基礎講座 「ユビキタスアプリケーションを支えるツールキット」(全4回)

第2回 OpenCVを利用した手軽な画像処理

関西学院大学 理工学部 河野 恭之

1. はじめに

Phidgetsに引き続いて今回は画像処理ライブラリOpenCVを紹介する。ユビキタスコンピューティングのためのアプリケーションを作る際にUSBカメラなどからの映像入力や映像ファイルを扱いたいということがあるだろう。OpenCVは代表的な画像処理をライブラリ化したものであり、上記のような場合にアプリケーションに画像処理機能を簡単に追加することができる。

OpenCVはIntelが開発した画像処理・コンピュータビジョン用ライブラリで、フリーで公開・配付されている。長らく「ベータ版」という位置づけであったが、2006年10月に正式版としてOpenCV1.0がリリースされ、[1]からWindows版とLinux版が入手可能である。OpenCVをインストールすると正式版のリファレンスマニュアルも同時にインストールされ、Webブラウザから参照可能となる。また、日本語の解説書[2]の制作チームが中心となって、ボランティアベースで日本語訳版リファレンスマニュアルが構築されつつある[3]。このサイトにはOpenCVに備えられた関数群を本格的に使用したサンプルコードとその解説も準備されているのでぜひ参照されたい。

2. 構成と動作環境

OpenCVには500もの関数が含まれており、画像処理を効率よく行うプログラムを書くことができる。OpenCVは次の4つのモジュールから構成されている。

CXCORE　OpenCVで扱うデータ構造(基本構造体)の定義、画像データや基本的なデータの操作関数(画像同士の演算処理や変換処理、コピーなど)が用意されている。

CV　画像処理、コンピュータビジョン、パターン認識に必要な様々な関数が実装されている。
画像処理(Image Processing):エッジ抽出、色変換、ヒストグラムなど基本的な画像処理関数群
構造解析(Structural Analysis):輪郭や矩形領域抽出など構造解析関数群
モーション解析と対象追跡(Motion Analysis and Object Tracking):オプティカルフローやカルマンフィルタなど動作解析・推定、対象追跡のための関数群
パターン認識(Pattern Recognition):物体検出などパターン認識関数群
カメラ校正と3次元再構成(Camera Calibration and 3D Reconstruction):カメラキャリブレーション、姿勢推定、エピポーラ幾何

機械学習　K最近傍やSVMなど、データの分類や回帰、クラスタリングに必要な関数とクラスをまとめたものである。

HighGUI　ウィンドウ作成や画像表示、トラックバーへの対応など簡単なGUI、ファイルやカメラからの画像・ビデオ入出力などの機能が提供される。

配布サイト[1]ではWindows版とLinux版が提供されている。本稿では、Windows環境(Windows2000以降)、Microsoft Visual C++(2005以降)を前提に解説するが、OpenCVはマルチプラットフォームのライブラリとして開発されており、一部の機能を除いてほとんどの環境でソース互換である*1。OpenCVに用意されているシンボルには次のような規則で接頭辞が付与されている。

関数　接頭辞“cv”
構造体・クラス　接頭辞“Cv”
定数・マクロ　接頭辞“CV_”

また、ほとんどの関数名は“cv<操作><操作対象>”という命名規則に従っている*2。更に、関数の引数は“<入力><出力><オプションパラメータ・フラグ>”という順に並んでいる。これらの規則により、慣れると知らない関数でもおおよその仕様を想像できる。また、多くのオプションやフラグにはデフォルト値が与えられており、デフォルトを使用する際はそれらの引数を省略して記述できる。

3. インストールと設定

3.1 OpenCVのインストール

Windows版のOpenCVは配布サイト[1]にインストール用のバイナリが用意されている*3。上記のSorceForgeサイトのopencv-winパッケージから最新のバイナリ(本稿執筆時点ではOpenCV_1.0.exe)をダウンロードして実行し、いくつかの質問に答えればあっけなくインストールされる。デフォルトのインストールフォルダはC:\Program Files\

*1 Windows環境においてもcygwin上での開発も可能である[5]が、Microsoftが提供するVisual C++ Express Edition[6]でもOpenCVベースのアプリケーションがフリーで開発可能となっている。
*2 更に<修飾子>が末尾に付与される関数もある。
*3 配布サイト[1]のopencv-docパッケージにLinux版のインストール方法やサンプルコードを記した資料や、解説プレゼン資料が用意されている。

OpenCVであり、dll等のバイナリ、開発用ライブラリやヘッダファイル、マニュアル類、ソースコード、サンプルプログラムとその実行ファイルなどの一切がインストールフォルダ以下に配置される*4。

3.2 PATHの設定と動作確認

システム変数PATHの設定を見てC:\ProgramFiles\OpenCV\binが追加されていることを確認する。追加されていなければ手動で追加する。この状態でC:\Program Files\OpenCV\samples\cにあるサンプルプログラムが実行できればインストール成功であるのでいくつか実行して動作確認するとよい。例えば、demhist.exeは画像ファイルのヒストグラムを表示するプログラムである。brightnessとcontrastのスライドバーを操作することによりヒストグラムも変化する。facedetectはカメラから入力される動画をリアルタイム処理して、Adaboost[4]によって予め学習した識別器により顔を検出するサンプルプログラムである。DirectShowに対応したカメラ*5をPCに接続して利用する。サンプルディレクトリにあるfacedetect.cmdというバッチファイルをダブルクリックすると、OpenCVに同梱されている予め学習させた辞書を用いる設定でfacedetectが起動される。facedetectを実行するとウィンドウが開いてカメラ入力画像が表示され、顔らしき領域を検出すると赤い枠が重畳される。このサンプルが動作すればOpenCVが正常にインストールされ、接続したカメラからの入力画像を処理できる状態であることが確認できる。

3.3 開発環境の設定

下記の設定をOpenCVインストール後に1回だけ行う。

3.3.1 ヘッダファイルのパスの設定

- Microsoft Visual Studioを立ち上げ、「ツール」－「オプション」メニューを開く。
- 「プロジェクト及びソリューション」－「VC++ディレクトリ」タブを開く。
- 「ディレクトリを表示するプロジェクト(S)」で「インクルードファイル」を選択し、以下を追加。
  - C:\Program Files\OpenCV\cv\include
  - C:\Program Files\OpenCV\cvaux\include
  - C:\Program Files\OpenCV\cxcore\include
  - C:\Program Files\OpenCV\otherlibs\highgui
  - C:\Program Files\OpenCV\otherlibs\cvcam\include

3.3.2 ライブラリファイルのパスの追加

- 前節と同様のメニューで、「ディレクトリを表示するプロジェクト(S)」で「ライブラリファイル」を選択し、以下を追加する。
  - C:\Program Files\OpenCV\lib

4. 基本的な使用方法

4.1 プロジェクトの作成

Microsoft Visual Studioでは「プロジェクト」を単位として作成アプリケーションを管理する。新しいOpenCVアプリケーションを作成したい場合にはまず下記の手順でプロジェクトを作成する。

1. 「ファイル」-「新規作成」-「プロジェクト」メニューを選択。
2. 「プロジェクトの種類」でWin32を選択し、「Win32コンソールアプリケーション」のテンプレートを選択。
3. プロジェクト名を入力(プロジェクト名は最終的に生成されるexeファイルの名前になる)。
4. リンクするライブラリを追加。
   「プロジェクト(P)」→「***のプロパティ(P)」→「構成プロパティ」→「リンカ」→「入力」を選択。「追加の依存ファイル」にリンクが必要なライブラリを追加する(構成タブを「すべての構成」としておくとデバッグ・リリースモードの両方が一度に変更可能)。典型的には下記が追加される。
   - cv.lib
   - cxcore.lib
   - cvcam.lib(カメラを使用しないプログラムでは不要)
   - highgui.lib(画像表示などのGUIを使用しないプログラムでは不要)
5. _tmain()をmain関数としてプログラムを作成。
   ここで、_tmain()でコマンドラインからの引数(argv)の解釈を行う場合、「プロジェクト(P)」→「***のプロパティ(P)」→「構成プロパティ」→「全般」→「文字セット」で「マルチバイト文字セットを使用する」を選択する。

4.2 プログラムの制御構造

図1に、カメラからの入力を1フレームずつキャプチャして逐次処理を行うプログラムのメインルーチンの典型例を示す(各プログラムに特有の処理は隠蔽してある)。このプログラムの_tmain()関数は次の3つのパートに分かれている。

1. 初期化部:キャプチャデバイスの初期化を行うと共に、プログラム特有の初期化処理を行うInitProcess()関数を呼び出す。
2. フレーム処理ループ:キーボードから'q'という文字が入力されるまで、次のフレームの画像をキャプチャし、その画像をプログラム特有の処理を行う関数ProcessFrame()に渡して処理する。すなわち、ProcessFrame()は1フレームキャプチャされる毎に呼び出される。
3. 終了処理部:キャプチャデバイスの解放を行うと共に、プログラム特有の終了処理を行う関数ExitProcess()を呼び出す。

*4 本稿では以降、デフォルトのC:\Program Files\OpenCVをインストールフォルダとして記述するので適宜読み替えていただきたい。

*5 PCショップ等で普通に販売しているUSBカメラは大抵対応している。

以下、本プログラムを例にとり動画像入力の基本的な取り扱い方を解説する。

4.3 ヘッダファイルの指定

図1のプログラムはOpenCVが提供する2つのヘッダファイルを読み込んでいる。OpenCVを用いたプログラムを書くには、OpenCVに収められた関数や各種構造体の宣言が収められているcv.hを必ずincludeする必要がある。また、OpenCVが提供するGUIツール群や、画像の読み込み・保存といった機能を利用するにはhighgui.hを読み込むことになる。

4.4 カメラ／動画ファイルからのキャプチャの準備

OpenCVではカメラや動画ファイルからの取り込みをサポートするcvCapture関数群が用意されており、動画入力をリアルタイム処理して動作するプログラムを簡単に作成できる。いわゆるWebカメラとして販売されている安価なUSBカメラは（大抵の場合）メーカーまたはMicrosoftから提供されるドライバがインストールされ、［コントロールパネル］－［デバイスマネージャ］上で「イメージングデバイス」として見えていれば、下記のデフォルト設定で使用することができる。動画像を扱うプログラムを記述する際には準備のためにCvCapture構造体*6を初期化する必要がある。初期化は下記のように行う。

```
CvCapture* capture;
capture = cvCreateCameraCapture(0);
```

cvCreateCameraCapture()はOpenCV内でのキャプチャデバイスとしてのカメラを初期化する関数で、引数はカメラのインデックスを表している。0を指定した場合は自動で使用可能なカメラを判断する。カメラが複数ある場合はこの引数を変えて指定することになるが、ここではカメラが1台の場合に限定して割愛することとする。なお、接続可能なデバイスがない場合cvCreateCameraCapture()はNULLを返す。

動画（AVI）ファイルからの入力は、初期化においてcvCreateCameraCapture()のかわりにcvCreateFileCapture()を呼び出し

```
CvCapture* capture;
capture = cvCreateFileCapture("ファイル名");
```

とするだけでプログラムの他の部分はそのままで、カメラ入力処理のプログラムを動画ファイルから画像を読み込んで処理するプログラムとすることができる。

4.5 キャプチャ処理

キャプチャデバイス（前項で取得したCvCapture構造体）が有効な間、カメラや動画ファイルからフレーム（1枚の画像）を取り込むことができる。この処理にはcvQueryFrame()を呼び出す。

```
IplImage* image;
image = cvQueryFrame(capture);
```

この関数が呼び出されるとカメラ又はAVIファイルから画像が1枚取り込まれ、IplImage構造体*7へのポインタが返される。ここで返ってくる画像（IplImage構造体）はデバイスの初期化関数（cvCaptureFromCAM()又はcvCaptureFromAVI()）の内部で確保されているので、プログラムの途中で解放してはならない。また、この返り値がNULLだと、AVIファイルが対象の場合ファイルの最後のフレームまで処理された後であることになり、カメラ入力が対象の場合は何らかの原因でカメラと切断されたことになるので終了処理を行う必要がある。

```
#include "stdafx.h"
#include <cv.h>
#include <highgui.h>

int _tmain(int argc, _TCHAR* argv[]) {
  CvCapture *capture = 0;
  IplImage *image =0;

  // キャプチャデバイス（カメラ）の初期化
  if (!(capture=cvCreateCameraCapture(0))) {
     return -1;
  }
  // 初期化処理：バッファ領域確保等
  InitProcess();

  // 'q' が押されるまで繰り返す
  int nCount = 0;
  while (cvWaitKey(10) != 'q') {
    // キャプチャ
    if (!(image=cvQueryFrame(capture)))
      break;
    // 入力フレーム画像の処理
    ProcessFrame(image, nCount);
    nCount++;
  }

  // 終了処理：バッファ領域解放，ウィンドウ消去等
  ExitProcess(image);
  // 画像領域解放（image は解放しなくてよい）
  // カメラデバイス解放
  cvReleaseCapture(&capture)
  return 0;
}
```

図1　典型的なカメラ入力の逐次処理プログラム

*6 カメラやキャプチャカード等の情報を格納する構造体だが、中身を知らなくてもおおむね問題はない。
*7 IplImage構造体については次節で詳説する。

4.6 終了処理

cvCreateCameraCapture()、又はcvCreateFileCapture()で取得したキャプチャデバイスは、デバイス使用終了時（典型的にはプログラム終了時）にcvReleaseCapture()を呼びだして解放する必要がある。

5. 画像データの取り扱い

5.1 画像データ表現と画素アクセス

OpenCVを使用する上で最も基本的なデータ構造が1枚／1フレームの画像を格納する構造体IplImageである。ファイルやカメラの入出力、CVモジュールの利用などはこの構造体（へのポインタ）を介して行う。IplImage構造体の主要メンバは下記である。

```
typedef struct _IplImage {
    int nChannels; // 画像のチャンネル数
    // OpenCV のほとんどの関数が、1, 2, 3
    //および4チャンネルをサポートする
    int depth; // ピクセルの色深度
    int width; // 画像の幅
    int height; // 画像の高さ
    int imageSize; // 画像サイズ
    char *imageData; // 画像データ本体
    int widthStep; // 1行のバイト数
       //あとは省略…
} IplImage;
```

これらのメンバを用いて画像の各画素に直接アクセスできる。画像データはimageDataというメンバ（一次元配列）に収められており、座標（x, y）の画素があるメモリの先頭アドレスは下記の式で計算できる。

```
typedef unsigned char BYTE;
inline BYTE* GetPixelPtr(IplImage *image,
                         int x, int y) {
  return (BYTE*)(image->imageData
    + y * image->widthStep
    + x * image->nChannels);
}
```

メンバwidthStepには1ライン分のバイト数が格納されている。メンバnChannelsは画像の色チャネル数であり、グレイスケール画像（濃淡画像）や白黒二値画像ではnChannelsの値が1、RGBカラー画像では3である。メンバdepthがピクセルの表現形式であるが、たいていの可視画像はIPL_DEPTH_8U（8bit符号なし整数）を用いる*8。このため濃淡画像や白黒二値画像では1画素あたり1バイト、RGBカラー画像は1画素あたり3バイトが使用されている。OpenCVでは通常、RGBカラー画像は上記で計算されたアドレスから青色成分（B）、緑色成分（G）、赤色成分（R）の順に並んでいる*9。例えば静止画像ファイルを読み込み、各画素を取り出して処理するプログラムは典型的には下記のようになる。

```
int x, y;
BYTE r, g, b;
BYTE l;
BYTE* pixel;
IplImage* image;

if (image=cvLoadImage("ファイル名", -1)){
    // 与えられたファイル名の画像を読み込む
  for (y=0; y < image->height; y++) {
    for (x=0; x < image->width; x++) {
      pixel = GetPixelPtr(image, x, y);
      if (image->nChannels == 1) {
        // グレイスケール画像（濃淡画像）や
           二値画像などの1チャネル画像
        l = *pixel;
        // 変数lは座標(x, y)の画素の輝度値
           …ここに処理を記述…
      }
      if (image->nChannels == 3) {
        // カラー画像：B, G, Rの順にその画素の
        // 色成分値が入っている
        b = pixel[0]; // 青色成分を取り出す
        g = pixel[1]; // 緑色成分を取り出す
        r = pixel[2]; // 赤色成分を取り出す
        // 変数r, g, bは座標(x, y)の画素の
        // 各色成分の値
           …ここに処理を記述…
      }
    }
  }
  cvReleaseImage(&image);
  // ファイル読み込み時に確保した画像データを開放
}
```

ここで、IplImage構造体の画像データ領域imageDataは文字列型（8bit符号あり整数）として宣言されているが、実際のデータはたいていの場合8bit符号なし整数として扱うべきである*10ことに注意する必要がある。不用意に文字列としてアクセスすると演算や比較の際に符号がらみのバグに悩まされる原因となるので、上記のようにBYTE（unsigned char、8bit符号なし）型を定義し、キャストしてからアクセスすることを推奨する。また、画素値同士の加算

*8 IPLはOpenCVのベースとなったライブラリIntel Image Processing Libraryの略である。より細かい操作が必要な場合はインストールするとよい。IPLは既に公式公開が終了しており、ミラーサイトhttp://www.cvmt.dk/~hn/Images/install/IPL/からインストールバイナリとマニュアルが入手可能である。

*9 OpenCVではBGRがデフォルトであるのに対し、同じく脚光を浴びているARToolkitはOpenGLのフォーマットに準拠しRGB順がデフォルトであるため、併用するには変換処理が必要である。

*10 前述のようにメンバdepthの値に依存する。

や乗算を行う場合、8bit変数のままでは容易に桁溢れが発生する。画素データの演算を行う際には桁溢れの可能性に注意しunsigned shortやunsigned int、さらにはdouble型の変数を場合に応じて使い分けたい。

前述のように画像データ（imageData）は一次元配列として実装されており、ポインタ変数を用いて画素データに直接アクセスすることができる。例えば静止画像ファイルを読み込み、グレイスケール画像に変換して保存するプログラムは下記のようになる*11。

```
int x, y;
IplImage* image;
IplImage* imgGray=0;
char *loadFileName;
char *saveFileName;

if (image=cvLoadImage(loadFileName,
     CV_LOAD_IMAGE_ANYCOLOR)) {
    imgGray = cvCreateImage(cvGetSize(image)
                            , IPL_DEPTH_8U, 1);
    BYTE* srcPixel = (BYTE*)image->imageData;
    BYTE* dstPixel = (BYTE*)imgGray->imageData;
    for (y=0; y < image->height; y++) {
     for (x=0; x < image->width; x++) {
      if (image->nChannels == 1) {
       // →そのまま画素値をコピー
       *dstPixel = *srcPixel;
       dstPixel++;
       srcPixel++; // ポインタ更新
       }
       if (image->nChannels == 3) {
        *dstPixel = ((*srcPixel)>>2)
                  + ((*(srcPixel+1))>>1)
                  + ((*(srcPixel+2))>>2);
          // グレイスケールへの簡易変換
          // B:G:R=1:2:1 の比で混合し
          // 濃淡画像の対応座標に格納
        dstPixel++;
        srcPixel+=3;
          // ポインタ更新
      }
     }
    }
   }
 }
 cvSaveImage(saveFileName, imgGray);
    // 濃淡画像をファイルに保存
 cvReleaseImage(&image);
    // ファイル読み込み時に確保された領域を開放
 cvReleaseImage(&imgGray);
    // グレイスケール用に確保した領域を開放
}
```

映像ではなく1枚の画像データのファイルからの読み込み、また保存のための関数がcvLoadImage()とcvSaveImage()である。cvLoadImage()はJPegやBMPなどの主要なフォーマットの画像ファイルから画像データを読み込み、IplImage構造体へのポインタを返す。第2引数は読み込みの際のフォーマットを指定するためのフラグであり、グレイスケールで読み込む場合はCV_LOAD_IMAGE_GRAYSCALEを、カラーで読み込む場合はCV_LOAD_IMAGE_COLORを、入力ファイルのままにしておく場合はCV_LOAD_IMAGE_ANYCOLORを指定する。cvLoadImage()は画像データを収める領域を確保して返すため、利用しなくなった領域はcvReleaseImage()を呼び出して解放する必要がある。

5.2 画像の作成と変換

画像データの実装を説明するために前節では画像データに直接アクセスする方法を記述したが、OpenCVには画像処理のための多くの機能がOpenCVには関数として用意されている。下記はカラー画像ファイル“image_color.jpg”を読み込み、グレイスケール画像に変換して“image_gray.jpg”に保存するという処理である。

```
IplImage *image, *imgGray;
image = cvLoadImage("image_color.jpg",
                    CV_LOAD_IMAGE_ANYCOLOR);
imgGray = cvCreateImage(cvGetSize(image),
                    IPL_DEAPTH_8U, 1);
    // 同じサイズの濃淡画像領域を確保
cvCvtColor(image, imgGray, CV_BGR2GRAY);
    // 濃淡画像に変換
cvSaveImage("image_gray.jpg", imgGray);
cvReleaseImage(&image);
    // 読み込み時に確保された領域を開放
cvReleaseImage(&imgGray);
    // 作成した画像領域を開放
```

OpenCVでは画像に対する処理や変換を行うために画像領域（IplImage）を作成することができる。そのための関数がcvCreateImage()である。前項のプログラム例にも記述があるが、作成した画像はプログラム終了までにcvReleaseImage()を呼びだして解放する必要がある。読み込んだ画像ファイルと同じサイズ（幅と高さ）の画像領域を作成するためにcvCreateImage()の第1引数にcvGetSize（image）と記述して、変数imageの幅と高さをそのまま与えている。cvCreateImage()の第2引数は作成する画像の色深度（depth）で、チャネルあたり0～255の256階調を意味するIPL_DEAPTH_8Uを与えている。更に、第3引数にチャネル数1を示すことで、1チャネル256階調のグレイスケール画像領域が作成される。cvCvtColor()は画像の色空間を変換する関数で、多種多様な変換がサポートされている。

*11 変数loadFileName、saveFileNameには既に値が与えられているとする。

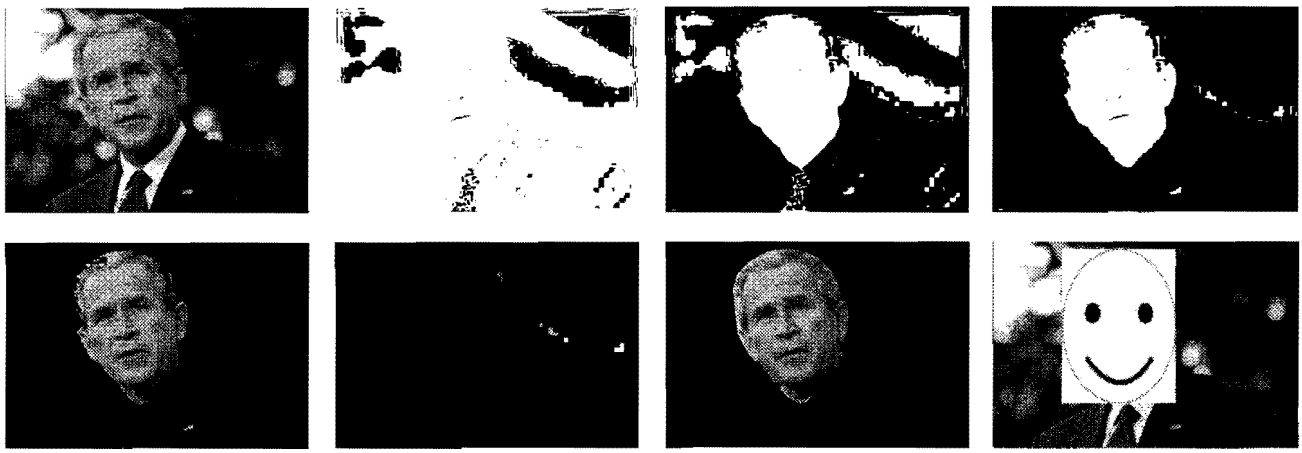

図2 マスク処理の様子（順にimgFrame、imgMaskHigh、imgMaskLow、imgMask、imgSkin、ラベリング結果の可視化、凸包化結果、顔領域の修飾結果）

第3引数に与える定数（一般形はCV_[src_color_space]2[dst_color_space]）により変換方法が指定できる。例えば、上のコードで使われているCV_BGR2GRAYは、BGRカラー画像をGRAY、すなわちグレイスケール画像（濃淡画像）に変換する。

6. 画像処理関数の利用

OpenCVの画像処理関数では画像中のある領域だけを処理対象とするROI（Region Of Interest）やある色成分（チャネル）だけを処理対象とするCOI（Channel Of Interest）を指定することができ、画像処理プログラムを効率よく記述することができる。HSV表色系で肌色を捉えた場合、どの人種でも色相（H値）が6°～38°の範囲に入るという統計[7]に基づき、入力画像から肌色領域を抽出した画像を作成するコードを次に示す*12。

```
int ProcessFrame(IplImage *imgFrame, int nId) {
  cvCvtColor(imgFrame, imgHsv, CV_BGR2HSV);
  cvSetImageCOI(imgHsv, 1);
  cvCopy(imgHsv, imgMask);
  cvThreshold(imgMask, imgMaskLow, (double)38/2,
                    255, CV_THRESH_BINARY_INV);
  cvThreshold(imgMask, imgMaskHigh, (double)6/2,
                      255, CV_THRESH_BINARY);
  cvAnd(imgMaskHigh, imgMaskLow, imgMask);
  cvSetZero(imgSkin);
  cvCopy(imgFrame, imgSkin, imgMask);
  return 0;
}
```

ここで、imgFrameと同じサイズの3チャネル画像imgHsv、imgSkinと1チャネル画像imgMask、imgMaskHigh、imgMaskLowが確保されていることが前提である。imgHsvは入力カラー画像imgFrameをHSV色空間に変換したものである。cvSetImageCOI()は画像データにCOIを設定する関数で、他の関数が処理対象とするチャネルを限定することができる。上のコードではimgHsvのH値を格納している第1チャネルをCOIに設定している。cvCopy()は単純には画像データ間のコピーを行う関数であるが、COIやROIを設定すると様々な用途に使用できる便利な関数である。例えばコピー元画像にCOIを設定し、3チャネル画像領域から1チャネル画像にcvCopy()すれば元画像の特定チャネルを抜き出せる。また、第3引数にROIとしてマスク画像を与えてcvCopy()すれば、マスク部分のみを取り出した画像を生成できる。上のコードではCOIが設定されたimgHsvをコピー元としてcvCopyを呼び出すことで色相情報のみの1チャネル画像imgMaskを作成している。

cvThreshold()は閾値を指定して濃淡画像を二値化する関数である。第5引数にCV_THRESH_BINARYを指定すると、第3引数に与えた閾値以上の値を持つ画素が抽出でき、CV_THRESH_BINARY_INVを指定すると閾値以下の画素が抽出できる。この関数を用いてH値が0°～38°の領域を示すimgMaskLowと6°～360°の領域を示すimgMaskHighを作成し、cvAnd()でそれらの論理積をとって肌色領域マスクimgMaskを作成する。更に、imgMaskを今度はROIに設定してcvCopy()を呼び出すことで、原画像から肌色領域のみを抽出したカラー画像imgSkinが得られる。

抽出結果から、例えば肌色領域マスクをラベリング*13して最大面積の領域を抽出するなどにより顔領域が得られる*14。更に領域を凸包化し*15その外接矩形を得れば、別の画像で顔部分だけ修飾するようなプログラムとなる。図2に一連の処理過程の画像例を示す。

*12 紙面の都合上、引数等のエラーチェックは省略する。
*13 OpenCVではラベリング機能が提供されていないが、奈良先端科学技術大学院大学の井村誠孝氏が画像ラベリングルーチン[8]を公開している。
*14 簡易的な方法であり、精度は高くない。
*15 凸包化にはcvConvexHull2()が用意されている。サンプルコードが[3]にあるので参照されたい。

7. GUIの利用

7.1 ウィンドウの生成と画像の表示

OpenCVには簡易GUIパッケージHighGUIが用意されている*16。凝った表示はできないが、ウィンドウを作成してカメラ入力画像やそれを処理した結果の画像をリアルタイム表示する程度の目的には十分使えるだけの基本的な機能を持っている。HighGUIはウィンドウを作成時の「名前」で管理する。ウィンドウはcvNamedWindow（"名前"，CV_WINDOW_AUTOSIZE）と呼び出せば生成される。第1引数に与える文字列が「名前」であり、第2引数にCV_WINDOW_AUTOSIZEと与えておけば表示画像サイズに合わせてウィンドウのサイズが自動調整される。ウィンドウに画像を表示するにはcvShowImage（"名前", iplImage）と呼べばよい。

```
// 画像領域はstatic変数として持つ
static IplImage *binary=0; // 初期化必要
static int thresh = 128;

void on_trackbar(int t) {
  cvThreshold(imgGray, binary, (double)thresh,
    255, CV_THRESH_BINARY);
  cvShowImage("ImageThersh", binary);
}

// ウィンドウ作成などの初期化
int InitProcess(void) {
  cvNamedWindow("Image", CV_WINDOW_AUTOSIZE);
  cvNamedWindow("ImageThersh", CV_WINDOW_AUTOSIZE);
  return 0;
}

// 1フレームの画像の処理
int ProcessFrame(IplImage *imgFrame, int nId) {
  if (!imgFrame) {
    return -1;
  }
  // 二値化画像の領域を用意（初回のみ）
  if (!binary)
    binary = cvCreateImage(cvGetSize(imgGray),
IPL_DEPTH_8U, 1);
  cvShowImage("Image", imgFrame);
  cvCreateTrackbar("threshold", "ImageThersh",
&thresh, 255, on_trackbar);
  on_trackbar(128);
  return 0;
}

// 終了処理：領域解放，ウィンドウ消去など
int ExitProcess(IplImage *imgFrame) {
  // 画像サンプルの保存
cvSaveImage("image_sampleBinary.jpg", binary);
  // バッファ領域解放
  cvReleaseImage( &binary );
  return 0;
}
```

図3　トラックバーの利用

7.2 トラックバーの利用

閾値などパラメータをリアルタイムに変更してその結果を見たいような場合には、cvCreateTrackbar()を使ってウィンドウにトラックバーを付ける。トラックバーで設定された値を閾値として入力画像を二値化し表示するプログラムを図3に示す。生成するトラックバーの名前とウィンドウの名前をcvCreateTrackbar()の引数として与えることでウィンドウとトラックバーが関連づけられる。第5引数のon_trackbarはトラックバーが操作されると呼び出されるコールバック関数の指定である。

図3のプログラムは図1と合わせるとそのまま動作する。また、図1からcvCreateCameraCapture()とcvReleaseCapture()を呼び出している部分を削除し、cvQueryFrame()の呼び出しの代わりにcvLoadImage（argv[1]、CV LOAD IMAGE ANYCOLOR）としてやると、コマンドラインから引数として与えられた画像ファイルを読み込んで表示し、トラックバーを操作して閾値を変更した二値化結果を表示するプログラムとなる。

8. おわりに

アプリケーションに手軽に画像処理機能を付加できる画像処理ライブラリOpenCVを紹介した。既述の通り、OpenCVには500もの関数が用意されており、本稿に記述した内容はそのほんの「さわり」である。OpenCVのパッケージには、輪郭抽出、エッジ抽出、対象追跡、本稿で紹介したような簡易版ではないCascade型識別器による本格的な顔検出などのサンプルプログラムが添付されているので参考にしていただきたい。

YouTubeやニコニコ動画などでも画像処理ベースの「作品」が注目を集めており、これまで十分とは言えなかった非専門家向けの情報提供体制が急速に整いつつある。より本格的に利用したい読者は、文献[2]や[9]を参照いただきたい。文献[2]の著者らがボランティアベースで運営するopencv.jp[3]に多くの実用的なサンプルコードと解説が掲載されている。ぜひ参照いただきたい。

*16 HighGUIを利用する際にはhighgui.hをincludeする必要がある。

参考文献

[1] Open Computer Vision Library: http://sourceforge.net/projects/opencvlibrary/

[2] 奈良先端科学技術大学院大学OpenCVプログラミングブック制作チーム: OpenCVプログラミングブック, 毎日コミュニケーションズ, 2007.

[3] http://opencv.jp/

[4] Viola, P., Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features, IEEE CVPR, 2001.

[5] OpenCV1.0 on Cygwin: http://www.dh.aist.go.jp/~kimura/opencv/opencv-1.0.html.ja

[6] Visual Studio 2008 Express Editions: http://www.microsoft.com/japan/msdn/vstudio/express/

[7] Sherrah, J., Gong, S.: Skin Colour Analysis, http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL COPIES/GONG1/html, 2001.

[8] 井村誠孝: 画像ラベリングルーチン, http://chihara.naist.jp/people/STAFF/imura/products/labeling

[9] Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision With the OpenCV Library, Oreilly & Associates Inc, Oct, 2008.

著者紹介

河野 恭之（こうの やすゆき）：

関西学院大学理工学部情報科学科教授。1994年大阪大学大学院基礎工学研究科博士後期課程修了。同年（株）東芝入社。同社関西研究所研究主務などを経て、2000年奈良先端科学技術大学院大学情報科学研究科助教授。2007年より現職。2006年～2008年IPA（情報処理推進機構）未踏ソフトウェア創造事業プロジェクトマネージャ兼任。専門は実世界インタラクション、体験記録とその利用、ウェアラブルとユビキタス。