CQ BUNKO SERIES

C++プログラムで人や物の動きを認識

# ラズパイ/PCで体験! 計測のためのカメラ画像処理

澤田 英宏 著
Hidehiro Sawada



CQ出版社

▶バイラテラル・フィルタ

```
cv::bilateralFilter(入力画像，出力画像，各ピクセル近傍領域の直径，色空間におけるシグマ・フィルタ，座標空間におけるシグマ・フィルタ)
```

エッジを鮮明に保ちながら，ノイズを抑えることができます(**図13**).

## 1-5　円の検出

フォルダ名：**05circle**
リンク・ファイル：opencv_core，opencv_highgui，opencv_imgproc，opencv_imgcodecs

円，楕円，四角の中から，円だけにマーカを表示する方法について紹介します(**図14**)．円形状の道路標識を認識させるなどの応用があります(**図15**).

画像中の円を検出を行うのはハフ変換と呼ばれる処理です．グレー・スケール変換などの画像処理は，関数を使うだけで結果が得られました．しかし円の検出をはじめとする高度な処理を行おうとすると，前処理が必要になります.

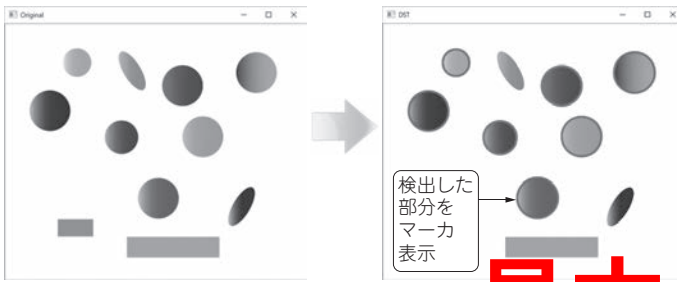



**図14　円の検出**


図15
円検出の応用…道路標識を検出してマーカで表示する


図16　円検出の前処理…グレー・スケール化してぼかし処理を行う

前処理では，カラー画像をグレー・スケール化し，ノイズ除去のためにぼかし処理をします（**図16**）．後は，HoughCircles()を使用して円形状を検知するだけです．検知情報は中心座標と半径の情報を受け取ります．受け取った座標データを基に円を描画します．

### ● プログラム

円を検出するプログラムを**リスト5**に示します．

**▶円を検出する**

```
cv::HoughCircles(入力画像，円の出力データ
        （3要素の浮動小数点），cv::HOUGH_GRADIENT（固定），
        画像分解能に対する出力解像度の比率の逆数，検出され
        円の中心同士の最小距離，しきい値1，しきい値2，
```

リスト5　円を検出するプログラム（05circle.cpp）

```
000: #include <iostream>
001: #include <opencv2/opencv.hpp>
002:
003:
004: int main()
005: {
006:     cv::Mat srcImg = cv::imread("../images/Circle.jpg");;
007:     cv::Mat dstImg = srcImg.clone();
008:     cv::Mat grayImg;
009:
010:     cv::cvtColor(srcImg, grayImg, cv::COLOR_BGR2GRAY);
011:     cv::GaussianBlur(grayImg, grayImg, cv::Size(5, 5),
                                                    10,10);
012:
013:     std::vector<cv::Vec3f> circles;
014:
015:     cv::HoughCircles(grayImg, circles,
                 cv::HOUGH_GRADIENT, 1, 50, 70, 30, 20, 40);
016:     for (std::vector<cv::Vec3f>::iterator ite =
             circles.begin(); ite != circles.end(); ite++) {
017:         cv::Point center(cv::saturate_cast<int>
               ((*ite)[0]), cv::saturate_cast<int> ((*ite)[1]));
018:         int radius(cv::saturate_cast<int> ((*ite)[2]));
019:         cv::circle(dstImg, center, radius,
                     cv::Scalar(255, 0, 255), 2, cv::LINE_AA);
020:     }
021:
022:     cv::namedWindow("Original");
023:     cv::namedWindow("DST");
024:     cv::namedWindow("GRAY");
025:     cv::imshow("Original", srcImg);
026:     cv::imshow("DST", dstImg);
027:     cv::imshow("GRAY", grayImg);
028:     cv::waitKey(0);
029:
030:     return 0;
031: }
```

結果の表示

最小円半径，最大円半径）

第4引き数の「画像分解能に対する出力解像度の比率の逆数」には，1または2を指定します．

第5引き数の「検出される円の中心同士の最小距離」は，小さすぎると正しいものに加えて，複数の近隣の円が誤って検出される可能性があります．大きすぎると，幾つかの円が見逃されることがあります．

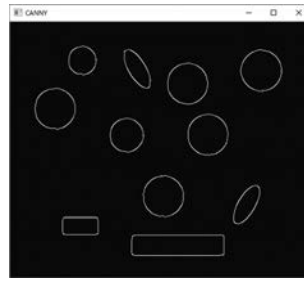



図17　エッジを強調するために2値化する

一般に円検出では，エッジを強調するために2値化してから処理を行います（**図17**）．

しかし，`HoughCircles()`では、グレー・スケールを入力画像としています．これは，`HoughCircles()`関数が内部処理で2値化を行って円検出を行っているためです．このため前処理として2値化を行う必要がありません．第6引き数の「しきい値1」と第7引き数の「しきい値2」は，この内部処理に関するパラメータです．

第6引き数の「しきい値1」は，Cannyエッジ検出器の大きい方のしきい値です．勾配がこのパラメータを超えている場合はエッジとして判定します．

第7引き数の「しきい値2」は，Cannyエッジ検出器の小さい方のしきい値です．勾配がこのパラメータを下回っている場合は非エッジとして判定します．

# 1-6　楕円フィッティング

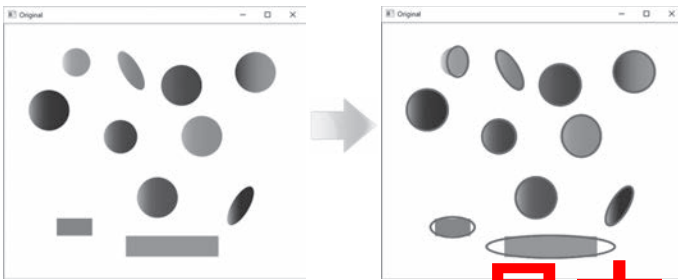
フォルダ名：06ellipse
リンク・ファイル：opencv_core，opencv_highgui，opencv_imgproc，opencv_imgcodecs

エッジ画像に対して楕円領域を検出する楕円フィッティングを紹介します（**図18**）．円検出のように形状検出ではないため，円も四角も楕円にフィッティングされます．検出方法は3段階の手順が必要です．

①元画像を2値化する
②`findContours()`を使用して2値化画像から輪郭を検索する
③`fitEllipse()`を使用して楕円領域を検出する

元画像を2値化した画像を**図19**に示します．グラデーションのかかった円のため，2値化画像では円の一部が欠けた形になっています．このエッジの領域に応じて楕円フィッティングされるため，**図18**のような検出画像になります．

**図18**では，幾何学模様を処理しているため，比較的分かりやすい結果が得られています．しかし，自然画を使用するとフィッティング・エリアが意味不明な感じになります（**図20**）．




元画像　　楕円フィッティング画像

**図18　楕円フィッティング**

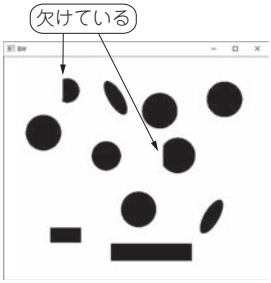



図19　楕円検出の前処理…元画像の2値化


図20　風景写真では分かりにくい結果が得られる

### ● プログラム

findContours()（**リスト6**）で画像内の輪郭座標を検索します．この座標は点のベクトル座標としてcontoursに格納されます．**図18**の画像では5000以上の座標データを受け取ることになります．

ellipse()で楕円の描画を行います．楕円の描画には，中心座標，高さと横幅，角度の情報が必要です（**図9**参照）．またfitEllipse()を使用して楕円領域の座標を取得します．

```
if (count < 100 || count > 1000) continue;
```

の部分はcontours[].size()を取得して楕円の大きさを制限しています．実際に使用する場合も，この大きさ制限は非常に役立ちます．

**▶輪郭検出**

```
cv::findContours(入力画像，輪郭データ出力，
                 輪郭検出方法，輪郭の近似手法)
```

第3引き数の輪郭検出方法には，以下を使用できます．

- RETR_EXTERNAL：最も外側の輪郭を検出する
- RETR_LIST：全ての輪郭を抽出する
- RETR_CCOMP：全ての輪郭を抽出し，2階層構造として保持する

リスト6　楕円フィッティングのプログラム（06e11ipse.cpp）

```
000: #include <iostream>
001: #include <opencv2/opencv.hpp>
002:
003: int main()
004: {
005:
006:     cv::Mat srcImg = cv::imread("../images/Circle.jpg");;
007:     cv::Mat dstImg = srcImg.clone();
008:     cv::Mat grayImg;
009:
010:     std::vector<std::vector<cv::Point> > contours;
011:     cv::cvtColor(srcImg, grayImg, cv::COLOR_BGR2GRAY);
012:     cv::threshold(grayImg, grayImg, 0, 255,
                          cv::THRESH_BINARY | cv::THRESH_OTSU);
013:     cv::findContours(grayImg, contours,
                         cv::RETR_LIST, cv::CHAIN_APPROX_NONE);
014:
015:     for (std::vector<std::vector<cv::Point>
                    >::iterator ite = contours.begin(); ite !=
                                        contours.end (); ite++) {
016:         size_t count = (*ite).size();
017:         if (count < 100 || count > 1000) continue;
018:         cv::Mat pointsf;
019:         cv::Mat(*ite).convertTo(pointsf, CV_32F);
020:         cv::RotatedRect box = cv::fitEllipse(pointsf);
021:         cv::ellipse(srcImg, box, cv::Scalar(255, 0,
                                          255), 2, cv::LINE_AA);
022:     }
023:
024:     cv::namedWindow("Original");
025:     cv::namedWindow("DST");
026:     cv::imshow("Original", srcImg);
027:     cv::imshow("DST", dstImg);
028:     cv::waitKey(0);
029:
030:     return 0;
031: }
```

第4引き数の輪郭の近似手法には以下を使用できます．

- CHAIN_APPROX_NONE：全ての輪郭点を格納する
- CHAIN_APPROX_SIMPLE：水平，垂直，斜めの部分を圧縮し，端点のみを残す．例えば，まっすぐな矩形輪郭は，4つの点にエンコードされる
- CHAIN_APPROX_TC89_L1，CHAIN_APPROX_TC89_KCOS：Teh-Chinチェーン近似アルゴリズムの1つを適用する

（a）フィッティング用２値画像

（b）CHAIN_APPROX_NONE

（c）CHAIN_APPROX_SIMPLE

（d）CHAIN_APPROX_TC89_L1

（e）CHAIN_APPROX_TC89_KCOS

（f）範囲拡（CHAIN_APPROX_NONE）

**図21　輪郭の近似手法による結果の違い**

輪郭の近似手法による結果の違いを**図21**に示します．また，CHAIN_APPROX_NONEのサイズの範囲をcount<10||count>2000と書き換えた場合を併せて示しています［**図21**（f）］．

検出サイズの大きさを制限している関係で，全ての楕円を描画していませんが，CHAIN_APPROX_NONEとCHAIN_APPROX_SIMPLEは違いが確認できます［**図21**（b）と（c）］．

ここでは違いが分かるように風景写真を使用していますが実際

にこのような画像での使い所はほとんどありません．

**▶楕円にフィッティング**

`cv::fitEllipse(2次元の点の集合)`

2次元の点集合に最もフィットする楕円座標を計算します．引き数の2次元の点集合は，Matに格納された2次元の点集合になります．

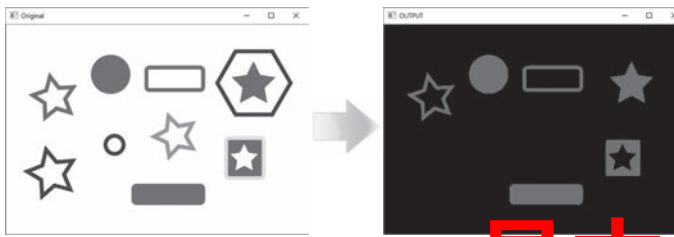
## 1-7　色によるマスク処理

フォルダ名：**07HSVmask**
リンク・ファイル：opencv_core，opencv_highgui，opencv_imgproc，opencv_imgcodecs

ここではHSVカラー・モデルを使用して目的の図形を抽出する方法を紹介します．赤色成分だけを抜き出して，その他をマスク処理した例を**図22**に示します．

フル・カラー画像では，RGB（OpenCVではBGR）で色を表現するカラー・モデルがよく用いられています．HSVは，色相（Hue），彩度（Saturation），明度（Value）で表現したカラー・モデルです（**図23**）．

RGBモデルでは，通常，R，G，Bの各色を0 ～ 255で表現しています．HSVモデルでは，Hを0.0 ～ 360.0で変化し，色環に




元画像　　赤色抽出し（マスク処理）

**図22　HSVを使ったマスク処理**

ISBN978-4-7898-5049-0
C3055 ￥1600E
CQ出版社

定価 1,760円（本体1,600円）⑩

9784789850490

1923055016002

ラズベリー・パイとその専用カメラ，オープンソースの画像処理ライブラリOpenCVを利用して，色合いや形状，明るさ，動き，顔，サイズの検出を行います．さらにこれらを組み合わせて，通行者の人数やスクワットの上下回数カウントなどの画像認識にトライします．