Processing による画像処理のために

Some Hints for Image Processing by “Processing”

ネットワーク情報学部　石原秀男

School of Network and Information　Hideo ISHIHARA

Keywords: Processing, Augmented Reality, OpenCV, Kinect

Abstract

Initially, the “Processing” programming language is created to teach fundamentals of computer programming for beginners, but now also evolved into a development tool[1] for professionals. Here, some useful libraries for “Processing”, such as, nyar4psg (for Augmented Reality), opencv-processing (wrapper for OpenCV functions), simple-openni(wrapper for Kinect Drivers) are introduced. These can be good tools for the people who want to build their own Image Processing Projects.

１．はじめに

入門者向けの言語と捉えられがちの Processing だが，外部のライブラリ群を利用すれば，かなり高度な処理も実行できる。ここでは，多くのライブラリの中から動画像処理に役立つものをいくつか紹介しよう。

なお，サンプルプログラムは，Processing 2.2.1 と Windows 7，8.1，Mac OS 10.10 の各 OS でテストを行っており，ほとんどの環境で動作する[1]はずである。

２．ビデオキャプチャと画素

最初に，動画像処理の基礎となるビデオキャプチャについて解説しておこう。Processing には標準の video ライブラリがあり，Capture クラスを使用して，PC に接続された USB カメラなどから動画像を取り込むことができる。

次は，キャプチャの簡単なサンプルである。

```
import processing.video.*;

Capture cam;

void setup(){
  size(640,480);
  cam = new Capture(this, 640, 480, 30);
  cam.start();
  frameRate(30);
}

void draw(){
  if(cam.available() == true)
    cam.read();
  image(cam, 0, 0);
}
```

Program.1

setup()の，new Capture()で 640×480 ドット，毎秒 30 フレームのキャプチャオブジェクトを作成し，cam.start()でキャプチャを開始している。draw()ループでは，cam.available()で 1 フレーム分のデータが取り込まれたことを確認し，cam.read()で読み込み，image()でウィンドウ上(0，0)の位置（左上隅）に表示している。

この方法では，使用しているカメラが解像度 640×480 ドット，毎秒 30 フレームをサポートしていなければ，new Capture()に失敗してしまうが，ほとんどのカメラでは問題にならないだろう。

画像の解像度やフレームレートを細かく指定したいときは，new Capture()の前で

```
String cams[] = Capture.list();
for( int i = 0 ; i < cams.length ; i++ )
  println( "[" + i + "]:" + cams[i] );
```

とすれば，サポートされるカメラのモードが

[1] 問題が発生した場合，Processing を最新の 3.x 系や逆に過去の 1.x 系に変更すると解消できることもある。

```
[0]:name=WebCamera HD,size=640x480,fps=15
[1]:name=WebCamera HD,size=640x480,fps=30
[2]:name=WebCamera HD,size=160x120,fps=15
                ・・・
```

などと表示されるので，たとえば

```
cam = new Capture(this, cams[1]);
```

というように，Capture の引数に cams[1]を指定すれば，このケースでは，640×480 ドット，毎秒 30 フレームで画像を取り込むことができる。

キャプチャ画像の保存は，カウンタ用の変数 int frame を宣言した上で，image(cam, 0, 0)に続けて

```
  saveFrame();
  frame++;
  if( frame >= 300 )
    exit();
```

とし，イメージシーケンスを使えばよい。この例なら，ソースコードと同じフォルダに，screen-0001.tif から screen-0300.tif まで 300 枚，時間にして 10 秒分のデータを保存できる。無圧縮なので 1 枚当たり 900KB とサイズは大きくなるが，ある程度の性能があるマシンならコマ落ちするようなことはないはずだ。

キャプチャしたデータから各画素のデータを取り出すには，loadPixels()を使用する。次の Program.2 は，画面の上から 1/3 について赤，1/3 から 2/3 までは緑，2/3 から下は青の各成分のみを表示するサンプルである。

```
//setup()までは Program.1 と同じなので省略
void draw(){
  if(cam.available() == true){
    cam.read();
    cam.loadPixels();
    for(int i = 0 ; i < 640*160 ; i++)
      cam.pixels[i]
      = color( red( cam.pixels[i] ) , 0 , 0 );
    for(int i = 640*160 ; i < 640*320 ; i++)
      cam.pixels[i]
      =color( 0,green(cam.pixels[i] ) , 0 );
    for(int i = 640*320 ; i < 640*480 ; i++)
      cam.pixels[i]
      = color( 0, 0 ,blue( cam.pixels[i] ) );
  }
  image(cam , 0 , 0);
}
```

Program.2

Program.2 では，cam.loadPixels()により各画素のデータが（画像の幅×高さ）個の要素を持つ一次元配列 cam.pixels[]に読み込まれ，(x,y)の位置の画素情報は cam.pixels[x+height*y]に記録される。サンプルからわかるように，RGB 要素の取り出しは red(), green(), blue()の各関数で行える。これを応用すれば，読み込んだ画像を自由に加工できる。

３．拡張現実

AR（Augmented Reality : 拡張現実）とは，現実世界を拡張し，仮想的な情報を付加する技術を意味するが，ARToolkit[2]はこの目的のために開発されたライブラリである。ARToolkit を用いれば，現実世界に実在するマーカと呼ばれる特定の画像を認識し，その 3 次元空間上の位置に任意の仮想的なオブジェクトを重ねて表示することができる。NyARToolkit はこの ARToolkit をベースに開発されたライブラリで，さまざまな言語向けのバージョンが存在するが，その中に NyARToolkit for Processing である nyar4psg がある。ここでは文献[3]に従って nyar4psg による拡張現実アプリケーションの作成について説明しよう。

まずライブラリのインストールだが

http://sourceforge.jp/projects/nyartoolkit/releases/

から nyar4psg-1.3.1.zip（現時点での nyar4psg の最新バージョン）をダウンロード・解凍し，nyar4psg とリネームして，フォルダごと Processing の Sketchbook フォルダ直下の libraries フォルダへコピー[2]する。なお，このように新規にライブラリを追加したときは，Processing 本体を再起動する必要があるので注意してもらいたい。

続いて認識の対象となるマーカを作成する。マーカは一辺 8cm 以下の正方形の黒枠（枠の太さは辺の 1/4）に囲まれた図形で，回転させたときに不都合を生じぬよう対称形を避ければ，特にデザイン上の制約はない。今回は，Fig.1 に示すように 80mm 四方，幅 20mm の黒枠内に，AR という文字をそのまま描いたものを作成し，AR.png として保存するとともに白紙に印刷し，適当な大きさ[3]に切り抜いておいた。

作成したマーカをプログラム中で処理するためには，

http://flash.tarotaro.org/ar
/MarkerGeneratorOnline.html

[2] もしドキュメントフォルダに Processing/libraries がない場合は，自分で作成する。
[3] 黒枠の周囲に一定の余白（白）が必要なので，そこまで含んだ形で保存や印刷しなければならない。

において，パターンファイルと呼ばれるデータファイルへと変換する。

この URL を開くとカメラへのアクセスを尋ねられるが，「拒否」を選んでから，ページ左上 Mode Select で Load Marker Image を選択し，作成した画像（AR.png）を指定する。マーカが認識され赤く囲まれていることを確認したら，ページ上中央の MarkerSegment が 16×16 となっていることをチェックし，Get Pattern を押す。Preview Marker にぼやけた感じの画像が表示されたら，適当な名前（今回は AR16.pat）で保存すればよい。これがパターンファイルである。一つのプログラム内で複数個のマーカを使用するときは，同様の方法で各マーカについてのパターンファイルを作成しておく。

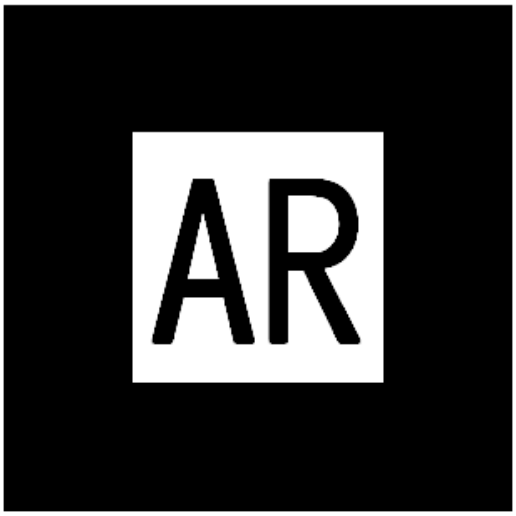

Fig.1　AR Marker

バターンファイルとは別に，カメラの特性を記録したカメラパラメータファイルも必要になるが，通常の環境であればライブラリに付属しているデフォルトの camera_para.dat を使用しても特に不都合[4]はない。

下は作成したマーカを認識し，その位置に青色の半透明な立方体を描画するプログラムである。ただし，実行前の準備として，メニューの Add Files から AR16.pat，camera_para.dat，NyAR4psg2b.jar を追加[5]しておく必要がある。

```
import processing.video.*;
import jp.nyatla.nyar4psg.*;

Capture cam;
MultiMarker ar;

int id;

void setup() {
  size(640,480,P3D);

  cam = new Capture(this,640,480,30);
  cam.start();
  ar = new MultiMarker(this,640,480,
             "camera_para.dat",
             NyAR4PsgConfig.CONFIG_PSG);
  id = ar.addARMarker("AR16.pat",80);
}

void draw()
{
  if(cam.available() ==true)
    cam.read();

  background(0);
  ar.drawBackground(cam);

  ar.detect(cam);
  if((ar.isExistMarker(id))){
    ar.beginTransform(id);
    fill(0,0,255,128);
    translate(0,0,20);
    box(40);
    ar.endTransform();
  }
}
```

Program.3

プログラムについて簡単に説明しておこう。MultiMarker は，複数マーカの認識が可能な ARToolkit の管理クラスで，ar = new MultiMarker()がコンストラクタである。引数 NyAR4PsgConfig.CONFIG_PSG は，マーカ座標系を左手座標系とし，姿勢推定アルゴリズムとして NyARToolkit を使用することを意味する定数である。ar.addARMarker()はマーカの登録部で，引数の"AR16.pat"がパターンファイル名，80 はサイズ（ミリ単位）を表しており，その管理番号を（普通は 0 から順に）整数 id に登録している。複数のマーカを使用するときは，この文を複数実行し，id に相当する変数もそれに応じて複数用意すればよい。

draw ループ内の ar.drawBackground()は，指定される画像をバックグラウンドに表示し，ar.detect()で指定される画像からマーカを検出する処理を開始する。ar.isExistMarker()は引数の示すマーカを認識できたか否かを判定し，ar.beginTransform()で座標軸を

[4] キャリブレーションをしてパラメータファイルを作成すれば認識率の向上が期待できる。
[5] AR16.pat と camera_para.dat は data フォルダに，NyAR4psg2b.jar は code フォルダに保存される

引数の示しているマーカ平面上に設定する。このマーカ平面に対して translate(0,0,20)が適用されるので，マーカの中心直上，z 軸方向（上方）へ+20 移動した点が原点になる。box(40)はその原点を中心として一辺 40 の立方体を描くため，Fig.2 に示すようにマーカの上に描画されるわけである。

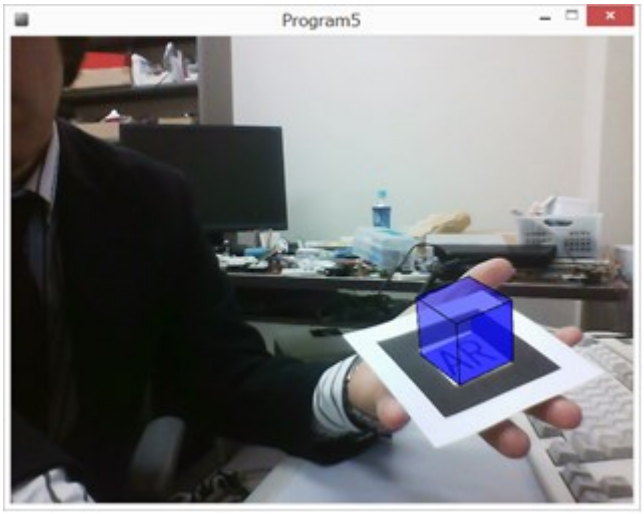

Fig.2　Augmented Reality

このプログラムを応用すれば，マーカ上でキャラクタを動かすなどというような AR らしいアプリケーションも作成できるだろう。

4．テンプレートマッチング

テンプレートマッチングは，入力画像から特定のパターン（テンプレート）画像を探し出す手法である。前節の AR におけるマーカ検出もテンプレートマッチングの一種であるが，NyARToolkit では事前にマーカからパターンファイルを作成[6]しておかなければならない。一方，パターンの拡大，縮小などの変化に対するロバストネスを犠牲にしても，特に事前準備することなしに，画像同士のマッチングをリアルタイムで実行したいというニーズもあるだろう。そのような比較的プリミティブな目的には，コンピュータビジョン全般に関するライブラリである OpenCV が適している。Processing ではそのサブセットである OpenCV for Processing が利用できるが，OpenCV 自体は膨大なライブラリなのでその詳細については文献[4]などを参考にしてもらいたい。

インストールは，Processing の Library Manager を利用して，メニューから Sketch...Import Library...Add Library と進んで OpenCV for Processing を選べばよい。環境によっては Library Maneger が失敗することもあるが，そのときは

https://github.cm/atduskgreg
/opencv-processing/releases

からダウンロードし，解凍した上で NyARToolkit の場合と同様に Processing の Sketchbook フォルダ直下の libraries フォルダへ置けばよい。

下は OpenCV を利用して作成したテンプレートマッチングのサンプルプログラムである。このプログラムでは，カメラからのキャプチャ画像がリアルタイムで表示されており，画面上でマウスをクリックすると，その点を左上端とする 48×48 ドットの画像をテンプレートに設定する。以降は，リアルタイムでキャプチャ画像からテンプレートを探索し，最もそれに近い領域を緑色の線で囲って表示する。具体的な用途としては，動き回る物体を追尾することを想定している。

```
import gab.opencv.*;
import processing.video.*;
import org.opencv.core.Mat;
import org.opencv.core.CvType;
import org.opencv.imgproc.Imgproc;
import
   org.opencv.core.Core.MinMaxLocResult;
import org.opencv.core.Core;

Capture cam;
OpenCV inputcv;
OpenCV templatecv;
Mat resultMat;
PImage templateImage;
int xxx, yyy;
final int SIZE = 48;

void setup(){
  size(640, 480);

  strokeWeight(3);
  noFill();
  stroke(0,255,0);

  cam = new Capture(this, 640, 480 ,30);
  inputcv = new OpenCV(this, 640, 480);
  templatecv
        = new OpenCV(this, SIZE, SIZE);
  resultMat
        = new Mat(640-SIZE+1, 480-SIZE+1,
                  CvType.CV_32FC1);
  cam.start();
```

[6] これは NyARToolkit の制限であって，AR 全般の制限というわけではない。

```
  frameRate(30);
}

void draw(){
  if(cam.available() == true)
    cam.read();
  image(cam, 0, 0);

  inputcv.loadImage(cam);
  Imgproc.matchTemplate(
      inputcv.getColor(),
      templatecv.getColor(),
      resultMat,
      Imgproc.TM_CCOEFF_NORMED  );
  MinMaxLocResult maxPoint
            = Core.minMaxLoc(resultMat);
  xxx = (int)maxPoint.maxLoc.x;
  yyy = (int)maxPoint.maxLoc.y;
  rect(xxx, yyy, SIZE, SIZE);
}

void mouseClicked(){
  xxx = mouseX;
  yyy = mouseY;
  templateImage = cam.get(
       mouseX, mouseY, SIZE, SIZE );
  templatecv.loadImage(templateImage);
}
```

Program.4

プログラムについて簡単に説明しておこう。inputcv は 640×480 ドットの入力画像，templatecv は 48×48 ドットのテンプレート画像のための変数である。resultMat はデータ管理用 Mat 型変数で，実体としては(640-48+1)×(480-48+1)の行列[7]になる。各オブジェクトのインスタンスは setup()で生成するが，resultMat = new Mat()では CvType.CV_32FC1 とし，保存するデータを 32 ビット浮動小数点に指定している。マッチング処理が行われるのは draw()ループの Imgproc.matchTemplate()で，TM_CCOEFF_NORMED はアルゴリズムとして正規化相互相関を使用することを意味する。各位置に対する相関係数の計算結果は，相当する resultMat に保存されるので minMaxLoc で maxPoint に最大・最小値を取り出し，さらに最大となる位置を maxLoc で求めている。要するに 640×480 ドットの画像中に，48×48 ドットの領域を移動させながら重ね合わせ，(640-48+1)×(480-48+1)個の相関係数を求めて，それが最大となる位置を探すわけである。大変な計算量になるが，効率的にコーディングされている OpenCV を用いればリアルタイムの処理も可能になる。

５．骨格の検出

人体の認識は OpenCV でも可能だが，細かな動作の検出などには，距離カメラでもある Kinect が適している。Processing から Kinect を使用する方法としては，Kinect4WinSDK, Open Kinect for Processing なども存在するが，ここでは文献[5]で解説[8]されている SimpleOpenNI を紹介しよう。

インストールは NyARToolkit や OpenCV の場合と同様に，Library Manager を使用して Add Library から SimpleOpenNI を選べばよい。うまく行かなければ，

https://code.google.com/p/simple-openni
/wiki/Installation

からダウンロードして手動でインストールすればよいが，その手順についても NyARToolkit や OpenCV と変わらない。

MacOS の場合はこれだけで Kinect 用のプログラムが動作するが，Windows 系 OS では，さらにドライバをインストールしなければならない。手順としては，事前に Visual Studio Express 2012 をインストールしておいて，Kinect for Windows SDK v1.8 を英語サイト

http://www.microsoft.com/en-us/download/

から検索してインストールすればよい。

下は，右手を追跡するサンプルプログラムである。

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup(){
  size(640, 480);

  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
  kinect.enableUser();
}

void draw(){
  kinect.update();
  PImage depth = kinect.depthImage();
```

[7] 640×480 ドットの範囲に存在する 48×48 ドットの領域の数に相当する。

[8] 残念ながらバージョンアップのために掲載されているサンプルの多くはそのままでは動作しない。

```
  image(depth, 0, 0);

  IntVector userList = new IntVector();
  kinect.getUsers(userList);

  if(userList.size() > 0){
    int userId = userList.get(0);

    if(kinect.isTrackingSkeleton(userId))
    {
      PVector rightHand = new PVector();
      kinect.getJointPositionSkeleton
      ( userId,
        SimpleOpenNI.SKEL_RIGHT_HAND,
        rightHand );
      PVector convertedRightHand
                        = new PVector();
      kinect.convertRealWorldToProjective
      ( rightHand,
        convertedRightHand );
      fill(255, 0, 0);

      ellipse(convertedRightHand.x,
              convertedRightHand.y,
              50,50 );

    }
  }
}

void onNewUser( SimpleOpenNI kinect,
                            int userId )
{
  println("Detection Start");
  kinect.startTrackingSkeleton(userId);
}
```

Program.5

プログラムについて簡単に説明しておこう。setup()の new SimpleOpenNI()で SimpleOpenNI オブジェクト kinect を宣言し，enableDepth()で距離カメラへのアクセスを，enableUser()でユーザの追跡を可能にしている。draw()ループの update()では Kinect から新しいデータを取得し，depthImage()で距離画像[9]を抽出し，image()で表示している。getUsers()でユーザ情報を userList へ読み込み，ユーザが一人でも認識できたら，一人目のユーザの番号を整数 userId へと代入する。isTrackingSkelton()でその人物の骨格情報が入手可能であること確認したら，getJoingPositionSkelton()で SKEL_RIGHT_HAND，すなわち右手の位置を PVector 型の変数 rightHand に取得する。こうして得られた 3 次元の空間座標を convertedReadlWorldToProjective()で 2 次元のスクリーン座標へと変換し，そこに直径 50 の円を描いているわけである。

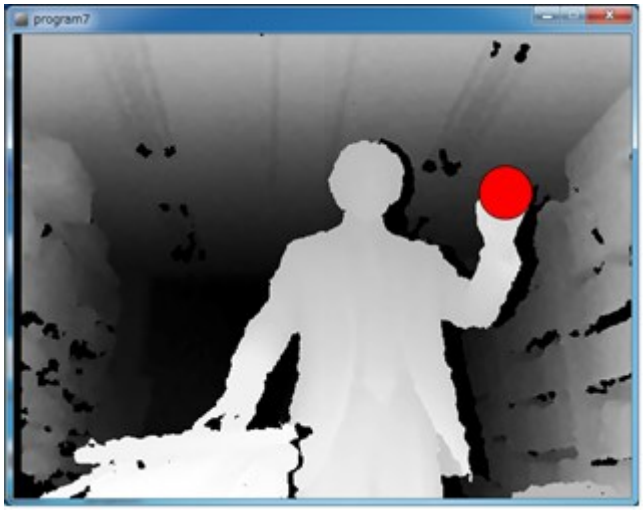

Fig.3　Skelton Tracking

Fig.3 はプログラムが動作している状況の図であるが，人物の左手（鏡像なので実際には右手）が認識されていることがわかるだろう。SKEL_RIGHT_HAND を必要に応じて変更[10]すれば，人体各部の動きをリアルタイムで追跡することも可能である。

６．おわりに

ここでは，NyARToolkit，OpenCV，SimpleOpenNI の各ライブラリを紹介した。いずれも，画像処理を含むプログラミングにおいて有益なツールとなるはずだ。興味が湧いたなら，ぜひ一度試してもらいたい。

参考文献

[1] http://processing.org
[2] http://www.hitl.washington.edu/artoolkit
[3] Processing でつくる拡張現実感のレシピ,橋本,オーム社(2012)
[4] Java で始める OpenCV プログラミング,北山,カットシステム(2013)
[5] Making Things See, Borenstein,オライリー(2013)

[9] カメラに近い点ほど白く表現されるモノクロ画像。

[10] 定数についてはインストールしたライブラリ内の documentation にある SimpleOpenNIConstants を参照。