

画像処理実習

宮本 信彦

国立研究開発法人産業技術総合研究所
インダストリアルCPS研究センター



資料

- 「WEBページ」フォルダのHTMLファイルを開く
 - チュートリアル(画像処理実習) _ OpenRTM-aist.html
- もしくは以下のリンク
 - <https://openrtm.org/openrtm/ja/node/7197>



はじめに

このページでは、OpenCVの画像処理により図形を検出して移動ロボット (Raspberry Piマウス) を追従させるRTCの作成手順を説明します。

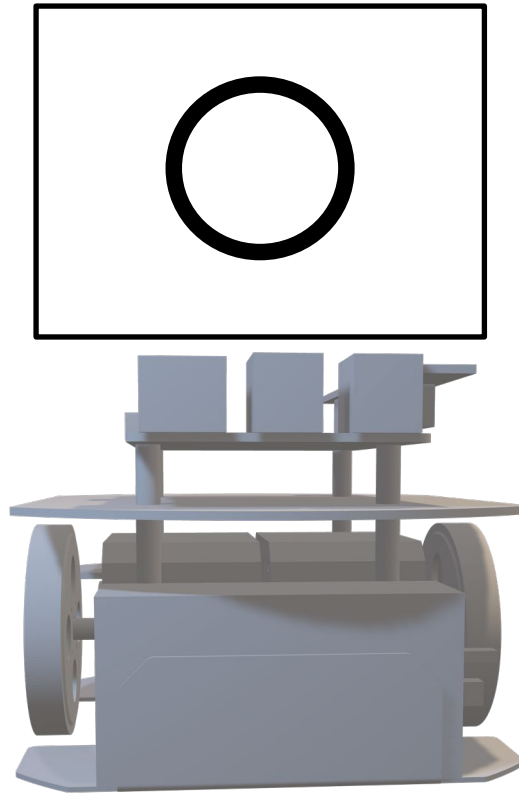
作成するRTコンポーネント

- CircleTracking コンポーネント : OpenCVライブラリのHoughCircles関数で円を検出して、検出した円の方向に移動ロボットが回転するように制御するRTC



画像処理コンポーネントの作成

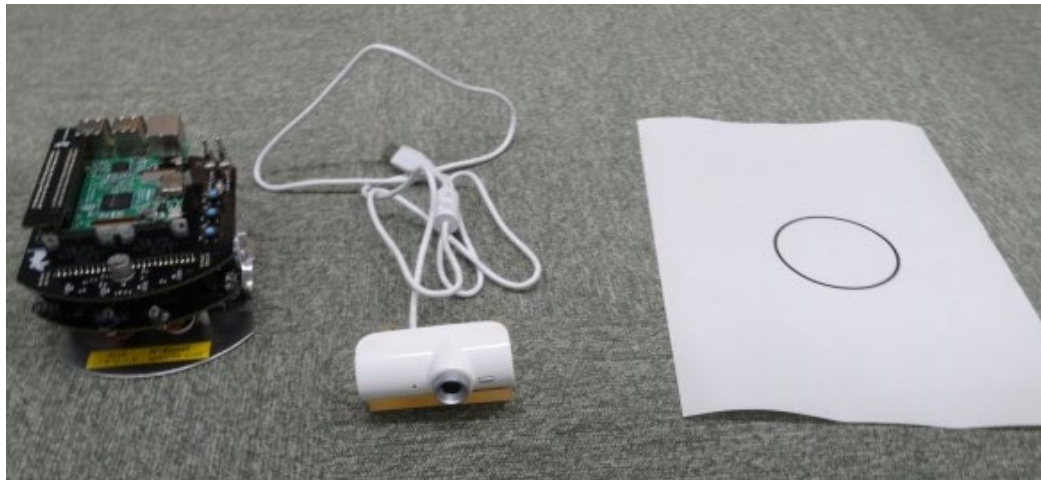
- 発展的な課題として、OpenCVを使った画像処理コンポーネントを作成する



課題：円形の図形を
カメラで検出して、
移動ロボットの向き
を追従させる

カメラ用マウントの取り付け

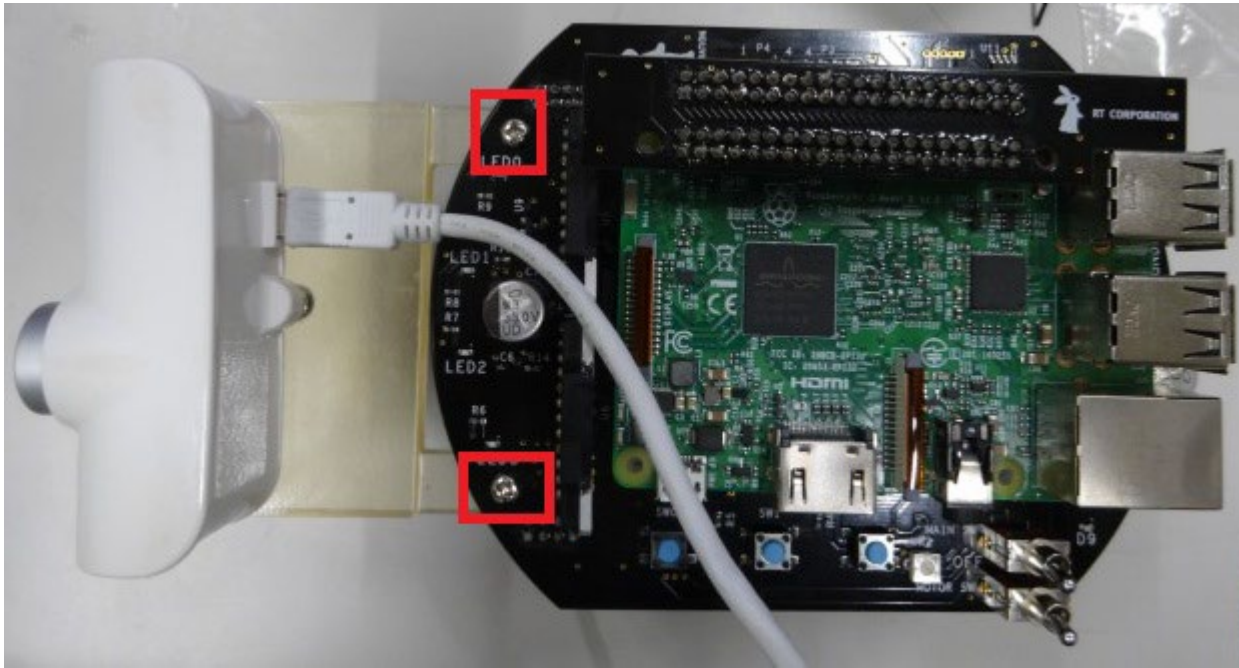
- 今回の課題で必要な機材は以下の3点
 - Raspberry Piマウス本体
 - USBカメラ (カメラ用マウントに固定済み)
 - 円を描いた紙



- Raspberry PiマウスにUSBカメラを取り付けるため、**LiDARは取り外す**

カメラ用マウントの取り付け

- LiDARマウントを固定する用のネジでカメラ用マウントを固定する



USBカメラとノートPC接続

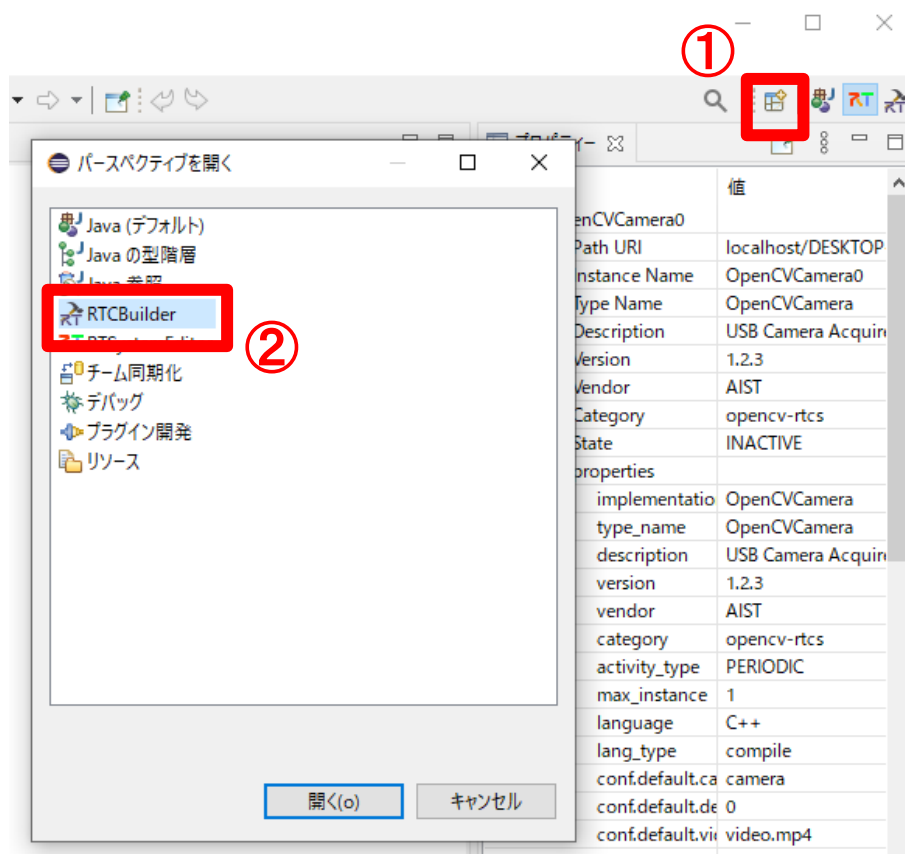
- ノートPCとUSBカメラをUSBポートで接続する



RTC Builderによる ひな型コード生成

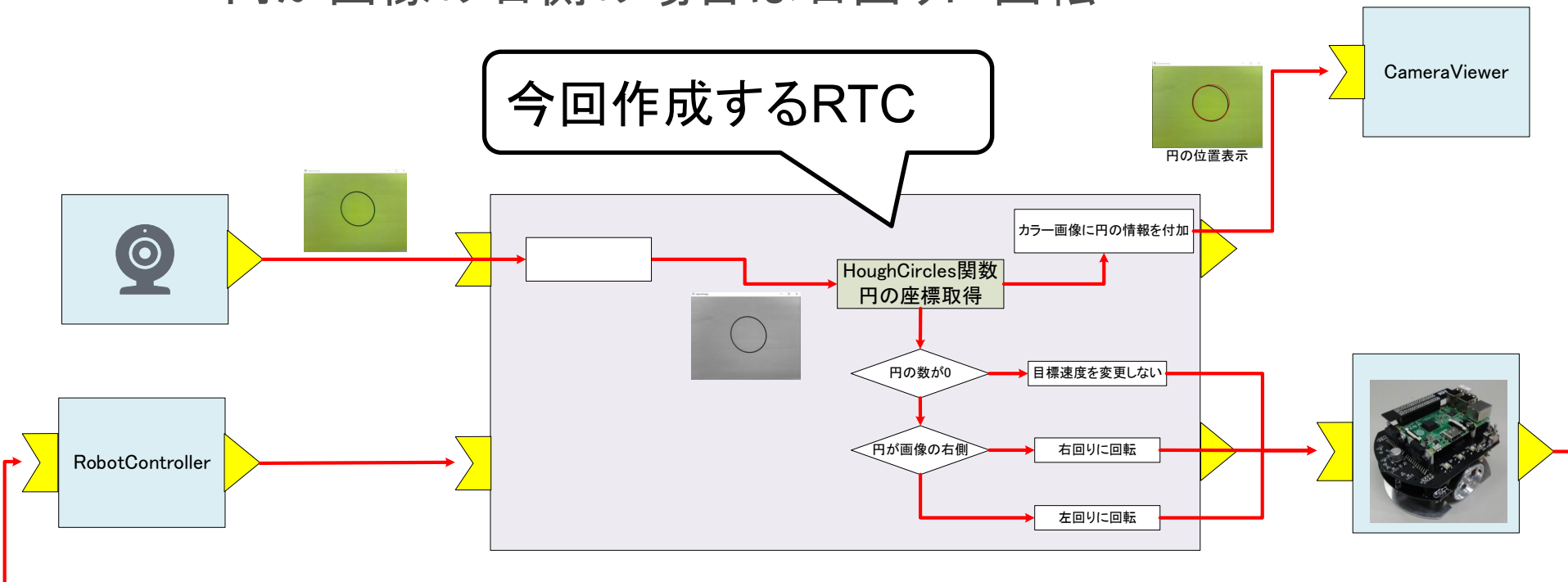
RTC Builder起動

- RT System Editorを起動中の場合は、「パースペクティブを開く」画面から起動する



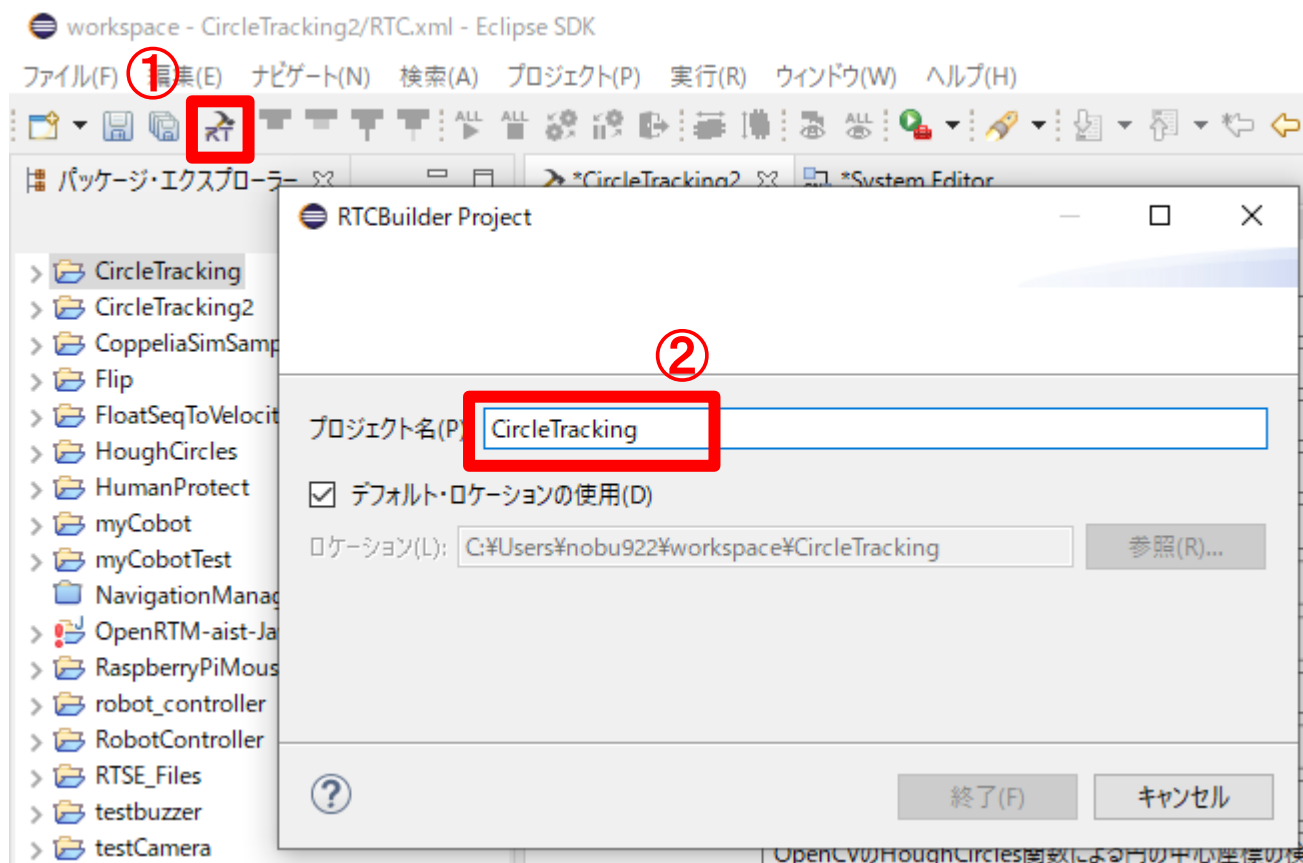
作成するRTCの概要

- カメラ画像から円形的位置を検出して、移動ロボットの向きを以下のように制御
 - 円が画像の左側の場合は左回りに回転
 - 円が画像の右側の場合は右回りに回転



プロジェクト作成

- 今回は「CircleTracking」という名前のプロジェクトを作成する



基本プロフィール入力

- コンポーネント名 :
 - CircleTracking
- 言語
 - C++
- 他の項目は今回は入力の必要なし

RT-Component Basic Profile

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

*コンポーネント名 :	CircleTracking
概要 :	Get Circles Position Component
*バージョン :	1.0.0
*ベンダ名 :	AIST
*カテゴリ :	ImageProcessing
コンポーネント型 :	STATIC
アクティビティ型 :	PERIODIC
最大インスタンス数 :	1
実行型 :	PeriodicExecutionContext
実行周期 :	1000.0
概要 :	OpenCVのHoughCircles関数による円の中心座標の検出を行うコンポーネント
RTC Type :	

▼ 言語

このセクションでは使用する言語を指定します

- ☒ C++
☐ Java
☐ Lua
☐ Python

アクティビティ

- 以下の3つを有効化
 - onActivated
 - onDeactivated
 - onExecute

アクティビティ

▼ アクティビティ

このセクションでは使用するアクションコールバックを指定します。

コンポーネントの初期化と終了処理に関するアクション

onInitialize

onFinalize

実行コンテキストの起動と停止に関するアクション

onStartup

onShutdown

alive状態でのコンポーネントアクション

onActivated

onDeactivated

onAborting

onError

onReset

Dataflow型コンポーネントのアクション

onExecute

onStateUpdate

onRateChanged

データポート

- 以下のInPortを設定する

- image_in

- データ型: RTC::CameraImage

- ※Img::CameraImageと間違えないように注意

- ※RTC::CameraInfoと間違えないように注意

- velocity_in

- データ型: RTC::TimedVelocity2D

The screenshot displays the RT Middleware configuration window. At the top, there are two panels: '*ポート名 (InPort)' and '*ポート名 (OutPort)'. The InPort panel lists 'image_in' and 'velocity_in', with 'image_in' selected. The OutPort panel lists 'image_out' and 'velocity_out', with 'velocity_out' selected. Below these panels is a 'Detail' section. It contains a text box with the selected port name 'image_in (InPort)'. Below that is a dropdown menu for '*データ型' (Data Type) set to 'RTC::CameraImage', with a 'ReLoad' button next to it. The interface also includes 'Add' and 'Delete' buttons for each port list.

データポート

- 以下のOutPortを設定する
 - **image_out**
 - データ型: RTC::CameraImage
 - ※Img::CameraImageと間違えないように注意
 - ※RTC::CameraInfoと間違えないように注意
 - **velocity_out**
 - データ型: RTC::TimedVelocity2D

The screenshot shows the RT Middleware configuration interface. At the top, there are two panels for managing ports. The left panel, labeled '*ポート名 (InPort)', contains a list with 'image_in' and 'velocity_in', and buttons for 'Add' and 'Delete'. The right panel, labeled '*ポート名 (OutPort)', contains a list with 'image_out' and 'velocity_out', and buttons for 'Add' and 'Delete'. Below these panels is a 'Detail' section. It contains a text box with the following text: 'このセクションではデータポート毎の概要を説明するドキュメントを記述します。上のデータポートを選択すると、それぞれのドキュメントが記述できます。' Below this text is a label 'ポート名:' followed by a text box containing 'image_in (InPort)'. At the bottom, there is a section for setting the data type. It has a label '*データ型' followed by a dropdown menu showing 'RTC::CameraImage' and a 'ReLoad' button.

コンフィギュレーションパラメータ

- 以下のコンフィギュレーションパラメータを設定する

– speed_r

- データ型: double
- デフォルト値: 0.5
- 制約条件: $0.0 < x < 2.0$
- Widget: slider
- Step: 0.01

*名称	
speed_r	
houghcircles_dp	
houghcircles_minDist	
houghcircles_param1	
houghcircles_param2	
houghcircles_minRadius	

▼ Detail

このセクションでは各コンフィギュレーション・パラメータの詳細情報を指定します。

パラメータ名:

*データ型	<input type="text" value="double"/>
*デフォルト値	<input type="text" value="0.5"/>
変数名:	<input type="text"/>
単位:	<input type="text" value="rad/s"/>
制約条件:	<input type="text" value="0.0 < x < 2.0"/>
Widget:	<input type="text" value="slider"/>
Step:	<input type="text" value="0.01"/>

コンフィギュレーションパラメータ

- 以下のコンフィギュレーションパラメータを設定する

- **houghcircles_dp**

- データ型: double
- デフォルト値: 2
- Widget: text

Widgetがsliderになっている場合は、必ずtextに変更する

- **houghcircles_minDist**

- データ型: double
- デフォルト値: 30
- Widget: text

- **houghcircles_param1**

- データ型: double
- デフォルト値: 100
- Widget: text

*名称	
speed_r	
houghcircles_dp	
houghcircles_minDist	
houghcircles_param1	
houghcircles_param2	
houghcircles_minRadius	

AddDelete

▼ Detail

このセクションでは各コンフィギュレーション・パラメータの詳細情報を指定します。

パラメータ名: houghcircles_dp

*データ型	double
*デフォルト値	2
変数名:	
単位:	
制約条件:	
Widget:	text
Step:	

コンフィギュレーションパラメータ

- 以下のコンフィギュレーションパラメータを設定する

- **houghcircles_param2**

- データ型 : double
 - デフォルト値 : 100
 - Widget : text

- **houghcircles_minRadius**

- データ型 : int
 - デフォルト値 : 0
 - Widget : text

- **houghcircles_maxRadius**

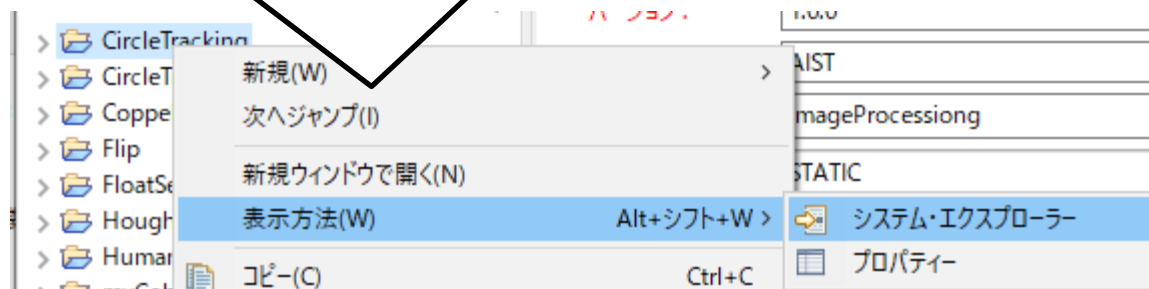
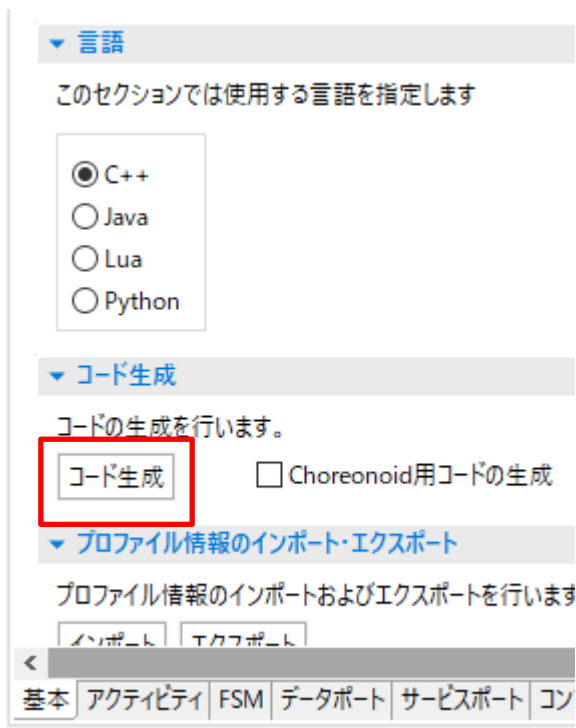
- データ型 : int
 - デフォルト値 : 0
 - Widget : text

The screenshot shows the RT Middleware configuration interface. At the top, there is a table listing parameters: houghcircles_minDist, houghcircles_param1, houghcircles_param2 (highlighted in blue), houghcircles_minRadius, and houghcircles_maxRadius. To the right of the table are 'Add' and 'Delete' buttons. Below the table, there is a 'Detail' section. It contains a text box for 'パラメータ名' (Parameter Name) with the value 'houghcircles_param2'. Below this, there are several input fields: '*データ型' (Data Type) set to 'double', '*デフォルト値' (Default Value) set to '100', '変数名' (Variable Name), '単位' (Unit), '制約条件' (Constraint), 'Widget' set to 'text', and 'Step'.

コード生成




- コード生成後、ファイルが生成されたかを確認

プロジェクトを右クリックして、「表示方法」
→「システムエクスプローラー」



CMakeLists.txtの編集

- **src**フォルダ内の**CMakeLists.txt**をメモ帳等で編集する
 - ※ CircleTrackingフォルダ直下のCMakeLists.txtではないので注意

名前	更新日時	種類	サイズ
 CircleTracking.cpp	2022/09/29 15:30	C++ ソース ファイル	9 KB
 CircleTrackingComp.cpp	2022/09/29 12:04	C++ ソース ファイル	4 KB
 CMakeLists.txt	2022/09/28 10:34	テキストドキュメント	3 KB

- OpenCVを使うため、find_packageでライブラリを検出する。

```
set(comp_srcs CircleTracking.cpp )  
set(standalone_srcs CircleTrackingComp.cpp)  
  
find_package(OpenCV REQUIRED) #追加
```

CMakeLists.txtの編集

- リンクするライブラリにOpenCVを追加する
 - 以下の2か所

```
# 以下は修正前
# target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES})
# 以下は修正後、${OpenCV_LIBS}を追加
target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
```

```
# 以下は修正前
# target_link_libraries(${PROJECT_NAME}Comp ${OPENRTM_LIBRARIES})
# 以下は修正後、${OpenCV_LIBS}を追加
target_link_libraries(${PROJECT_NAME}Comp ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
```

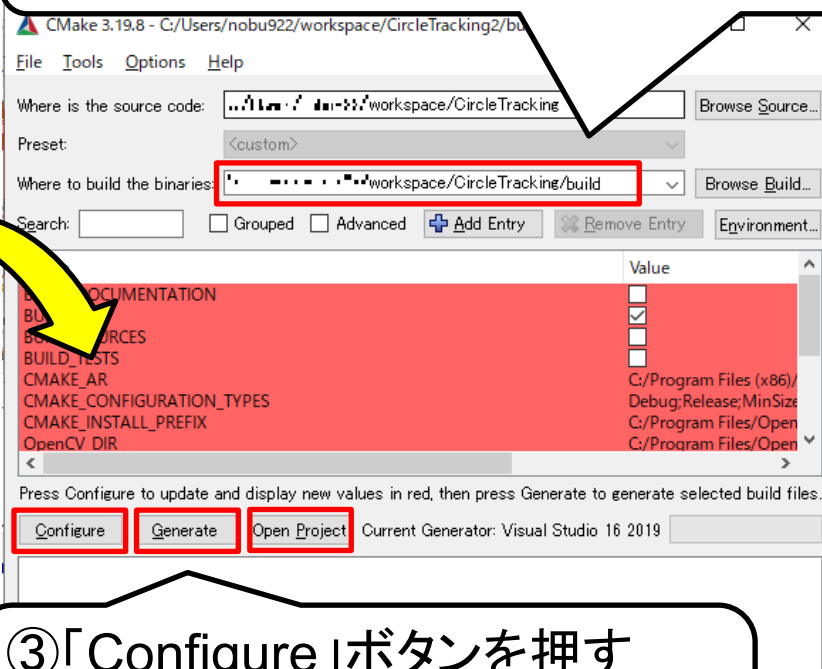
ビルドに必要なファイルの生成

- CMake(cmake-gui)でビルドに必要なファイル (Visual Studioのプロジェクトファイル等)を生成

① CMakeLists.txtをcmake-guiにドラッグアンドドロップする
※ CircleTrackingフォルダ直下のCMakeListsを使う

build	2022/09/29 12:07	ファイル フォルダー	
cmake	2022/09/29 12:04	ファイル フォルダー	
doc	2022/09/29 12:04	ファイル フォルダー	
idl	2022/09/29 12:04	ファイル フォルダー	
include	2022/09/29 12:04	ファイル フォルダー	
scripts	2022/09/29 12:04	ファイル フォルダー	
src	2022/09/29 15:30	ファイル フォルダー	
test	2022/09/29 12:04	ファイル フォルダー	
.appveyor.yml	2022/09/29 12:04	Yaml ソース ファイル	4 KB
.project	2022/09/29 12:03	PROJECT ファイル	1 KB
CircleTracking.conf	2022/09/29 12:04	CONF ファイル	13 KB
CMakeLists.txt	2022/09/29 12:04	テキスト ドキュメント	6 KB
COPYING	2022/09/29 12:04	ファイル	35 KB
COPYING.LESSER	2022/09/29 12:04	LESSER ファイル	8 KB
README.md	2022/09/29 12:04	Markdown ソース フ...	9 KB
rtc.conf	2022/09/29 12:04	CONF ファイル	51 KB
RTC.xml	2022/09/29 20:55	XML ドキュメント	7 KB
RTC.xml20220929120409	2022/09/29 12:03	XML ドキュメント	1 KB

② 「Where to build the binaries」のパスの後ろに「/build」を追加



③ 「Configure」ボタンを押す
④ 「Generate」ボタンを押す
⑤ 「Open Project」ボタンを押す

ソースコードの編集

- CircleTracking.hの編集

- opencv2/opencv.hppをインクルードする

```
34: #include <rtm/DataInPort.h>
35: #include <rtm/DataOutPort.h>
36:
37: #include <opencv2/opencv.hpp> //追加
```

- 「private:」の下あたりに、以下の3行を追加する

入力画像データ、出力画像データを格納する変数を宣言

移動ロボットの回転方向を格納する変数を宣言

- 0: 回転方向を指定しない
- 1: 左回りの回転
- 2: 右回りの回転

```
314: private:
315:     cv::Mat m_imageBuff; //追加
316:     cv::Mat m_outputBuff; //追加
317:     int m_direction; //追加
```

ソースコードの編集

- CircleTracking.cppの編集

```
RTC::ReturnCode_t CircleTracking::onActivated(RTC::UniqueId /*ec_id*/)
{
    // OutPortの画面サイズを0に設定
    m_image_out.width = 0;
    m_image_out.height = 0;

    //進行方向を0(回転方向を指定しない)に設定
    m_direction = 0;
    return RTC::RTC_OK;
}
```

```
RTC::ReturnCode_t CircleTracking::onDeactivated(RTC::UniqueId /*ec_id*/)
{
    if (!m_outputBuff.empty())
    {
        // 画像用メモリの解放
        m_imageBuff.release();
        m_outputBuff.release();
    }

    return RTC::RTC_OK;
}
```

ソースコードの編集

- CircleTracking.cppの編集

```
RTC::ReturnCode_t CircleTracking::onExecute(RTC::UniqueId /*ec_id*/)
{
    if (m_image_inIn.isNew())
    {
        cv::Mat gray;
        std::vector<cv::Vec3f> circles;
        // 画像データの読み込み
        m_image_inIn.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_image_in.width != m_image_out.width || m_image_in.height != m_image_out.height)
        {
            m_image_out.width = m_image_in.width;
            m_image_out.height = m_image_in.height;

            m_imageBuff.create(cv::Size(m_image_in.width, m_image_in.height), CV_8UC3);
            m_outputBuff.create(cv::Size(m_image_in.width, m_image_in.height), CV_8UC3);
        }

        // InPortの画像データをm_imageBuffにコピー
        std::memcpy(m_imageBuff.data,
                    (void *)&(m_image_in.pixels[0]),
                    m_image_in.pixels.length());
    }
}
```

InPort (image_in) でデータを受信して、
変数m_imageBuffに画像データをコピー
するまでの処理

ソースコードの編集

- CircleTracking.cppの編集

```
//カラー画像をグレースケールに変換
cv::cvtColor(m_imageBuff, gray, cv::COLOR_BGR2GRAY);

// HoughCircles関数で円を検出する
cv::HoughCircles(gray, circles, cv::HOUGH_GRADIENT, m_houghcircles_dp,
                  m_houghcircles_minDist, m_houghcircles_param1,
                  m_houghcircles_param2, m_houghcircles_minRadius,
                  m_houghcircles_maxRadius);
```

グレースケール画像から円を検出するまでの処理

HoughCircles関数の引数にコンフィギュレーションパラメータの変数を指定する

ソースコードの編集

• CircleTracking.cppの編集

```
//円を検出できた場合の処理
if (!circles.empty())
{
    //円の位置が画像の左側の場合は左回りに回転するように設定
    if (circles[0][0] < gray.cols / 2)
    {
        m_direction = 1;
    }
    //円の位置が画像の右側の場合は右回りに回転するように設定
    else
    {
        m_direction = 2;
    }
}
//円を検出できなかった場合は回転方向の指定をしないように設定
else
{
    m_direction = 0;
}
```

以下の条件で移動ロボットの回転方向を変更

- 円が検出できた
 - 円が画像の左側
 - 円が画像の右側
- 円が検出できなかった

ソースコードの編集

• CircleTracking.cppの編集

```
//元のカラー画像をコピーして円の情報を画像に追加  
std::memcpy(m_outputBuff.data,  
            (void *)&(m_image_in.pixels[0]),  
            m_image_in.pixels.length());
```

確認用に検出した円を元の画像に付加して、OutPort (image_out) から出力する処理

```
for (auto circle : circles)  
{  
    cv::circle(m_outputBuff,  
              cv::Point(static_cast<int>(circle[0]), static_cast<int>(circle[1])),  
              static_cast<int>(circle[2]),  
              cv::Scalar(0, 0, 255), 2);  
}
```

```
// 画像データのサイズ取得  
int len = m_outputBuff.channels() * m_outputBuff.cols * m_outputBuff.rows;  
m_image_out.pixels.length(len);
```

```
// 円の情報を付加した画像データをOutPortにコピー  
std::memcpy((void *)&(m_image_out.pixels[0]), m_outputBuff.data, len);  
//画像データを出力  
m_image_outOut.write();  
}
```

ソースコードの編集

- CircleTracking.cppの編集

```
if (m_velocity_inIn.isNew())
{
    //速度指令値を読み込み
    m_velocity_inIn.read();
    m_velocity_out = m_velocity_in;

    //円が画像の左側にある場合、左回りに回転する
    if (m_direction == 1)
    {
        m_velocity_out.data.va = m_speed_r;
    }
    //円が画像の右側にある場合、右回りに回転する
    else if (m_direction == 2)
    {
        m_velocity_out.data.va = -m_speed_r;
    }
    //速度指令値を出力
    m_velocity_outOut.write();
}
return RTC::RTC_OK;
}
```

速度指令をInPort (velocity_in) から読み込んで、検出した円の位置により回転速度を変更する処理

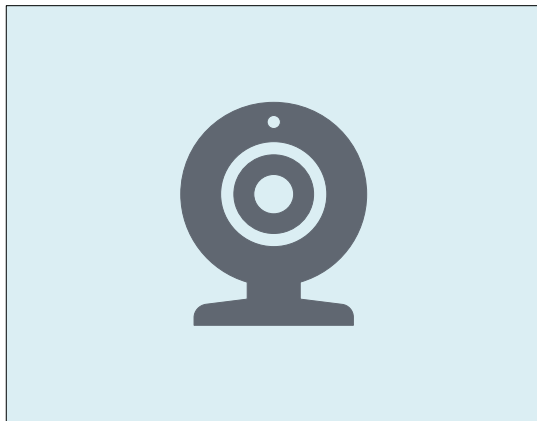
ここまで追加編集が完了したらビルドする。

RTCの起動

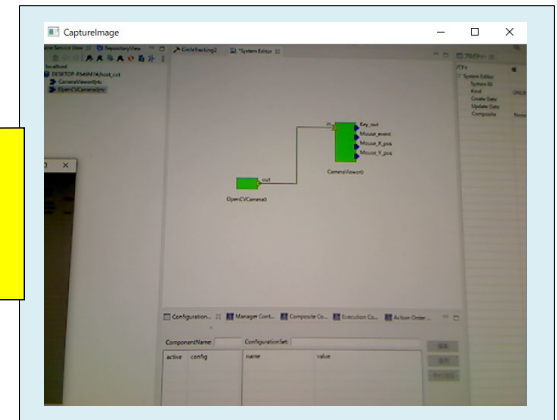
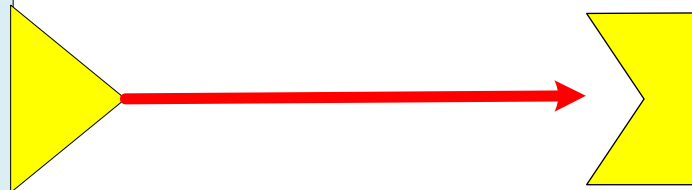
- 以下の5つのRTCを起動する
 - **RaspberryPiMouseRTC**
 - **RobotController**
 - 「RTコンポーネントの作成入門」の資料を参照
 - <https://openrtm.org/openrtm/ja/node/6550>
 - <https://openrtm.org/openrtm/ja/node/6551>
 - **OpenCVCamera**
 - **CameraViewer**
 - OpenRTM-aistのサンプルコンポーネント
 - 次のスライドで説明
 - **CircleTracking**
 - Windows
 - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内の実行ファイル(Comp.exe)
 - Ubuntu
 - **Build/src**フォルダ内の実行ファイル

サンプルコンポーネント概要

- OpenCVCamera
 - カメラから取得した画像をOutPortから出力する
- CameraViewer
 - InPortで受信した画像データを表示する



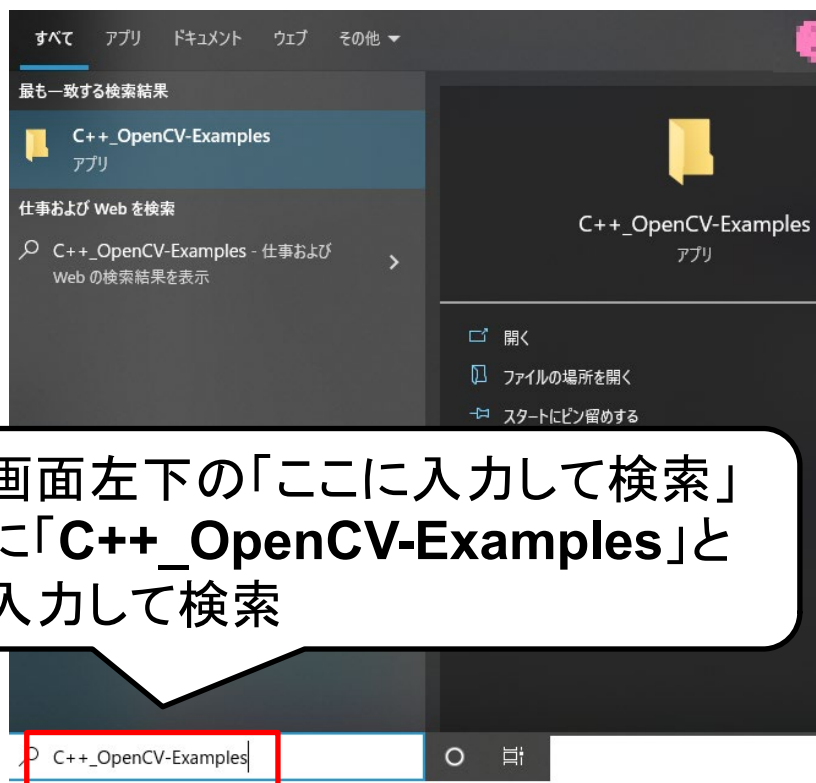
OpenCVCamera



CameraViewer

サンプルコンポーネントの起動

- 起動手順
 - Windows



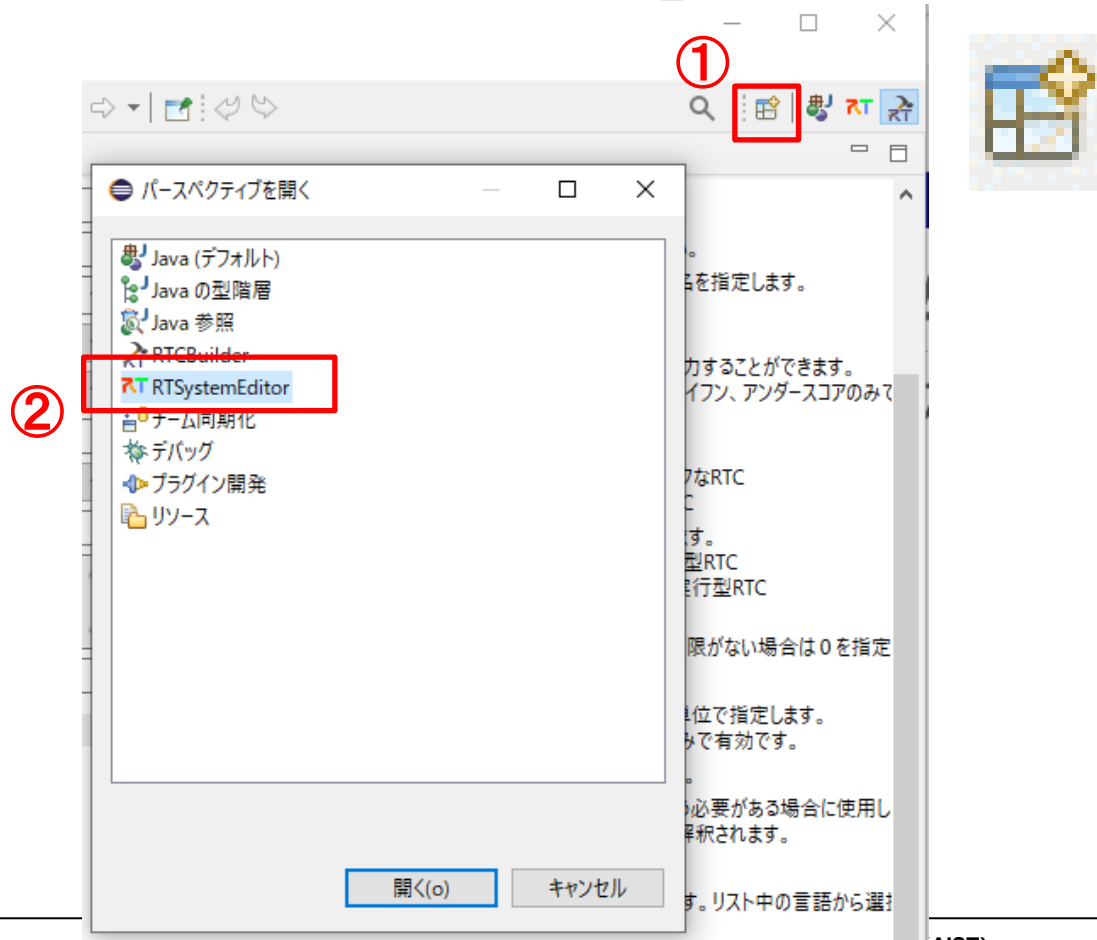
エクスプローラーから、以下のファイルをダブルクリックする

- CameraViewer.bat
- OpenCVCamera.bat

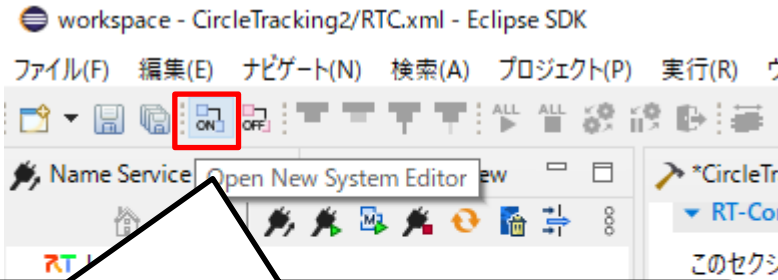
vc14	2022/09/24 21:22	ファイル フォルダー	
vc16	2022/09/24 21:22	ファイル フォルダー	
Affine.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
BackGroundSubtractionSimple.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Binarization.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
CameraViewer.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
ChromaKey.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
DilationErosion.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Edge.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Findcontour.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Flip.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Histogram.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Hough.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
ImageCalibration.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
ImageSubtraction.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
ObjectTracking.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
OpenCVCamera.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB
Perspective.bat	2022/03/24 20:46	Windows バッチ ファイル	1 KB

RTシステム構築

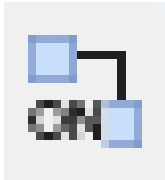
- RT System Editorを起動する。
 - 「パースペクティブを開く」画面から起動する



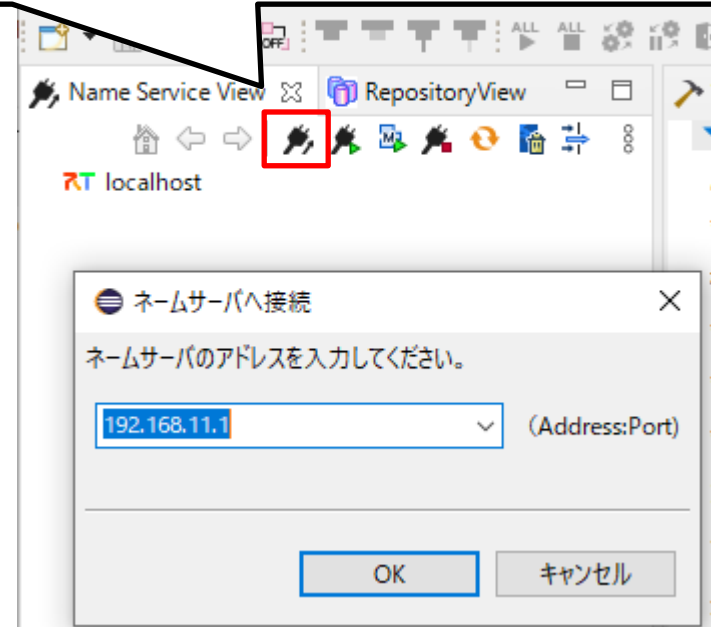
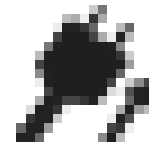
動作確認



システムダイアグラムを表示する

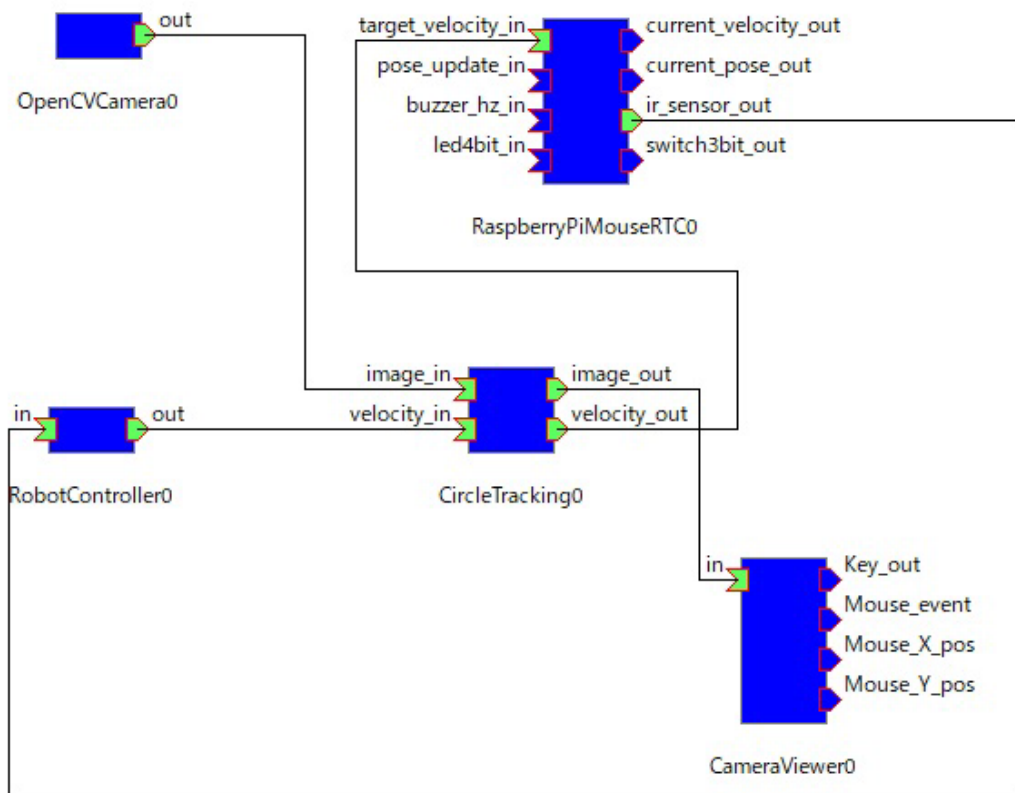


ネームサービスビューに「192.168.11.1」がない場合は、192.168.11.1のネームサーバーへ接続する



動作確認

- RT System Editorで以下のようにポートを接続してアクティブ化する

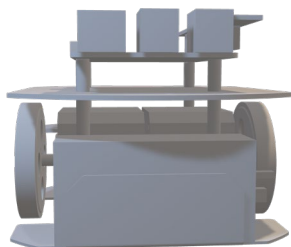
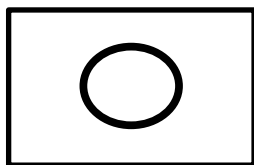


アクティブ化には
「Activate Systems」ボタン
を押す。

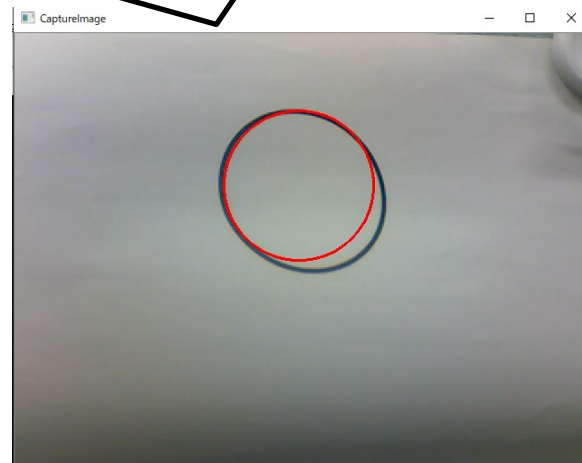


動作確認

- 円を描いた紙をカメラの前で動かして、Raspberry Piマウスの動作を確認する



円が正常に検出できているかを、CameraViewerで確認する



カメラのID指定

- PC内蔵カメラなどの画像が表示されている場合は、コンフィギュレーションパラメータでカメラのIDを指定する

① OpenCVCamera0をクリックして、下のコンフィギュレーションビューの「編集」ボタンを押す

name	value
capture_mode	camera
device_num	0
video_file	video.mp4
URL	
frame_width	640
frame_height	480
frame_rate	30
brightness	128
contrast	32
saturation	32
hue	0
gain	64

② 「device_num」の値を変更する

HoughCircles関数のパラメータ調整

- 誤検出が多い場合は、HoughCircles関数のパラメータを変更する
 - HoughCircles関数の詳細は以下を参照
 - http://opencv.jp/opencv-2svn/cpp/feature_detection.html#cv-houghcircles

