

XG-335x

XG-3358 OpenCV サンプルプログラム解説

Rev1.0 2014/11/12

目次

1. 概要	1
1.1 はじめに.....	1
1.2 サンプルプログラム概要.....	1
1.3 開発環境.....	2
1.4 プログラム構成.....	3
2. 動作確認	4
2.1 事前準備.....	4
2.2 PC (Ubuntu)上の動作確認.....	5
2.3 XG-3358 上の動作確認.....	9
3. プログラム説明	15
3.1 クラス一覧.....	15
3.2 クラス図.....	15
3.3 OpenCV による画像処理.....	16

表記

●バージョンに関する表記

弊社提供のソース等に関しては、弊社の管理するバージョン番号がファイル名やフォルダ名に付いている場合があります。そのバージョン番号に関しては、本ドキュメントでは、『X』を使用して表現しております。そのため、以下のような表記になりますので、その部分は読み替えてください。

例：

以下の表記がある場合

```
helloworld-X.X.tar.bz2
```

Ver1.0 での実際のファイル名は、以下になります。

```
helloworld-1.0.tar.bz2
```

●コマンドラインの表記

本ドキュメントには、コマンドラインで入力する操作手順が記載されております。操作は PC 及び XG ボードで行います。それぞれの記述について以下に記載します。

ゲスト OS(Ubuntu)での操作


プロンプトは、『\$』で記載します。

実際のプロンプトには、カレントディレクトリ等が表示されますが、本ドキュメントでは省略します。

なお、省略時には、コマンドプロンプトの前に、**省略** と表記します。

XG ボード上の Linux での操作


プロンプトは、『#』で記載します。

本ドキュメント中での入力では、以下のように表現し、入力の最後には、 があります。


例：ゲスト OS(Ubuntu)上で make コマンドを実行する場合の表記

```
省略 $ make 
```

コマンドによっては 1 つのコマンドが複数行で記載されている場合もあります。

その場合には、2 行目以降の入力では ENTER キーを押さずに続けて入力し、 の表記がある行の最後で ENTER キーを入力してそのコマンドを実行してください。

例：2 行続いてコマンド入力がある表記

```
省略 $ mkimage -A arm -O linux -T ramdisk -C gzip -d output/images/rootfs.cpio.gz output/images/uInitrd-xg3730 
```

●ソースコードの表記

ソースコードを表示して説明する場合があります。以下の表記になります。先頭に行番号を表示します。

例：

```
95: cv::morphologyEx(img, img, CV_MOP_OPEN, cv::Mat(), cv::Point(-1,-1), 1);
```

1. 概要

1.1 はじめに

本ドキュメントでは、弊社製の XG-3358 上で動作する OpenCV サンプルプログラムについて解説します。

- ・ サンプルプログラム動作確認手順
 - PC 上でのサンプルプログラム実行手順
 - XG-3358 上のサンプルプログラム実行手順
- ・ プログラム説明



本ドキュメントでは、VirtualBox 含めた開発環境が WindowsPC にインストールされていることが前提となっています。Qt、OpenCV のための Linux カーネル作成・クロス開発環境が必要です。次のマニュアルを参照して、先に環境設定を行ってください。

『Linux 開発 インストールマニュアル for XG-335x』、『LK-3358 Linux ソフトウェアマニュアル』、『Qt のためのルートファイルシステム作成方法』、『Qt(4.8.5)プログラミング』、『OpenCV アプリケーション開発』、『Qt+OpenCV の PC 用アプリケーション開発』。

1.2 サンプルプログラム概要

基板画像からビス止め用の穴を検出する画像処理プログラムです。

画像処理に Open CV を、GUI は Qt を利用しています。

- | | |
|----------|---|
| 「読み…」ボタン | : 基板画像ファイルを選択し読み込み、画像を表示します。 |
| 「検出」ボタン | : 画像処理が行われ、検出された基板枠を黄枠、ビス止め穴を赤丸で描画します。
検出に要した時間を表示します。 |

1.3 開発環境

ソフトウェア構成

Ubuntu 12.04 LTS
Qt Creator 2.4.1

<PC 環境>

Qt 4.8.1
OpenCV 2.4.9

<XG 環境>

Qt 4.8.5
OpenCV 2.4.9

ハードウェア構成

- ・ホスト PC
- ・XG-3358
- ・LCD-KIT-B01 または LCD-KIT-C01
- ・PC-USB-04
- ・接続ケーブル類

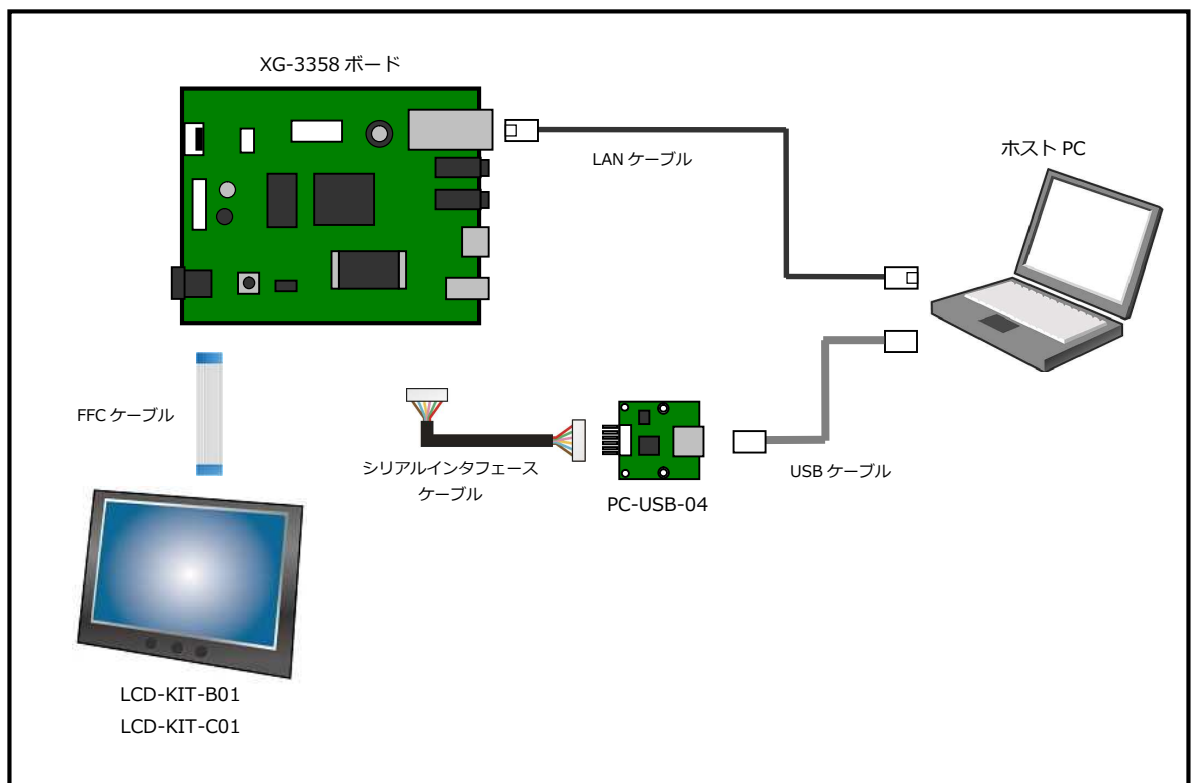


Fig 1.3-1 ハードウェア構成図

1.4 プログラム構成

ファイル構成

ap_cv_sample_X.X.tar.bz2 OpenCV サンプルプログラム（ソース、画像 など）の圧縮ファイル

圧縮ファイル内は以下のファイルで構成されています。

sample/	OpenCV サンプルプログラム ソース一式
holedetector.cpp	基板穴検出クラス ソース (OpenCV 処理)
holedetector.h	基板穴検出クラス ヘッダ
mainwindow.cpp	メインウィンドウクラス ソース
mainwindow.h	メインウィンドウクラス ヘッダ
mainwindow.ui	メインウィンドウクラス ui
main.cpp	プログラムメイン
sample.pro	Qt プロジェクトファイル
data/	OpenCV サンプルプログラム用 画像ファイル (jpeg、1280×960)
ap_board1.jpg	基板画像 1
ap_board2.jpg	基板画像 2
ap_board3.jpg	基板画像 3
exe/	実行モジュール フォルダ
sample_xg3358	XG-3358 用 実行モジュール

2. 動作確認

本章では、サンプルプログラムのビルド・実行までの手順を説明します。

- ・事前準備
- ・PC(Ubuntu)上での動作確認
- ・XG-3358 上の動作確認

2.1 事前準備

環境設定

XG-3358 をターゲットとする Qt+OpenCV アプリのプログラム開発・実行するには以下に挙げる準備が必要となります。

- ① XG-3358 で稼働させるカスタマイズをしたカーネルの作成 (LCD グラフィック表示など)
- ② Qt・OpenCV を動かすために必要なライブラリをインストールしたルートファイルシステムの作成
- ③ Ubuntu 上でクロス開発するための開発環境の設定 (Qt など)
- ④ Ubuntu に OpenCV 2.4.9 をインストール (PC 上での動作確認に必要)

本ドキュメントでの詳細解説は行いませんので、各マニュアルを参照してください。

- ・「Linux 開発インストールマニュアル」
- ・「XG-3358 Linux ソフトウェアマニュアル」
- ・「Qt のためのルートファイルシステム作成方法」
- ・「Qt(4.8.5)プログラミング」
- ・「OpenCV アプリケーション開発」
- ・「Qt+OpenCV の PC 用アプリケーション開発」

<ネットワーク設定>

XG-3358 で実行確認する際には、ネットワーク設定が必要です。

本ドキュメントではネットワークの IP アドレスは次の値であるものとして説明します。適宜、読み替えてください。

PC(Ubuntu)	:	192.168.128.210
XG-3358	:	192.168.128.200

サンプルファイル準備

“ap_cv_sample_X.X.tar.bz2” を PC (Ubuntu) 上の作業ディレクトリに解凍しておきます。

2.2 PC (Ubuntu)上の動作確認

XG-3358 上での動作確認の前に、開発 PC (Ubuntu) での実行の手順を説明します。

プロジェクトを開く

- ① Qt Creator で、sample.pro ファイルを開きます。
「ようこそ」画面の「プロジェクトを開く…」ボタンを押すとファイル選択ダイアログが現れますので、サンプルプログラムフォルダ内の sample.pro を指定します。

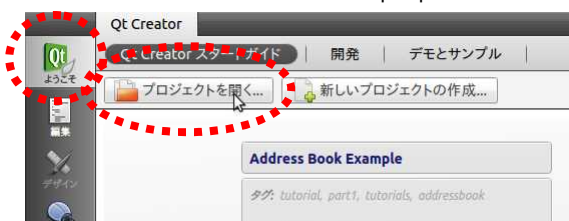


Fig 2.2-1 「プロジェクトを開く…」

- ② ファイル選択後に現れる「プロジェクト設定」画面では、デフォルトのままに「完了」します。



Fig 2.2-2 「プロジェクト設定」画面

- ③ sample プロジェクトファイルが開かれます。

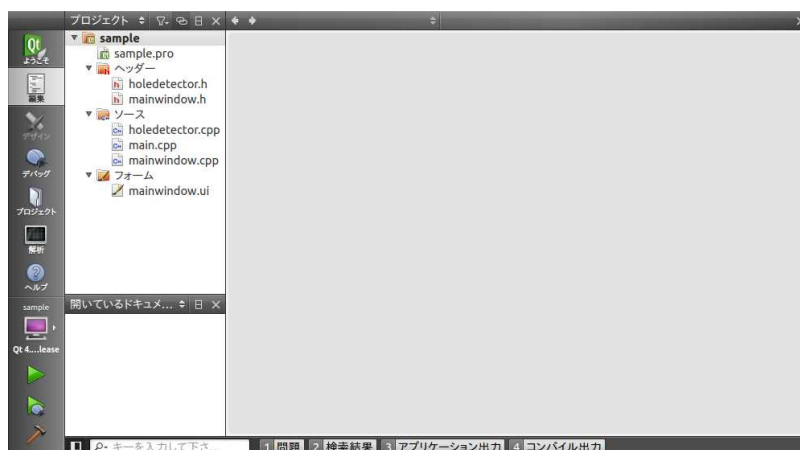


Fig 2.2-3 プロジェクトオープン後

ビルド

- ① ビルド構成を PC(Ubuntu)用に設定します。初期状態では下図のように、“Qt 4.8.5 (XG-3358) Release” が選択されていますので、“PATH(システム)に含まれる Qt 4.8.1 Release” に変更します。

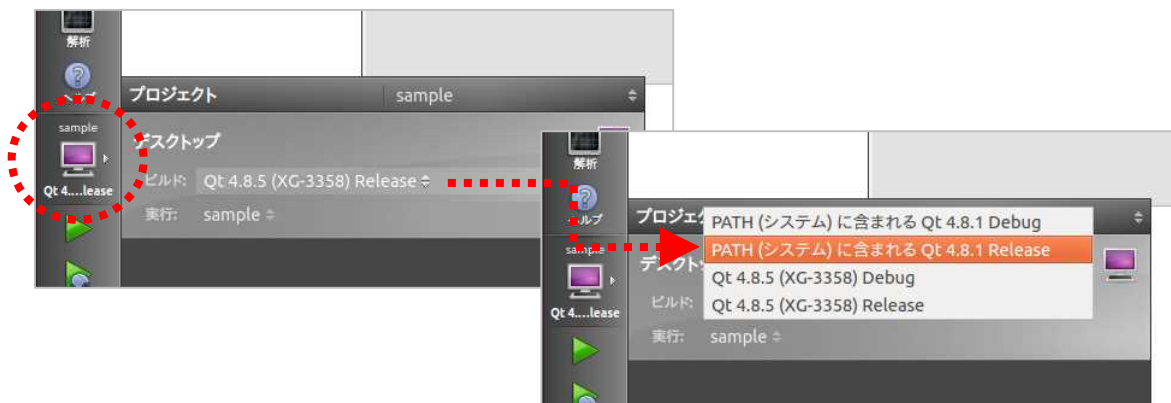


Fig 2.2-4 ターゲット変更

- ② Qt Creator 左下の「ビルド」ボタンを押してビルドします。



Fig 2.2-5 「ビルド」ボタン

- ③ ビルドに成功すると、「コンパイル出力」の画面に“…正常に終了しました”と表示されます。

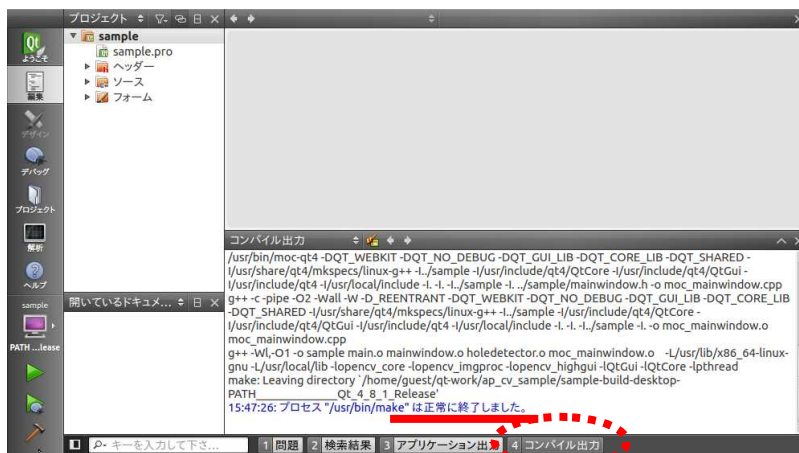


Fig 2.2-6 ビルドが正常終了

実行（起動）

- ① Qt Creator 左下の実行ボタン(緑の三角アイコン)を押して実行スタートします。



Fig 2.2-7 「実行」ボタン

- ② PC(Ubuntu)上でサンプルプログラムが起動します。



Fig 2.2-8 サンプルプログラムの実行初期画面

実行（検出）

本サンプルプログラムに付属している画像ファイルを使います。

- ① 「読み込み」ボタンを押して、画像ファイルを選択します。
data フォルダに3つのファイルがあります。まず、“ap_board1.jpg”を選択します。

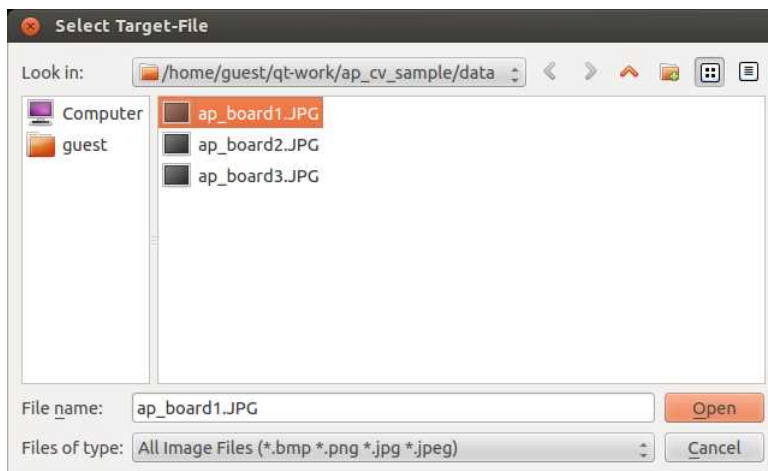


Fig 2.2-9 「読込…」の画像ファイル選択ダイアログ

- ② 画像ファイルが読み込まれ、画面上に画像が表示されます。

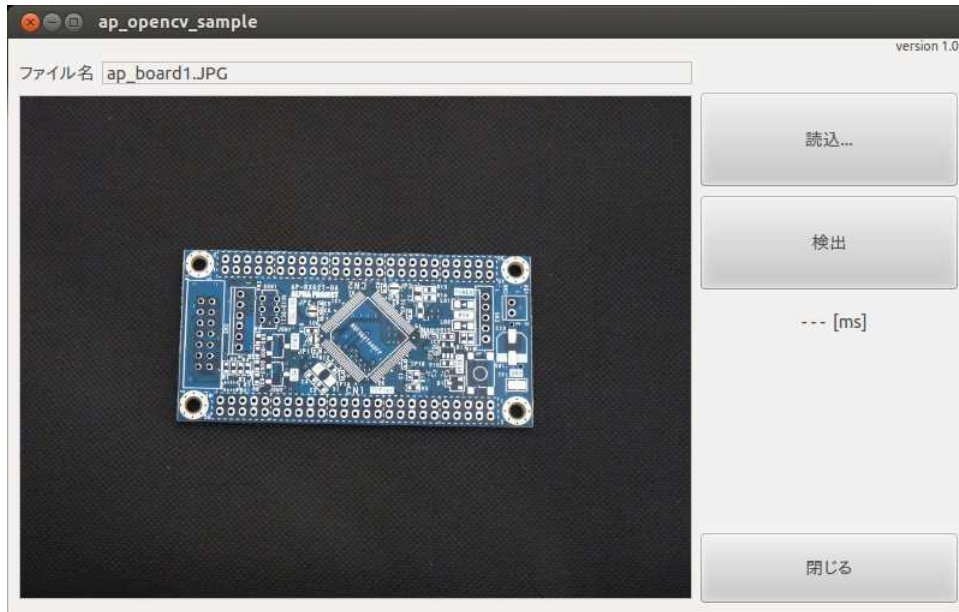


Fig 2.2-10 画像ファイル読込後

- ③ 「検出」ボタンを押すと、基板穴検出が行われ、結果が画面上に表示されます。
- ・ 基板枠 = 黄枠、ビス止め穴 = 赤い丸 で表示されます。
 - ・ 検出処理に要した時間が「検出」ボタンの下に表示されます。

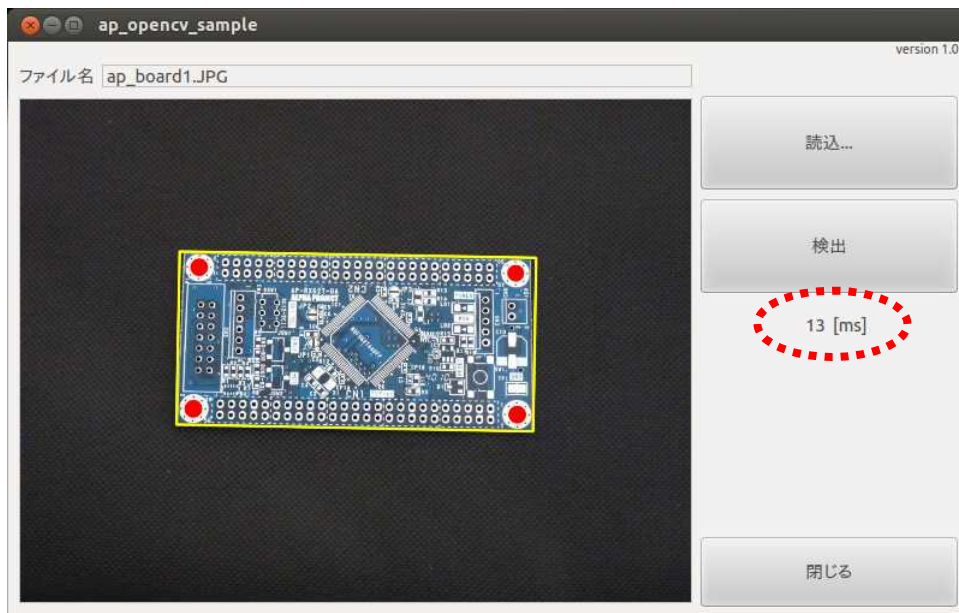


Fig 2.2-11 検出後

OpenCV を利用した基板位置検出とビス穴検出の画像処理が行われています。
OpenCV 画像処理の解説は第 3 章をご覧ください。

付属の他のサンプル画像でも同様の処理が可能です。

2.3 XG-3358 上の動作確認

XG-3358 用のビルドと、XG-3358 で実行するまでの手順を説明します。

ここで説明する実行方法は、実行バイナリファイルを開発 PC(NFS サーバ)側の/nfs ディレクトリに置き、XG-3358(NFS クライアント)は NFS マウントしたディレクトリ内に存在するファイルを実行します。

ビルド設定変更

XG-3358 用のビルド設定を変更します。

ビルド後にビルドされた実行モジュールを XG-3358 が NFS マウントするディレクトリにコピーする設定をします。

- ① プロジェクトモードに切り替えて、「ビルド構成を編集:」を「Qt 4.8.5 (XG-3358) Release」に変更します。



Fig 2.3-1 ビルド設定の編集対象を変更

- ② “Qt 4.8.5 (XG-3358) Release” のビルド構成が表示されます。



Fig 2.3-2 XG-3358 のビルド設定

ビルドディレクトリが赤く表示されていますが、この時点でディレクトリがまだないためです。問題ありませんので、このまま進めてください。

「概要」の「Qtバージョン」が“Qt 4.8.5 (XG-3358)”となっているはずですが。

- ③ 「ビルドステップ」の下にある、「ビルドステップを追加」ボタンを押して“独自プロセスステップ”を選択します。



Fig 2.3-3 独自プロセスステップの追加

- ④ 独自プロセスステップの入力欄が新しくできますので、次の設定をします。
- ・「独自プロセスステップの有効化」のチェックボックスをチェックします。
 - ・「コマンド」欄に、“cp” と入力します。
 - ・「コマンド引数」欄に、“sample /nfs/ap_cv_sample” と入力します。

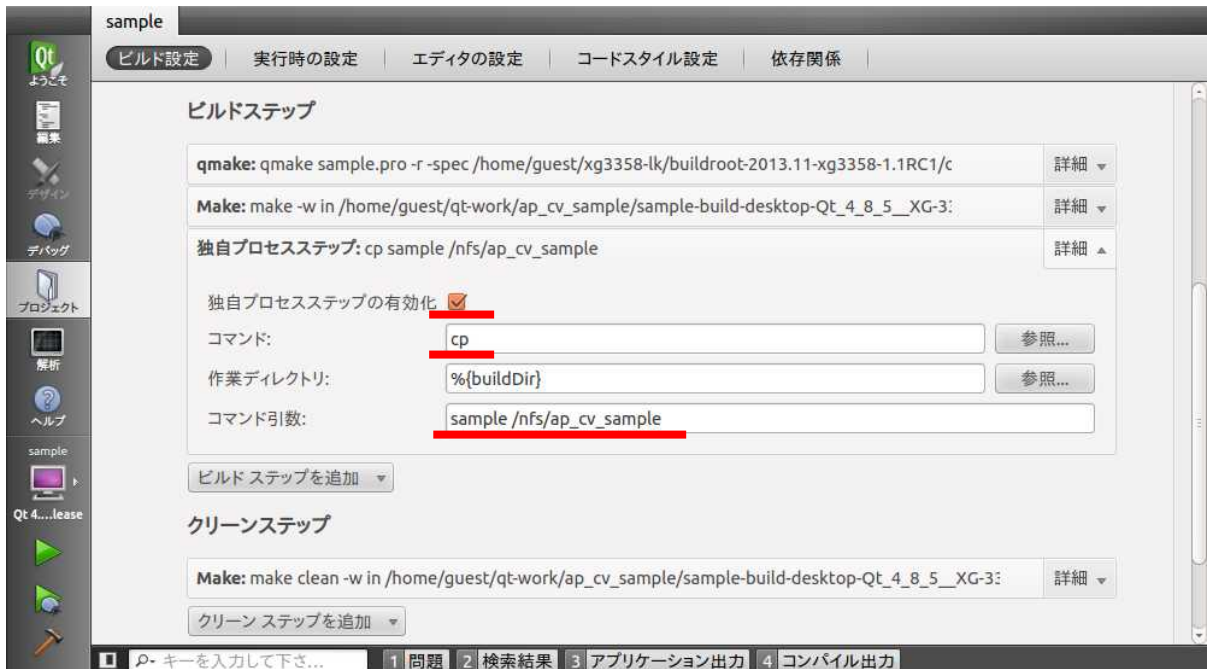


Fig 2.3-4 独自プロセスステップの編集

ビルド

- ① Qt Creator 左下の「ビルド」ボタンを押してビルドします。



Fig 2.3-5 「ビルド」ボタン

- ② 「コンパイル出力」のログを見ると、“make は正常に終了しました。”と“cp は正常に終了しました。”と表示されていることが確認できます。

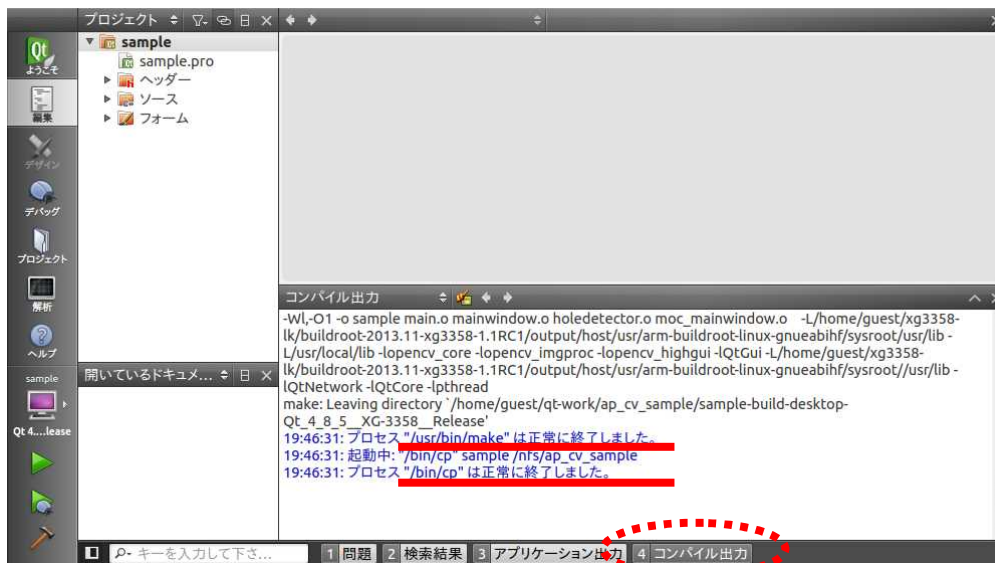


Fig 2.3-6 独自プロセスステップの編集

- ③ 念のため、ビルドファイルが/nfs/ap_cv_sample にコピーされていることを確認します。
例えば、PC(Ubuntu)のターミナルで次のように入力します。“-l” オプションでタイムスタンプも確認します。

```
省略 $ ls /nfs/ap_cv_sample -l ← 入力
-rwxrwxr-x 1 guest guest SSSSSS MM DD HH:MM /nfs/ap_cv_sample
```

上記の“SSSSSS MM DD HH:MM”の部分は、サイズ・日時が表示されます。

実行（起動）

- ① PC(Ubuntu)の/nfsディレクトリを XG-3358 の /mnt/nfs ディレクトリ にマウントします。
既にマウントしている場合は不要です。

```
# mount -t nfs -o nolock 192.168.128.210:/nfs /mnt/nfs
```

- ② ap_cv_sample ファイルが存在することを確認します。

```
# cd /mnt/nfs ←入力
# ls ap_cv_sample ←入力
ap_cv_sample
```

- ③ ap_cv_sample を実行します。

```
# ./ap_cv_sample -qws ←入力
```

- ④ 次のような画面が表示されます。

PC(Ubuntu)上での実行時とは異なり、タイトルバーがありません。



Fig 2.3-7 サンプルプログラムの実行初期画面

実行（検出）

ここからの検出実行は PC(Ubuntu)での実行と変わりません。同じように実行できること確認します。

- ① 検出に使用する画像はあらかじめ XG-3358 でアクセスできる場所にコピーしておきます。
ここでは、NFS の /mnt/nfs/ap_images ディレクトリに置いてあるとして説明します。
(XG-3358 の /mnt/nfs/ap_images ディレクトリ = PC(Ubuntu)の /nfs/ap_images ディレクトリ です)

- ② 「読み…」 ボタンをタップして、画像ファイルを選択します。

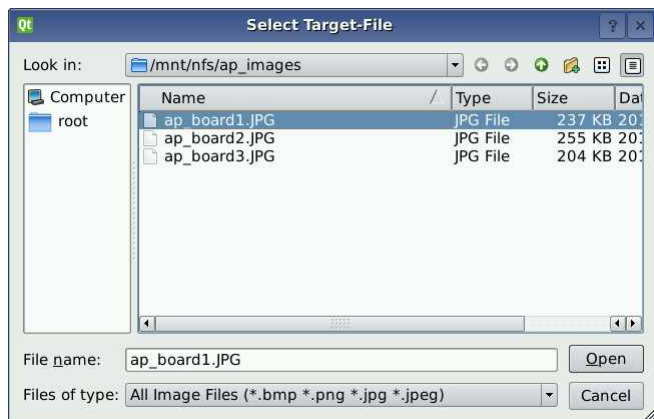


Fig 2.3-8 「読み…」の画像ファイル選択ダイアログ

- ③ 選択した画像ファイルが読み込まれ、画面上に表示されます。

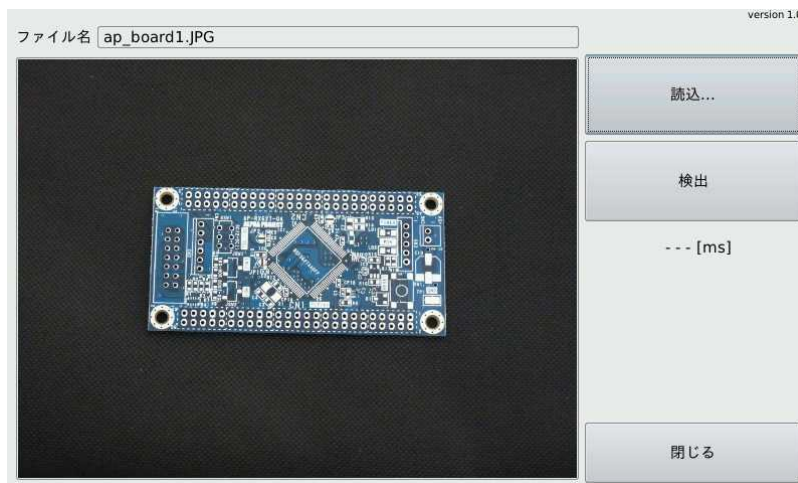


Fig 2.3-9 画像ファイル読み込後

- ④ 「検出」 ボタンをタップして検出処理を実行します。検出結果が基板画像上に表示されます。

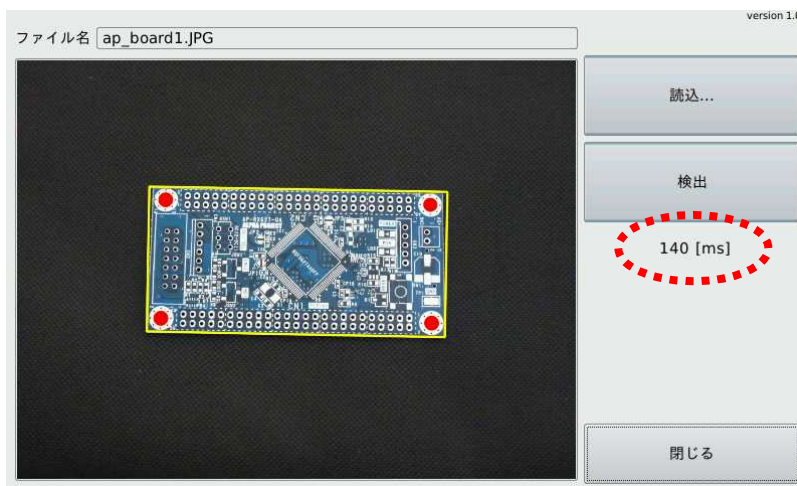


Fig 2.3-10 検出後

3. プログラム説明

本章では、サンプルプログラムの内部処理について説明します。

3.1 クラス一覧

構成するクラスは以下の通りです。

クラス名	定義ファイル	説明
HoleDetector	holedetector.cpp holedetector.h	基板穴検出クラス。 検出元となる基板画像から、基板枠の位置・ビス止め穴位置を検出します。 検出には OpenCV による画像処理を行います。
MainWindow	mainwinodow.cpp mainwinodow.h mainwinodow.ui	本サンプルプログラムのメインウィンドウクラス。 QMainWindow の派生クラス。 元画像の選択と表示、検出結果の表示を行います。 画面サイズは 800×480 固定です。
DetectThread	mainwinodow.h	検出用のスレッドクラス。QThread の派生クラス。 メインウィンドウの検出ボタン押下の 1 回毎に 1 個生成され、検出が終了すると破棄されます。

3.2 クラス図

クラス関連図を示します。(変数とメソッドは、主要なメンバのみ記載します)

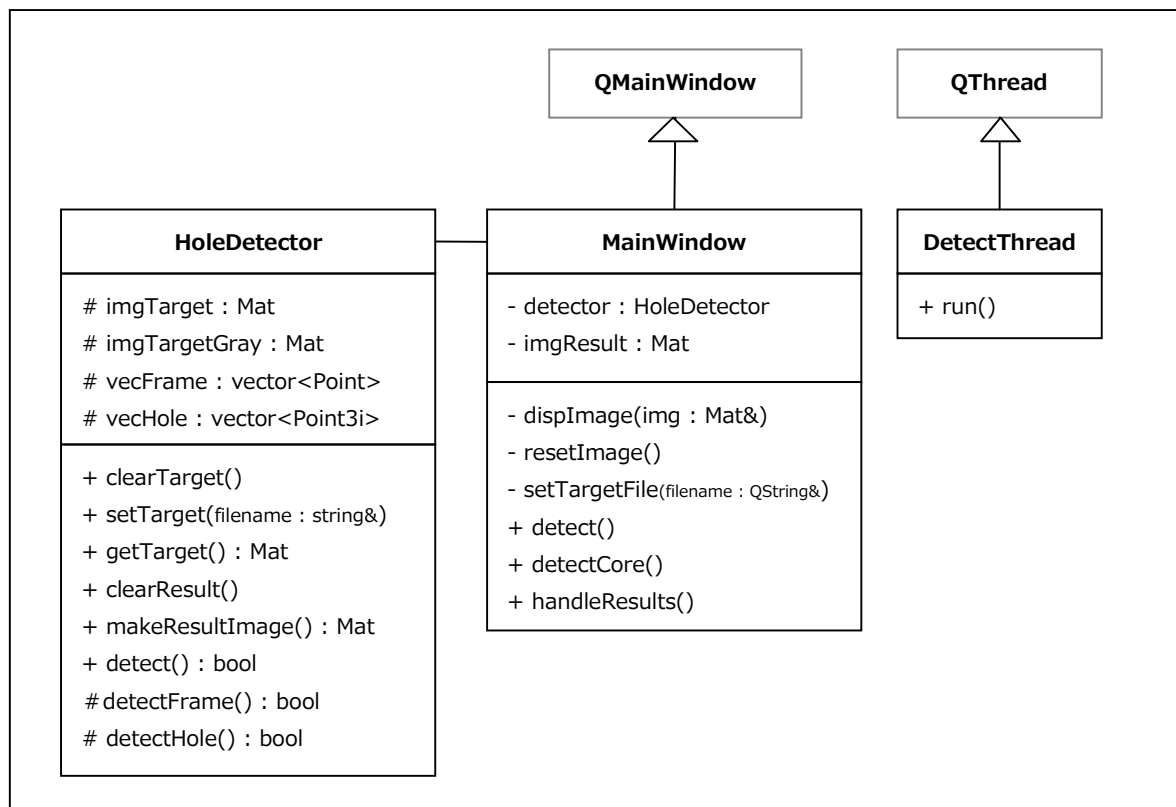


Fig 3.2-1 クラス図

3.3 OpenCV による画像処理

基板穴検出処理は HoleDetector クラスで行っています。

下図の ap-board1.jpg を例に、基板穴検出の画像処理について説明します。

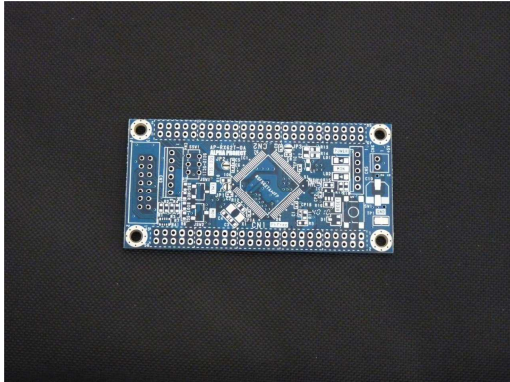


Fig 3.3-1 元画像 ap-board1.jpg

本サンプルプログラムでは、二値化と輪郭線抽出を利用して検出処理を実現していますが、検出には様々な手法を使うことができます。ハフ変換による直線検出・円検出なども今回のケースでは有用な手法です。また、基板の傾き補正を行っていませんが、入力撮影条件によっては、必要になる場合もあります。

本サンプルプログラムで行っている手法は、ある条件下での一例として参考にしてください。

画面に表示している処理時間は、ここで説明する基板枠検出とビス穴検出にかかる時間です。

基板枠検出

基板枠検出は、HoleDetector クラスの detectFrame() メソッドで行います。定義ファイルは holedetector.cpp です。

① グレースケール変換

```
88: cvtColor(imgTarget, imgTargetGray, CV_BGR2GRAY);
```

元画像はカラー画像ですが、色情報は使用しません。

(ビス穴検出処理でも、ここで変換したグレースケール変換画像を使用します。)

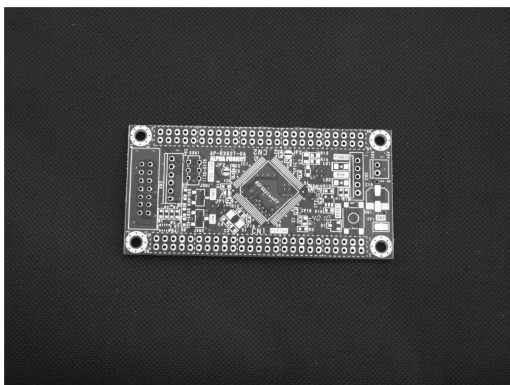


Fig 3.3-2 グレースケール変換

② 二値化

```
92: cv::threshold(imgTargetGray, img, kBrightnessMin, 255, CV_THRESH_BINARY);
```

二値化しきい値は定数の kBrightnessMin(=70)を与えています。

各ピクセル値が、70 以上なら 255(白)に、70 未満なら 0(黒)のどちらかに変換します。

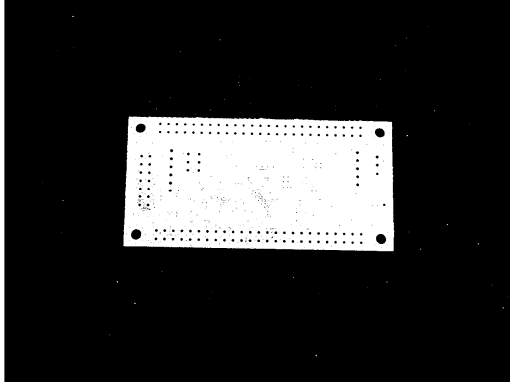


Fig 3.3 -3 二値化

(このしきい値は、添付のサンプル画像に適した値になっています。)

③ ノイズ除去

```
95: cv::morphologyEx(img, img, CV_MOP_OPEN, cv::Mat(), cv::Point(-1,-1), 1);
```

1 画素分のオープニング処理を行います。

オープニング処理とは、収縮 (Erosion) して膨張(Dilation)する処理です。

白い部分が 1 画素分収縮して 1 画素分膨張するので、白い小さいノイズが除去されます。

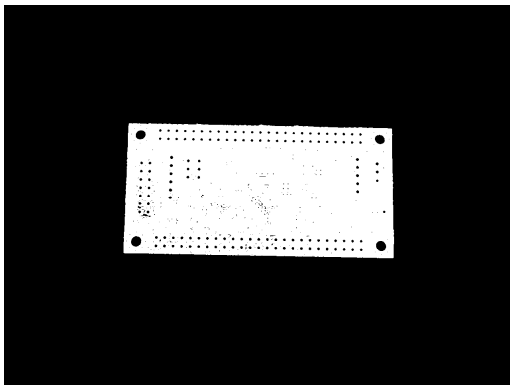


Fig 3.3 -4 ノイズ除去

④ 輪郭抽出

```

99:     std::vector<std::vector<cv::Point> > contours;
100:    std::vector<cv::Vec4i> hierarchy;
101:    cv::findContours(img, contours, hierarchy, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);

```

cv::RETR_EXTERNAL を指定して、最も外側の輪郭のみを抽出しています。

第二引数の contours に検出された輪郭点列が返ります。

下図は cv::findContours で求めた輪郭線を描画した図です。この画像のケースでは 1 個の輪郭のみです。

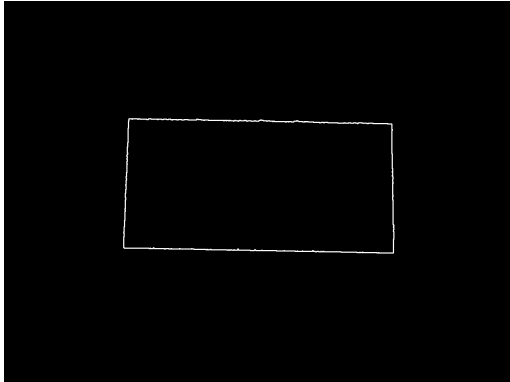


Fig 3.3-5 輪郭抽出

⑤ 最大の輪郭線を判別

```

110:    cv::Rect rect = cv::boundingRect(contours[idx]);
111:    double value = rect.area();

```

各輪郭線の輪郭点列から面積を計算し、最大の輪郭線を判別します。

- ・ boundingRect() : 輪郭線点列の取り囲む矩形を求める
- ・ area() : 矩形の面積を求める

下図は基板枠の輪郭点列と判定された輪郭線を描画した図です。

この画像ケースでは前の処理で求めた輪郭線が 1 個だけですので、前の図と変わりません。

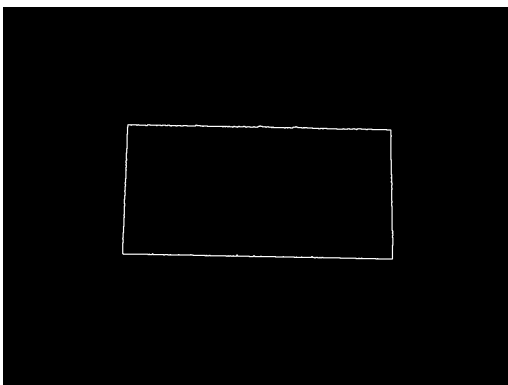


Fig 3.3-6 最大輪郭線

⑥ 基板枠輪郭線を確定

```
121:     vecFrame = contours[indexHit]; // 基板部分の点列
```

最大輪郭と判定した点列を vecFrame メンバに格納して基板枠検出は完了します。

下図は、「輪郭線が収まる最小矩形」を青色枠で、「回転を考慮した矩形」を黄色枠で描画した画像です。

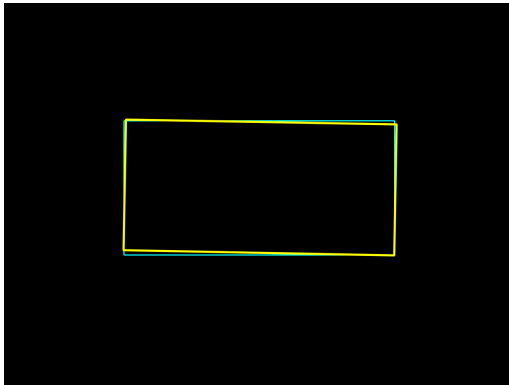


Fig 3.3-7 基板枠検出結果

ビス穴検出

基板枠の検出が成功したら、ビス穴検出処理を行います。

ビス穴検出は、HoleDetector クラスの detectFrame() メソッドです。

ビス穴は、基板のふち付近にあるという条件下での処理となっています。

① 基板部分の切り出し

```
134:   cv::Rect rect = cv::boundingRect(vecFrame);
135:   cv::Mat img = imgTargetGray(rect).clone();
```

vecFrame は先の基板枠検出で求めた基板外形の点列です。cv::boundingRect で点列が入る最小矩形を求めています。

imgTargetGray はグレースケール画像で「Fig 3.3-2 グレースケール変換」です。

imgTargetGray(rect).clone() では、基板部分のみを切り出した画像を作成しています。

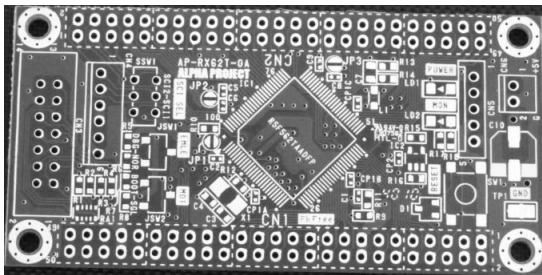


Fig 3.3-8 切り出し画像

② 検出範囲外の塗りつぶし

```
139:   int width = kHoleSearchBand;
140:   cv::Rect rectFill(width, width, img.cols - width * 2, img.rows - width * 2);
141:   cv::rectangle(img, rectFill, cv::Scalar(80), -1); // グレー(80)で塗り物す
```

検出範囲以外をグレーに塗りつぶし、後の処理で余分な輪郭線とならないようにします。

グレーの値は、次の二値化処理で黒になる値となっています。

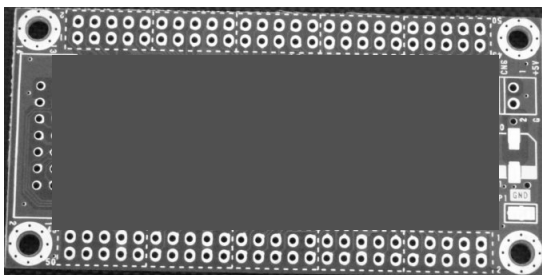


Fig 3.3-9 検出範囲外の塗りつぶし

③ 二値化 (しきい値=自動)

```
144: cv::threshold(img, img, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
```

二値化をします。

オプションフラグ CV_THRESH_OTSU を設定しているため、しきい値は画像データより自動算出されます。

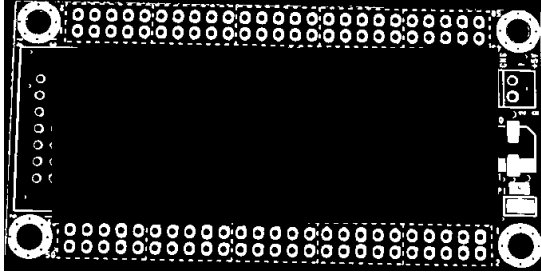


Fig 3.3-10 二値化

④ 輪郭抽出

```
149: cv::findContours(img, contours, hierarchy, cv::RETR_LIST, cv::CHAIN_APPROX_NONE);
```

cv::RETR_LIST を指定して、すべての輪郭を抽出します。

(輪郭の階層構造は利用しないので、階層構造を返さない cv::RETR_LIST にしています。)

下図は cv::findContours で求めた全ての輪郭線を描画した図です。

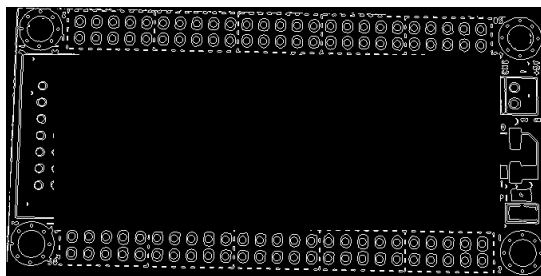


Fig 3.3-11 輪郭抽出

⑤ 輪郭線の形状・サイズから穴を判別

前の処理で求めた輪郭線群の個々の輪郭について、面積やアスペクト比を検査します。

```
155:         double area = cv::contourArea(contour);
```

cv::contourArea で輪郭線の面積を求めて、範囲内か否かを判定しています。

```
161:         cv::Rect rect = cv::boundingRect(contour);
```

cv::boundingRect で輪郭線が入る最小矩形を求め、縦横のアスペクト比を検査します。

下図は条件に入る輪郭線のみを描画した図です。このケースでは4つの輪郭線が残ります。

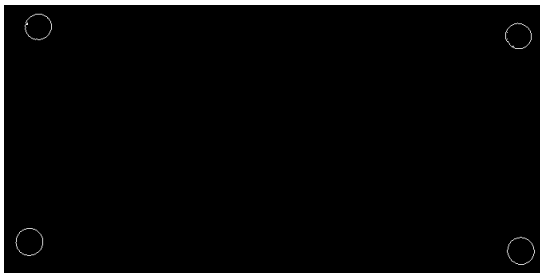


Fig 3.3-12 穴の輪郭線

⑥ 穴と判別した輪郭線から、穴の中心と半径を求める

```
170:         cv::RotatedRect rect2 = cv::minAreaRect(contour);
171:         cv::Point ptCenter = rect2.center;
172:         float radius = (rect2.size.height + rect2.size.width) / 4.0;
173:         cv::Point3i hole(ptCenter.x + ptOffset.x, ptCenter.y + ptOffset.y, radius);
```

cv::minAreaRect で回転を考慮した最小矩形を求めて、円の中心と半径を算出します。

```
176:         vecHole.push_back(hole);
```

ビス穴情報は、vecHole メンバに格納します。

下図は vecHole を赤丸で描画した図です。

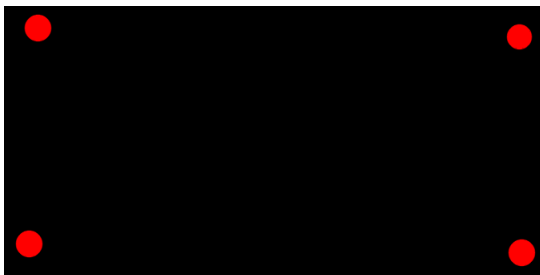


Fig 3.3-13 穴検出結果

ご注意

- ・本文書の著作権は、株式会社アルファプロジェクトが保有します。
- ・本文書の内容を無断で転載することは一切禁止します。
- ・本文書に記載されているサンプルプログラムの著作権は、株式会社アルファプロジェクトが保有します。
- ・本文書に記載されている内容およびサンプルプログラムについての技術サポートは一切受け付けておりません。
- ・本文書の内容およびサンプルプログラムに基づき、アプリケーションを運用した結果、万一損害が発生しても、弊社では一切責任を負いませんのでご了承下さい。
- ・本文書の内容については、万全を期して作成いたしました。が、万一ご不審な点、誤りなどお気付きの点がありましたら弊社までご連絡下さい。
- ・本文書の内容は、将来予告なしに変更されることがあります。

商標について

- ・VirtualBox は、Oracle Corporation の登録商標、商標または商品名称です。
- ・Linux は、Linus Torvalds の米国およびその他の国における登録商標または商標です。
- ・Windows®の正式名称は Microsoft®Windows®Operating System です。
- ・Microsoft、Windows、Windows NT は、米国 Microsoft Corporation.の米国およびその他の国における商標または登録商標です。
- ・その他の会社名、製品名は、各社の登録商標または商標です。