コンピュータビジョンライブラリー OpenCV3.0

my recommendations on research & development tools.

浦西友樹†

キーワード：OpenCV, C++, Python, 画像処理, コンピュータビジョン

1. まえがき

OpenCV (Open Source Computer Vision Library)¹⁾ はBSD (Berkeley Software Distribution) ライセンスに基づいて公開されている(図1), コンピュータビジョンのためのオープンソースのライブラリーであり, 上述の分野においてはデファクトスタンダードの地位を築いている。OpenCVはバージョン3.4まで公開されており(2018年6月執筆現在), 多くのプラットフォームや言語をサポートしている。本稿ではOpenCVの概要や, 最新のメジャーバージョンであるOpenCV3.0系の特徴について述べた後, 主要なオペレーティングシステムにおけるOpenCVのインストール方法やサンプルコードについて解説する。

2. OpenCVの概要

OpenCVプロジェクトの開始は, 公式には1999年とされている。最初のアルファ版は, コンピュータビジョンとパターン認識の主要な会議であるIEEE Conference on Computer Vision and Pattern Recognition (CVPR2000)において公開された。その後, ベータ版の公開を経て, 2006年に正式版となるバージョン1.0が公開された。OpenCVは当初C言語向けのライブラリーであったが, 2009年10月に公開されたバージョン2.0ではC++言語を新たにサポートした。C++言語インタフェースによりモダンな記述が可能となり, OpenCVの利便性が大幅に向上した。続いて2010年にリリースされたバージョン2.1ではPythonをサポートし, 翌年のバージョン2.3ではGPUでの処理に正式対応するなど, バージョンが上がるごとに着実に利用範囲を拡大してきている。累計ダウンロードは1,700万回を超えるなど, コンピュータビジョン向けライブラリーとしてデファクトスタンダードの地位を築いている。

OpenCVはWindows, Linux, AndroidおよびmacOSをサポートし, C++, Python, JavaおよびMATLAB向け

図1 OpenCVのウェブサイト¹⁾

のインタフェースを有している。さらに並列化のため, NVIDIA CUDAやOpenCLを利用するインタフェースも活発に開発されている状況にあり, インストール時に利用を選択できる。OpenCVにはコンピュータビジョンに必要な多くの機能が用意されており, ユーザは手軽にこれらの機能を利用できる。提供されている機能の一部を以下に列挙する。

- ・コア機能(データ型の定義, 行列演算, 例外処理など)
- ・入出力機能, インタフェース
- ・基本的な画像処理
- ・幾何変換(図2)
- ・フィッティング
- ・オプティカルフロー, トラッキング
- ・領域分割
- ・カメラキャリブレーション
- ・特徴点検出(図3)
- ・深度推定, ステレオ視
- ・物体認識
- ・機械学習
- ・コンピュータビジョンフォトグラフィ

† 大阪大学

"OpenCV: Open Source Computer Vision Library" by Yuki Uranishi (Osaka University, Osaka)

図2 幾何変換

図3 特徴点検出とマッチング
(ORB: Oriented FAST and Rotated BRIEF)

3. OpenCV3.0

2015年6月にCVPR2015に合わせて公開されたOpenCV3.0は、APIの構造の整理やパフォーマンスの向上を目的としている。OpenCV3.0の特徴は多岐にわたるが、本稿ではその一部を以下で紹介する。

3.1 C言語サポートの段階的な終了

OpenCV3.0においてはC言語インタフェースから段階的に脱却し、C++言語へユーザを全面移行させることを目指しており、C言語インタフェースは部分的に削除されているとともに、今後新たなインタフェースは実装されない。

3.2 ライブラリー構造の刷新

OpenCVのバージョン2系では、バージョンアップが進む中でライブラリーの大型化・多機能化が著しく、結果として「枯れた」コアな画像処理の部分に変更がないにも関わらず、実験的なモジュールの追加やバグフィックスのためにライブラリー全体のバージョンが上がる、という点が問題となっていた。OpenCV3.0においては、図に示すよう

にライブラリーを大きくコア部分とcontributions部分に分けており、実験的なモジュールはcontributions部分に含まれる。Windows向けのコンパイル済みパッケージなどを用いると、基本的にはコア部分のみがインストールされる。contributions部分はGitHub上で公開されており²⁾、手元の環境でコンパイルして利用する。

3.3 T-API (Transparent API)

OpenCV3.0において特に注目すべき点の一つが、T-API (Transparent API)の実装である。T-APIとは、プログラムを実行する環境に合わせてバックエンドでの動作を切り替えるテクノロジーであり、ユーザはプログラムを動作させる際に実行環境の違いを意識しなくてよい。

過去のバージョン2系においては、例えば、CannyフィルタをCPUで動作させるときは`cv::Canny()`、OpenCLによる最適化を適用する際は`ocl::Canny()`、そしてNVIDIA CUDAを用いる際は`gpu::Canny()`を用いるように、ターゲットとなる環境に合わせてソースプログラムを書き換える必要があった。T-APIテクノロジーでは、プログラムの実行時にOpenCLが利用可能であればOpenCLを使用し、利用できなければCPUで動作させる。この選択は実行時に行われるため、実行環境ごとに再コンパイルする必要がない。

4. OpenCVのインストール

OpenCVのインストールはプラットフォームごとに方法が確立されており、本稿では、Microsoft Windows 10、macOS XおよびUbuntu Linuxへのインストール方法の一部を紹介する。なお、先述の通り、OpenCVにはcontributionsパッケージやNVIDIA CUDAを利用するための応用的なインストール方法が多数存在するが、本稿では紙面の都合上説明を割愛し、基本的な方法のみを記載することとした。その他のインストール方法については、文献³⁾やWeb版記事などを適宜参照されたい。

4.1 Windows & Visual Studioへのインストール

Microsoft Windowsと最新のVisual Studioの組合せにおいては、公式ウェブサイトで提供されているビルド済みパッケージを用いるのが簡単である。なお、下記はVisual Studio 2017の例を示している。

- ・インストーラをダウンロード
- ・インストーラを実行し、生成されたディレクトリを任意の位置に配置（以下、このディレクトリの位置を\$OPENCV\$と記述）
- ・(\$OPENCV\$/build/x64/vs15/bin/)を環境変数PATHに追加

その後、Visual Studioにおけるプロジェクトに下記の値を設定する。

- ・追加のインクルードディレクトリに、OpenCVのヘッダファイルが格納されているファイルパス

(\$OPENCV\$/build/include/)を設定

- ・追加のライブラリディレクトリに、OpenCVのライブラリファイルが格納されているファイルパス (\$OPENCV\$/build/x64/vs15/lib/)を設定

4.2 Python環境へのインストール

Python環境へのインストールは、パッケージ管理ツール pipを用いる。Gohlkeらのウェブサイト⁴⁾などから入手した whl ファイル (ここでは opencv_python_3.4.1_cp37_cp37m_win_amd64.whl を例とする) を pip 経由でインストールするには、下記のようなコマンドを実行する。

```
$ pip install opencv_python_3.4.1_cp37_cp37m_win_amd64.whl
```

4.3 macOSへのインストール

Homebrew⁵⁾ がインストールされている環境であれば、ターミナルで下記のコマンドを実行することにより、OpenCVがインストールされる。環境によっては時間がかかるので注意されたい。

```
$ brew install opencv
```

インストールされたバージョンや依存環境などの詳細は、下記のコマンドを実行することで得られる。

```
$ brew info opencv
```

ターミナルからコンパイルする際には、下記のようなコマンドを実行する。なお、yourbin および yourcode.cpp は適宜変更される。

```
$ clang++ -o yourbin `pkg-config --cflags opencv` `pkg-config --libs opencv` yourcode.cpp
```

5. OpenCVの使用例

これまでに述べたように、OpenCVにはさまざまなコンピュータビジョンに関するモジュールが用意されている。本稿ではその一例として、カメラ画像を入力として、Farnebackの手法⁶⁾によりオプティカルフローを計算するサンプルプログラムを示す。

C++によるサンプルプログラムは下記の通りである。

```
#include <opencv2/opencv.hpp>

int main(int argc, char** argv)
{
    cv::Mat frame1, frame2, prvs, next, hsv, rgb,
    flow, mag, ang;
    std::vector<cv::Mat> flow_split, hsv_split;

    //カメラキャプチャの宣言
    cv::VideoCapture capture(0);

    //事前に1フレームだけ画像を読み込みグレースケール化
    do {
        capture > frame1;
    } while (frame1.empty());
```

```
cv::cvtColor(frame1, prvs, cv::COLOR_BGR2GRAY);
hsv = frame1.clone();
```

```
for(;;) {
    //現在フレームを取得してグレースケール化
    capture > frame2;
    cv::cvtColor(frame2, next, cv::COLOR_BGR2GRAY);
```

```
// Farnebackの手法でオプティカルフローを計算
cv::calcOpticalFlowFarneback(prvs, next, flow,
0.5, 3, 15, 3, 5, 1.2, 0);
```

```
//フローの向きを色相に対応させる
cv::split(flow, flow_split);
cv::cartToPolar(flow_split[0], flow_split[1],
mag, ang);
cv::normalize(mag, mag, 0, 255, cv::NORM_MINMAX);
```

```
for(int y = 0; y < hsv.rows; y++) {
    cv::Vec3b* ptr_hsv = hsv.ptr<cv::Vec3b>(y);
    float* ptr_ang = ang.ptr<float>(y);
    float* ptr_mag = mag.ptr<float>(y);
    for(int x = 0; x < hsv.cols; x++) {
        ptr_hsv[x][0] = (ptr_ang[x] * 180) / (2 * CV_PI);
        ptr_hsv[x][1] = 255;
        ptr_hsv[x][2] = (uchar)ptr_mag[x];
    }
}
```

```
cv::cvtColor(hsv, rgb, cv::COLOR_HSV2BGR);
```

```
//表示
cv::imshow("frame", frame2);
cv::imshow("Calculated Opt Flow", rgb);
```

```
// escが入力されたら終了
if(cv::waitKey(10) == 27) {
    break;
}
```

```
//現在フレームの画像を次の基準画像にする
next.copyTo(prvs);
}
```

```
return 0;
}
```

同様に、Pythonでオプティカルフローを検出するサンプルプログラムは下記の通りである。

```
import numpy as np
import cv2

#カメラキャプチャの宣言
cap = cv2.VideoCapture(0)
#事前に1フレームだけ画像を読み込みグレースケール化
_, frame1 = cap.read()
prvs = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[..., 1] = 255

while(1):
#現在フレームを取得してグレースケール化
_, frame2 = cap.read()
next = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
# Farnebackの手法でオプティカルフローを計算
flow = cv2.calcOpticalFlowFarneback(prvs, next, None, 0.5, 3, 15, 3, 5, 1.2, 0)

#フローの向きを色相に対応させる
mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
hsv[..., 0] = ang*180 / np.pi / 2
hsv[..., 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
rgb = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

#表示
cv2.imshow("frame", frame2)
cv2.imshow("Calculated Opt Flow", rgb)

# escが入力されたら終了
k = cv2.waitKey(10)
if k == 27:
break
#現在フレームの画像を次の基準画像にする
prvs = next

cap.release()
cv2.destroyAllWindows()
```

いずれのサンプルプログラムにおいても、図4に示すようなカメラ画像が入力であるとき、図5に示すような実行結果が得られる。カメラ画像内での前フレームとのオプティカルフローの大きさや角度に応じて、オプティカルフローが色付けされている様子が確認できる。

6. むすび

本稿では、オープンソースのコンピュータビジョン向けライブラリーOpenCVについて紹介した。OpenCVは基

図4 フレーム画像

図5 可視化されたオプティカルフロー

本的な画像処理から最新のアルゴリズムに至るまで、数多くの処理を多様なインタフェースで実現でき、バージョンアップも活発に行われている。コンピュータビジョンプログラミングの際は、OpenCVを積極的に利用することを勧める。

(2018年6月15日受付)

〔文 献〕

- 1) OpenCV, <https://opencv.org/>
- 2) GitHub - Repository for OpenCV's extra modules, https://github.com/opencv/opencv_contrib
- 3) 藤本ら：「OpenCV 3 プログラミングブック」，マイナビ（2015）
- 4) Unofficial Windows Binaries for Python Extension Packages, <https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv>
- 5) Homebrew, https://brew.sh/index_ja
- 6) G. Farneback: "Two-Frame Motion Estimation Based on Polynomial Expansion", Proceedings of the 13th Scandinavian Conference on Image Analysis, pp.363-370 (2003)

浦西 友樹 2008年、奈良先端科学技術大学院大学情報科学研究科情報処理学専攻博士後期課程修了。現在、大阪大学サイバーメディアセンター准教授。2011年～2012年、フィンランド・オウル大学客員研究教授。コンピュータビジョン、拡張現実感、ヒューマンインタフェースに関する研究に従事。博士（工学）。