OpenCV ユーザガイド

v2.2

平成 23 年 2 月 14 日

目 次

第 I 部

C++ API ユーザガイド

第1章　cv::Mat. 画像の操作.

1.1　入出力

画像

ファイルから画像を読み込みます：

```
Mat img = imread(filename);
```

jpg ファイルを読み込む場合，デフォルトで 3 チャンネル画像が作成されます．グレースケール画像が必要な場合は，次のようにしてください：

```
Mat img = imread(filename, 0);
```

画像をファイルに保存します：

```
Mat img = imwrite(filename);
```

XML/YAML

1.2　画像の基本操作

ピクセル輝度値にアクセスする

ピクセルの輝度値を取得するには，画像の種類とチャンネル数を知る必要があります．ここでは，シングルチャンネルのグレースケール画像（8UC1 型）で，座標が x, y のピクセルの例を示します：

```
Scalar intensity = img.at<uchar>(x, y);
```

`intensity.val[0]` には，0 から 255 の値が入ります．

次に，カラーの順序が `bgr` となっている 3 チャンネル画像（これは imread が返すデフォルトのフォーマットです）を考えます：

```
Vec3b intensity = img.at<Vec3b>(x, y);
uchar blue = intensity.val[0];
uchar green = intensity.val[1];
uchar red = intensity.val[2];
```

浮動小数点型の画像（例えば，3 チャンネル画像に対して Sobel を実行したときに得られる画像）に対しても同様のメソッドを利用できます：

```
Vec3f intensity = img.at<Vec3f>(x, y);
float blue = intensity.val[0];
float green = intensity.val[1];
float red = intensity.val[2];
```

このメソッドは，ピクセルの輝度値を変更するときにも利用できます：

```
img.at<uchar>(x, y) = 128;
```

OpenCV には，2 次元や 3 次元の点群の配列を Mat 形式としてとる関数があります．これらは，特に calib3d モジュールに存在し， projectPoints などがその一例です．行列は，必ず 1 列で，各行が 1 つの点に対応します．行列の型は，それぞれ 32FC2 または 32FC3 です．このような行列は，std::vector から簡単に作ることができます．

```
vector<Point2f> points;
//... 配列を埋めます
Mat pointsMat = Mat(points);
```

画像の場合と同様のメソッド Mat::at を利用して，行列の点にアクセスすることができます：

```
Point2f point = pointsMat.at<Point2f>(i, 0);
```

メモリ管理と参照カウント

Mat は，行列や画像の性質（行数や列数，データの種類など）と，データへのポインタを保存する構造体です．同一のデータに対して，複数の Mat インスタンスを作っても全く問題ありません． Mat は参照カウントを持ち，これは，Mat のあるインスタンスが破棄されたときに，そのデータが解放されるべきかどうかを教えてくれます．ここでは，データをコピーせずに 2 つの行列を作る例を示します：

```
std::vector<Point3f> points;
//... 配列を埋めます
Mat pointsMat = Mat(points).reshape(1);
```

この結果，1 列の 32FC3 行列の代わりに，3 列の 32FC1 行列を得ます． pointsMat は， points のデータを利用し，破棄されるときもこのメモリを解放しません．しかし，この場合，開発者は points が pointsMat よりも長いライフタイムを持つことを保証する必要があります．データをコピーする必要があれば，例えば， Mat::copyTo や Mat::clone を利用すれば良いでしょう：

```
Mat img = imread("image.jpg");
Mat img1 = img.clone();
```

開発者が出力画像を作成する必要があった C API とは対照的に，空の出力 Mat を各関数に渡すことができます．それぞれの実装は，出力行列に対して Mat::create を呼び出します．このメソッドは，行列が空なら，その行列にデータを割り当てます．行列が空ではなく，そのデータサイズと種類

が正しければ，このメソッドは何もしません．しかし，サイズまたは種類が入力引数と異なる場合，そのデータは解放され（消えて），新しいデータが割り当てられます．例を示しましょう：

```
Mat img = imread("image.jpg");
Mat sobelx;
Sobel(img, sobelx, CV_32F, 1, 0);
```

基本的な演算

行列には，便利な演算子が多数定義されています．ここでは，いくつかの例を示します．既に存在するグレースケール画像 `img` を，真っ黒な画像にする：

```
img = Scalar(0);
```

ROI の指定：

```
Rect r(10, 10, 100, 100);
Mat smallImg = img(r);
```

`Mat` から C API のデータ構造への変換：

```
Mat img = imread("image.jpg");
IplImage img1 = img;
CvMat m = img;
```

ここでは，データがコピーされないことに注意してください．

カラーからグレースケールへの変換：

```
Mat img = imread("image.jpg"); // 8UC3 画像の読み込み
Mat grey;
cvtColor(img, grey, CV_BGR2GRAY);
```

8UC1 から 32FC1 への型の変換：

```
convertTo(src, dst, CV_32F);
```

画像の可視化

これは，開発処理中のアルゴリズムの中間結果を見る際に，非常に役立ちます．OpenCV には，画像を可視化する便利な方法があります．8U 画像の場合は，次の様にして表示できます：

```
Mat img = imread("image.jpg");

namedWindow("image", CV_WINDOW_AUTOSIZE);
imshow("image", img);
waitKey();
```

waitKey() を呼び出すことで，"image" ウィンドウ内でキーが押されるのを待つ，メッセージパッシングサイクルが開始されます．32F 画像の場合は，8U 型に変換する必要があります．例えば，次の様にします：

```
Mat img = imread("image.jpg");
Mat grey;
cvtColor(img, grey, CV_BGR2GREY);

Mat sobelx;
Sobel(grey, sobelx, CV_32F, 1, 0);

double minVal, maxVal;
minMaxLoc(sobelx, &minVal, &maxVal); // 輝度値の最小と最大を見つけます.
Mat draw;
sobelx.convertTo(draw, CV_8U, 255.0/(maxVal - minVal), -minVal);

namedWindow("image", CV_WINDOW_AUTOSIZE);
imshow("image", draw);
waitKey();
```

第2章　Features2d.

2.1　検出器

2.2　ディスクリプタ

2.3　キーポイントのマッチング

コード

まずは opencv/samples/cpp/matcher_simple.cpp のサンプルから見ていきます：

```
Mat img1 = imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
Mat img2 = imread(argv[2], CV_LOAD_IMAGE_GRAYSCALE);
if(img1.empty() || img2.empty())
{
        printf("Can't read one of the images\n");
        return -1;
}

// キーポイントを検出します.
SurfFeatureDetector detector(400);
vector<KeyPoint> keypoints1, keypoints2;
detector.detect(img1, keypoints1);
detector.detect(img2, keypoints2);

// ディスクリプタを計算します.
SurfDescriptorExtractor extractor;
Mat descriptors1, descriptors2;
extractor.compute(img1, keypoints1, descriptors1);
extractor.compute(img2, keypoints2, descriptors2);

// ディスクリプタ同士をマッチングします.
BruteForceMatcher<L2<float> > matcher;
vector<DMatch> matches;
matcher.match(descriptors1, descriptors2, matches);

// 結果を描画します.
namedWindow("matches", 1);
Mat img_matches;
drawMatches(img1, keypoints1, img2, keypoints2, matches, img_matches);
```

```
imshow("matches", img_matches);
waitKey(0);
```

コードの解説

コードを分解してみましょう.

```
Mat img1 = imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
Mat img2 = imread(argv[2], CV_LOAD_IMAGE_GRAYSCALE);
if(img1.empty() || img2.empty())
{
        printf("Can't read one of the images\n");
        return -1;
}
```

2 つの画像を読み込み，読込が正しく行われているかどうかを確認します.

```
// キーポイントを検出します.
FastFeatureDetector detector(15);
vector<KeyPoint> keypoints1, keypoints2;
detector.detect(img1, keypoints1);
detector.detect(img2, keypoints2);
```

まず，キーポイント検出器のインスタンスを作成します．すべての検出器は，FeatureDetector 抽象インタフェースから派生していますが，コンストラクタはアルゴリズム依存です．各検出器に対する 1 番目の引数は，通常，検出できるキーポイント数と検出の安定性のバランスを制御するためのものです．検出器が異なると，この値の範囲も異なりますので，[1] 意味が分からない場合はデフォルト値を利用してください.

```
// ディスクリプタを計算します.
SurfDescriptorExtractor extractor;
Mat descriptors1, descriptors2;
extractor.compute(img1, keypoints1, descriptors1);
extractor.compute(img2, keypoints2, descriptors2);
```

ディスクリプタ抽出器のインスタンスを作成します．OpenCV のディスクリプタの多くは，DescriptorExtractor 抽象インタフェースから派生します．次に，各キーポイントに対して，ディスクリプタを求めます．`DescriptorExtractor::compute` メソッドが出力する `Mat` には，i-番目のキーポイントのディスクリプタが，i 行目に格納されています．このメソッドは，ディスクリプタが定義されないようなキーポイント（通常，画像の境界付近にあるキーポイント）を削除して，キーポイントベクトルを変更する可能性があることに注意してください．このメソッドは，それぞれの出力キーポイントとディスクリプタが，互いに対応がとれるようにします（キーポイント数とディスクリプタ行列の行数が等しくなるようにします）.

[1] 例えば， FAST の閾値はピクセル輝度の差を意味し，通常，領域によって変化します $[0, 40]$ ．SURF の閾値は，画像の Hessian に対して適用されるもので，通常，100 よりも大きい値をとります.

```
// ディスクリプタ同士をマッチングします.
BruteForceMatcher<L2<float> > matcher;
vector<DMatch> matches;
matcher.match(descriptors1, descriptors2, matches);
```

そして，2 つの画像のディスクリプタが分かると，それらを比較することができます．まず，画像 2 の各ディスクリプタに対し，ユークリッド距離を用いて画像 1 のディスクリプタを全探索し，最近値を見つける matcher を作成します．Brief ディスクリプタにおけるハミング距離と同様に，マンハッタン距離も実装されています．出力ベクトル `matches` には，対応点同士のインデックスが格納されます．

```
// 結果を描画します.
namedWindow("matches", 1);
Mat img_matches;
drawMatches(img1, keypoints1, img2, keypoints2, matches, img_matches);
imshow("matches", img_matches);
waitKey(0);
```

このサンプルの最後の部分は，マッチング結果を可視化しています．

第3章　Highgui.

3.1　Kinect センサの利用

Kinect センサは， VideoCapture クラスによってサポートされています．ユーザは慣れ親しんだ VideoCapture インタフェースを使って，depth マップや rgb 画像，その他のフォーマットの Kinect 出力を取り出すことができます．

OpenCV で Kinect を利用するには，ユーザは以下のステップに従って準備しなければいけません：
1) OpenNI ライブラリ，および PrimeSensor Module for OpenNI http://www.openni.org/downloadfiles をインストールします．各ソフトのインストラクションで表示されるデフォルトフォルダに，インストールしてください．

```
OpenNI:
        Linux & MacOSX:
                Libs into: /usr/lib
                Includes into: /usr/include/ni
        Windows:
                Libs into: c:/Program Files/OpenNI/Lib
                Includes into: c:/Program Files/OpenNI/Include
PrimeSensor Module:
        Linux & MacOSX:
                Bins into: /usr/bin
        Windows:
                Bins into: c:/Program Files/Prime Sense/Sensor/Bin
```

片方あるいは両方がこれとは別のフォルダにインストールされている場合，ユーザは対応する CMake の変数を変更しなくてはいけません（ OPENNI_LIB_DIR, OPENNI_INCLUDE_DIR または/かつ OPENNI_PRIME_SEN ）．
2) CMake で WITH_OPENNI フラグをセットし，OpenCV が OpenNI をサポートするように設定します．OpenNI がデフォルトのインストールフォルダに見つかれば，OpenCV は，PrimeSensor Module の有無に関係なく OpenNI ライブラリを利用してビルドされます．PrimeSensor Module が見つからない場合，その旨の警告が CMake ログに出力されます．PrimeSensor Module が検出されなくても，OpenCV は OpenNI ライブラリを利用してコンパイルされますが，この場合， VideoCapture オブジェクトは Kinect センサからのデータを取得できません．

3) Build OpenCV.

VideoCapture は，次の Kinect データを取り出す事ができます：

```
a.) depth 生成部からのデータ：
        OPENNI_DEPTH_MAP          - mm 単位で表される depth 値 (CV_16UC1)
```

```
        OPENNI_POINT_CLOUD_MAP    - m 単位で表される XYZ (CV_32FC3)
        OPENNI_DISPARITY_MAP      - ピクセル単位で表される視差 (CV_8UC1)
        OPENNI_DISPARITY_MAP_32F  - ピクセル単位で表される視差 (CV_32FC1)
        OPENNI_VALID_DEPTH_MASK   - 有効ピクセル（オクルージョンではない,
                                    . 陰になっていない，など）を表すマスク (CV_8UC1)
b.) RGB 画像生成部からのデータ：
        OPENNI_BGR_IMAGE          - カラー画像 (CV_8UC3)
        OPENNI_GRAY_IMAGE         - グレースケール画像 (CV_8UC1)
```

Kinect から depth マップを得るには， VideoCapture::operator >> を利用します．例えば，次の様になります．

```
VideoCapture capture(0); // または CV_CAP_OPENNI
for(;;)
{
        Mat depthMap;

        capture >> depthMap;

        if( waitKey( 30 ) >= 0 )
                break;
}
```

複数の Kinect マップを得るには， VideoCapture::grab + VideoCapture::retrieve を使ってください．例えば，次の様になります．

```
VideoCapture capture(0); // または CV_CAP_OPENNI
for(;;)
{
        Mat depthMap;
        Mat rgbImage

        capture.grab();

        capture.retrieve( depthMap, OPENNI_DEPTH_MAP );
        capture.retrieve( bgrImage, OPENNI_BGR_IMAGE );

        if( waitKey( 30 ) >= 0 )
                break;
}
```

詳細は， sample フォルダの kinect_maps.cpp を参照してください．