OpenCV 2.0 (3)

～さまざまな画像処理機能の紹介と顔検出～

my recommendations on research & development tools.

末永 剛^{†1}, 小枝正直^{†2}, 上田悦子^{†3}, 怡土順一^{†4}

キーワード: OpenCV, 画像処理, コンピュータビジョン, オープンソースライブラリー, マルチプラットフォーム, プログラミング

1. ま え が き

本シリーズでは、コンピュータビジョンライブラリーであるOpenCV¹⁾を題材としてこれまで2回にわたって解説を進めてきた。第1回では導入としてOpenCVの概要とインストール方法について、第2回では画像メモリの扱いと画像ファイルやカメラからの画像読み込みを取り上げ、プログラミング言語(C, C++, Python)によるソースコード記述の比較について説明してきた。シリーズ最後となる第3回では、まずクラスや関数、処理結果画像を挙げながら、OpenCVで利用可能なコンピュータビジョンの機能について紹介する。また、顔検出機能を例にして、ソースコードを挙げて具体的な処理手順を解説する。

2. 画像処理機能の紹介

OpenCVでは、いくつかの関数を呼び出すことでコンピュータビジョンの機能を簡単に利用することができる。ここではCVモジュールに含まれる関数を中心に、処理結果画像とともに、いくつかのコンピュータビジョンの機能を紹介する。紙面の都合上、個々のアルゴリズムやソースコードを掲載できないが、必要な場合には、OpenCVに付属するサンプルや文献2)～4)などを参照されると良いだろう。

2.1 画像フィルタリングと幾何学変換

画像フィルタリング処理については、平滑化(GaussianBlur, bilateralFilter, medianBlur)、エッジ抽出(Sobel, Laplacian)や任意の線形フィルタ(filter2D)、画像の膨張収縮(dilate, erode)などが利用できる。また画像の幾何学変換では、画像のスケーリング(resize)、アフィン変換(warpAffine)、射影変換

図1 画像修復

図2 ハフ変換を用いた直線検出

(warpPerspective)が利用でき、これら変換行列を対応点の座標から直接得ること(getAffineTransform, getPerspectiveTransform)もできる。

2.2 画像変換とヒストグラム

色変換(cvtColor)や閾値処理(threshold, adaptiveThreshold)、画像修復(inpaint, 図1)、画像領域分割(watershed)などの画像変換のほか、ヒストグラム処理(calcHist, calcBackProject)も用意されている。

2.3 特徴検出

特徴抽出アルゴリズムとしては、Cannyエッジ抽出(Canny)やHarrisのコーナー検出(cornerHarris)、LK法による特徴点追跡(goodFeaturesToTrack)、ハフ変換(HoughCircles, HoughLines, 図2)、MSER特徴抽出(MSERクラス)、SURF特徴抽出(SURFクラス, 図3)などが利用可能である。また、FAST、自己相似特徴、HoG(図4)、LBP、平面物体検出など多くの特徴検出アルゴリズムがOpenCV 2.0からCVAUXモジュールに追加されている。

†1 奈良先端科学技術大学院大学 情報科学研究科

†2 大阪電気通信大学 総合情報学部 メディアコンピュータシステム学科

†3 奈良工業高等専門学校 電子制御工学科

†4 川田工業株式会社

"OpenCV 2.0 (3): introduction of various image processing functions and face detection" by Tsuyoshi Suenaga (Nara Institute of Science and Technology, Graduate School of Information Science, Nara), Masanao Koeda (Osaka Electro-Communication University, Department of Computer Science, Osaka), Etsuko Ueda (Nara National College of Technology, Department of Control Engineering, Nara) and Junichi Ido (Kawada Industries, Inc., Tochigi)

図6 Condensationによる赤色トラッキング

図3 SURF特徴量に基づく物体検出

図5 モーションテンプレートによる動き推定

図4 HoG特徴量に基づく人物検出

図7 点列を包含する矩形検出

2.4 モーション解析と物体追跡

動画の動き推定処理も備えており、オプティカルフロー (calcOpticalFlowPyrLK) やモーションテンプレート (calcMotionGradient, calcGlobalOrientation, 図5), CAMSHIFTやMean-Shiftによる物体追従 (CamShift, meanShift), カルマンフィルタ (KalmanFilterクラス) やパーティクルフィルタ (cvCreateConDensation, cvConDensUpdateByTime, 図6) などの推定器が利用可

能である。

2.5 構造解析, 平面細分化と物体検出

輪郭抽出 (findContours) や点列を包含する矩形検出 (boundingRect, 図7), 2次元や3次元のアフィン行列の推定 (estimateRigidTransform, estimateAffine3D), M推定による直線フィッティング (fitLine) などの構造解析やドローネー分割による平面細分化 (cvCreateSubdivDelaunay2D) が利用可能である。また、学習に基づくカスケード分類器に

図8 カメラキャリブレーション

図9 ステレオキャリブレーションと対応問題
(上段：ステレオカメラの入力画像、下段左：平行化された画像とエピポラ線、下段右：(奥行きを表す) 視差画像)

図10 2次元の画像と3次元の扱い

図14 顔検出⁶⁾

よる物体検出 (CascadeClassifierクラス) やテンプレートマッチング (matchTemplate) などを備える。学習に基づく物体検出については、次章で詳細な使用方法を示す。

2.6 カメラキャリブレーションと3次元再構築

画像 (2次元) から空間 (3次元) を扱うこともできる。図8

のように、スケールが既知のチェスパターンを複数の異なる位置から撮影し、チェスボードのコーナー検出 (findChessboardCorners) に基づき、画像座標とチェスボード上の座標をカメラキャリブレーション関数 (CalibrateCamera2) に与えることで、カメラパラメー

タ(焦点距離やレンズ歪みなど)を推定することができる。求められたパラメータを用いることで、レンズの歪み補正(cvUndistort2, cvInitUndistortRectifyMap)ができるほか、画像上のピクセルを実スケールに換算して利用できるようになる。また、ステレオキャリブレーション機能(cvStereoCalibrate)も用意されており、図9のように、ステレオ画像の対応関係から左右のカメラの位置関係を理解することができ(cvFindStereoCorrespondenceBM)、これにより、例えば図10のように異なる視点で撮影された画像から3次元空間を扱うことができる。

3. 顔 検 出

本章では、顔検出のプログラムを例に挙げ、具体的な処理の手順について説明する。ここで扱う物体検出では、物体の有無を判別するための分類器を学習により生成するステップと、分類器を利用して物体検出を行うステップの2ステップ必要であるが、今回はOpenCVで用意されている

正面顔の分類器を用いて顔検出を行う。この分類器は、Haar-like特徴を用いた特徴抽出とAdaboost学習アルゴリズムにより作成されているが、詳細なアルゴリズムは文献5)を参照されたい。また新たな分類器を学習により生成する方法については文献3)4)でも紹介されている。

図11、図12にC++で記述した顔検出のソースコードを示す。プログラム内での処理内容を以下に示す。

- A. 検出したい物体(正面顔)の分類器の読み込み
- B. 画像ファイルから画像の読み込み
- C. ウィンドウの生成
- D. 顔検出と検出領域の描画
 - D1. 前処理: 画像のグレースケール変換
 - D2. 前処理: 高速化のための画像縮小
 - D3. 前処理: 画像のヒストグラム平均化
 - D4. 物体検出(顔検出)
 - D5. 検出領域に楕円を描画
- E. 画像表示
- F. キー入力待つ

```

1: #include <cv.h>
2: #include <highgui.h>
3: #include <iostream>
4:
5: using namespace std;
6: using namespace cv;
7:
8: void detectAndDraw( Mat& img, CascadeClassifier& cascade,
9:                    double scale);
10:
11: String cascadeName =
12: "haarcascades/haarcascade_frontalface_alt2.xml"; // 分類器
13: String imgFilename = "GroupPhoto.jpg"; // 画像ファイルの指定
14:
15: int main( int argc, const char** argv )
16: {
17:   Mat image;
18:   CascadeClassifier cascade;
19:   double scale = 1.2; // 画像縮小により高速化
20:
21:   // A. 検出したい物体の分類器を読み込む
22:   if( !cascade.load( cascadeName ) ) {
23:     cerr << "Could not load classifier cascade." << endl;
24:     return -1;
25:   }
26:
27:   image = imread( imgFilename, 1 ); // B. 画像ファイルを開く
28:   if( image.empty() ) {
29:     cerr << "Could not read " << imgFilename << endl;
30:     return -1;
31:   }
32:   namedWindow( "result", CV_WINDOW_AUTOSIZE ); // C. ウィンドウ生成
33:   detectAndDraw( image, cascade, scale ); // D. 顔検出と描画
34:   imshow( "result", image ); // E. 画像を表示
35:   waitKey( 0 ); // F. キー入力待つ
36:
37:   return 0;
38: }
```

図11 顔検出のソースコード(1/2)

```

39: void detectAndDraw( Mat& img, CascadeClassifier& cascade,
40:                    double scale)
41: {
42:   vector<Rect> faces;
43:   const static Scalar color = CV_RGB( 255, 255, 255 );
44:   Mat gray, smallImg( saturate_cast<int>( img.rows/scale ),
45:                      saturate_cast<int>( img.cols/scale ),
46:                      CV_8UC1 );
47:
48:   // D1. グレースケールに変換
49:   cvtColor( img, gray, CV_BGR2GRAY );
50:   // D2. 高速化のため、scaleだけ画像を縮小する
51:   resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
52:   // D3. ヒストグラムを平均化する
53:   equalizeHist( smallImg, smallImg );
54:
55:   // D4. 顔検出を行う
56:   cascade.detectMultiScale( smallImg, faces,
57:                             1.1, 2, 0
58:                             | cascade.DO_CANNY_PRUNING
59:                             // | cascade.FIND_BIGGEST_OBJECT
60:                             ,
61:                             Size(30, 30) );
62:
63:   // D5. 検出エリアに円を描画
64:   for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end();
65:        ++r ) {
66:     Point center;
67:     int radius;
68:     center.x = saturate_cast<int>( ( r->x + r->width * 0.5 ) * scale );
69:     center.y = saturate_cast<int>( ( r->y + r->height * 0.5 ) * scale );
70:     radius = saturate_cast<int>( ( ( r->width + r->height ) * 0.25 *
71:                                     scale );
72:     circle( img, center, radius, color, 3, 8, 0 );
73:   }
```

図12 顔検出のソースコード(2/2)

```
void detectMultiScale( const Mat& image, vector<Rect>& objects,
                      double scaleFactor=1.1, int minNeighbors=3,
                      int flags=0, Size minSize=Size());
```

引数 image 入力画像
 objects 検出した物体の矩形領域
 scaleFactor 複数スケールの検出のために利用されるスケール
 minNeighbors 物体領域が含むべき近傍矩形の最小数
 flags 処理モード (複数選択可能)
 DO_CANNY_PRUNING : Cannyのエッジ検出利用
 FIND_BIGGEST_OBJECT : 最も大きい物体のみ検出
 minSize 検出する物体候補の最小サイズ

図13 関数detectMultiScaleの関数リファレンス

処理B, Cおよび処理E, Fについては第2回でも紹介しているのでここでは省略する。物体検出の主要部分となる処理A, D4では, CascadeClassifierクラスのメンバ関数load, detectMultiScaleにより物体検出が実現される。関数loadでは, 分類器ファイルへのパスを指定しており, 本プログラムでは正面顔の分類器を利用している。OpenCVでは他に目や全身などの分類器が用意されており, このファイルを切替えることで各物体を検出することが可能である。今回の分類器で用いられるHaar-like特徴では, 輝度値の差を利用しているため, 処理D1でグレースケール変換が必要となる。また処理D2では高速化のために画像を縮小しており, 縮小率は20行目の変数scaleによって指定する。処理D3では, 画像ヒストグラムを均一化している。次に実際の検出処理となる処理D4の関数detectMultiScaleについて, 主要な引数を図13に示す。この関数では, 複数サイズの物体を検出可能であり, 第3引数scaleFactorに応じたスケーリングにより任意サイズの物体検出が行われる。また, 第5引数flagsには複数のオプションを指定可能である。例えば, 検出対象が一つだけであるとわかっている場合, flagにFIND_BIGGEST_OBJECTを指定することで, 処理速度を向上させることができ, 一番大きな物体のみが検出される。次に, 処理D5では, 検出された矩形領域に円を描画している。saturate_cast<int>()では, 小数点以下を四捨五入し, int型に丸めている。これで一連の処理は終わりである。

本プログラムを使用する場合には, OpenCVのインストールフォルダ以下にある分類器フォルダ"data\haarcascades"と画像ファイルを, 事前にプロジェクトフォルダにコピーし, 13行目で画像ファイル名を指定する。プログラム実行結果を図14(68ページに掲載)に示す。カメラ画像に対する顔検出も第2回の内容を参考にすれば容易であろう。

4. む す び

本シリーズでは3回にわたって画像処理ライブラリーOpenCV 2.0について紹介してきた。最終回となる第3回では, OpenCVに実装されている画像処理について, いくつかの処理結果画像とともに紹介し, 顔検出を例にソースコードの解説を行った。OpenMPやIPPライブラリーを利用できる環境では, インストール時のcmakeで有効化することで, ソース互換のまま高速化することも可能である。2010年4月には2.0系の安定バージョン2.1がリリースされており, Grabcut画像セグメンテーションアルゴリズムやSGBM(Semi-Global Block Matching)による準大域的ステレオマッチングアルゴリズムなど, 新機能も追加されている。OpenCVにより, 手軽に最新の画像処理アルゴリズムを導入するだけでなく, このライブラリーを新しい画像処理アルゴリズム開発のためのツールとして利用されると良いだろう。

(2010年6月30日受付)

〔文 献〕

- 1) OpenCV, <http://opencv.willowgarage.com/>
- 2) opencv.jp, <http://opencv.jp/>
- 3) OpenCVプログラミングブック第2版, 毎日コミュニケーションズ, ISBN978-4839931599
- 4) G. Bradski, A. Kaehler著, 松田晃一訳: “詳解OpenCV—コンピュータビジョンライブラリーを使った画像処理・認識”, O'Reilly Japan
- 5) Paul Viola and Michael J. Jones: "Rapid Object Detection Using a Boosted Cascade of Simple Features", IEEE CVPR2001 (2001)
- 6) Group photo 1 (<http://www.flickr.com/photos/andypowell1/211935378/>) <http://www.flickr.com/photos/andypowell1/> / CC BY 2.0

すなが つよし 2008年, 奈良先端科学技術大学院大学情報科学研究科博士後期課程修了。同年, 同大学情報科学研究科研究員。3次元ディスプレイや知能ロボットの機能モジュール化などの研究に従事。博士(工学)。

こまき まさなお 2005年, 奈良先端科学技術大学院大学情報科学研究科博士後期課程修了。同年, 立命館大学情報理工学部知能情報学科講師。2008年, 大阪電気通信大学総合情報学部メディアコンピュータシステム学科准教授。ヒューマノイドや力覚デバイスなどの研究に従事。博士(工学)。

うえだ えつこ 2003年, 奈良先端科学技術大学院大学情報科学研究科博士後期課程修了。2004年, 同大学助手。2007年, 奈良産業大学情報学部准教授。2010年, 奈良工業高等専門学校電子制御工学科教授。ビジョンベースインタフェースなどの研究に従事。博士(工学)。

い と じゅんいち 2009年, 奈良先端科学技術大学院大学研究員。同年, 産業技術総合研究所特別研究員。2010年, 川田工業(株)に勤務。ヒューマンロボットインタラクション, ロボットビジョンに関する研究に従事。博士(工学)。