上級者向け講習会課題３

会津大学ＲＴミドルウェア講習会

カメラ画像から物体の輪郭を検出するコンポーネントを作成する

目次

※ 文中の「x.y」や「x.y.z」の表記は使用環境の OpenRTM-aist のバージョンに読み替えてください。

1 カメラコンポーネントの復習

今回はカメラコンポーネントを使用するのでカメラコンポーネントの起動の仕方を復習として記載します。

1.1 コンポーネントを起動する

1.1.1 NameService を起動する

コンポーネントの参照を登録するためのネームサービスを起動します。
[スタート]メニューから[すべてのプログラム]→[OpenRTM-aist x.y]→ [tools]→[Start Naming Service]をクリックしてください。

※ [Start Naming Service]をクリックしても omniNames が起動されない場合は、フルコンピュータ名が 14 文字以内に設定されているかを確認してください。

※ OpenRTM-aist C++ 1.1.1 使用の方は[Start C++ Naming Service]クリックしてください。

※ Windows8 の場合下記パスを参考にあります。
C:¥ProgramData¥Microsoft¥Windows¥Start Menu¥Programs¥OpenRTM-aist x.y¥Tools

1.1.2 カメラコンポーネントとビューアコンポーネントと Flip コンポーネントを起動する

USB カメラのキャプチャ画像を OutPort から出力する OpenCVCameraComp と InPort で受け取った画像を反転して OutPort から出力する FlipComp、InPort で受け取った画像を画面に表示する CameraViewerComp を起動します。
[スタート]メニューから[すべてのプログラム]→[OpenRTM-aist x.y]→ [C++]→[Components]→[OpenCV-Examples] 内にあるのでダブルクリックで起動してください。

・OpenCVCameraComp.exe
・FlipComp.exe
・CameraViewerComp.exe

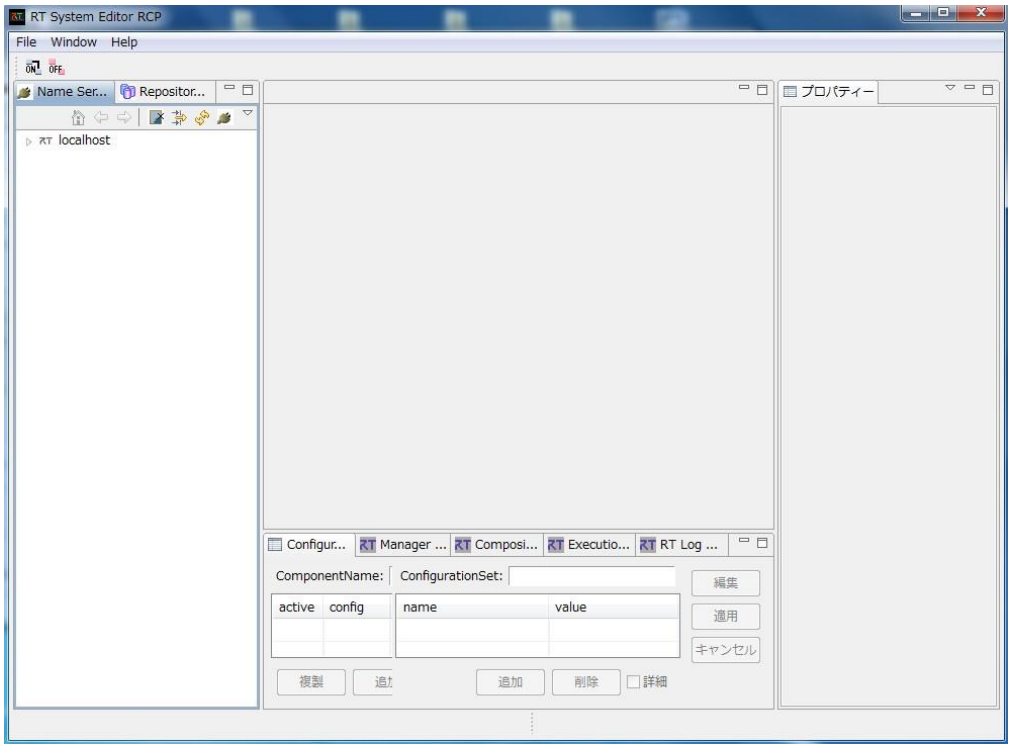
1.1.3 コンポーネントを接続する

WEB カメラから画像が取得できることを確認します。

1.1.3.1 RTSystemEditor を起動する

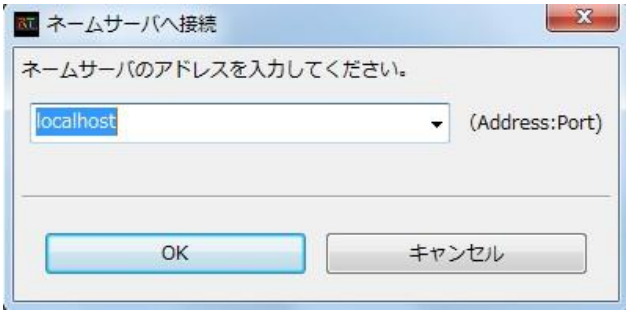
最初に RTSystemEditor を起動します。

起動方法は RTCBuilder 画面右上の「パースペクティブを開く」を選択し、さらに[その他]を選択します。そして「パースペクティブ」の中から[RT System editor]を選択して起動させます。またはスタートメニューの「OpenRTM-aist x.y」→「tools」→[RTSystemEditorRCP]から起動します。



1.1.3.2 コンポーネントを接続する

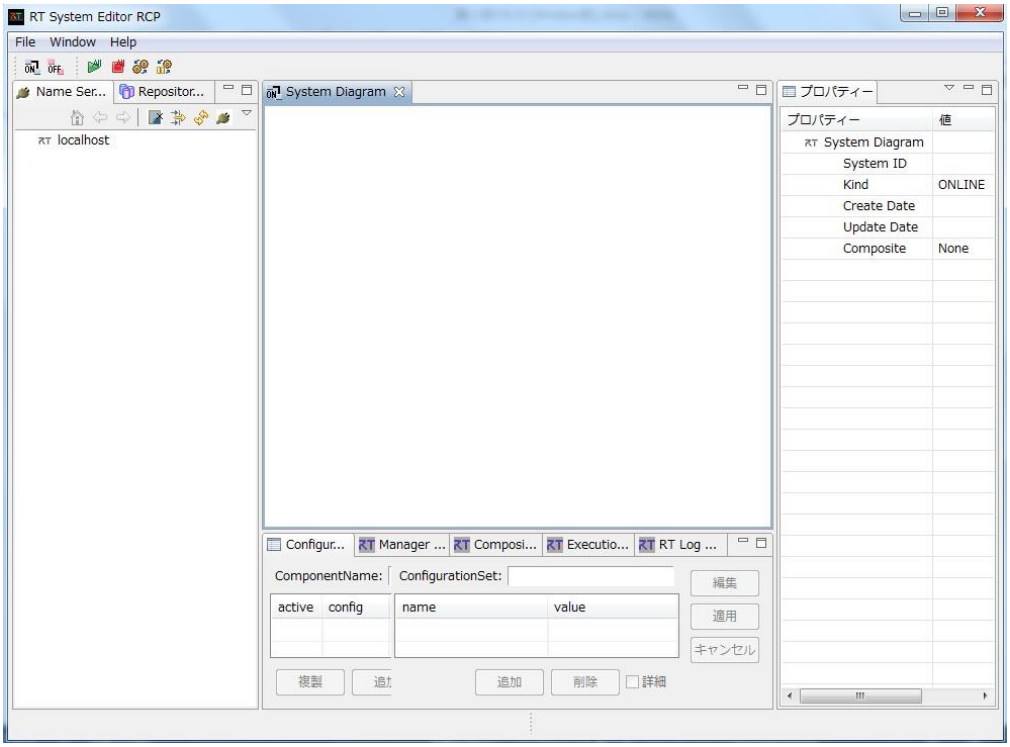
Name Service View に何も表示されていない場合は、RTSystemEditor の左側の Name Service View のコンセントアイコンをクリックし、ネームサーバへ接続します。表示された接続ダイアログに localhost と入力します。



Name Service View に[localhost]のリストが表示されます。

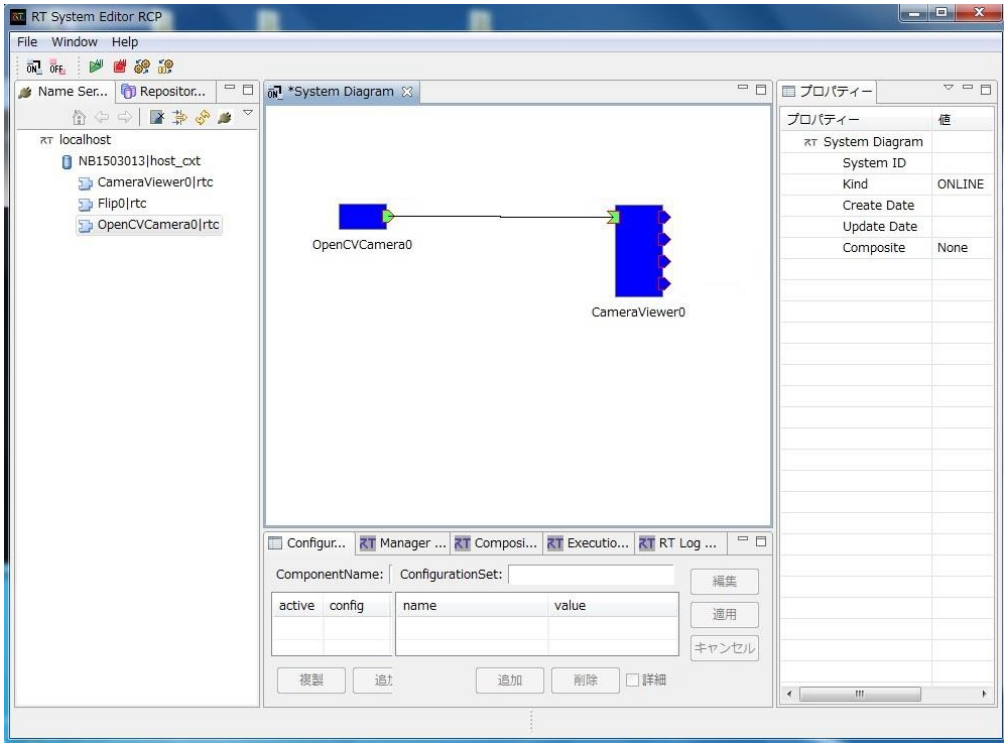
1.1.3.3 SystemDiagram

メニューバーの「Open New System Editor」ボタンをクリックし、SystemDiagram を開きます。



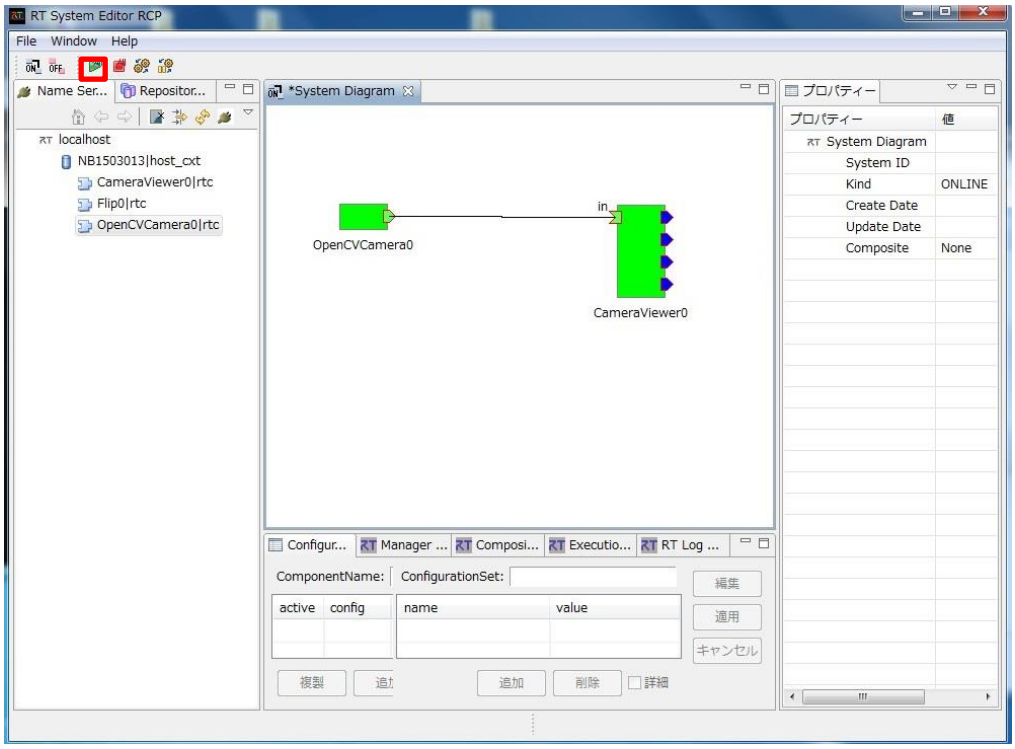
1.1.3.4 コンポーネントを配置する

次に Name Service View から OpenCVCameraComp と CameraViewerComp をドラックアンドドロップして SystemDiagram 上に配置してください。
次に、コンポーネントのデータポート同士を接続します。片方のデータポート上でドラッグすると線が伸びるので、接続したいデータポート上まで線を伸ばし接続します。
接続すると接続プロファイルが表示されるので OK をクリックします。
接続が完了すると下図の様になります。



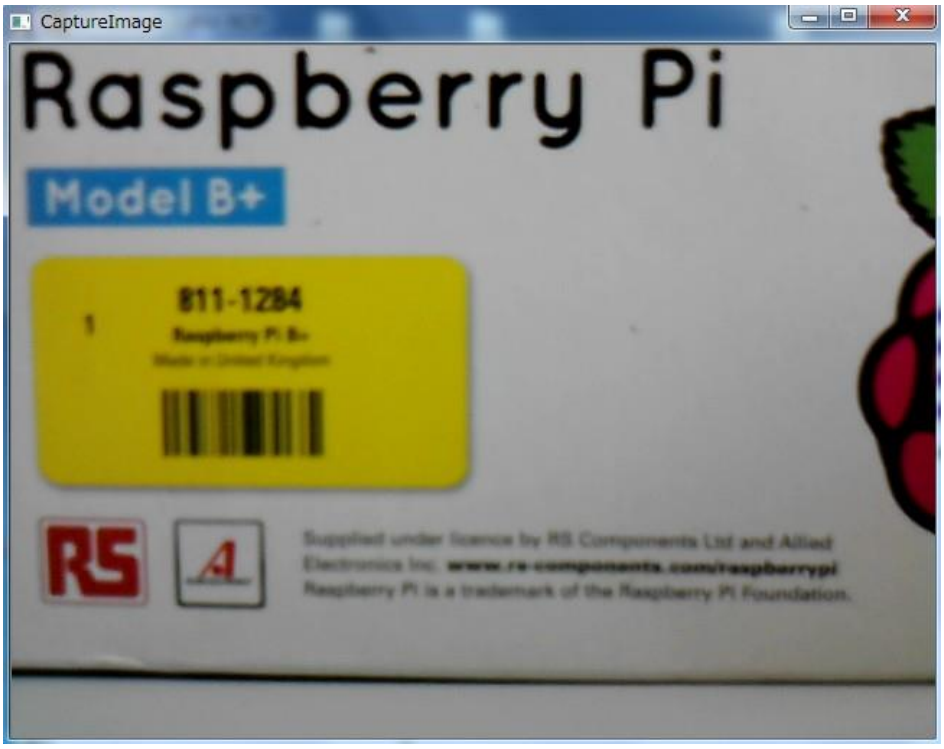
1.1.4 コンポーネントをアクティブ状態にする

RTSystemEditor の緑色の再生ボタンをクリックし、 全てのコンポーネントをアクティブにします。正常にアクティブになると、下図のように黄緑色でコンポーネントが表示されます。



1.1.5 動作確認

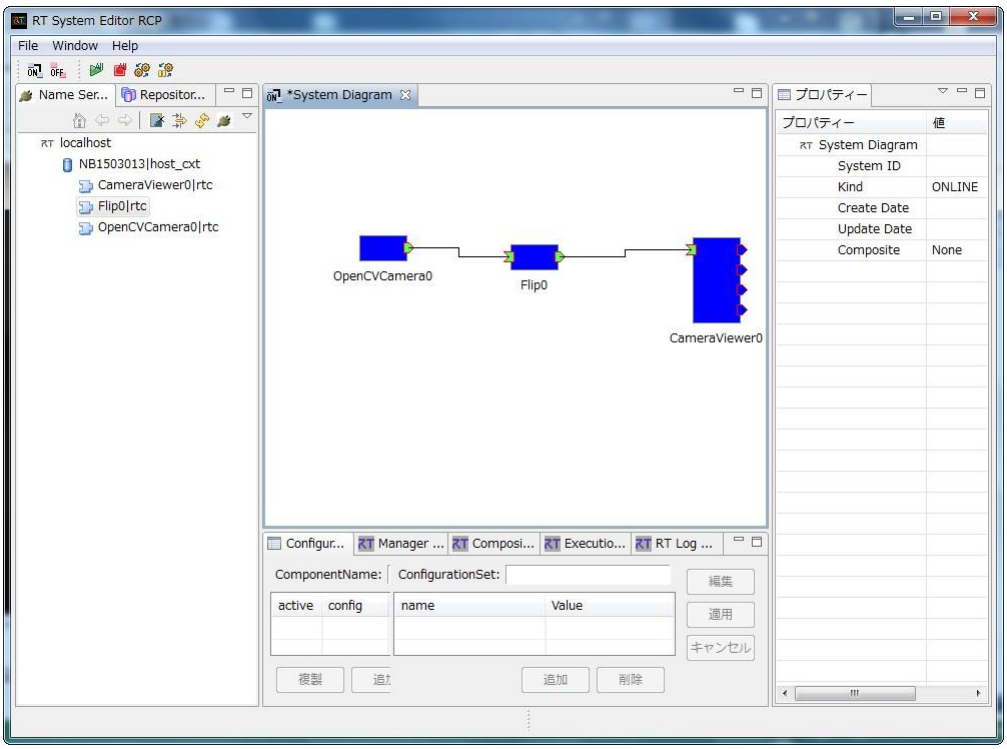
ウィンドウが出てきてカメラから取得した画像が表示されることを確認してください。
下の画像はカメラから取得したものです。色々なものを写し、画像が表示されることを確認してみてください。



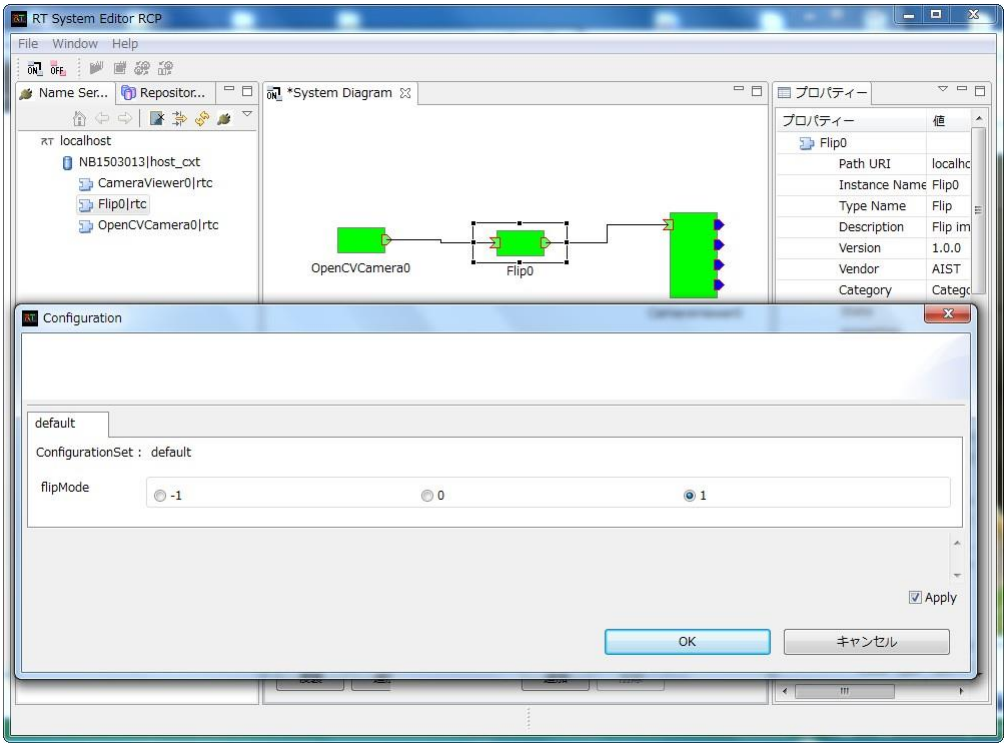
確認できましたら赤色の停止ボタンをクリックしてください。コンポーネントがディアクティブ状態になり、動作が終了します。

1.1.6 Flip コンポーネントの動作確認

線を delete で消した後、下図の様に Flip コンポーネントを挿入して緑の再生ボタンをクリックしてください。



下図のようにコンフィギュレーションビューにて実行時にコンフィギュレーションを変更することができます。Flip コンポーネントをクリックしてコンフィギュレーションビューの編集を押すと下記ダイアログが表示されます。「flipMode」を「0」や「-1」などに変更し画像が反転することを確認してください。



2 画像表示用コンポーネントのダウンロード（RTC-Library-FUKUSHIMA より）

画像表示時に使用するコンポーネントを RTC-Library-FUKUSHIMA の下記の URL からダウンロードしてください。

・UVCCameraViewer コンポーネント

https://rtc-fukushima.jp/component/2080/

サンプルコンポーネントの OpenCVCameraComp はグレースケール画像が表示されないので、グレースケール確認時にはこのコンポーネントで確認をします。

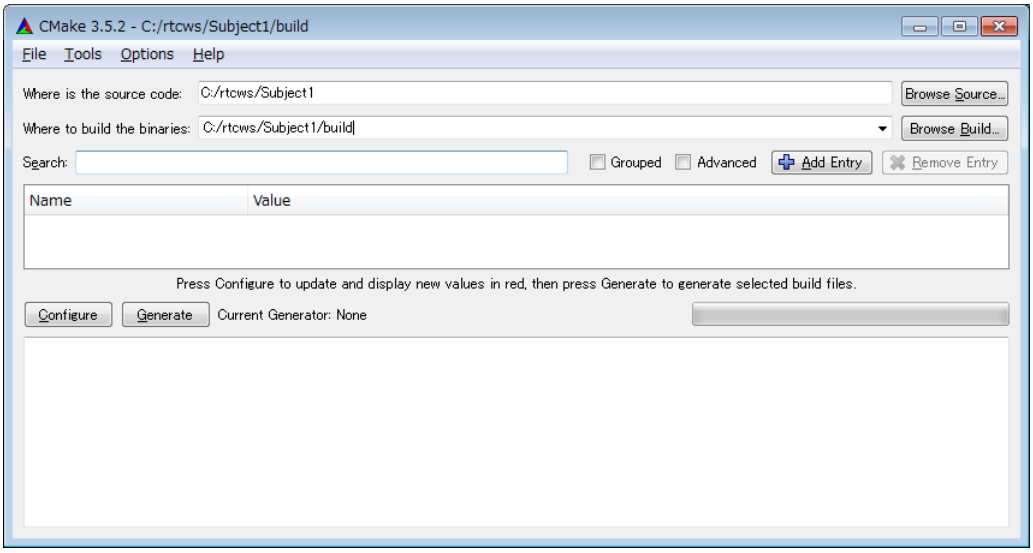
2.1 （参考）ソリューションファイルの作成

復習として実行ファイルの作成の仕方を説明します。 [UVCCameraViewer]を使用して説明します。フォルダを C ドライブ直下に置いたと仮定して話を進めます。

2.1.1 CMake

2.1.1.1 フォルダの指定

CMake を利用してビルド環境の Configure を行います。
スタートメニューなどから CMake (cmake-gui)を起動します。



画面上部に以下のようなテキストボックスがあります。

・Where is the soruce code

・Where to build the binaries

「Where is the soruce code」に CMakeList.txt が有る場所、「Where to build the binaries」にビルドディレクトリを指定します。

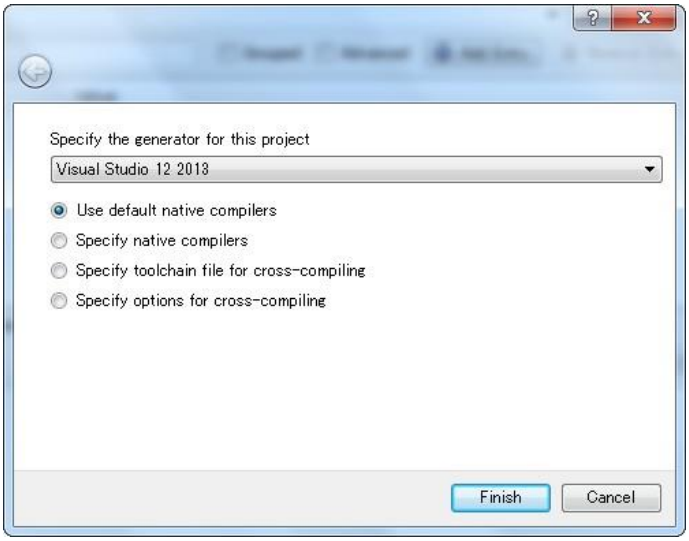
CMakeList.txt はデフォルトでは[UVCCameraViewer]のルートフォルダ内になります。
ビルドディレクトリとは、ビルドするためのプロジェクトファイル やオブジェクトファイル、バイナリを格納する場所のことです。場所は任意ですが、この場合[UVCCameraViewer /build]のように分かりやすい名前をつけた UVCCameraViewer のサブディレクトリを指定することをお勧めします。
ディレクトリは自動で作成されるので指定前に作成する必要はありません。
今回は以下の様になるはずです。

Where is the soruce code	C:¥UVCCameraViewer
Where to build the binaries	C:¥UVCCameraViewer¥build

指定したら、下の Configure ボタンを押します。

2.1.1.2 生成したいプロジェクトの種類の指定

前項で Configure ボタンを押すと、下図のようなダイアログが表示されます



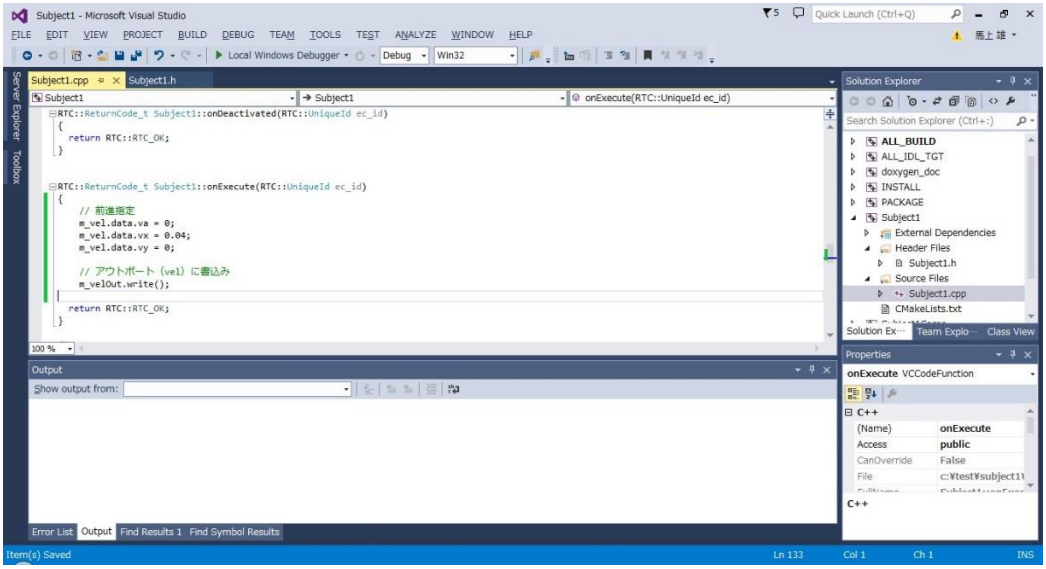
ここでは生成したいプロジェクトの種類を指定します。
生成したいプロジェクトの種類(Specify the generator for this project)とインストールされている VisualStudio の対応は以下の表の通りです。

Visual Studio バージョン	32/64 bit	生成したいプロジェクトの種類
Visual Studio 2013	32 bit	Visual Studio　12 2013
	64 bit	Visual Studio　12 2013 Win 64
Visual Studio 2015	32 bit	Visual Studio　14 2015
	64 bit	Visual Studio　14 2015 Win 64

Finish を押すと Configure が始まります。問題がなければ下部のログウインドウに「Configuring done」と出力されますので、続けて Generate ボタンを押します。
「Generating done」と出ればプロジェクトファイル・ソリューションファイル等の出力が完了します。

2.1.2 Visual Studio

次に「Where to build the binaries」で指定した build ディレクトリの中の UVCCameraViewer.sln をダブルクリックして Visual Studio を起動します。



2.1.2.1 ビルド

起動した VisualStudio のソリューションエクスプローラーで、 ALL_BUILD を右クリックしビルドを選択してビルドします。
その後作成されたコンポーネント (C:¥UVCCameraViewer¥build¥src¥Debug¥UVCCameraViewerComp.exe)をダブルクリックして起動してください。

3 システム概要

3.1 システム概要

カメラから入力された画像から箱の輪郭を検出し、それを矩形で囲うコンポーネントを作成します。

3.2 輪郭検出の手法

3.2.1 輪郭検出の手法

画像から輪郭を検出する基本的な手順は、

① 画像のグレースケール化
② グレースケール画像の 2 値化
③ 2 値化した画像から輪郭を検出

の 3 つの工程に分類されます。

① 画像のグレースケール化
入力された画像をグレースケールに変換します。グレースケール画像とはカラーチャンネルを 1 つだけ持ち、明るさを 0～255 までの 256 階調で表現したモノクロ画像です。2 値化処理を行うためにはグレースケール画像が必要になります。

② グレースケール画像の 2 値化
輪郭検出を行うためには 2 値画像が必要になります。2 値化とは閾値を設定しピクセルの値が閾値を超えた場合は白、超えない場合は黒に変換する処理です。検出対象の物体を白で表示することができれば輪郭検出が可能になります。

③ 2 値化した画像から輪郭を検出
2 値化をすると画像は白と黒の 2 色のみで表現されます。その画像から関数を利用して白い部分を上手く検出することで輪郭検出を行うことができます。

3.2.2 カラーチャンネル

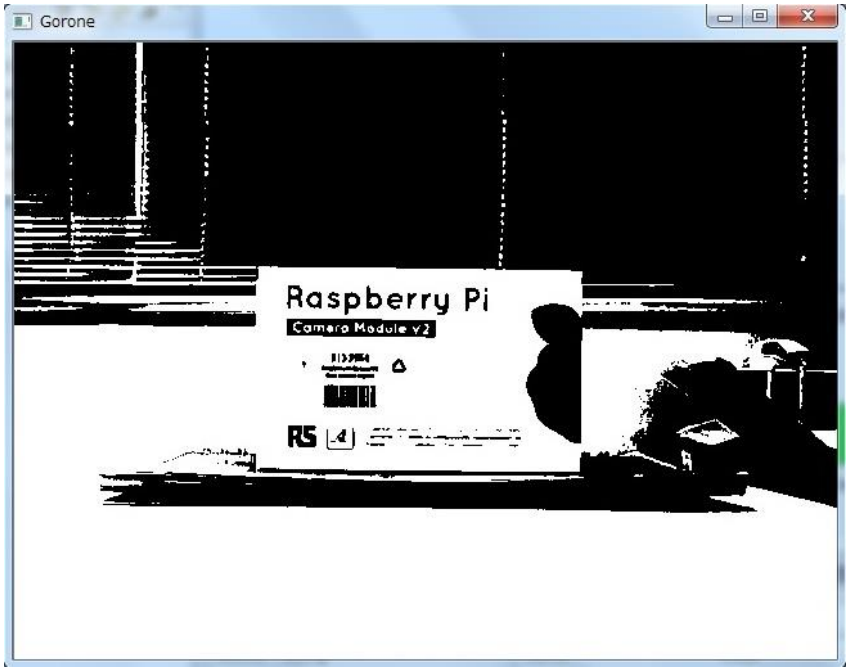
デジタルカラー画像はそれぞれ 256 階調を持つ RGB を利用して色の表現をしています。カラーチャンネルとは画像の持つ RGB それぞれの輝度値をピクセルごとに格納したものです。
カラー画像には赤チャンネル、緑チャンネル、青チャンネルの 3 つのチャンネル、グレースケール画像や 2 値画像には 1 つのチャンネルが存在します。（他にも画像の透明度を格納するアルファチャンネルもありますが今回は使用しません。）

3.2.3　平滑化フィルタを利用したノイズ除去

グレースケール画像を 2 値化すると、輪郭検出の妨げとなる細かい模様など（ノイズ）を検出してしまうことがあります。
その場合、平滑化フィルタを利用することでノイズを減らすことができます。

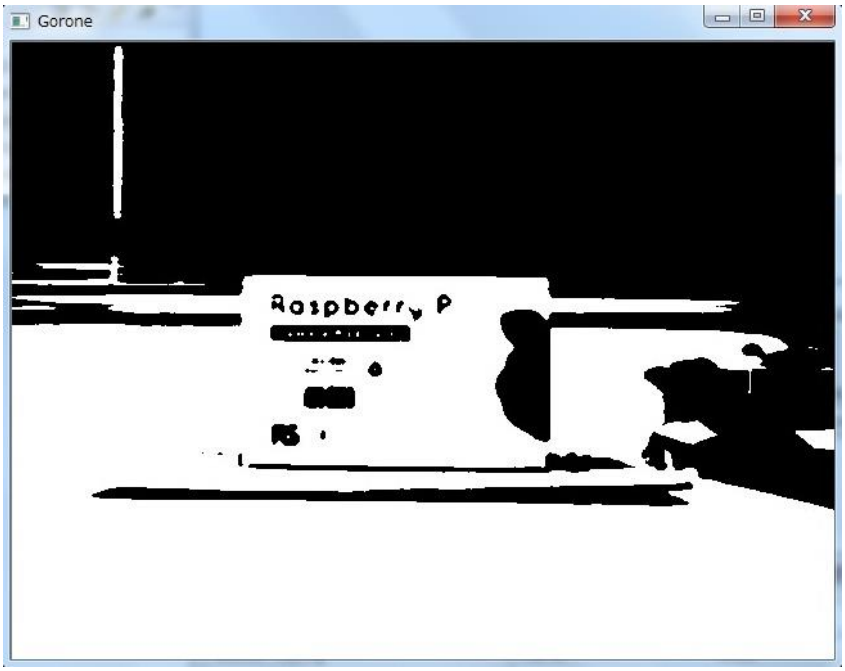
・グレースケール画像

・ノイズ処理をしていない 2 値画像



上級者向け講習会課題 3

・ノイズ処理をした 2 値画像



4 グレースケールコンポーネント作成

4.1 グレースケール化コンポーネントの仕様説明

カメラから入力されたカラー画像をグレースケール画像に変換するコンポーネントを作成します。仕様は以下の通りです。

・GrayScale コンポーネント（システム図）

GrayScale コンポーネント 仕様	
コンポーネント名	GrayScale
InPort	
ポート名	originalImage
データ型	RTC::CameraImage
変数名	originalImage
OutPort	
ポート名	grayscaledImage
データ型	RTC::CameraImage
変数名	grayscaledImage

実際にプログラムをするとき、ポート名と変数は以下の様に変化します。

InPort ポート名	InPort 変数名	OutPort ポート名	OutPort 変数名
m_ InPort 変数名 In	m_変数名	m_OutPort 変数名 Out	m_変数名

4.2 グレースケール化コンポーネントの作成

コンポーネントは以下の手順で作成します。

① RTCBuilder で雛形作成
② CMakeList 編集
③ CMake でソリューションファイルを作成
④ ソースコード編集
⑤ ビルド

4.2.1 サンプルコード

4.2.1.1 CMakeList 編集

C:¥rtcws¥GrayScale¥src¥CMakeLists.txt を編集します。
このコンポーネントでは OpenCV を利用していますので、OpenCV のヘッダのインクルードパス、ライブラリやライブラリサーチパスを与える必要が有ります。以下の 2 点を追加・変更後、CMake でソリューションファイルを作成すると OpenCV のライブラリがリンクされ使えるようになります。

① find_package(OpenCV REQUIRED)を追加

② 最初の target_link_libraries に ${OpenCV_LIBS} を追加
※target_link_libraries は 2 ヶ所あります。
※追加するときは${OpenCV_LIBS}の前に半角スペースを入れてください。

```
set(comp_srcs Flip.cpp )
set(standalone_srcs FlipComp.cpp)

find_package(OpenCV REQUIRED) ←この行を追加
  : 中略
add_dependencies(${PROJECT_NAME} ALL_IDL_TGT)
target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES} ${OpenCV_LIBS}) ←${OpenCV_LIBS}を追加
  : 中略
add_executable(${PROJECT_NAME}Comp ${standalone_srcs}
  ${comp_srcs} ${comp_headers} ${ALL_IDL_SRCS})
target_link_libraries(${PROJECT_NAME}Comp ${OPENRTM_LIBRARIES} ${OpenCV_LIBS}) ←${OpenCV_LIBS}を追加
```

4.2.1.2 GrayScale.h

OpenCV のライブラリを使用するため、OpenCV のライブラリファイルをインクルードします。下記内容をインクルードしている所に追加してください。

```
//OpenCV 用ライブラリファイルのインクルード
#include<cv.h>
#include<cxcore.h>
#include<highgui.h>
```

上級者向け講習会課題 3

画像の保存用にメンバー変数を追加します。下記内容を class の private:の中(// <rtc-template block="private_attribute">の下)に追加してください。

```
//画像保存用のメンバ変数
IplImage* m_imageBuff;
IplImage* m_grayscaleImageBuff;
```

4.2.1.3 GrayScale.cpp

カラー画像をグレースケール画像に変換するために、OpenCV のライブラリ関数「cvCvtColor」を利用します。

void cvCvtColor(const CvArr* src, CvArr* dst, int code);	
src	入力画像
dst	出力画像
code	変換方法の指定 グレースケール変換する場合は定数 CV_RGB2GRAY を指定します。

参考 URL：http://opencv.jp/opencv-2svn/c/miscellaneous_image_transformations.html?highlight=cvcvtcolor

グレースケール、又は 2 値画像用のメモリを確保する場合は、チャンネル数の変更を行う必要があります。

IplImage* cvCreateImage(CvSize size, int depth, int channels);	
size	画像の幅と高さ
depth	画素要素のビット幅
channels	利用するチャンネル数 グレースケール画像、又は 2 値画像の場合 1 を指定 カラー画像の場合 3 を指定

※作成時のヒント

下記のように、onActivated(),onDeactivated(),onExecute()を実装します。

onActivated()

```
RTC::ReturnCode_t Flip::onActivated(RTC::UniqueId ec_id)
 {
   // イメージ用メモリの初期化
   m_imageBuff = NULL;
   m_grayscaleImageBuff = NULL;

   // OutPort の画面サイズの初期化
   m_grayscaledImage.width = 0;
   m_grayscaledImage.height = 0;
   return RTC::RTC_OK;
 }
```

onDeactivated()

```
RTC::ReturnCode_t Flip::onDeactivated(RTC::UniqueId ec_id)
 {
   if(m_imageBuff != NULL)
   {
     // イメージ用メモリの解放
     cvReleaseImage(&m_imageBuff);
     cvReleaseImage(&m_grayscaleImageBuff);
   }

   return RTC::RTC_OK;
 }
```

onExecute()

下記のサンプルコードの複数行コメント部分（/*　*/）の処理を作成してください。

```
RTC::ReturnCode_t GrayScale::onExecute(RTC::UniqueId ec_id)
{
    // 新しいデータのチェック
    if (m_originalImageIn.isNew()) {
        // InPort データの読み込み
        m_originalImageIn.read();

        // InPort と OutPort の画面サイズ処理およびイメージ用メモリの確保
        if (m_originalImage.width != m_grayscaledImage.width || m_originalImage.height !=
m_grayscaledImage.height)
        {
            m_grayscaledImage.width = m_originalImage.width;
            m_grayscaledImage.height = m_originalImage.height;

            // InPort のイメージサイズが変更された場合
            if (m_imageBuff != NULL)
            {
                cvReleaseImage(&m_imageBuff);
                cvReleaseImage(&m_grayscaleImageBuff);
            }

            // イメージ用メモリの確保
            m_imageBuff = cvCreateImage(cvSize(m_originalImage.width, m_originalImage.height),
IPL_DEPTH_8U, 3);
            // グレースケール用メモリを確保
            m_grayscaleImageBuff = cvCreateImage(/*グレースケール用メモリを確保*/);
        }

        // InPort の画像データを IplImage の imageData にコピー
        memcpy(m_imageBuff->imageData, (void *)&(m_originalImage.pixels[0]),
m_originalImage.pixels.length());

        // InPort からの画像データをグレースケール化する
        /* 画像データをグレースケール化する処理を記載*/
        // 画像データのサイズ取得
        int len = m_grayscaleImageBuff->nChannels * m_grayscaleImageBuff->width * m_grayscaleImageBuff-
>height;
        m_grayscaledImage.pixels.length(len);

        // グレースケール化した画像データを OutPort にコピー
        memcpy((void *)&(m_grayscaledImage.pixels[0]), m_grayscaleImageBuff->imageData, len);

        // グレースケール化した画像データを OutPort から出力する。
        m_grayscaledImageOut.write();
    }

  return RTC::RTC_OK;
}
```

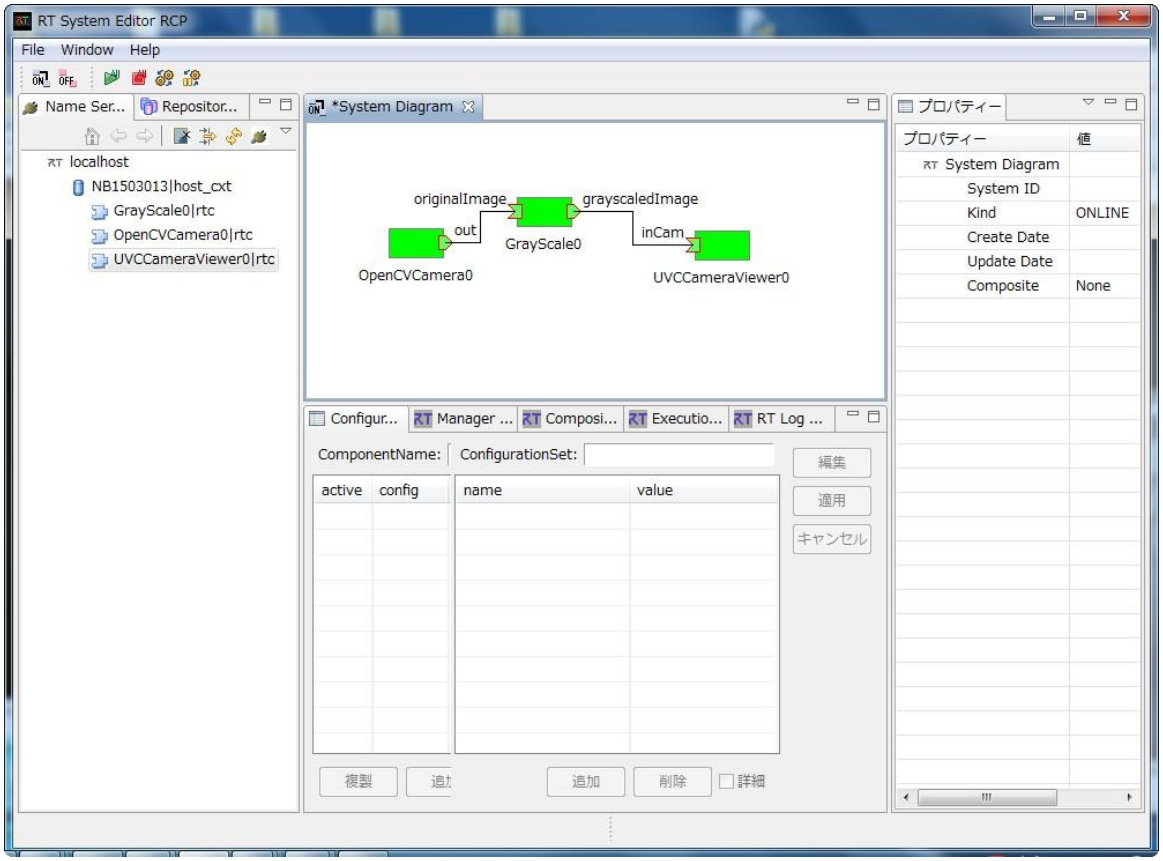
4.2.2 接続方法

以下のコンポーネントを下図の様にコンポーネントを接続してください。

・OpenCVCameraComp

・GrayScaleComp

・UVCCameraViewerComp



4.2.3 動作確認

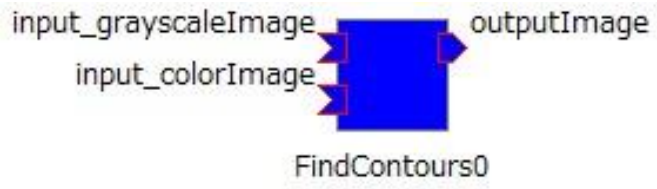
下図の様な 256 階調のモノクロ画像が表示されたら成功です。

5 輪郭抽出の作成

5.1 輪郭検出用コンポーネントの仕様説明

入力されたグレースケール画像から輪郭検出を行い、検出部分を矩形で囲んだ画像を出力するコンポーネントを作成します。仕様は以下の通りです。

・FindContours コンポーネント（システム図）



FindContours コンポーネント 仕様	
コンポーネント名	FindContours
InPort1	
ポート名	input_grayscaleImage
データ型	RTC::CameraImage
変数名	input_grayscaleImage
InPort2	
ポート名	input_colorImage
データ型	RTC::CameraImage
変数名	input_colorImage
OutPort	
ポート名	outputImage
データ型	RTC::CameraImage
変数名	output_Image

実際にプログラムをするとき、ポート名と変数は以下の様に変化します。

InPort ポート名	InPort 変数名	OutPort ポート名	OutPort 変数名
m_InPort 変数名 In	m_変数名	m_OutPort 変数名 Out	m_変数名

5.2 輪郭検出コンポーネントの作成

このコンポーネントでは OpenCV のライブラリを利用します。GrayScale コンポーネントと同じ手順で作成してください。

5.2.1 プログラムの流れ

5.2.1.1 画像入力

先ほど作成した GrayScale コンポーネントから出力されたデータと、カメラから出力されたデータを受け取ります。

5.2.1.2 ノイズ除去

入力されたグレースケール画像に平滑化処理を行い、2 値化処理時のノイズを減らします。
画像の平滑化には cvSmooth 関数を利用します。

```
void cvSmooth(
        const CvArr* src,
        CvArr* dst,
        int smoothtype,
        int param1,
        int param2,
        double param3,
        double param4
);
```

src	入力画像
dst	出力画像
smoothtype	平滑化に使用するフィルタを選択 ・CV_GAUSSIAN：ガウシアンフィルタ ・CV_BLUR　　：ブラーフィルタ ・CV_ MEDIAN：メディアンフィルタ
param1	カーネルの水平方向サイズ、奇数のみ指定できます。 平滑化処理は畳み込み演算で行われており、その時に利用される係数行列をカーネルと呼びます。
param2	カーネルの垂直方向サイズ ブラー、又はガウシアンフィルタ使用時に 0 を指定した場合、param1 の値が使用されます。
param3	ガウシアンフィルタ使用時の標準偏差を指定します。 0 を指定した場合、param1、param2 で指定した値を基に自動で計算します。
param4	非正方形のガウシアンカーネルを使用するとき、垂直方向に param3 と異なる標準偏差値を指定します。

上級者向け講習会課題 3

参考 URL：http://opencv.jp/opencv-1.0.0/document/opencvref_cv_filters.html

5.2.1.3 グレースケール画像の 2 値化

ノイズ処理を行ったグレースケール画像を 2 値画像へ変換します。画像の 2 値化には cvThreshold 関数を利用します。

void cvThreshold(const CvArr* src, CvArr* dst, double threshold, double max_value, int threshold_type);	
src	入力画像
dst	出力画像
threshold	閾値の指定 後述する CV_THRESH_OTSU を使用する場合は、指定した値は無視されます。
max_value	最大値 ピクセルの明るさが閾値を超えた場合、明るさを max_value で指定されている値にします。
threshold_type	処理方法の指定 2 値化処理を行う場合は CV_THRESH_BINARY を指定します。 また、CV_THRESH_OTSU を指定すると自動で最適な閾値の選択を行います。利用時は第五引数に下記を記述してください。 CV_THRESH_BINARY | CV_THRESH_OTSU

参考 URL：http://opencv.jp/opencv-1.0.0/document/opencvref_cv_filters.html

5.2.1.4 2 値画像から輪郭を検出

2 値画像から輪郭を検出します。輪郭検出には cvFindContours 関数を利用します。

int cvFindContours(CvArr* image, CvMemStorage* storage, CvSeq** first_contour, int header_size, int mode, int method, CvPoint offset);	
戻り値	検出された輪郭の個数
image	入力画像
storage	抽出された輪郭を格納するストレージ
first_contour	出力パラメータ 最初に検出された輪郭へのポインタが入っています。
header_size	シーケンスヘッダのサイズ method=CV_CHAIN_CODE の場合、>=sizeof(CvChain)，それ以外の場合は >=sizeof(CvContour) を指定してください。
mode	輪郭抽出モードの選択 ・CV_RETR_EXTERNAL 画像内で検出された輪郭の中で最も外側に存在する輪郭（白い部分）のみを抽出します。 輪郭が複数存在する場合、 次の輪郭を参照するには first_contour ->h_next 前の輪郭を参照するには first_contour ->h_prev と指定することで別の輪郭を選択することができます。 ・CV_RETR_LIST すべての輪郭を抽出し，それらをリストに保存します。 ・CV_RETR_CCOMP すべての輪郭を抽出し、2 つのレベルを持つ階層構造を構成します。輪郭（白い部分）を第 1 階層とし、その輪郭の中に輪郭（黒い部分）が存在するとき、それらを第 2 階層へ格納します。 ・CV_RETR_TREE すべての輪郭を抽出し、枝分かれした輪郭を完全に表現する階層構造を構成します。
method	近似手法の選択 ・CV_CHAIN_APPROX_SIMPLE 輪郭を形成している角の座標を端点として取得します。

	・CV_CHAIN_APPROX_NONE 輪郭を形成しているすべての点を取得します。 ・CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS Teh-Chin チェーン近似アルゴリズムの 1 つを適用します。 ・CV_CHAIN_CODE 抽出した輪郭をフリーマンチェーンコードで取得します。 ・CV_LINK_RUNS 値が 1（白い部分）のセグメントを水平に接続し、上記と異なった近似アルゴリズムで輪郭を取得します。 抽出モードが CV_RETR_LIST の場合のみ指定可能です。 ※CV_CHAIN_APPROX_TC89 系と CV_CHAIN_CODE、 CV_LINK_RUNS を選択すると矩形の計算が行われませんので、 CV_CHAIN_APPROX_SIMPLE 又は、CV_CHAIN_APPROX_NONE を選択してください。
offset	指定した位置分、輪郭の描写位置を移動させることができます。 今回は使用しません。

参考 URL：http://opencv.jp/opencv-1.0.0/document/opencvref_cv_segmentation.html
参考 URL（輪郭抽出モードと近似手法について）：
http://imagingsolution.blog107.fc2.com/blog-entry-202.html

5.2.1.5 検出された輪郭の中から最も大きな面積を持つ輪郭を見つける

輪郭検出を行うと大小複数の輪郭が検出されます。複数存在すると次の矩形計算の工程で目的の輪郭を囲うことが困難になるため、今回は面積が最も大きくなる輪郭を矩形で囲います。
輪郭の面積を求めるには cvContourArea 関数を利用します。

```
double cvContourArea(
        const CvArr* contour,
        CvSlice slice
);
```

戻り値	引数に設定した輪郭の面積が求められます。
contour	面積を求めたい輪郭
slice	輪郭の始点と終点を設定し、その範囲内の面積を求めます。 デフォルトでは輪郭全体の面積を求めます。

参考 URL：http://opencv.jp/opencv-1.0.0/document/opencvref_cv_contours.html

5.2.1.6 検出した輪郭を囲む矩形を描写

検出された輪郭から矩形を計算し、InPort から入力されているカラー画像へ描写します。矩形の計算には cvBoundingRect 関数を利用し、描写には cvRectangle 関数を利用します。

<table>
<tr><td colspan="2">CvRect cvBoundingRect(
　　CvArr* points,
　　int update
);</td></tr>
<tr><td>戻り値</td><td>描写された矩形の左下頂点の x 座標と y 座標、矩形の幅と高さが戻されます。</td></tr>
<tr><td>points</td><td>シーケンス（CvSeq*, CvContour*）か，点のベクトル（CvMat*），非 0 のピクセルが点列とみなされる 8 ビット 1 チャンネルマスク画像 （CvMat*, IplImage*）のいずれかで表現された 2 次元の点列を指定します。</td></tr>
<tr><td>update</td><td>更新フラグ
points に指定した型と update に指定するフラグによって、処理の内容を変更できます。
<table>
<tr><td>points</td><td>update</td><td>処理内容</td></tr>
<tr><td rowspan="2">CvCounter*</td><td>0</td><td>矩形を求める計算が行われず、CvCounter の rect フィールドから値が読み込まれます。</td></tr>
<tr><td>1</td><td>矩形を求める計算が行われ、CvCounter の rect フィールドに計算した値が書き込まれます。</td></tr>
<tr><td>CvSeq*、又は CvMat*</td><td colspan="2">update に指定したフラグは無視され、矩形が計算されてその座標が返されます。</td></tr>
</table>
</td></tr>
</table>

参考 URL：http://opencv.jp/opencv-1.0.0/document/opencvref_cv_contours.html

void cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int thickness, int line_type, int shift);	
img	矩形が描写される画像
pt1	矩形の左下の頂点
pt2	矩形の右上の頂点
color	描写する矩形の色 CV_RGB(0, 0, 0)を使用して指定します。
thickness	描写する矩形の太さ 負の値、又は CV_FILLED を指定すると矩形が塗りつぶされます。
line_type	線の種類 8　　線を 8 近傍で描写します。デフォルトではこちらが適用されます。 4　　線を 4 近傍で描写します。 CV_AA アンチエイリアス処理された線を描写します。
shift	頂点の座標の小数点以下の桁数を表すビット数を指定します。

参考 URL：http://opencv.jp/opencv-2svn/c/drawing_functions.html

5.2.1.7　矩形が描写された画像と矩形の座標の出力

矩形が描写された画像と矩形の座標をそれぞれ別の OutPort から出力します。

5.2.2 サンプルコード

5.2.2.1 FindContours.h

OpenCV のライブラリを使用するため、OpenCV のライブラリファイルをインクルードします。下記内容をインクルードしている所に追加してください。

```
//OpenCV 用ライブラリファイルのインクルード
#include<cv.h>
#include<cxcore.h>
#include<highgui.h>
```

画像の保存用にメンバー変数を追加します。下記内容を class の private:の中(// <rtc-template block="private_attribute">の下)に追加してください。

```
IplImage* m_grayscaleImageBuff;      // グレースケール画像用
IplImage* m_colorImageBuff;          // カラー画像入出力用
IplImage* m_noise_reductionBuff;     // ノイズ処理用
IplImage* m_thresholdImageBuff;      // ２値化用

CvMemStorage *parent_storage;        // メモリ解放バグ回避用
CvMemStorage *storage;               // 抽出された輪郭を保存する領域
CvSeq *first_contours;               // 最も外側に存在する輪郭へのポインタ
CvSeq *contour;                      // 最も大きな面積を持つ輪郭を保持
CvRect rect;                         // 計算された矩形保持用
```

5.2.2.2 FindContours.cpp

・onActivated

```
RTC::ReturnCode_t FindContours::onActivated(RTC::UniqueId ec_id)
{
  // イメージ用メモリの初期化
  m_grayscaleImageBuff = NULL;
  m_colorImageBuff = NULL;
  m_noise_reductionBuff = NULL;
  m_thresholdImageBuff = NULL;

  // OutPort の画面サイズの初期化
  m_output_Image.width = 0;
  m_output_Image.height = 0;

  // 輪郭保持親メモリの初期化
  parent_storage = cvCreateMemStorage(0);

  return RTC::RTC_OK;
}
```

・onDeactivated

```
RTC::ReturnCode_t FindContours::onDeactivated(RTC::UniqueId ec_id)
{
  // イメージ用メモリの解放
  if (m_colorImageBuff != NULL)
  {
    cvReleaseImage(&m_grayscaleImageBuff);
    cvReleaseImage(&m_colorImageBuff);
    cvReleaseImage(&m_noise_reductionBuff);
    cvReleaseImage(&m_thresholdImageBuff);
  }

  //輪郭保持領域の解放
  if (storage != NULL)
  {
    cvReleaseMemStorage(&storage);
  }

  //親領域の解放
  if (parent_storage != NULL)
  {
    cvReleaseMemStorage(&parent_storage);
  }
  return RTC::RTC_OK;
}
```

上級者向け講習会課題 3

・onExecute

下記のサンプルコードの複数行コメント部分（/*　*/）の処理を作成してください。

```
RTC::ReturnCode_t FindContours::onExecute(RTC::UniqueId ec_id)
{
  // 新しいデータのチェック
  if (m_input_colorImageIn.isNew() && m_input_grayscaleImageIn.isNew())
  {
    // InPort データの読み込み
    m_input_colorImageIn.read();
    m_input_grayscaleImageIn.read();

    /*  使用する画像のメモリ領域を確保  */

    // InPort からの画像データをメモリにコピー
    memcpy(m_grayscaleImageBuff-
    >imageData,(void*)&(m_input_grayscaleImage.pixels[0]),
    m_input_grayscaleImage.pixels.length());
    memcpy(m_colorImageBuff->imageData,(void*)&(m_input_colorImage.pixels[0]),
    m_input_colorImage.pixels.length());

    /*  入力されたグレースケール画像のノイズ除去  */

    /*  グレースケール画像の２値化処理  */

    //抽出した輪郭を保持する領域
    storage = cvCreateChildMemStorage(parent_storage);
    first_contours = NULL;

    /*  ２値画像から輪郭を検出  */
```

次のページへ続きます

```
    if (first_contours != NULL)
    {
    contour = NULL;
    /*  最も大きな面積を持つ輪郭を探し出す  */

    /*  矩形の計算と描写処理  */
    }

    // 画像データのサイズ取得
    int len = m_colorImageBuff->nChannels * m_colorImageBuff->width *
    m_colorImageBuff->height;
    m_output_Image.pixels.length(len);

    // 画像データを OutPort にコピー
    memcpy((void *)&(m_output_Image.pixels[0]), m_colorImageBuff->imageData,
    len);

    // 画像データを OutPort から出力する。
    m_output_ImageOut.write();

    // 輪郭保持用メモリの解放
    if (storage != NULL){
        cvReleaseMemStorage(&storage);
    }
  }
  return RTC::RTC_OK;
}
```

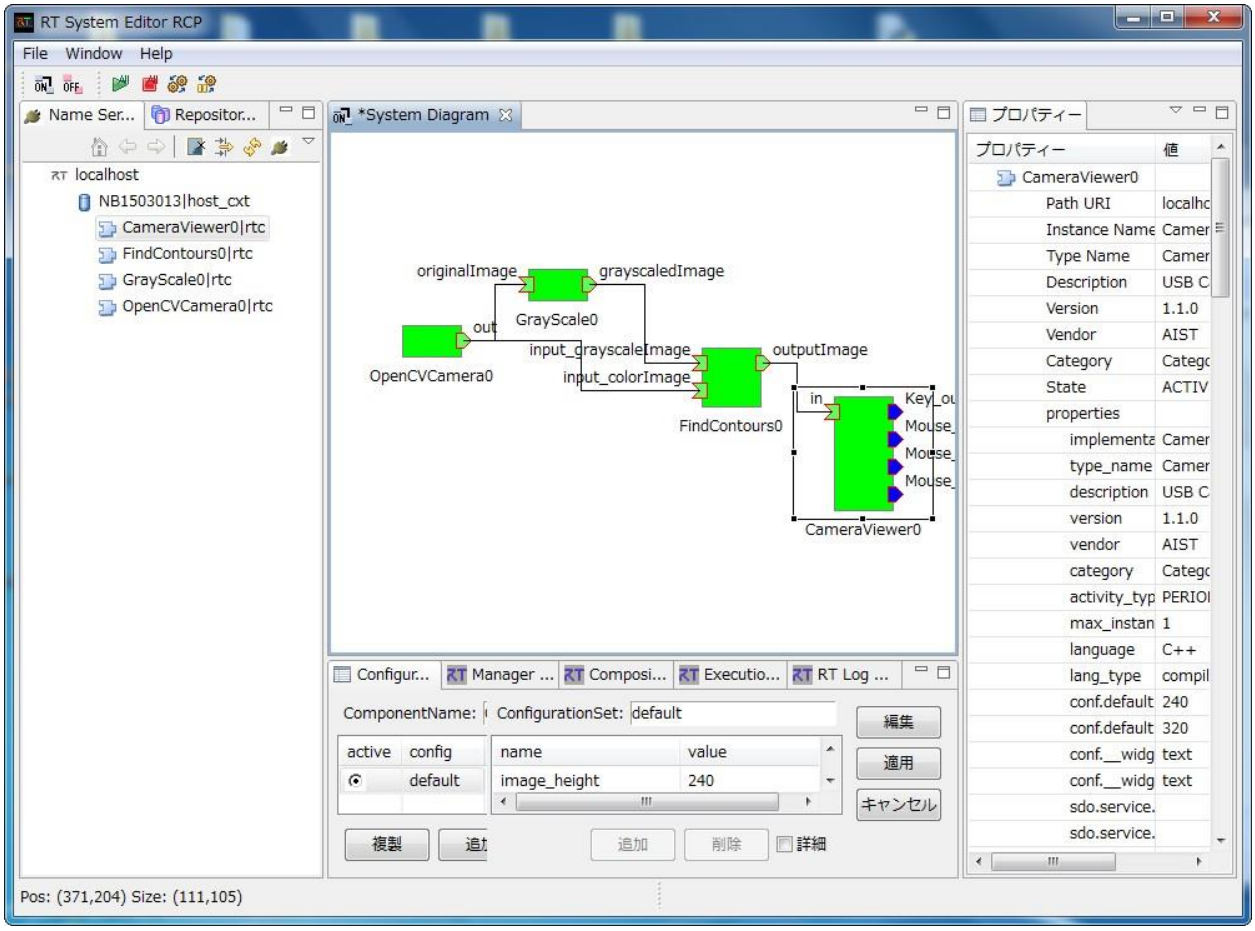
5.2.3　作成時のヒント

輪郭検出が上手く行われない場合は高めの閾値を設定してください。

5.2.4　接続方法

以下のコンポーネントを下図の様に接続してください。

・OpenCVCameraComp.exe
・GrayScaleComp.exe
・FindContoursComp.exe
・CameraViewerComp.exe



5.2.5　動作確認

上級者向け講習会課題 3