, 4> Vec4b;  typedef Vec<short, 2> Vec2s;  typedef Vec<short, 3> Vec3s;  typedef Vec<short, 4> Vec4s;  typedef Vec<int, 2> Vec2i;  typedef Vec<int, 3> Vec3i;  typedef Vec<int, 4> Vec4i;  typedef Vec<float, 2> Vec2f;  typedef Vec<float, 3> Vec3f;  typedef Vec<float, 4> Vec4f;  typedef Vec<float, 6> Vec6f;  typedef Vec<double, 2> Vec2d;  typedef Vec<double, 3> Vec3d;  typedef Vec<double, 4> Vec4d;  typedef Vec<double, 6> Vec6d; |

Vecの各要素には、operator[]によってアクセスできます。

* Scalar

4要素ベクトルのクラステンプレート。

例えば、ピクセル値を指定する場合に使用されています。

利便性のために，以下のように型の別名が定義されています。

|  |
| --- |
| typedef Scalar\_<double> Scalar; |

* + コンストラクト

|  |
| --- |
| Scalar white(255,255,255); /\* 白 \*/  Scalar black(0,0,0); /\* 黒 \*/  Scalar blue(255,0,0); /\* 青 \*/  Scalar green(0,255,0); /\* 緑 \*/  Scalar red(0,0,255); /\* 赤 \*/ |

ピクセル値の指定では、通常3要素のみ使用しますので、上記の例では4要素目は省略されています。

さらなる詳細や、他の基本構造体については、参考文献等を参照してください。

## 3.2 配列操作

画像データは、2次元の配列(Mat)に保持されているため、配列内のデータを操作することによって簡単な画像処理ができます。いくつかの配列操作例を挙げます。

* 白黒画像をネガポジ反転

CV\_8UC1の白黒画像は、各ピクセル値は符号なし8ビットです。それらの値を NOT 演算することで、ネガポジ反転となります。

|  |
| --- |
| dst\_img = ~src\_img; |

図3.2ネガポジ反転

(左から src\_img, dst\_img)

* 画像の合成 (足し算)

配列の足し算によって、単純に画像を合成することができます。

|  |
| --- |
| add(src\_img, src2\_img, dst\_img); |

これは、下記の式と全く同じです。

|  |
| --- |
| dst\_img = src\_img + src2\_img; |

各ピクセル値(白黒)、各色の値(BGRカラー) が足されますので、全体的に白っぽくなります。

図3.3画像の合成 (足し算)

(左から src\_img, src2\_img, dst\_img)

(注)

各ピクセル値が整数の場合、これらの演算は飽和演算になります。例えば符号なし8ビット整数型の場合、255(最大値)を越える場合は、255(最大値)になります。逆に、0(最小値)を下回る場合は、0(最小値)になります。通常のC++の四則演算で、飽和演算をするには下記のように、saturate\_castを使ってください。

|  |
| --- |
| unsigned char x = saturate\_cast<unsigned char>(a + b); |

* 画像の合成 (重みづけ)

addのように白っぽくならないように、適度な重みづけをして合成するには、addWeighted を使用します。

以下の例では、src\_img を 40%、src2\_img を 60% の重みづけをして、合成しています。(第5引数のガンマ値は各要素に加算する値)

|  |
| --- |
| addWeighted(src\_img, 0.4, src2\_img, 0.6, 0, dst\_img); |

これは、下記の式と全く同じです。

|  |
| --- |
| dst\_img = src\_img \* 0.4 + src2\_img \* 0.6 + 0; |

図3.4画像の合成 (重みづけ)

(左から src\_img, src2\_img, dst\_img)

* 画像の合成 (max)

各ピクセル値(白黒)、各色の値(BGRカラー) 毎に、大きな値を採用します。

|  |
| --- |
| max(src\_img, src2\_img, dst\_img); |

図3.5画像の合成 (max)

(左から src\_img, src2\_img, dst\_img)

* 画像の合成 (min)

各ピクセル値(白黒)、各色の値(BGRカラー) 毎に、小さな値を採用します。

|  |
| --- |
| min(src\_img, src2\_img, dst\_img); |

図3.6画像の合成 (min)

(左から src\_img, src2\_img, dst\_img)

* 上下反転、左右反転

配列要素の並びを変えることによって、上下反転、左右反転ができます。

flipの第3引数が、

0の場合、上下反転

正の数の場合、左右反転(鏡像)

負の数の場合、上下左右反転(180度回転)

となります。

|  |
| --- |
| flip(src\_img, dst\_img, 0); /\* 上下反転 \*/  flip(src\_img, dst2\_img, 1); /\* 左右反転(鏡像) \*/  flip(src\_img, dst3\_img, -1); /\* 上下左右反転(180度回転) \*/ |

図3.7上下反転、左右反転

(左から src\_img, dst\_img, dst2\_img, dst3\_img)

さらなる詳細や、他の配列操作については、参考文献等を参照してください。

## 3.3 描画

簡単な描画関数が用意されています。ここではそれら描画関数について、いくつか使用例を挙げます。

* 全面塗り潰し

横640ピクセル縦480ピクセルのカラー画像データを宣言して、指定した単色に塗り潰します。

|  |
| --- |
| Mat img(Size(640,480), CV\_8UC3);  img = Scalar(200,200,200); |



図3.8全面塗り潰し

* 線分

始点(0,0)と終点(640,480)を指定して、太さ5の線分を描画します。

|  |
| --- |
| line(img, Point(0,0), Point(640,480), Scalar(255,0,0), 5); |

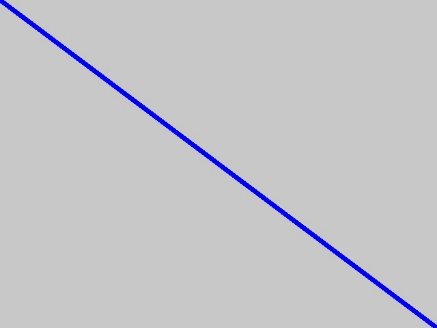


図3.9線分

* 矩形

左上(320,0)と右下(640,240)を指定して、太さ5の矩形を描画します。

|  |
| --- |
| rectangle(img, Point(320,0), Point(640,240), Scalar(0,255,0), 5); |

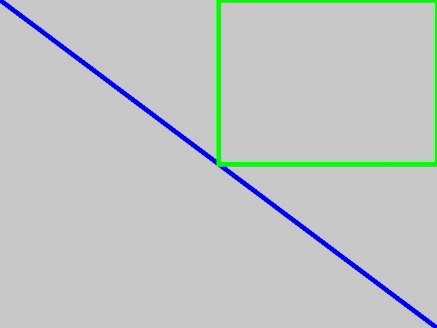


図3.10矩形

左上(340,20)でサイズ(280,200)の矩形を指定して、塗り潰した矩形を描画します。

|  |
| --- |
| rectangle(img, Rect(Point(340,20),Size(280,200)), Scalar(0,255,0), -1); |

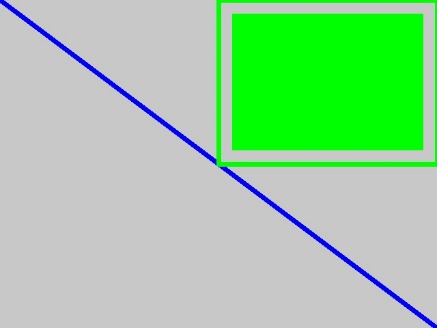


図3.11矩形の塗り潰し

* 円

中心(320,240)と半径(200)を指定して、太さ5の円を描画します。

|  |
| --- |
| circle(img, Point(320,240), 200, Scalar(0,0,255), 5); |

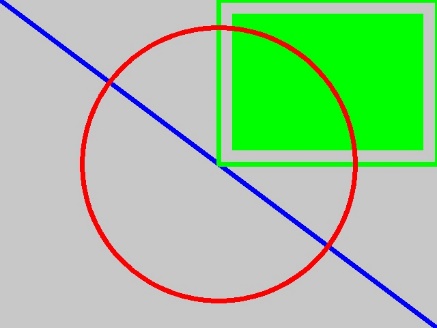


図3.12円

中心(320,240)と半径(150)を指定して、塗り潰した円を描画します。

|  |
| --- |
| circle(img, Point(320,240), 150, Scalar(0,0,255), -1); |

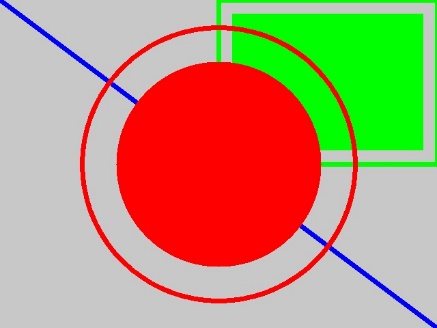


図3.13円の塗り潰し

* 多角形

描画する2つの多角形のためのデータを用意します。

|  |
| --- |
| const int npt[2] = { 4, 6 };  Point pts0[] = {  Point(0,220), Point(160,220),  Point(320,300), Point(160,300),  };  Point pts1[] = {  Point(100,20), Point(200,25), Point(150,10),  Point(140,130), Point(190,120), Point(110,100),  };  const Point \*ppt[2] = { pts0, pts1 }; |

上記データで表わされた多角形 2つを、太さ3で描画します。

|  |
| --- |
| polylines(img, ppt, npt, 2, true, Scalar(255,255,0), 3); |

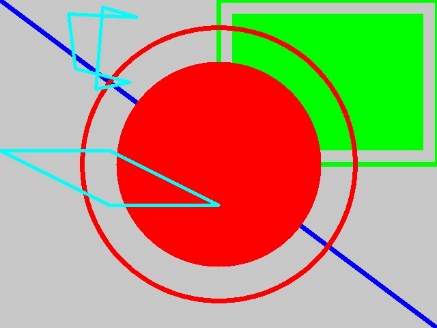


図3.14多角形

上記データで表わされた多角形 2つに囲まれた領域を、塗り潰します。

|  |
| --- |
| fillPoly(img, ppt, npt, 2, Scalar(255,0,255)); |

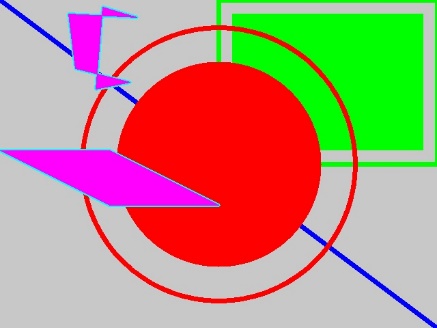


図3.15多角形の塗り潰し

* 文字列

ASCIIコード(ISO646)の印字可能な文字からなる文字列を表示します。日本語は指定できません。

|  |
| --- |
| putText(img, "OpenCV", Point(10,460),  CV\_FONT\_HERSHEY\_SIMPLEX, 2.5, Scalar(0,255,255), 5, CV\_AA); |

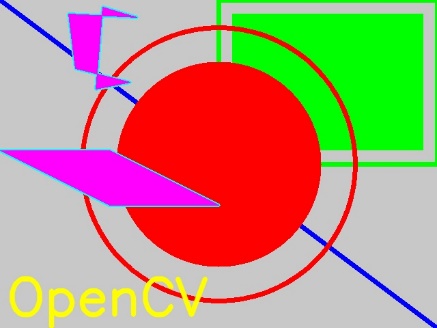


図3.16文字列

さらなる詳細(各引数や省略している引数など)や、他の描画関数については、参考文献等を参照してください。

# 4. 画像処理

画像処理の機能は、主にimgprocモジュールによって提供されています。

## 4.1 画像変換

画像変換について、いくつか使用例を挙げます。

* カラーモデルの変換

OpenCVで通常扱うカラー画像は BGR (青(Blue)、緑(Green)、赤(Red)) です。

cvtColor() 関数を用いて、BGR から下記のようなカラーモデルに変換(または逆変換)をすることができます。

GRAY 白黒(グレースケール)

HSV 色相(Hue)、彩度(Saturation)、明度(Value)

YCrCb 輝度(Y)、色差(CrとCb)

(他にもありますが、詳細は参考文献を参照してください)

cvtColor() の第3引数は、CV\_{変換前}2{変換後} のように指定します。

BGR からグレースケールに変換します。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY); |

図4.1グレースケール変換

(左から src\_img, dst\_img)

BGR から HSV に変換して、HSVの成分を変更し、BGRへ戻します。

|  |
| --- |
| cvtColor(src\_img, work\_img, CV\_BGR2HSV);  work\_img += Scalar(0,100,0); /\* 彩度を変更 \*/  cvtColor(work\_img, dst\_img, CV\_HSV2BGR); |

図4.2 HSV変換して彩度を変更

(左から src\_img, dst\_img)

BGR から YCrCb に変換して、YCrCbの成分を変更し、BGRへ戻します。

|  |
| --- |
| cvtColor(src\_img, work\_img, CV\_BGR2YCrCb);  work\_img += Scalar(100,0,0); /\* 輝度を変更 \*/  cvtColor(work\_img, dst\_img, CV\_YCrCb2BGR); |

図4.3 YCrCb変換して輝度を変更

(左から src\_img, dst\_img)

* 2値画像に変換

threshold() 関数によって、グレースケールの画像を、大津のアルゴリズムで決定された最適な閾値を用いて、2値画像に変換します。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY);  threshold(dst\_img, dst2\_img, -1, 255, THRESH\_BINARY|THRESH\_OTSU); |

図4.4 2値画像に変換

(左から src\_img, dst\_img, dst2\_img)

第5引数の THRESH\_BINARY によって、2値化を指定しています。閾値以下のピクセルは0になり、閾値を越えるピクセルは、第4引数の値(255)になります。第3引数が閾値ですが、ここでは、第5引数に THRESH\_OTSU を指定することで、大津のアルゴリズムで決定された最適な閾値を使用しています。(第3引数に、-1を指定していますが、この値は閾値として使用されていません)

さらなる詳細(各引数や省略している引数など)や、他の画像変換の関数については、参考文献等を参照してください。

## 4.2 幾何学変換

画像の幾何学変換について、いくつか使用例を挙げます。

* サイズの変更

指定サイズに変更します。

|  |
| --- |
| resize(src\_img, dst\_img, Size(640,480)); |

図4.5指定サイズに変更

(左から src\_img, dst\_img)

倍率を指定してサイズを変更します。

|  |
| --- |
| resize(src\_img, dst\_img, Size(), 0.8, 1.2); |

図4.6指定倍率に変更

(左から src\_img, dst\_img)

* アフィン変換

3組の対応点からアフィン変換行列を求めて、アフィン変換します。

|  |
| --- |
| const Point2f src\_pt[] = {  Point2f(200, 200), Point2f(300, 200), Point2f(200, 300)  };  const Point2f dst\_pt[] = {  Point2f(200, 250), Point2f(250, 250), Point2f(250, 350)  };  const Mat affine\_matrix = getAffineTransform(src\_pt, dst\_pt);  warpAffine(src\_img, dst\_img, affine\_matrix, src\_img.size()); |

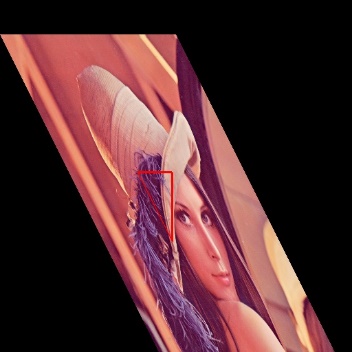
 

図4.7アフィン変換 (3組の対応点)

(左から src\_img, dst\_img)

中心座標と回転角(度)と倍率からアフィン変換行列を求めて、アフィン変換(回転)します。回転角が正の場合に反時計回りになります。

|  |
| --- |
| Point2f center(src\_img.cols\*0.5, src\_img.rows\*0.5);  float angle = 45.0, scale = 1.0;  const Mat affine\_matrix2 = getRotationMatrix2D(center, angle, scale);  warpAffine(src\_img, dst\_img, affine\_matrix2, src\_img.size()); |

図4.8アフィン変換 (回転)

(左から src\_img, dst\_img)

* 透視変換

4組の対応点から透視変換行列を求めて、透視変換します。

|  |
| --- |
| const Point2f pts1[] = {  Point2f(200,200), Point2f(300,200),  Point2f(300,300), Point2f(200,300)  };  const Point2f pts2[] = {  Point2f(210,200), Point2f(290,210),  Point2f(290,290), Point2f(210,300)  };  Mat perspective\_matrix = getPerspectiveTransform(pts1, pts2);  warpPerspective(src\_img, dst\_img, perspective\_matrix, src\_img.size()); |

図4.9透視変換

(左から src\_img, dst\_img)

さらなる詳細(各引数や省略している引数など)や、他の幾何学変換の関数については、参考文献等を参照してください。

## 4.3 フィルタリング

画像のフィルタリングについて、いくつか使用例を挙げます。

* 膨張 (Dilation)

各ピクセルについて、隣接するピクセルのうち最大値(白っぽい)ものを、そのピクセル値にします。これを複数回繰り返します。たとえば単純に白黒画像であれば、白っぽいところの周辺が白くなり、膨張したように見えます。黒い小さな点や細い線は消えることがあります。

以下の例では、膨張処理を1回と、3回します。

|  |
| --- |
| dilate(src\_img, dst\_img, Mat(), Point(-1,-1), 1);  dilate(src\_img, dst2\_img, Mat(), Point(-1,-1), 3); |

図4.10膨張

(左から src\_img, dst\_img, dst2\_img)

ここでは、第3,4引数が Mat(), Point(-1,-1) なので、3x3矩形構造要素で、アンカー位置はその中心になっています。

* 収縮 (Erosion)

各ピクセルについて、隣接するピクセルのうち最小値(黒っぽい)ものを、そのピクセル値にします。これを複数回繰り返します。たとえば単純に白黒画像であれば、黒っぽいところの周辺が黒くなり、収縮したように見えます。白い小さな点や細い線は消えることがあります。

以下の例では、収縮処理を1回と、3回します。

|  |
| --- |
| erode(src\_img, dst\_img, Mat(), Point(-1,-1), 1);  erode(src\_img, dst2\_img, Mat(), Point(-1,-1), 3); |

図4.11収縮

(左から src\_img, dst\_img, dst2\_img)

ここでは、第3,4引数が Mat(), Point(-1,-1) なので、3x3矩形構造要素で、アンカー位置はその中心になっています。

* Opening処理

収縮を複数回した後、膨張を同じ回数行なうことを、Opening処理といいます。たとえば単純に白黒画像であれば、白い小さな点や細い線が目立たなくなり、元の大きさに戻ることになります。

以下の例では、Opening処理の1回と、3回を実行します。

|  |
| --- |
| morphologyEx(src\_img, dst\_img, MORPH\_OPEN, Mat(), Point(-1,-1), 1);  morphologyEx(src\_img, dst\_img, MORPH\_OPEN, Mat(), Point(-1,-1), 3); |

図4.12 Opening処理

(左から src\_img, dst\_img, dst2\_img)

ここでは、第3,4引数が Mat(), Point(-1,-1) なので、3x3矩形構造要素で、アンカー位置はその中心になっています。

* Closing処理

膨張を複数回した後、収縮を同じ回数行なうことを、Closing処理といいます。たとえば単純に白黒画像であれば、黒い小さな点や細い線が目立たなくなり、元の大きさに戻ることになります。

以下の例では、Closing処理の1回と、3回を実行します。

|  |
| --- |
| morphologyEx(src\_img, dst\_img, MORPH\_CLOSE, Mat(), Point(-1,-1), 1);  morphologyEx(src\_img, dst\_img, MORPH\_CLOSE, Mat(), Point(-1,-1), 3); |

図4.13 Closing処理

(左から src\_img, dst\_img, dst2\_img)

ここでは、第3,4引数が Mat(), Point(-1,-1) なので、3x3矩形構造要素で、アンカー位置はその中心になっています。

* 平滑化

正規化されたボックスフィルタを用いて画像を平滑化します。指定サイズの周辺ピクセル値の平均値を、中央のピクセル値にします。

|  |
| --- |
| blur(src\_img, dst\_img, Size(5,5));  blur(src\_img, dst2\_img, Size(2,100)); |

図4.14 blurで平滑化

(左から src\_img, dst\_img, dst2\_img)

ガウシアンフィルタを用いて画像を平滑化します。指定サイズの周辺ピクセル値に対してガウス分布に従った係数で加重平均した値を、中央のピクセル値にします。

|  |
| --- |
| GaussianBlur(src\_img, dst\_img, Size(11,11), 10, 10);  GaussianBlur(src\_img, dst2\_img, Size(51,3), 80, 3); |

図4.15ガウシアンフィルタで平滑化

(左から src\_img, dst\_img, dst2\_img)

メディアンフィルタを用いて画像を平滑化します。指定サイズの正方形の周辺ピクセル値のうちの中央値を、中央のピクセル値にします。

|  |
| --- |
| medianBlur(src\_img, dst\_img, 11);  medianBlur(src\_img, dst2\_img, 51); |

図4.16メディアンフィルタで平滑化

(左から src\_img, dst\_img, dst2\_img)

さらなる詳細(各引数や省略している引数など)や、他のフィルタリング関数については、参考文献等を参照してください。

## 4.4 構造や形状の解析

構造や形状の解析について、いくつか使用例を挙げます。

* 輪郭の検出

2値化し、findContours() によって輪郭を検出します。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY);  threshold(dst\_img, dst\_img, -1, 255, THRESH\_BINARY|THRESH\_OTSU);  std::vector<std::vector<Point> > contours;  findContours(dst\_img, contours, CV\_RETR\_LIST, CV\_CHAIN\_APPROX\_SIMPLE); |

ここで contours は、Point型のベクトルのベクトルであり、検出された輪郭が複数、格納されます。各輪郭は、Point型のベクトルで描線の順になっています。この contours を、drawContours() によって描画します。

|  |
| --- |
| dst2\_img = src\_img.clone();  drawContours(dst2\_img, contours, -1, Scalar(0,255,0), 2); |

図4.17 輪郭の検出

(左から src\_img, dst\_img, dst2\_img)

* 外接する図形

ここでは、乱数の点座標集合 pts を用意し、それらに外接する図形を検出します。

|  |
| --- |
| std::vector<Point> pts(30);  randu(pts, 100, 400);  for (int i = 0; i < 30; i++) {  circle(src\_img, pts[i], 3, Scalar(255,255,0), -1);  } |

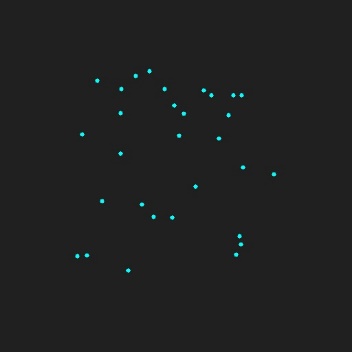


図4.18 乱数で点を描画

boundingRect() を用いて、点座標集合の外接矩形を求めます。

|  |
| --- |
| Rect rect = boundingRect(pts); |

ここでrectには、検出された矩形データが格納されているので、この値を用いて矩形を描画します。

|  |
| --- |
| dst\_img = src\_img.clone();  rectangle(dst\_img, rect, Scalar(0,0,255), 2); |

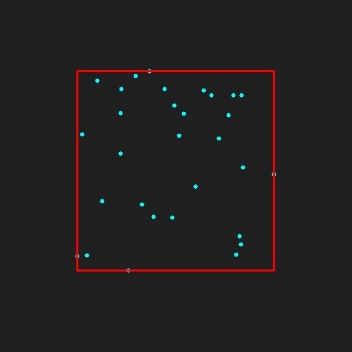


図4.19 外接矩形

minEnclosingCircle() を用いて、点座標集合の外接円を求めます。

|  |
| --- |
| Point2f center;  float radius;  minEnclosingCircle(pts, center, radius); |

ここでcenterとradius には、検出された円の中心座標と半径が格納されているので、この値を用いて円を描画します。

|  |
| --- |
| dst\_img = src\_img.clone();  circle(dst\_img, center, radius, Scalar(0,255,0), 2); |

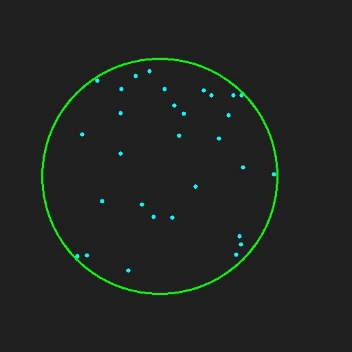


図4.20 外接円

convexHull() を用いて、点座標集合の凸包を求めます。

|  |
| --- |
| std::vector<Point> hull;  convexHull(pts, hull); |

ここでhullは、Point型のベクトルであり、検出された凸包を構成する点が描線の順に格納されているので、各点を結ぶ線分を描画します。

|  |
| --- |
| dst\_img = src\_img.clone();  int hnum = hull.size();  for (int i = 0; i < hnum - 1; i++) {  line(dst\_img, hull[i], hull[i+1], Scalar(255,0,0), 2);  }  line(dst\_img, hull[hnum-1], hull[0], Scalar(255,0,0), 2); |

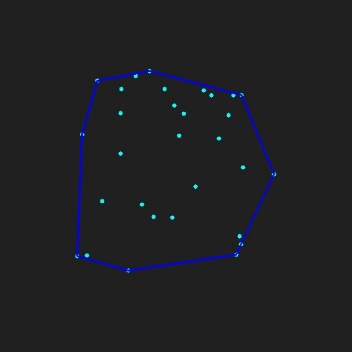


図4.21 凸包

さらなる詳細(各引数や省略している引数など)や、他の構造や形状の解析のための関数については、参考文献等を参照してください。

## 4.5 特徴検出

特徴検出について、いくつか使用例を挙げます。

* エッジ検出

Canny() 関数によって、グレースケールの画像から、閾値(50と200)を指定して、エッジ検出します。閾値の小さい方の値がエッジの接続に利用され、大きい方の値が明確なエッジの初期セグメントを検出するのに利用されます。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY);  Canny(dst\_img, dst2\_img, 50, 200); |

図4.22 エッジ検出

(左から src\_img, dst\_img, dst2\_img)

* 直線検出

直線検出のために、元画像のエッジ検出しておきます。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY);  Canny(dst\_img, dst\_img, 50 200); |

図4.23 直線検出のためのエッジ検出

(左から src\_img, dst\_img)

HoughLines() 関数によって古典的[Hough変換](http://ja.wikipedia.org/wiki/%E3%83%8F%E3%83%95%E5%A4%89%E6%8F%9B)し、エッジ検出画像から直線を検出します。ここでは、長さの分解能1、角度の分解能π/180(1度に相当)、直線と判断する閾値200を指定しています。

HoughLines()の内部ではHough空間の各直線毎に、存在する可能性を見付けるたびにカウントアップする仕組みがあります。その結果が「直線と判断する閾値」を超えた場合に、直線を検出したことになります。同じ画像でも、サイズを大きくすると直線の可能性を見付ける回数が多くなるので、閾値を高く調整する必要があります。

|  |
| --- |
| std::vector<Vec2f> lines;  HoughLines(dst\_img, lines, 1, CV\_PI/180, 200); |

Hough空間で直線は、2要素(ρ,θ)で表わされます。検出された直線が無限に長いと仮定して、ρは原点(画像の左上コーナー)からの最短距離(垂線の距離)、θはラジアン単位で表される直線の角度を示しています。角度は、0が垂直方向を示し、y軸に対して時計回りになっています。

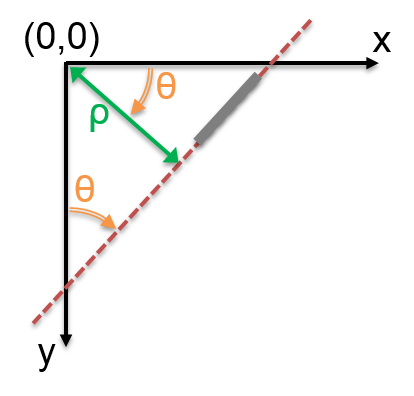


図4.24 Hough空間の直線

ここで lines は、Vec2f型(ρ,θ)のベクトルであり、検出された直線が複数、格納されます。それぞれのρとθから、直線上の原点から最も近い位置(x0,y0)を求め、そこから十分離れた両端(pt1,pt2)を求めて描画します。

|  |
| --- |
| dst2\_img = src\_img.clone();  std::vector<Vec2f>::iterator it = lines.begin();  for (; it != lines.end(); it++) {  float rho = (\*it)[0];  float theta = (\*it)[1];  double a = cos(theta), b = sin(theta);  double x0 = a \* rho, y0 = b \* rho;  Point2d pt1(x0 + 1000 \* (-b), y0 + 1000 \* (a));  Point2d pt2(x0 - 1000 \* (-b), y0 - 1000 \* (a));  line(dst2\_img, pt1, pt2, Scalar(0,0,255), 2);  } |

図4.25直線検出

(左から dst\_img, dst2\_img)

HoughLinesP() 関数によって確率的[Hough変換](http://ja.wikipedia.org/wiki/%E3%83%8F%E3%83%95%E5%A4%89%E6%8F%9B)し、エッジ検出画像から線分を検出します。ここでは、距離の分解能1、角度の分解能π/180(1度に相当)、線分と判断する閾値200を指定しています。また、最小線分長50、同一線分みなす2点間の最大距離10 も指定しています。

HoughLinesP()の内部でも HoughLines()と同様に、線分が存在する可能性を見付けるたびにカウントアップする仕組みがあります。その結果が「線分と判断する閾値」を超えた場合に、線分を検出したことになります。同じ画像でも、サイズを大きくすると線分の可能性を見付ける回数が多くなるので、閾値を高く調整する必要があります。

|  |
| --- |
| std::vector<Vec4i> linesP;  HoughLinesP(dst\_img, linesP, 1, CV\_PI/180, 200, 50, 10); |

ここで linesP は、Vec4i型(x1,y1,x2,y2)のベクトルであり、検出された線分が複数、格納されます。(x1,y1)と(x2,y2)が線分の端点を示しています。

これら線分の端点から、線分を描画します。

|  |
| --- |
| dst2\_img = src\_img.clone();  std::vector<Vec4i>::iterator it = linesP.begin();  for (; it != linesP.end(); it++) {  Point pt1((\*it)[0], (\*it)[1]);  Point pt2((\*it)[2], (\*it)[3]);  line(dst2\_img, pt1, pt2, Scalar(0,0,255), 2);  } |

図4.26線分検出

(左から dst\_img, dst2\_img)

* 円検出

円検出のために、元画像をグレースケールに変換し、誤検出を減らすために平滑化しておきます。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY);  GaussianBlur(dst\_img, dst\_img, Size(11,11), 2, 2); |

図4.27円検出のための平滑化

(左から src\_img, dst\_img)

HoughCircles() 関数によって[Hough変換](http://ja.wikipedia.org/wiki/%E3%83%8F%E3%83%95%E5%A4%89%E6%8F%9B)し、円を検出します。ここでは、[Hough変換](http://ja.wikipedia.org/wiki/%E3%83%8F%E3%83%95%E5%A4%89%E6%8F%9B)方式は CV\_HOUGH\_GRADIENT しかないため、それを指定しています。また、画像分解能に対する投票分解能の比率の逆数1、円の中心同士の最小距離100を指定しています。

HoughCircles()の内部では円の可能性を見付けるたびにカウントアップする仕組みがあり、これを投票と呼んでいます。「画像分解能に対する投票分解能の比率(の逆数)」によって、画像の大きさよらず円が検出できることになります。

|  |
| --- |
| std::vector<Vec3f> circles;  HoughCircles(dst\_img, circles, CV\_HOUGH\_GRADIENT, 1, 100); |

ここで circles は、Vec3f型(x,y,r)のベクトルであり、検出された円が複数、格納されます。(x,y)が円の中心座標、rが円の半径を示しています。

|  |
| --- |
| dst2\_img = src\_img.clone();  std::vector<Vec3f>::iterator it = circles.begin();  for (; it != circles.end(); it++) {  Point2f center((\*it)[0], (\*it)[1]);  float radius = (\*it)[2];  circle(dst\_img, center, radius, Scalar(0,0,255), 2);  } |

図4.28円検出

(左から dst\_img, dst2\_img)

さらなる詳細(各引数や省略している引数など)や、他の特徴検出のための関数については、参考文献等を参照してください。

## 4.6 物体検出

テンプレートマッチングによる物体検出の使用例を挙げます。

* テンプレートマッチング

matchTemplate()関数によって、画像(src\_img)から、テンプレート画像(tmpl\_img)を探します。

|  |
| --- |
| matchTemplate(src\_img, tmpl\_img, dst\_img, CV\_TM\_CCOEFF\_NORMED); |

ここで dst\_img には、対応する座標における類似度が格納されます。minMaxLoc() 関数で、dst\_img 内の最大値と、最大値がある座標を求めます。すると、その座標が、テンプレートに一番類似している部分となります。

|  |
| --- |
| Point pt;  double max;  minMaxLoc(dst\_img, NULL, &max, NULL, &pt);    dst\_img = src\_img.clone();  rectangle(dst2\_img, Rect(pt, tmpl\_img.size()), Scalar(0,0,255), 2); |

図4.29テンプレートマッチング (左から src\_img, tmpl\_img, dst\_img, dst2.img)

ただし、最大値を選ぶだけですので、全く類似しているものがない場合もどれか1箇所が候補として挙がります。その最大値によって、マッチングしたとみなしてよいかどうかを判断する必要があるでしょう。

さらなる詳細(各引数など)については、参考文献等を参照してください。

# 5. ユーザインタフェースとメディア入出力

ユーザインタフェースとメディア入出力の機能は、主にhighguiモジュールによって提供されています。

## 5.1 ユーザインタフェース

ユーザインタフェースについて、いくつか使用例を挙げます。

* ウインドウ

namedWindow() 関数によってウインドウを作成します。引数の文字列は、ウィンドウを識別するためのものであり、ウィンドウのタイトルバーに表示されます。ウインドウには、画像とトラックバーを置くことができます。

|  |
| --- |
| namedWindow("ShowImage"); |

ここでは、第2引数を省略しています。省略時は CV\_WINDOW\_AUTOSIZE が設定されていることになり、表示される画像サイズに合わせてウィンドウサイズが自動的に調整され、ユーザが手動でウィンドウサイズを変更することはできません。



図5.1 ウインドウ

* 画像の表示

imshow() 関数によって、指定したウィンドウに、画像を表示します。引数の文字列でウィンドウを指定します。

|  |
| --- |
| imshow("ShowImage", img); |

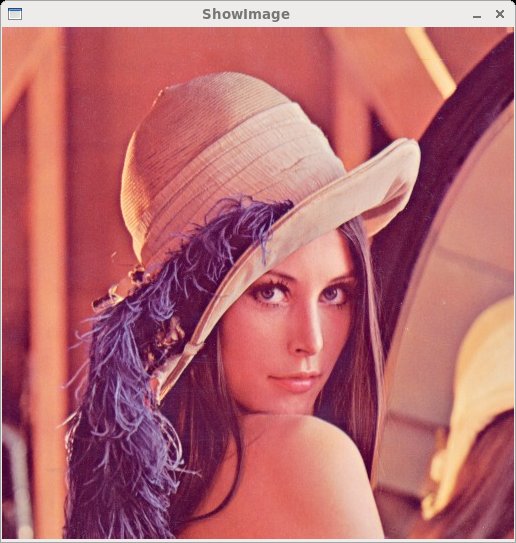


図5.2 画像表示したウインドウ

ウインドウ作成時に CV\_WINDOW\_AUTOSIZE が指定されていれば、表示される画像サイズに合わせてウィンドウサイズが自動的に調整されます。

* キー入力

waitKey() 関数で、キー入力を待つことできます。キー入力があれば、押されたキーコードを返します。引数がないか 0 以下の場合、無限に待ちます。引数が 0 より大きい場合、その値だけミリ秒単位で待ちます。指定時間を過ぎてもキー入力がなければ、-1 を返します。

以下の例では、表示した画像を確認し、キー入力待ちをし、次のステップへ進みます。例えば、プログラムが終了してしまうとウインドウが全て消えてしまいますので、表示した画像を確認してから終了する場合にキー入力待ちをするとよいでしょう。

|  |
| --- |
| imshow("ShowImage", img);  std::cout << "Hit Any Key!" << std::endl;  waitKey(); |

画像の変化していく様子を、約15フレーム/秒の動画として表示し続け、'q' をキー入力することで終了します。

|  |
| --- |
| namedWindow("ShowImage");  for (;;) {  /\* ここで、img に対して、なにか処理する \*/  imshow("ShowImage", img);  if (waitKey(66) == 'q') break;  } |

* トラックバー

createTrackbar() 関数によって、トラックバーを指定したウィンドウの上部に配置します。トラックバー内のスライダを動かすことができ、スライダの位置に応じた値を得ることができます。

|  |
| --- |
| int value = 128;  createTrackbar("Parameter", "ShowImage", &value, 255); |

第1引数の文字列 ("Parameter") は、トラックバーを識別するためのものであり、トラックバーの左に表示されます。第2引数の文字列 ("ShowImage") は、ウインドウを指定します。第3引数のポインタで指された変数 (value) の値が、スライダの動きに応じて変化します。第4引数が、最大値 (255) を示します。最小値は 0 固定です。

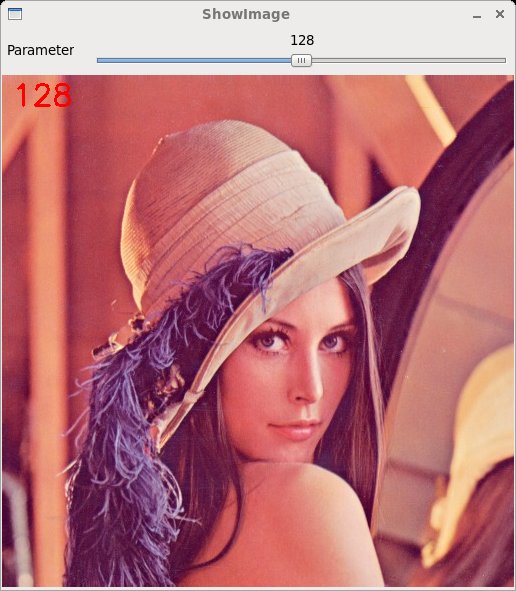


図5.3 トラックバー付きウインドウ

getTrackbarPos() 関数によって、指定トラックバーの値を読み出すことができます。上記の value のような変数が読み出せないスコープから、指定トラックバーの値を読み出すために使用できます。

|  |
| --- |
| now\_value = getTrackbarPos("Parameter", "ShowImage"); |

さらなる詳細(各引数や省略している引数など)や、他のユーザインタフェースの関数については、参考文献等を参照してください。

## 5.2 メディア入出力

メディア入出力について、いくつか使用例を挙げます。

* 静止画ファイルを読み込む

imread() 関数によって、静止画ファイルを読み込むことができます。BMP, JPEG, PNG, TIFF などの様々な画像フォーマットが扱えます。(システムによって多少異なる)

ファイル名の拡張子ではなく、ファイルの内容によって、画像フォーマットが決まります。

"sample.png" ファイルを、グレースケール画像として、src\_img に読み込みます。

|  |
| --- |
| src\_img = imread("sample.png", 0); |

"sample.jpg" ファイルを、BGRカラー画像として、src2\_img に読み込みます。

|  |
| --- |
| src2\_img = imread("sample.jpg", 1); |

* 静止画をファイルへ書き込む

imwrite() 関数によって、静止画ファイルとして書き込むことができます。BMP, JPEG, PNG, TIFF などの様々な画像フォーマットが扱えます。(システムによって多少異なる)

ファイル名の拡張子によって、画像フォーマットが決まります。

画像データを "result.jpg" というファイルに書き込みます。

|  |
| --- |
| imwrite("result.jpg", dst\_img); |

* カメラから画像を読み込む

VideoCapture クラスによって、カメラから画像を読み込むことができます。

0番目のカメラデバイスをオープンし、オープンが成功したかどうかを確認します。

|  |
| --- |
| VideoCapture cap(0);  if (!cap.isOpened()) {  return -1;  } |

または、以下のような手順でも、同様です。

|  |
| --- |
| VideoCapture cap;  if (!cap.open(0)) {  return -1;  } |

約15フレーム/秒のタイミングで、カメラ(cap)から1フレーム読み込み、そのままウインドウに表示します。'q'キーが押されるまで、表示し続けます。

|  |
| --- |
| for (;;) {  Mat img;  cap >> img;  imshow("Output", img);  if (waitKey(66) == 'q') break;  } |

imgに対して、なにか操作を行なうことで、リアルタイムに画像処理ができます。

* 動画ファイルを読み込む

VideoCapture クラスによって、動画ファイルから画像を読み込むことができます。MPEG-4.2, DVIX, FLV1, MothonJPEG などの様々な動画フォーマットが扱えます。(システムによって異なる)

動画ファイルをオープンし、オープンが成功したかどうかを確認します。

|  |
| --- |
| VideoCapture cap("sample.avi");  if (!cap.isOpened()) {  return -1;  } |

または、以下のような手順でも、同様です。

|  |
| --- |
| VideoCapture cap;  if (!cap.open("sample.avi")) {  return -1;  } |

動画ファイルのフレームレートを取得します。

|  |
| --- |
| double fps = cap.get(CV\_CAP\_PROP\_FPS); |

だいたいフレームレートに合わせて、動画ファイル(cap)から1フレーム読み込み、そのままウインドウに表示します。ファイルの最後になるか、'q'キーが押されるまで、表示し続けます。

|  |
| --- |
| for (;;) {  Mat img;  cap >> img;  if (img.empty()) break;  imshow("Output", img);  if (waitKey(1000/fps) == 'q') break;  } |

imgに対して、なにか操作を行なうことで、リアルタイムに画像処理ができます。

* 動画をファイルへ書き込む

VideoWriter クラスによって、ファイルに動画を書き込むことができます。MPEG-4.2, DVIX, FLV1, MothonJPEG などの様々な動画フォーマットが扱えます。(システムによって

異なる)

書き込む動画ファイルを、オープンします。ここでは、ファイル名に"result.avi"、コーデックにMPEG-4.2、フレームレートに15フレーム/秒、サイズに(320,240)を指定しています。

|  |
| --- |
| VideoWriter rec("result.avi",  CV\_FOURCC('M','P','4','2'), 15, Size(320,240)); |

オープンが成功したかどうかを確認します。

|  |
| --- |
| if (!rec.isOpened()) {  return -1;  } |

または、以下のような手順でも、同様です。

|  |
| --- |
| VideoWriter rec;  if (!rec.open("result.avi", CV\_FOURCC('M','P','4','2'), 15, Size(320,240))) {  return -1;  } |

1フレームずつ、動画ファイル(rec)に書き込んでいきます。

|  |
| --- |
| for (;;) {  Mat img;  /\* img画像を作成 \*/  rec << img;  if (waitKey(1) == 'q') break;  } |

さらなる詳細(各引数や省略している引数など)や、他のメディア入出力の関数については、参考文献等を参照してください。

# 6. オブジェクト検出と機械学習

オブジェクト検出の機能は、主にobjdetectモジュールによって提供され、機械学習の機能は、主にmlモジュールによって提供されています。

## 6.1 オブジェクト検出

オブジェクト検出の方法として、カスケード型分類器を使ったものとLatent SVMを使ったものを紹介します。

* カスケード型分類器とは

オブジェクト検出する単純な方法では、画像内を1ピクセルずらしながら、全てのピクセルについて合致するかどうか調べ、さらに対象のオブジェクトがどういう大きさで存在するかもわからないので、サイズを変更したものについても合致するかどうか調べる必要があります。このような方法ではかなり時間がかかってしまいます。

分類器というのは、検出対象のオブジェクトが含まれている可能性があるかないかを分類する機能を持ちます。カスケード型分類器は、複数の分類器を一列につなぎ、可能性のないものを排除していきます。簡単な(計算コストの低い)分類器から順に適用することによって、検出にかかる時間を短くします。

* カスケード型分類器によるオブジェクト検出の手順

分類器をファイルから読み込みます。ここでは、OpenCVに用意されている、顔検出用に学習されたものを使用します。

|  |
| --- |
| std::string cascadeName =  "/usr/local/share/OpenCV/haarcascades/"  "haarcascade\_frontalface\_alt.xml";  CascadeClassifier cascade;  if (!cascade.load(cascadeName)) {  return -1;  } |

検出元の画像をグレースケールに変換します。

|  |
| --- |
| cvtColor(src\_img, dst\_img, CV\_BGR2GRAY); |

CascadeClassifier::detectMultiScale() によって、顔検出を行います。

|  |
| --- |
| std::vector<Rect> objs;  cascade.detectMultiScale(dst\_img, objs); |

ここで objs に、検出された顔領域を囲む矩形が複数、格納されます。以下のコードで、各矩形を描画します。

|  |
| --- |
| dst2\_img = src\_img.clone();  std::vector<Rect>::iterator it = objs.begin();  for (; it != objs.end(); it++) {  rectangle(dst2\_img, \*it, Scalar(0,0,255), 2);  } |

図6.1 顔検出

(左から src\_img, dst\_img, dst2\_img)

別のオブジェクトを認識したい場合には、そのオブジェクトに対応した分類器を用意して、読み込めば認識できます。

* Latent SVM

Latent SVM は、SVM (Support Vector Machine)の一種です。

(以下Wikipediaより抜粋)

**サポートベクターマシン**([英](http://ja.wikipedia.org/wiki/%E8%8B%B1%E8%AA%9E): Support vector machine、**SVM**)は、[教師あり学習](http://ja.wikipedia.org/wiki/%E6%95%99%E5%B8%AB%E3%81%82%E3%82%8A%E5%AD%A6%E7%BF%92)を用いる[識別](http://ja.wikipedia.org/wiki/%E7%B5%B1%E8%A8%88%E7%9A%84%E8%AD%98%E5%88%A5)手法の一つである。[パターン認識](http://ja.wikipedia.org/wiki/%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3%E8%AA%8D%E8%AD%98)や[回帰分析](http://ja.wikipedia.org/wiki/%E5%9B%9E%E5%B8%B0%E5%88%86%E6%9E%90)へ適用できる。

サポートベクターマシンは、現在知られている多くの手法の中で一番認識性能が優れた[学習](http://ja.wikipedia.org/wiki/%E5%AD%A6%E7%BF%92)モデルの一つである。サポートベクターマシンがすぐれた認識性能を発揮することができる理由は、未学習データに対して高い識別性能を得るための工夫があるためである。

* Latent SVMによるオブジェクト検出の手順

指定ファイルから複数のモデルを読み込み、それぞれのクラス名を読み出しておきます。

|  |
| --- |
| std::vector<std::string> models;  models.push\_back("../model/person.xml");  models.push\_back("../model/car.xml");  LatentSvmDetector detector;  if (!detector.load(models)) {  return -1;  }  std::vector<std::string> name = detector.getClassNames(); |

ここでは、人と車のモデルを読み込んでいます。学習済みのモデルとして、下記ページにあるのものを使用しました。

<https://code.ros.org/trac/opencv/browser/trunk/opencv_extra/testdata/cv/latentsvmdetector/models_VOC2007/>

LatentSvmDetector::detect() によって、オブジェクト検出を行います。

|  |
| --- |
| std::vector<LatentSvmDetector::ObjectDetection> detections;  detector.detect(src\_img, detections); |

ここで detections に、検出されたオブジェクトの情報が複数、格納されます。検出されたオブジェクトの情報は、score, rect, classID を要素に持つLatentSvmDetector::ObjectDetection 構造体です。score は検出の信頼度、rect は検出されたオブジェクトを囲む矩形、classID はどのモデルとして検出されたかを示します。

以下のコードでは、scoreが負の数の場合は無視し、rectの矩形を描画し、その矩形の左上にクラス名を描画します。

|  |
| --- |
| dst\_img = src\_img.clone();  std::vector<LatentSvmDetector::ObjectDetection>::iterator it = detections.begin();  for (; it != detections.end(); it++) {  if (it->score < 0) continue;  rectangle(dst\_img, it->rect, Scalar(0, 0, 255), 2);  Point pt(it->rect.x + 5, it->rect.y + 15);  putText(dst\_img, name[it->classID], pt,  FONT\_HERSHEY\_SIMPLEX, 0.5, Scalar(255, 255, 255), 2);  } |

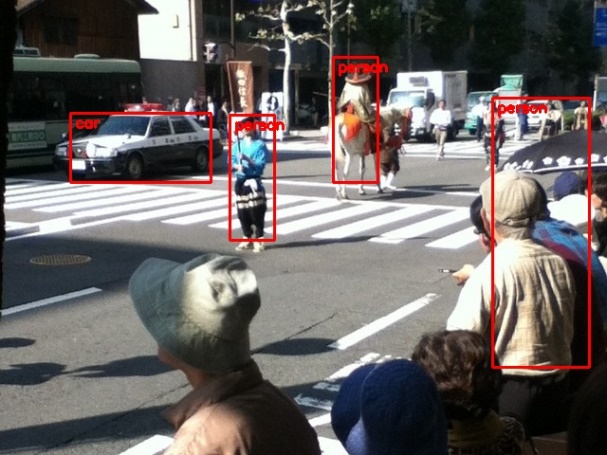
 

図6.2 人と車の検出

(左から src\_img, dst\_img)

さらなる詳細(各引数や省略している引数など)や、他のオブジェクト検出用の関数については、参考文献等を参照してください。

## 6.2. 機械学習

mlモジュールによって作成された学習用コマンドが用意されており、特にプログラムを作成しなくとも、機械学習をすることができます。

ここでは、学習用に用意されたコマンドによる学習の[手順](http://docs.opencv.org/doc/user_guide/ug_traincascade.html)を示します。

* 検索対象

ここでは、検出対象をOpenCVのロゴとします。

<http://opencv.org/wp-content/themes/opencv/images/logo.png>

上記ファイルを取得し、ImageMagickのコマンドconvertを使用して、ロゴマークだけ切り出します。

|  |
| --- |
| convert -chop 10000x10000+82+76 logo.png OpenCV-logo.png |



図6.3 検出対象の画像

* 学習に必要なもの

学習のためには、ポジティブ画像とネガティブ画像が必要です。ポジティブ画像というのは、検出対象を含む画像です。ネガティブ画像というのは、検出対象を含まない画像です。認識率を高めるために、これらの画像を、多く用意しなければなりません。枚数が少なく学習が不十分であれば、認識率が低くなります。しかし枚数が多くなれば、学習に時間がかかるようになります。

ここでは、ポジティブ画像を4000枚、ネガティブ画像を3060枚用意しました。この枚数で学習させると、

CPU Intel Core-i7 3.40GHz

メモリ 16GB

OS CentOS 6.5 (64bit)

を使用して、25時間程度かかりました。

* ネガティブ画像の準備

検出対象を含まない画像として、下記ページにある画像を使用します。

<http://www.vision.caltech.edu/Image_Datasets/Caltech101/Caltech101.html>

101と書いてありますが、実際には102のカテゴリの画像が含まれています。それぞれのカテゴリから30枚ずつ使用することで、ネガティブ画像を3060枚用意します。

ネガティブ画像の指定には、各ファイルパスを1行ずつ並べた以下のようなファイルを使用します。

|  |
| --- |
| ...略...  101\_ObjectCategories/accordion/image\_0028.jpg  101\_ObjectCategories/accordion/image\_0029.jpg  101\_ObjectCategories/accordion/image\_0030.jpg  101\_ObjectCategories/airplanes/image\_0001.jpg  101\_ObjectCategories/airplanes/image\_0002.jpg  ...略... |

Linux コマンドラインで以下のように実行すれば、3060枚のネガティブ画像を指定するための bg.txt というファイルが生成できます。

|  |
| --- |
| ls -1 101\_ObjectCategories/\*/image\_00{[0-2]?,30}.jpg > bg.txt |

* ポジティブ画像の生成

ポジティブ画像をたくさん用意するのはなかなか大変です。

ここでは、opencv\_createsamples コマンドを使用します。このコマンドで、検出対象の画像を機械的に変形(回転させたり、明るさを変えたり、任意の背景の上に重ねたり)して、適切な枚数のポジティブ画像を生成することができます。

opencv\_createsamplesで、ポジティブ画像を生成するには、背景画像も必要です。背景画像には、ネガティブ画像が利用できます。先程作成しておいたbg.txtをそのまま使用します。

以下のように実行すると、OpenCV-logo.vec が生成されます。これをポジティブ画像として使用します。

|  |
| --- |
| opencv\_createsamples -vec OpenCV-logo.vec -img OpenCV-logo.png \  –bg –bg.txt -num 4000 |

* 学習

学習用のコマンドとして、opencv\_haartraining とopencv\_traincascade の2つのコマンドがあります。opencv\_haartraining は古いコマンドで、Haar-like特徴量のみ対応しています。opencv\_traincascadeはHaar-like特徴量だけでなく、LBP特徴量も対応しています。opencv\_traincascade は新しく、OpenCVのVer2以降のC++言語インタフェースで書かれています。ここでは、新しいopencv\_traincascade を使用します。

用意したポジティブ画像4000枚とネガティブ画像3060枚を用いて、以下のように、学習を開始します。

|  |
| --- |
| mkdir OpenCV-logo  opencv\_traincascade -data OpenCV-logo -vec OpenCV-logo.vec \  -numPos 3000 -numNeg 3060 -bg bg.txt -featureType HAAR -mode ALL |

(注) -numPos 1000 としているのは、opencv\_traincascade では、vecファイル内の画像枚数の8割～9割程度しか使用できないため。

指定するパラメータによりますが、学習には、数日かかることもあります。マシンのトラブル等で学習コマンドが停止させられてもよいように、-data で指定したディレクトリ内に、途中経過を置いています。マシンが復帰してから同じコマンドを実行すると、続きから学習ができます。

学習結果は、-date で指定したディレクトリ内の cascade.xml です。このファイルをカスケード分類器として読み込んで、オブジェクト検出できます。

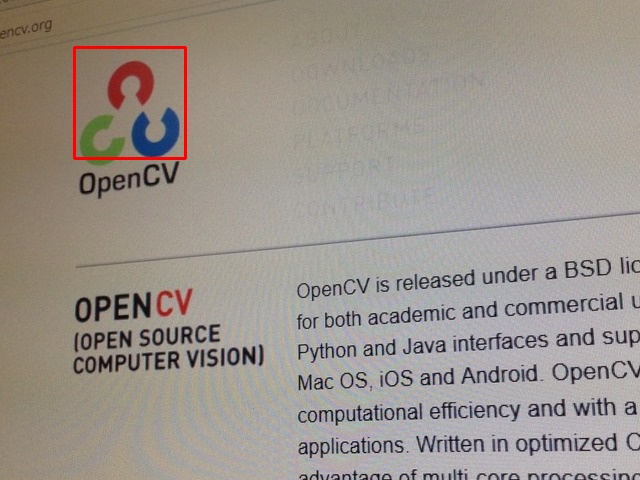


図6.4 学習結果を用いてオブジェクト検出

さらなる詳細(各パラメータや省略しているパラメータなど)については、参考文献等を参照してください。

# 7. 付録

## 7.1 OpenCVのインストール

ここでは、CentOS 6.5 64bit に OpenCV-2.4.8 をインストールする手順を示します。

ソースコードからビルドするものがありますが、最新バージョンが追加されているかもしれません。その場合は、適切なバージョンに置き変えて適用してください。

sudoコマンドでroot権限が得られる、インターネットに接続していてyumコマンドが使用可能、使用するシェルはbash を仮定しています。適切な作業ディレクトリで実行してください。

* ビルドに必要なツールの導入

OpenCV/ffmpegビルドのために必要なツール群を導入します。(ffmpegは、OpenCVの動画処理のために必要になります)

|  |
| --- |
| sudo yum install autoconf automake gcc gcc-c++ git libtool \  make nasm pkgconfig zlib-devel libgtk2.0-devel |

ビルドには直接関係ありませんが、静止画を扱うために ImageMagick、カメラを扱うために cheese、動画を扱うためにtotem を導入しておきます。

|  |
| --- |
| sudo yum install ImageMagick cheese totem |

この他に、必要に応じて、好みのテキストエディタを導入しておいてください。

* ビルドのための設定

ソースコードからビルドするものは、/usr/local 以下にインストールされます。コマンドサーチパスには、/usr/local/bin を追加しておいてください。古いバージョンの同名コマンドがあるかもしれませんので、それよりも優先されるようにしておいてください。

共有ライブラリは /usr/local/lib 以下にインストールされますので、/etc/ld.so.conf に /usr/local/lib を追加して、以下のコマンドを実行してください。

|  |
| --- |
| sudo ldconfig |

* 最新ffmpeg のビルドとインストール

CentOSでは、ffmpeg が yum で導入できません。また、ffmpeg 1.0 以上が必要ですので、最新のソースコードからビルドします。

ffmpeg をビルドする前に yasm, x264, fdk-aac, opus, ogg, vorbis, vpx をインストールします。

* + yasm (x264 と ffmpeg のためのアセンブラ)

|  |
| --- |
| wget http://www.tortall.net/projects/yasm/releases/yasm-1.2.0.tar.gz  tar xfz yasm-1.2.0.tar.gz  cd yasm-1.2.0  ./configure  make  sudo make install  cd .. |

* + x264 (H.264 video encoder)

|  |
| --- |
| git clone git://git.videolan.org/x264  cd x264  ./configure --enable-shared --enable-pic  make  sudo make install  cd .. |

* + fdk-aac (AAC audio encoder)

|  |
| --- |
| wget http://sourceforge.net/projects/opencore-amr/files/latest/download?source=files  tar xfz fdk-aac-0.1.3.tar.gz  cd fdk-aac-0.1.3  ./configure  make  sudo make install  cd .. |

* + lame (MP3 audio encoder)

|  |
| --- |
| wget http://sourceforge.net/projects/lame/files/latest/download?source=files  tar xfz lame-3.99.5.tar.gz  cd lame-3.99.5  ./configure --enable-nasm --enable-efence --enable-mp3x --enable-mp3rtp  make  sudo make install  cd .. |

* + opus (OPUS audio encoder and decoder)

|  |
| --- |
| wget http://downloads.xiph.org/releases/opus/opus-1.1.tar.gz  tar xfz opus-1.1.tar.gz  cd opus-1.1  ./configure  make  sudo make install  cd .. |

* + ogg, vorbis (OGG bitstream library, VORBIS audio encoder)

|  |
| --- |
| sudo yum install libvorbis-devel |

* + vpx (vp8/vp9 video encoder)

|  |
| --- |
| git clone http://git.chromium.org/webm/libvpx.git  cd libvpx  ./configure --enable-shared --enable-pic --disable-examples  make  sudo make install  cd .. |

これらのインストールの後、以下のように ffmpeg をビルド、インストールします。

|  |
| --- |
| PKG\_CONFIG\_PATH="/usr/local/lib/pkgconfig"; export PKG\_CONFIG\_PATH  git clone git://source.ffmpeg.org/ffmpeg.git  cd ffmpeg  ./configure --enable-shared --enable-gpl --enable-nonfree \  --enable-libfdk\_aac --enable-libmp3lame --enable-libopus \  --enable-libvorbis --enable-libvpx --enable-libx264  make  sudo make install  cd .. |

* cmake のビルドとインストール

OpenCVをビルドするには、cmake バージョン2.8 以上が必要なので、ソースコードからビルドしインストールします。

|  |
| --- |
| wget http://www.cmake.org/files/v2.8/cmake-2.8.12.1.tar.gz  tar xfz cmake-2.8.12.1.tar.gz  cd cmake-2.8.12.1  ./configure  make  sudo make install  cd .. |

* opencv-2.4.8 のビルドとインストール

OpenCVは以下のようにビルドしインストールします。

|  |
| --- |
| wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.8/opencv-2.4.8.zip/download  unzip opencv-2.4.8.zip  cd opencv-2.4.8  cmake .  make  sudo make install  cd .. |

## 7.2 サンプルコード

サンプルコードとそのビルド方法を示します。

* 静止画の顔検出サンプル

カスケード分類器によるオブジェクト検出をします。

|  |
| --- |
| /\* sample1.cpp \*/  #include <iostream>  #include <opencv2/opencv.hpp>  using namespace cv;  #define WIN\_NAME\_INPUT "Input"  #define WIN\_NAME\_OUTPUT "Output"  int  main(int argc, char \*argv[])  {  const char \*input\_file = "sample.jpg";  if (argc > 1) input\_file = argv[1];  namedWindow(WIN\_NAME\_INPUT);  namedWindow(WIN\_NAME\_OUTPUT);  Mat src = imread(input\_file, 1);  if (src.empty()) {  std::cerr << "input file error" << std::endl;  exit(1);  }  imshow(WIN\_NAME\_INPUT, src);  Mat dst, work;  dst = src.clone();  cvtColor(src, work, CV\_BGR2GRAY);  std::string cascadeName =  "/usr/local/share/OpenCV/haarcascades/"  "haarcascade\_frontalface\_alt.xml";  CascadeClassifier cascade;  if (!cascade.load(cascadeName)) {  std::cerr << "cascade file error" << std::endl;  exit(1);  }  std::vector<Rect> objs;  cascade.detectMultiScale(work, objs);  std::vector<Rect>::iterator it = objs.begin();  for (; it != objs.end(); it++) {  rectangle(dst, \*it, Scalar(0,0,255), 2);  }  imshow(WIN\_NAME\_OUTPUT, dst);  imwrite("result.jpg", dst);  waitKey(0);  exit(0);  } |

以下の手順でビルドできます。

|  |
| --- |
| c++ sample1.cpp –o sample1 –lopencv\_objdetect |

objdetectモジュールとの依存関係により、core, imgproc, highgui モジュールもリンクされます。

* 動画のサンプル

カメラ入力または動画ファイルから読み出し、鏡像で表示し、動画ファイルに保存します。

|  |
| --- |
| /\* sample2.cpp \*/  #include <iostream>  #include <opencv2/opencv.hpp>  using namespace cv;  #define WIN\_NAME "ShowVideo"  #define CODEC CV\_FOURCC('M','P','4','2')  int  main(int argc, char \*argv[])  {  VideoCapture cap;  if (argc > 1) {  cap.open(argv[1]);  } else {  cap.open(0);  }  if (!cap.isOpened()) {  std::cerr << "capture error" << std::endl;  exit(1);  }  namedWindow(WIN\_NAME);  Size size(cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH),  cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT));  std::cout << "size=" << size << std::endl;  double fps = cap.get(CV\_CAP\_PROP\_FPS);  std::cout << "fps=" << fps << std::endl;  if (fps <= 0.0) fps = 15.0;  VideoWriter rec("result.avi", CODEC, fps, size);  if (!rec.isOpened()) {  std::cerr << "writer error" << std::endl;  exit(1);  }  Mat image;  for (;;) {  cap >> image;  if (image.empty()) break;  flip(image, image, 1);  imshow(WIN\_NAME, image);  rec << image;  if (waitKey(1000/fps) == 'q') break;  }  exit(0);  } |

以下の手順でビルドできます。

|  |
| --- |
| c++ sample1.cpp –o sample1 –lopencv\_highgui |

highguiモジュールとの依存関係により、core, imgprocモジュールもリンクされます。

## 7.3演習問題

上記サンプルコードを参考にして、本書で紹介した画像処理をいくつか試してみましょう。

* 演習1
  + 静止画ファイルを、白黒画像として読み込み、ネガポジ反転
  + カラー画像として読み込み、白黒画像へ変換し、ネガポジ反転
* 演習2
  + 静止画ファイルで画像の合成、上下反転、左右反転のいずれか
  + 余裕があれば、カメラ画像にも適用
* 演習3
  + 静止画ファイルを読み込み、画像の上に、直線、矩形、円、文字列を描画
  + 余裕があれば、カメラ画像にも適用
* 演習4
  + 静止画ファイルで2値化、幾何学変換、フィルタリングのいずれか
  + 余裕があれば、カメラ画像にも適用
* 演習5
  + 静止画ファイルで輪郭の抽出、外接図形描画のいずれか
  + 余裕があれば、カメラ画像にも適用
* 演習6
  + 静止画ファイルでエッジ検出、直線や円の検出のいずれか
  + 余裕があれば、カメラ画像にも適用
* 演習7
  + 静止画ファイルでオブジェクト検出のいずれか
  + 余裕があれば、カメラ画像にも適用
* 演習8
  + トラックバーを配置し、演習4〜7のいずれかのパラメータを動的に変化

# 8. 参考文献

* OpenCV 2 プログラミンングブック製作チーム 著, OpenCV 2 プログラミンングブック / OpenCV 2.2/2.3対応, マイナビ, 2011

※ OpenCVのバージョンが少し古いのですが、新しいC++言語インタフェースでコードが紹介されています。

* 永田雅人 著, 実践 OpenCV 2.4 映像処理&解析, カットシステム, 2013

※ OpenCV 2.4系列対応とのことですが、古い言語インタフェースでコードが紹介されています。新しいC++言語インタフェースではありませんが、OpenCVの概念を学ぶことができます。

* Adrian Kaehler, Gary Bradski 著, Learning OpenCV / Computer Vision in C++ with the OpenCV Library, O'Reilly Media, 2013

※ この本の日本語訳はまだ出版されていません。同じ表紙でサブタイトルの違う旧版については、日本語に訳された本があります。言語インタフェースは古いです。

* OpenCV サイト

<http://opencv.org/>

<http://docs.opencv.org/> (APIリファレンスやユーザガイドなど)

* OpenCV日本語サイト

<http://opencv.jp/>

# 9. 変更履歴

|  |  |  |
| --- | --- | --- |
| **版数** | **変更内容** | **年月日** |
| 1.0 | 新規作成 | 2014/MAR/2 |