

2026年10月16日

高度ポリテクセンター・能力開発セミナー (E0771)

「RTミドルウェアによるロボットプログラミング技術」

4. 総合演習 (1)



はじめに

- 本日の講習会のWebページ
 - <https://openrtm.org/openrtm/ja/node/7366>
 - 短縮URL
 - <https://x.gd/tjC0y>
- こちらに、必要な情報がまとまっていますので、ブックマークしてください。



概要

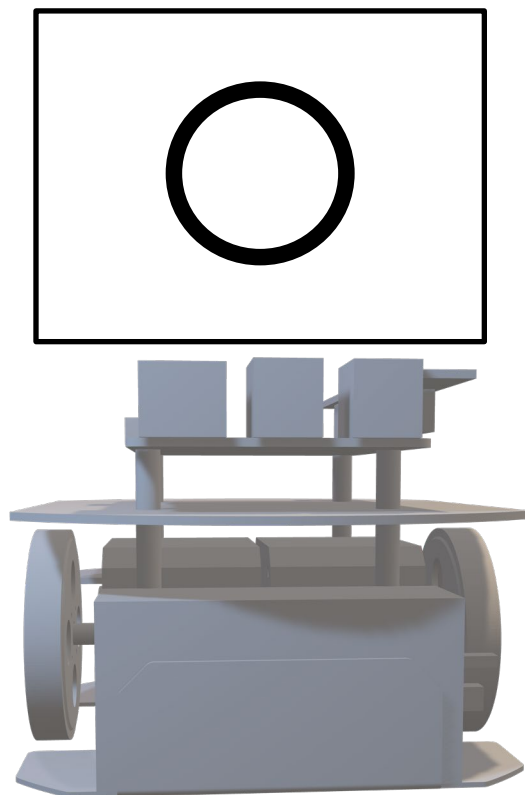
1. 画像処理コンポーネントの作成
2. システム構築とテスト

目標：

1. 画像データを格納するデータ型について理解する
2. 外部ライブラリを使う場合の設定方法を理解する
3. 画像処理を移動ロボットの制御に応用するという、より高度なシステムの構築手順を理解する

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: カメラ画像から円形の図形を検出して、移動ロボットの向きを追従させる

資料

• WEBページ

- <https://openrtm.org/openrtm/ja/node/7197>
- 短縮URL
 - <https://x.gd/vWx6A>

12:30-16:45	3. RTCプログラミング演習 (1) RTCBuilderによるひな形コードの生成 (2) プログラムの作成とコンパイル (3) シミュレータロボットと接続してテスト (4) 実機ロボットと接続してテスト ・チュートリアル(RTコンポーネントの作成入門、Raspberry Pi Mouse、Windows) ・チュートリアル (RaspberryPiマウス、Joystick操作) 資料: 241121-03.pdf シミュレータ等: RTM_Tutorial.zip
-------------	---

11月22日 (金)	
10:00 -11:45	4. 総合演習 (1) (1) 画像処理コンポーネントの作成 (2) システム構築とテスト ・チュートリアル(画像処理実習) 資料: 241122-04.pdf
11:45 -12:30	昼食
12:30 -16:45	5. 総合演習 (2) (1) SLAMについて (2) 地図作成・ナビゲーション実習 ・チュートリアル (MRPT RTC群によるSLAMナビゲーションシステム) 資料: 241122-05.pdf SLAM用RTC等(RTCプログラミング演習のRTM_Tutorial.zipと同じ): RTM_Tutorial.zip

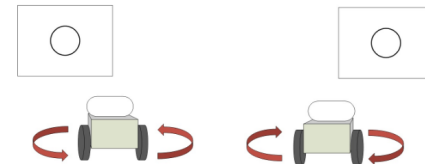


はじめに

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

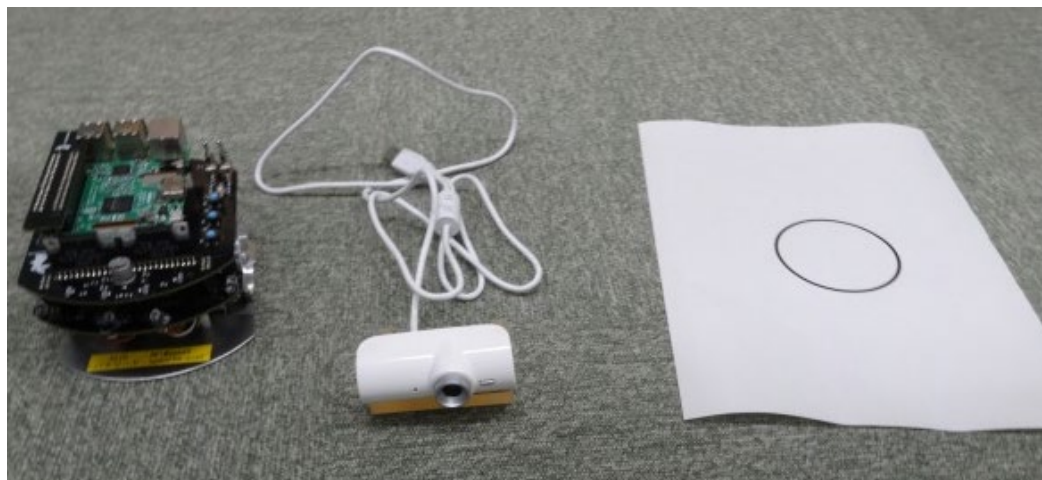
- ・CircleTracking コンポーネント: OpenCVライブラリのHoughCircles関数で画像から円を検出して、検出した円の方向に移動ロボットが回転するように制御するRTC



HoughCircles関数について

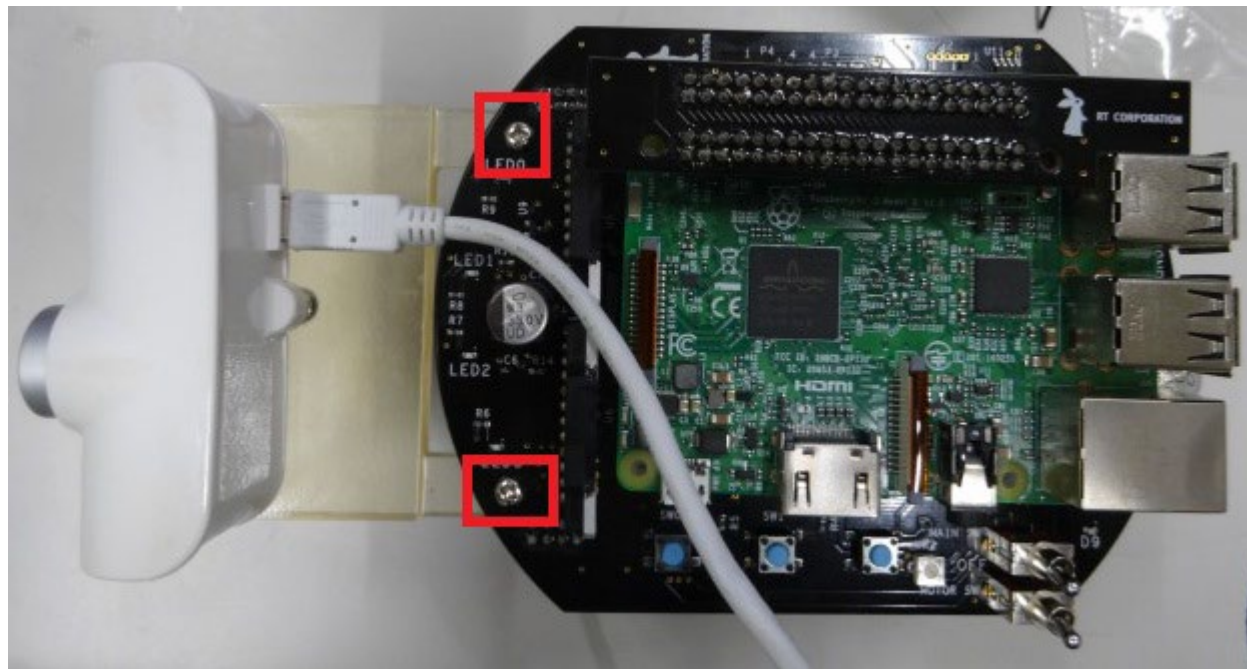
カメラ用マウントの取り付け

- 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



カメラ用マウントの取り付け

- LiDARマウント固定用のネジでカメラ用マウントを固定する



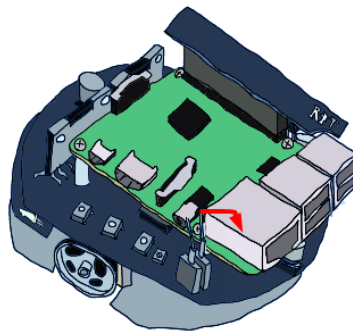
USBカメラとノートPC接続

- ノートPCとUSBカメラをUSBポートで接続する

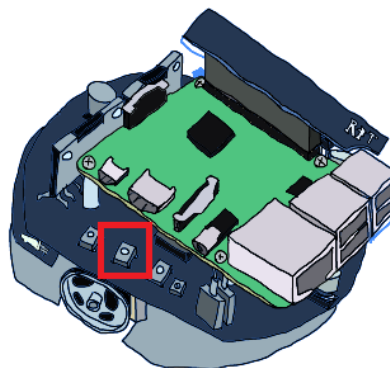


Raspberry Piの起動

- Raspberry Piの電源投入
 - 内側のスイッチをオンにする



- 電源を切る場合
 - 3つ並んだスイッチの中央のボタンを1秒以上押す
 - 10秒ほどでシャットダウンするため、その後に電源スイッチをオフにする



Raspberry Piとの接続

- 無線LANアクセスポイントとの接続
 - SSID、パスワードはRaspberry Piマウス上のシールに記載
 - 接続手順(Windows)
 - 画面右下のネットワークアイコンをクリック



- raspberrypi_xx(もしくはRPiMouse_xx)に接続後、パスワードを入力

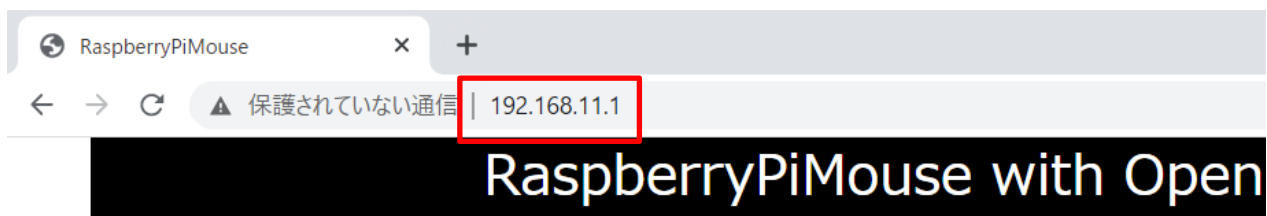


RaspberryPi Mouse LiDAR付きを選択した場合

- **RaspberryPi Mouse V3** (LiDARが付属している) の場合、ネームサーバー、RTCが自動起動しないため、WEBブラウザからの操作で起動する必要がある。



- Chrome、Firefox、Edge等で**192.168.11.1**にアクセスする。



RT-Components

- | | |
|---|--|
| ☐ RaspberryPiMouseRTC | [Start] [Stop] [Activate] [Deactivate] |
| ☐ RPLidarRTC | [Start] [Stop] [Activate] [Deactivate] |
| ☐ Mapper_MRPT | [Start] [Stop] [Activate] [Deactivate] |
| ☐ Localization_MRPT | [Start] [Stop] [Activate] [Deactivate] |
| ☐ PathPlanner_MRPT | [Start] [Stop] [Activate] [Deactivate] |
| ☐ SimplePathFollower | [Start] [Stop] [Activate] [Deactivate] |
| ☐ MapServer(Java) | [Start] [Stop] [Activate] [Deactivate] |
| ☐ NavigationManager(Java) | [Start] [Stop] [Activate] [Deactivate] |

RaspberryPi Mouse LiDAR付きを選択した場合

- ネームサーバーを起動する
 - Start NameServerボタンを押す

RaspberryPiMouse with OpenRTM-aist

RT-Components

☐ RaspberryPiMouseRTC	[Start] [Stop] [Activate] [Deactivate]
☐ RPLidarRTC	[Start] [Stop] [Activate] [Deactivate]
☐ Mapper_MRPT	[Start] [Stop] [Activate] [Deactivate]
☐ Localization_MRPT	[Start] [Stop] [Activate] [Deactivate]
☐ PathPlanner_MRPT	[Start] [Stop] [Activate] [Deactivate]
☐ SimplePathFollower	[Start] [Stop] [Activate] [Deactivate]
☐ MapServer(Java)	[Start] [Stop] [Activate] [Deactivate]
☐ NavigationManager(Java)	[Start] [Stop] [Activate] [Deactivate]

RTC List

- RaspberryPiMouseRTCを起動する
 - RaspberryPiMouseRTCのStartを押す

RaspberryPiMouse with OpenRTM-aist

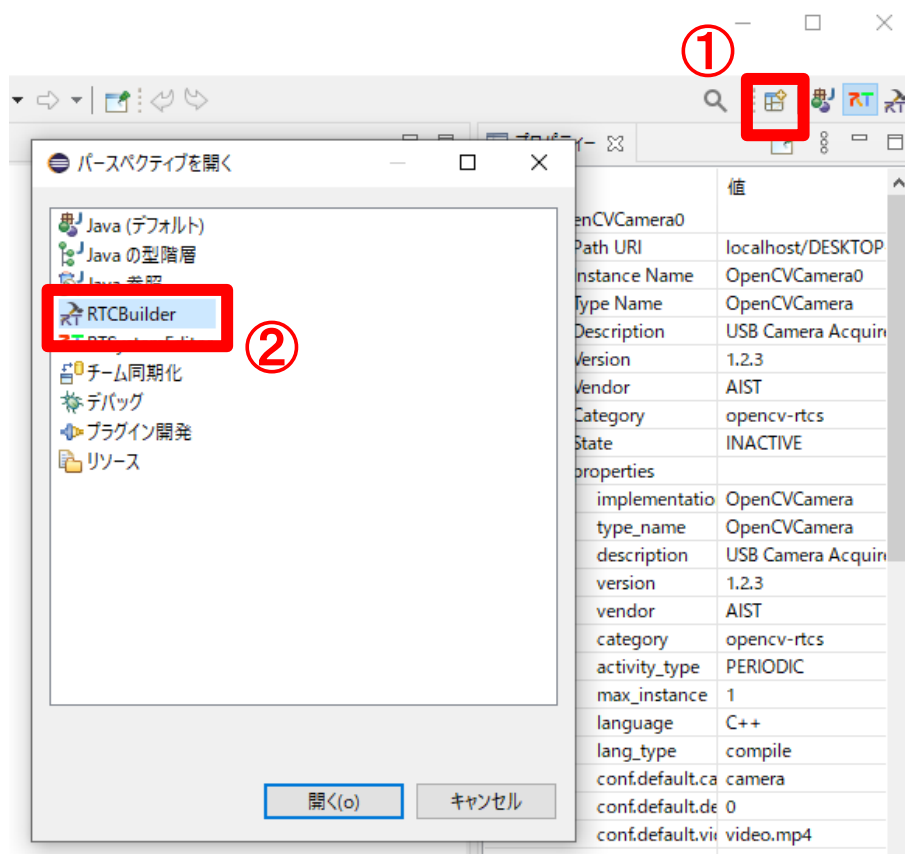
RT-Components

☐ RaspberryPiMouseRTC	[Start] [Stop] [Activate] [Deactivate]
☐ RPLidarRTC	[Start] [Stop] [Activate] [Deactivate]
☐ Mapper_MRPT	[Start] [Stop] [Activate] [Deactivate]
☐ Localization_MRPT	[Start] [Stop] [Activate] [Deactivate]
☐ PathPlanner_MRPT	[Start] [Stop] [Activate] [Deactivate]
☐ SimplePathFollower	[Start] [Stop] [Activate] [Deactivate]
☐ MapServer(Java)	[Start] [Stop] [Activate] [Deactivate]
☐ NavigationManager(Java)	[Start] [Stop] [Activate] [Deactivate]

RTC Builderによる ひな型コード生成

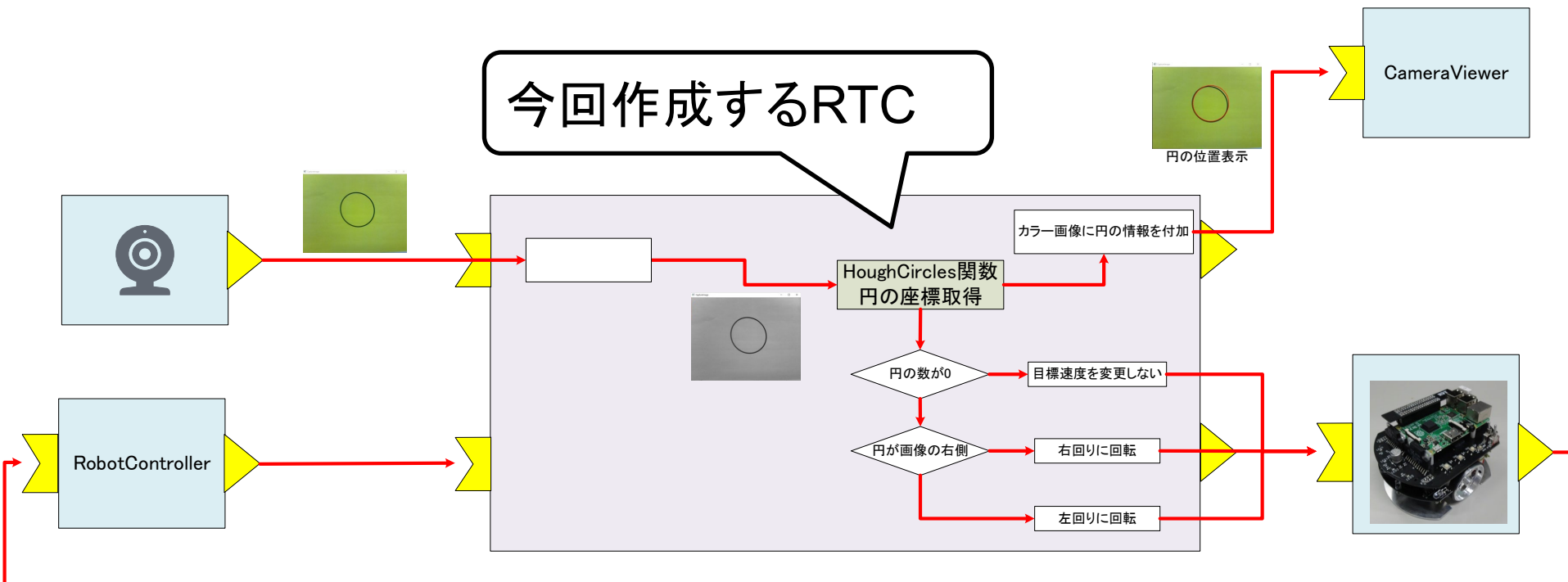
RTCBuilder起動

- RTSystemEditor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



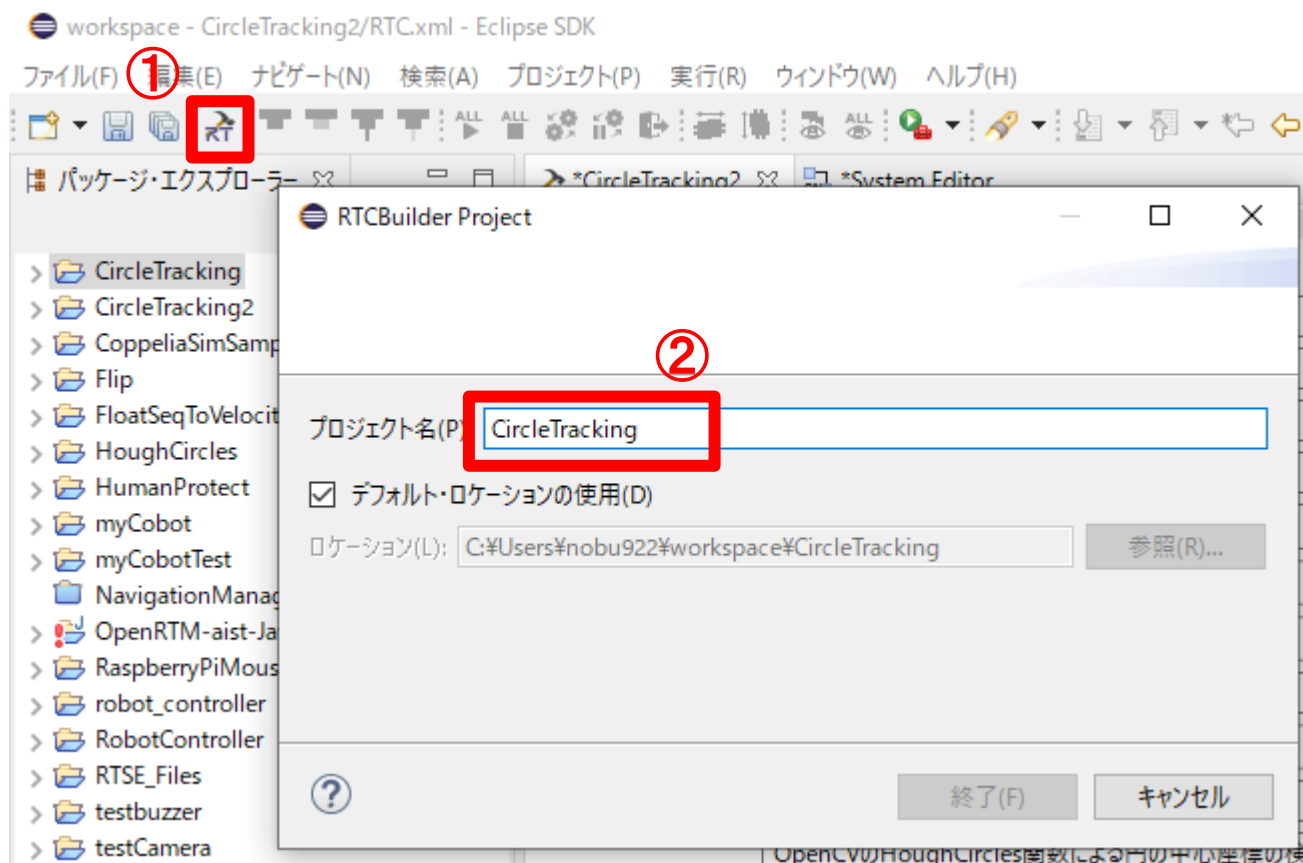
作成するRTCの概要

- 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
*バンド名:	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 interface. It features two main sections for port management: InPort and OutPort. Each section has a list of ports, 'Add' and 'Delete' buttons, and a scroll bar. Below these is a 'Detail' section with a dropdown menu for selecting a port. The 'image_in (InPort)' is selected, and its data type is set to 'RTC::CameraImage'. A 'ReLoad' button is also present.

*ポート名 (InPort)

image_in
velocity_in

Add
Delete

*ポート名 (OutPort)

image_out
velocity_out

Add
Delete

▼ Detail

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。

ポート名 : image_in (InPort)

*データ型 RTC::CameraImage ReLoad

NATIONAL INSTITUTE OF ADVANCED INDUSTRIAL SCIENCE AND TECHNOLOGY (AIST)

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 for data ports. It features two main sections: 'InPort' and 'OutPort'. The 'InPort' section has a list with 'image_in' and 'velocity_in', and buttons for 'Add' and 'Delete'. The 'OutPort' section has a list with 'image_out' and 'velocity_out', and buttons for 'Add' and 'Delete'. Below these is a 'Detail' section with a description: '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。' (This section describes the documents for each data port. Selecting the data port above allows you to describe the documents for each). The 'ポート名' (Port Name) field is set to 'velocity_out (OutPort)'. At the bottom, the '*データ型' (Data Type) is set to 'RTC::TimedVelocity2D' with a 'ReLoad' button.

*ポート名 (InPort)	*ポート名 (OutPort)
image_in	image_out
velocity_in	velocity_out

▼ Detail

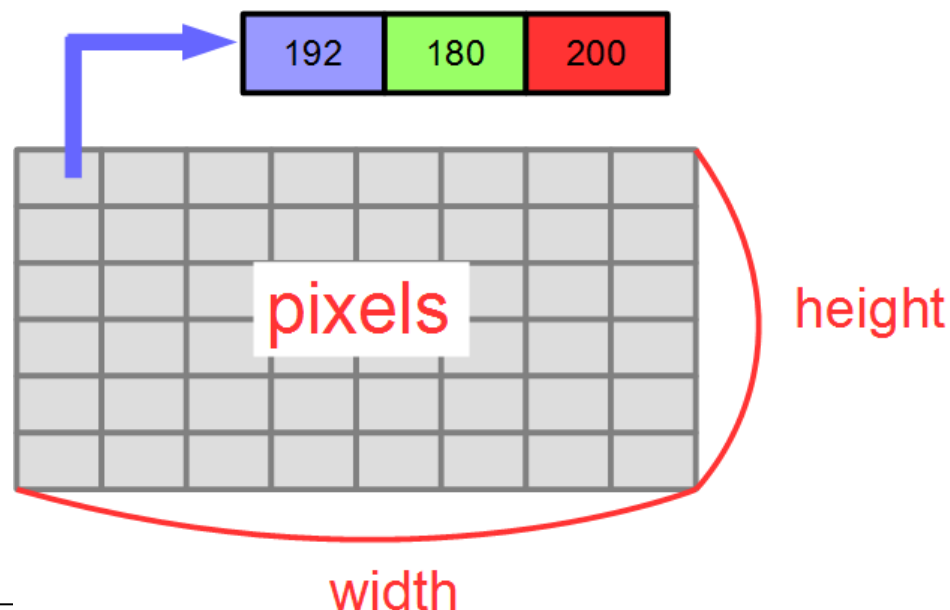
このセクションではデータポート毎の概要を説明するドキュメントを記述します。
上のデータポートを選択すると、それぞれのドキュメントが記述できます。

ポート名: velocity_out (OutPort)

*データ型: RTC::TimedVelocity2D ReLoad

RTC::CameraImage型

- 画像データをやり取りするためのデータ型
 - **width** : 画像の幅
 - **height** : 画像の高さ
 - **bpp** : 色深度
 - **format** : フォーマット名(jpeg、png)
 - **fDiv** : スケールファクタ
 - **pixels** : 画像データ



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

右回転、左回転する時の回転速度

*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 : speed_r

*データ型	double
*デフォルト値	0.5
変数名	
単位	rad/s
制約条件	$0.0 < x < 2.0$
Widget	slider
Step	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	

▼ 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 a configuration window with a table of parameters. The parameter 'houghcircles_param2' is selected and highlighted in blue. Below the table, there is a 'Detail' section for the selected parameter. The detail section contains the following fields:

*名称	
houghcircles_minDist	
houghcircles_param1	
houghcircles_param2	
houghcircles_minRadius	
houghcircles_maxRadius	

Buttons: Add, Delete

▼ 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_param2

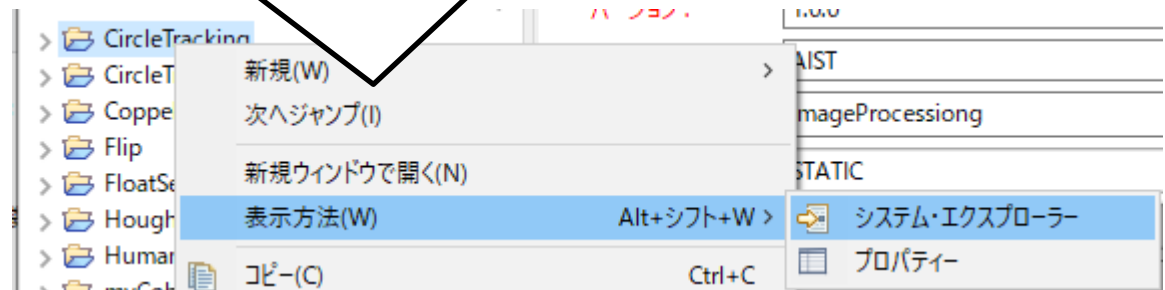
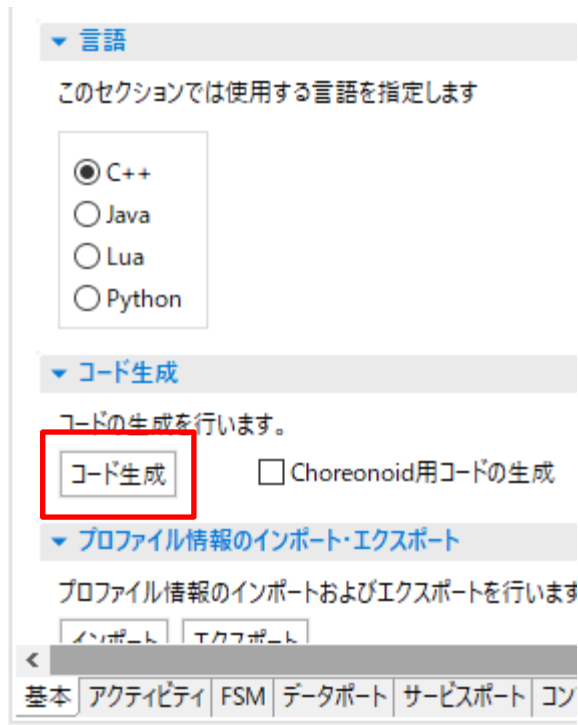
*データ型	double
*デフォルト値	100
変数名:	
単位:	
制約条件:	
Widget:	text
Step:	

HoughCircles関数で指定する
パラメータ (6個)

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

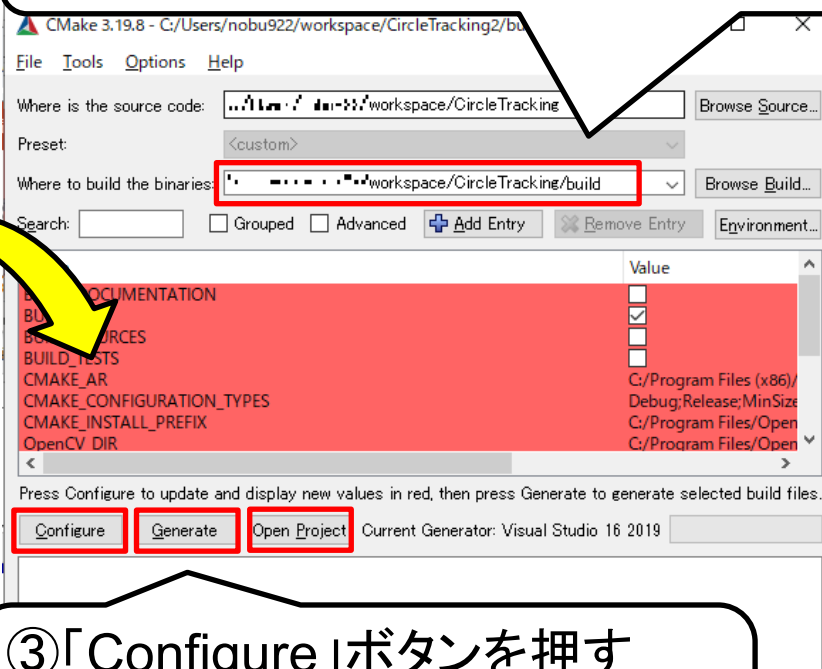
- CMake(cmake-gui)でビルドに必要なファイル (Visual Studioのプロジェクトファイル等)を生成

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	
cmake	2022/09/29 12:04	ファイル フォルダ	
doc	2022/09/29 12:04	ファイル フォルダ	
idl	2022/09/29 12:04	ファイル フォルダ	
include	2022/09/29 12:04	ファイル フォルダ	
scripts	2022/09/29 12:04	ファイル フォルダ	
src	2022/09/29 15:30	ファイル フォルダ	
test	2022/09/29 12:04	ファイル フォルダ	
.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

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

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

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  
m_outputBuff = m_imageBuff.clone();
```

```
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);
```

検出した円の情報を元の画像に付加して、OutPort (image_out) から出力する処理

```
}
```

```
// 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

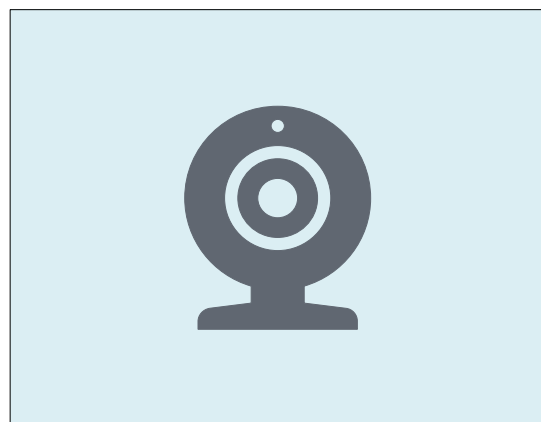
ここまで追加編集が完了したらビルドする。

RTCの起動

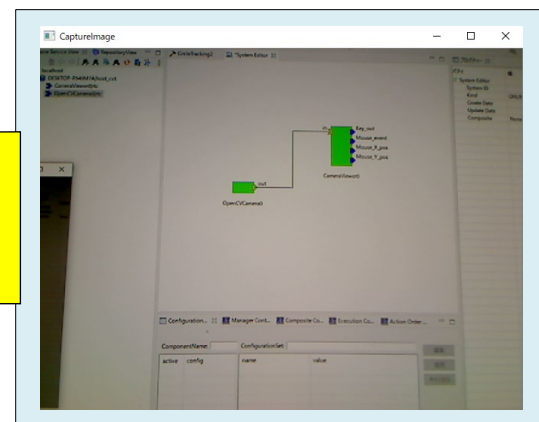
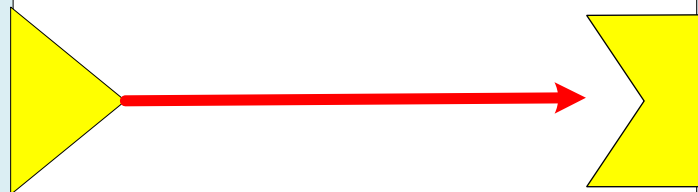
- 以下の5つのRTCを起動する
 - **RaspberryPiMouseRTC** (Raspberry Piで起動)
 - **RobotController** (第3部の実習で作成)
 - 「RTコンポーネントの作成入門」の資料を参照
 - <https://openrtm.org/openrtm/ja/node/6550>
 - **OpenCVCamera**
 - **CameraViewer**
 - OpenRTM-aistのサンプルコンポーネント
 - 次のスライドで説明
 - **CircleTracking**
 - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内の実行ファイル(CircleTrackingComp.exe)

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



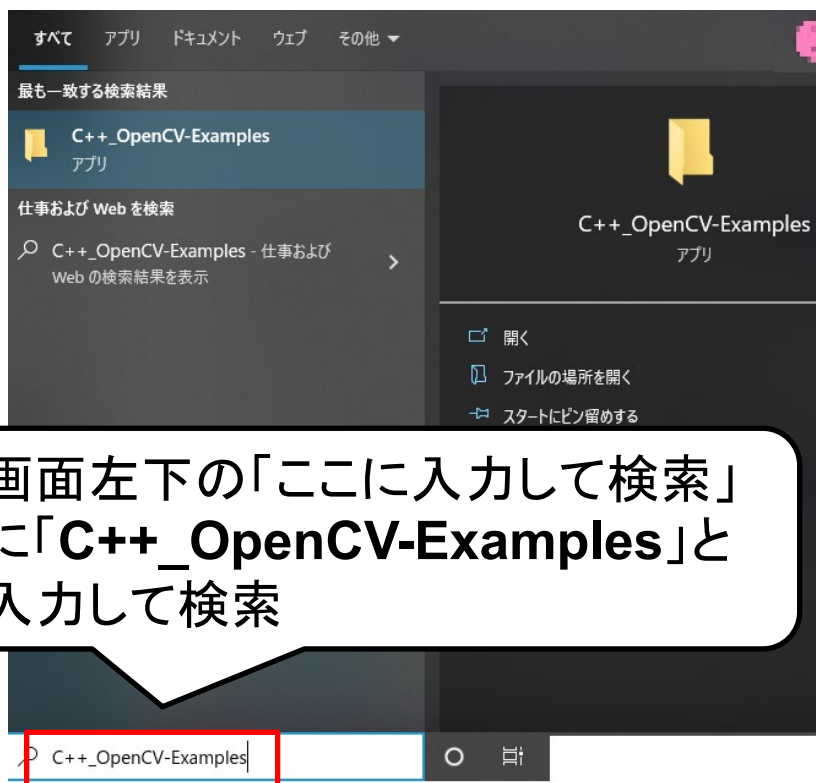
OpenCVCamera



CameraViewer

サンプルコンポーネントの起動

- 起動手順
 - Windows



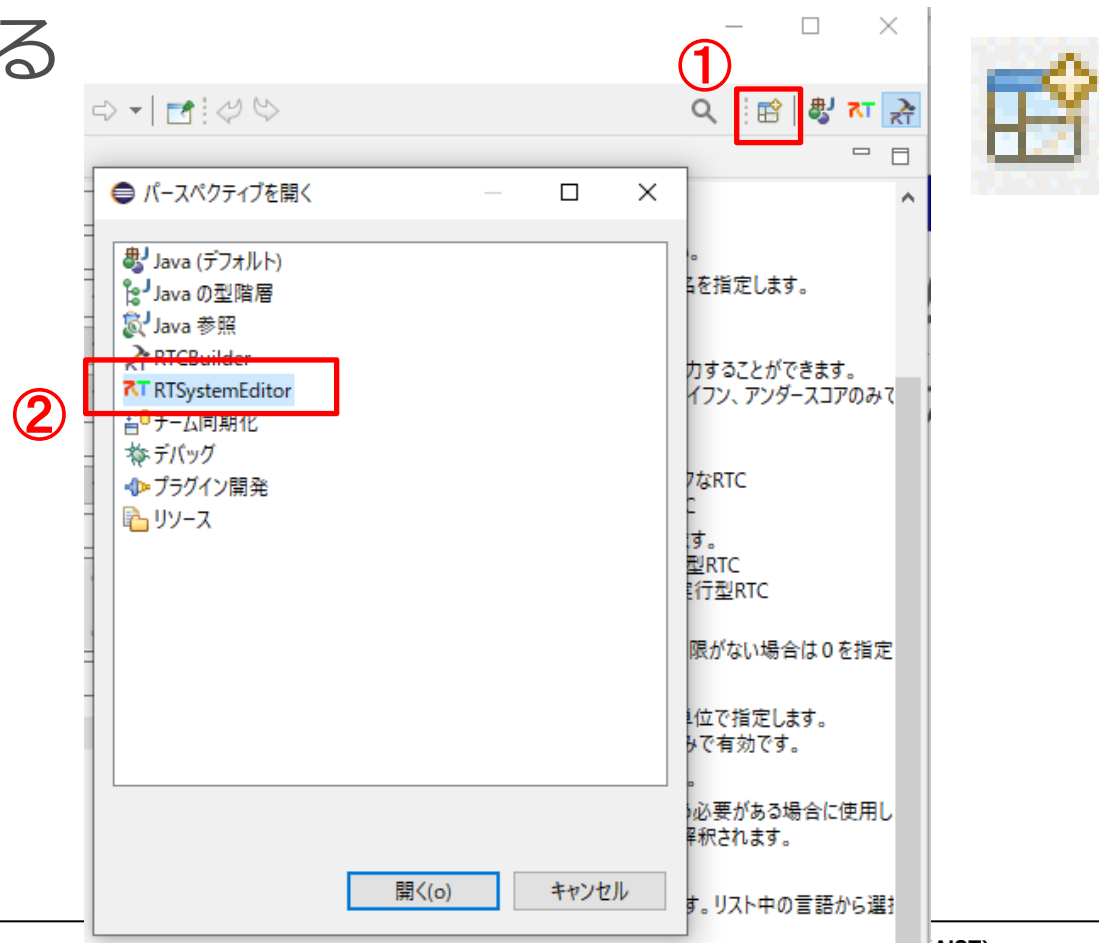
エクスプローラーから、以下のファイルをダブルクリックする

- 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...	1 KB
BackGroundSubtractionSimple.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Binarization.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
CameraViewer.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
ChromaKey.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
DilationErosion.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Edge.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Findcontour.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Flip.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Histogram.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Hough.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
ImageCalibration.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
ImageSubtraction.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
ObjectTracking.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
OpenCVCamera.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB
Perspective.bat	2022/03/24 20:46	Windows バッチ ファ...	1 KB

RTシステム構築

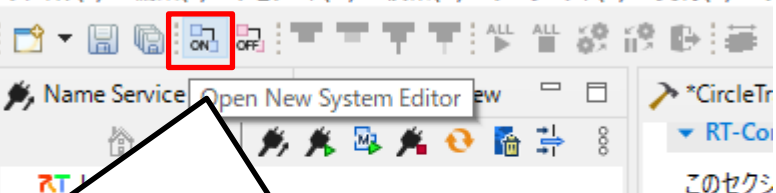
- RTSystemEditorを起動する。
 - 「パースペクティブを開く」画面から起動する



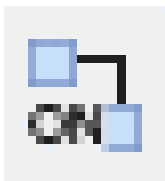
動作確認

workspace - CircleTracking2/RTC.xml - Eclipse SDK

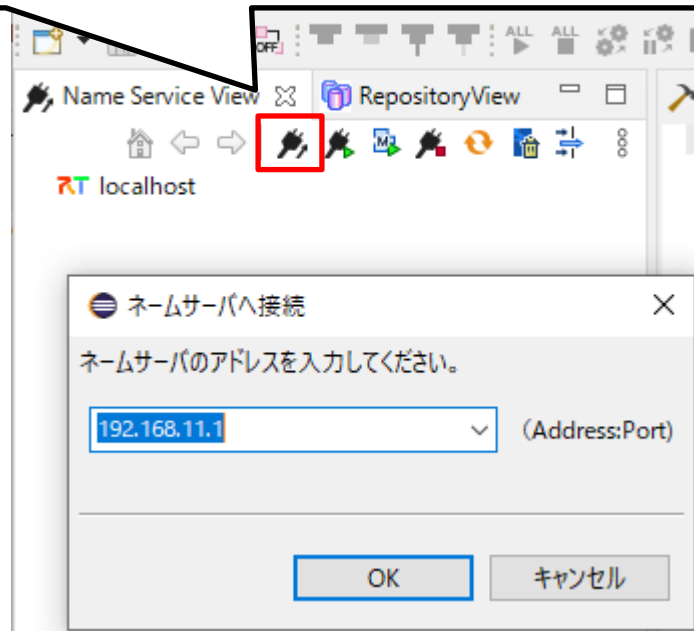
ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウ



システムダイアグラムを表示する

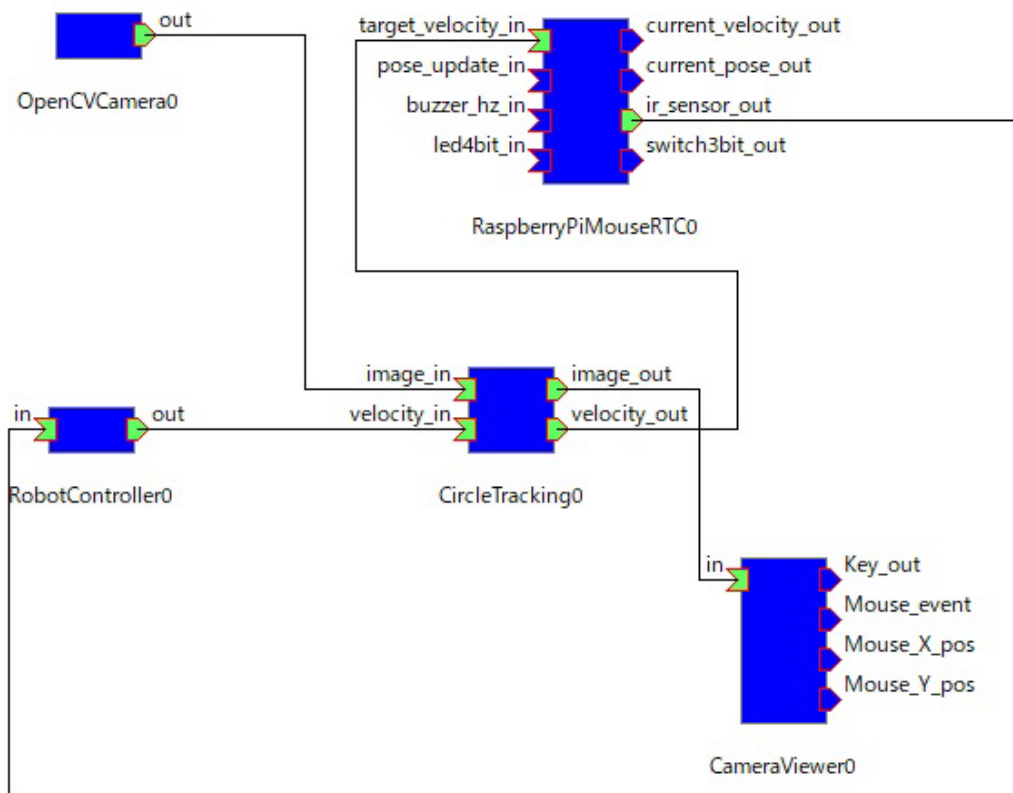


ネームサービスビューに「192.168.11.1」がない場合は、192.168.11.1のネームサーバーへ接続する



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

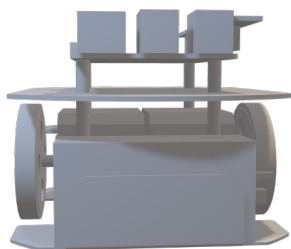
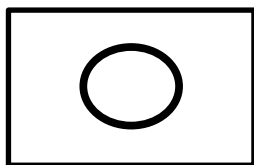


アクティブ化には
「Activate Systems」ボタン
を押す。

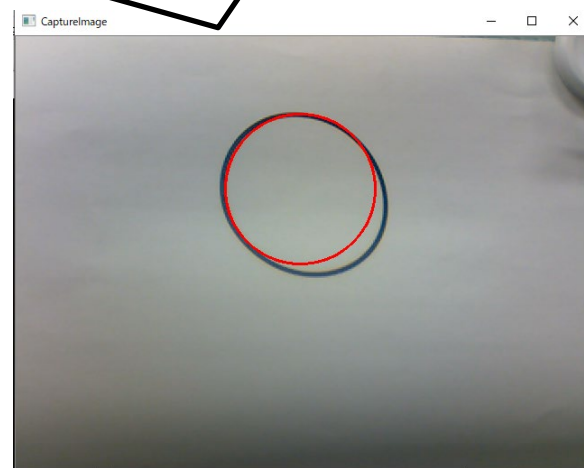


動作確認

- 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



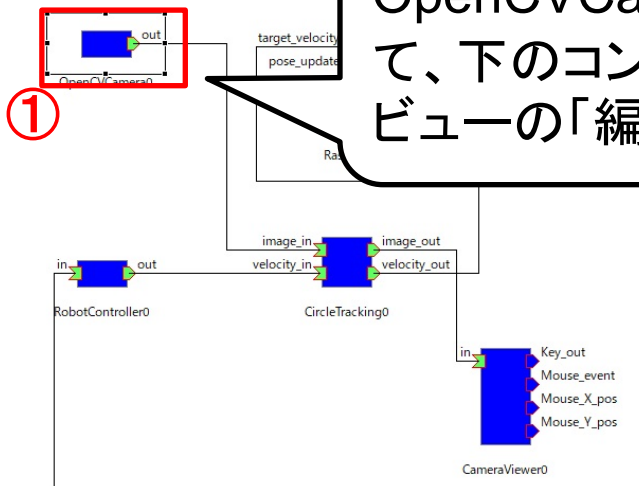
円が正常に検出できているかを、CameraViewerで確認する



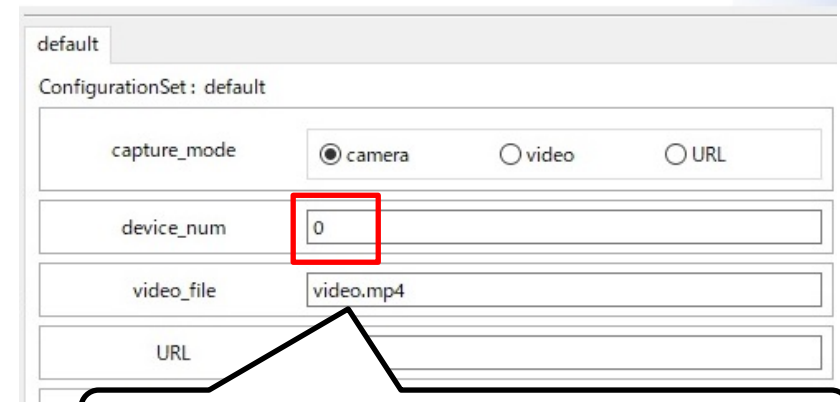
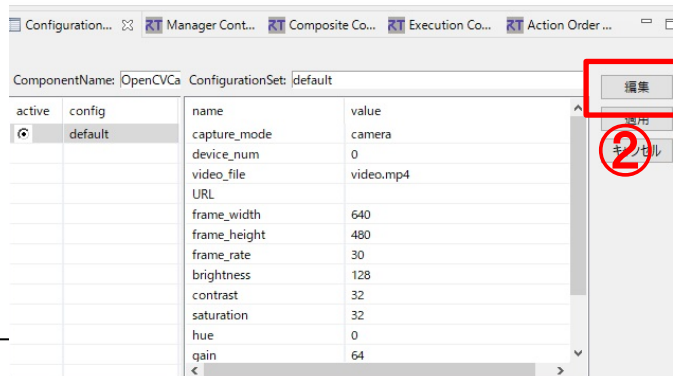
- 課題：左右に小刻みに動く問題を修正する

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



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

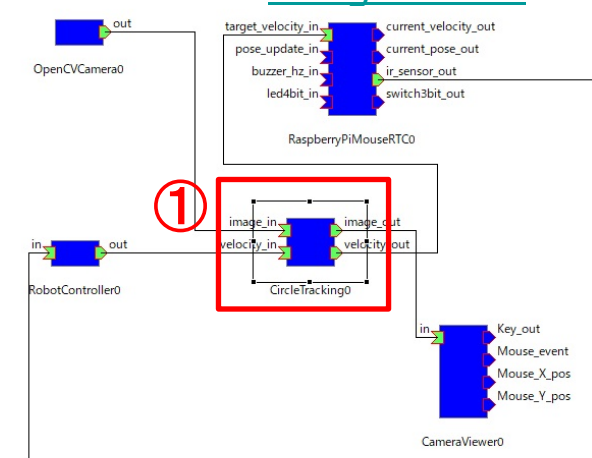


「device_num」の値を変更する

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



Configuration window for CircleTrack component.

active	config	name	value
●	default	speed_r	0.5
		houghcircles_dp	2
		houghcircles_minDist	30
		houghcircles_param1	100
		houghcircles_param2	100
		houghcircles_minRadius	0
		houghcircles_maxRadius	0

Buttons: 編集 (Edit), 適用 (Apply), キャンセル (Cancel)

Configuration window for CircleTrack component.

default ConfigurationSet: default

speed_r: 0.5

houghcircles_dp: 2

houghcircles_minDist: 30

houghcircles_param1: 100

houghcircles_param2: 100

houghcircles_minRadius: 0

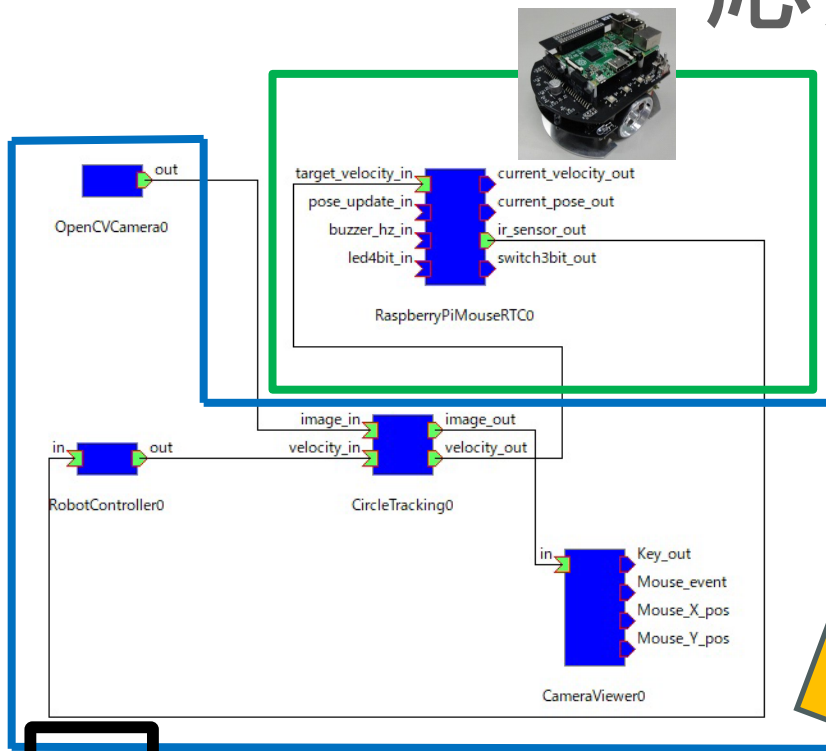
houghcircles_maxRadius: 0

各パラメータを変更する

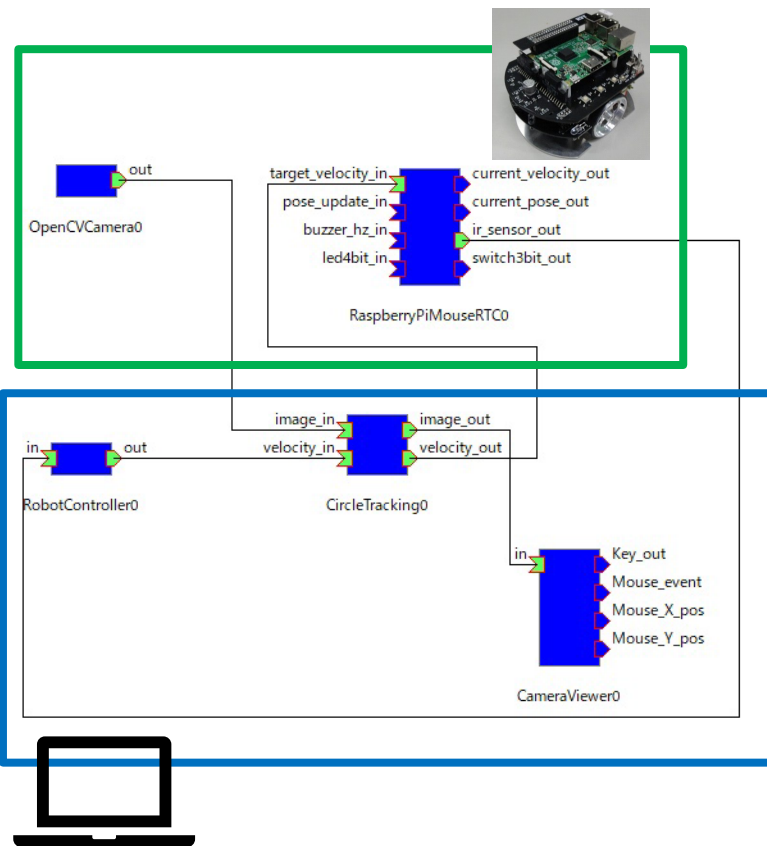
応用課題

- Raspberry Pi上のOpenCVCameraコンポーネントと接続する
- TeraTermでRaspberry PiにSSH接続し、OpenCVCameraコンポーネントを起動するコマンドを実行する
 - 起動後、Raspberry Pi上のOpenCVCameraコンポーネントを使ったシステムを構築する。
 - データの転送が遅い場合は、画像を圧縮する。

応用課題



Raspberry PiにUSBカメラ
を接続する



リモートログイン

- Raspberry Pi上のLinuxにログインして、
コマンドによりOpenCVCameraコンポー
ネントを起動する

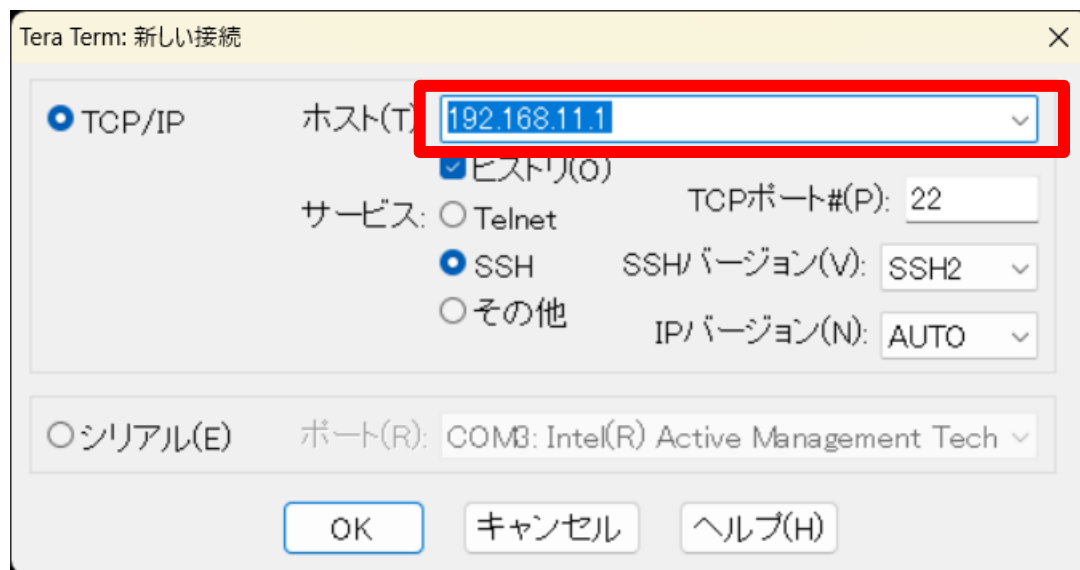


- Tera Termを起動する
– デスクトップのショートカットから起動



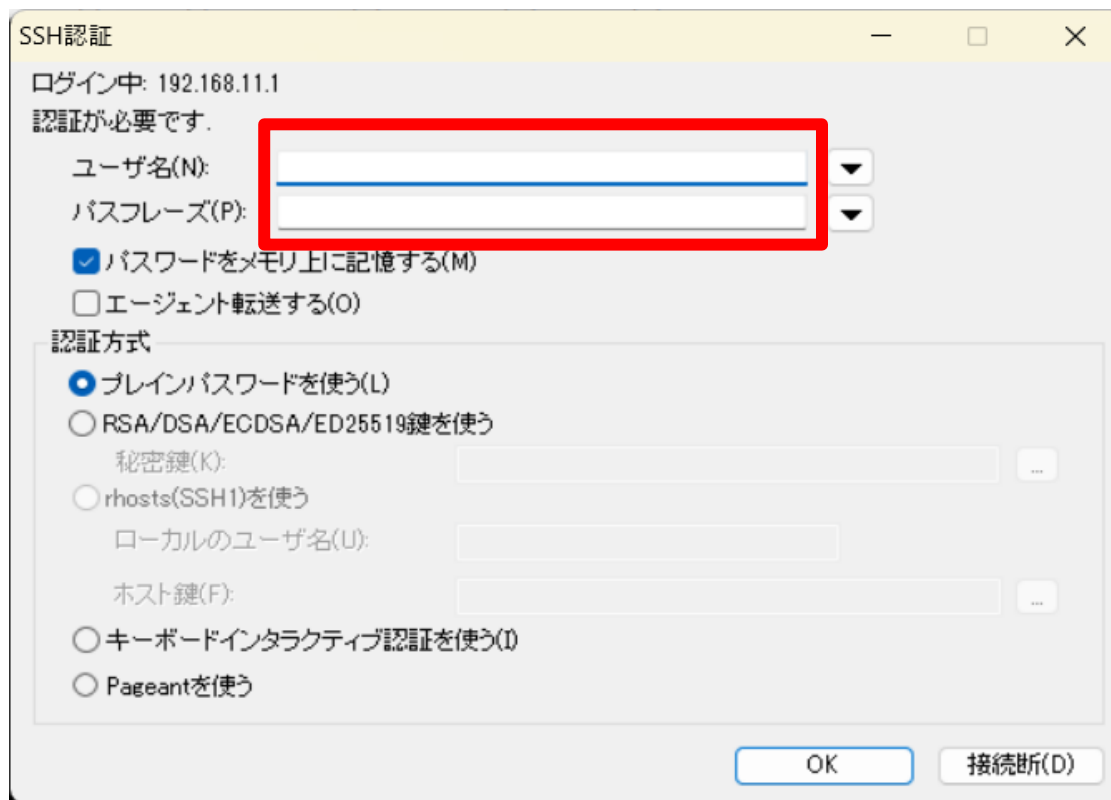
Tera Termの操作

- 「ホスト」に**192.168.11.1**を入力して「OK」をクリック



Tera Termの操作

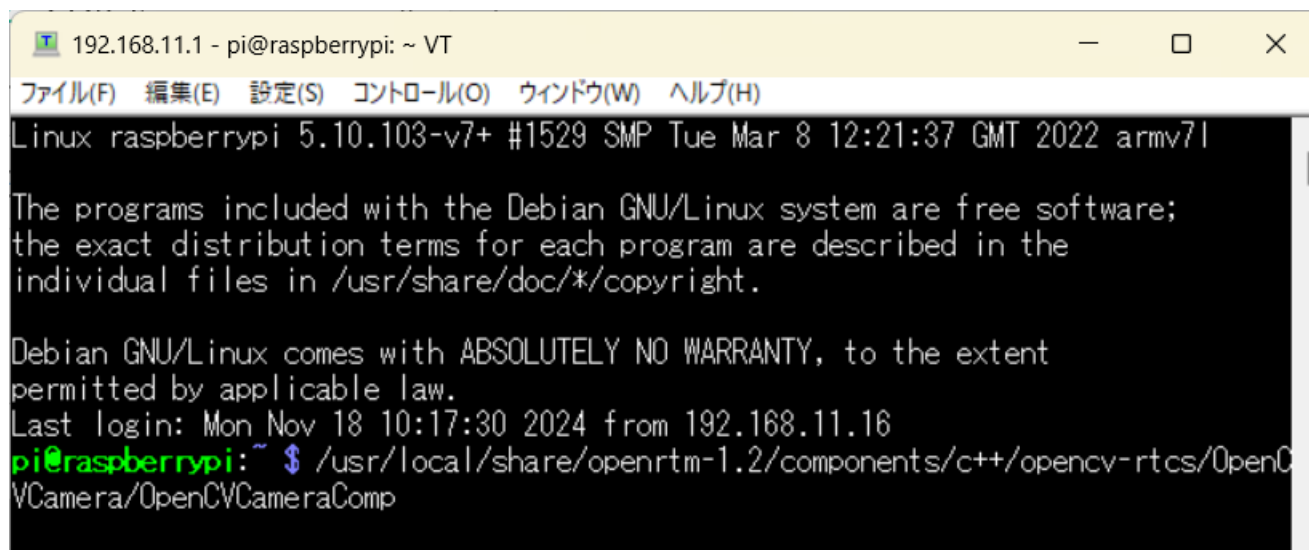
- ユーザー名とパスワードを入力して「OK」をクリック



Tera Termの操作

- OpenCVCameraCompをコマンドで起動する

```
/usr/local/share/openrtm-1.2/components/c++/opencv-rtcs/OpenCVCamera/OpenCVCameraComp
```



The screenshot shows a Tera Term terminal window titled "192.168.11.1 - pi@raspberrypi: ~ VT". The terminal displays the following text:

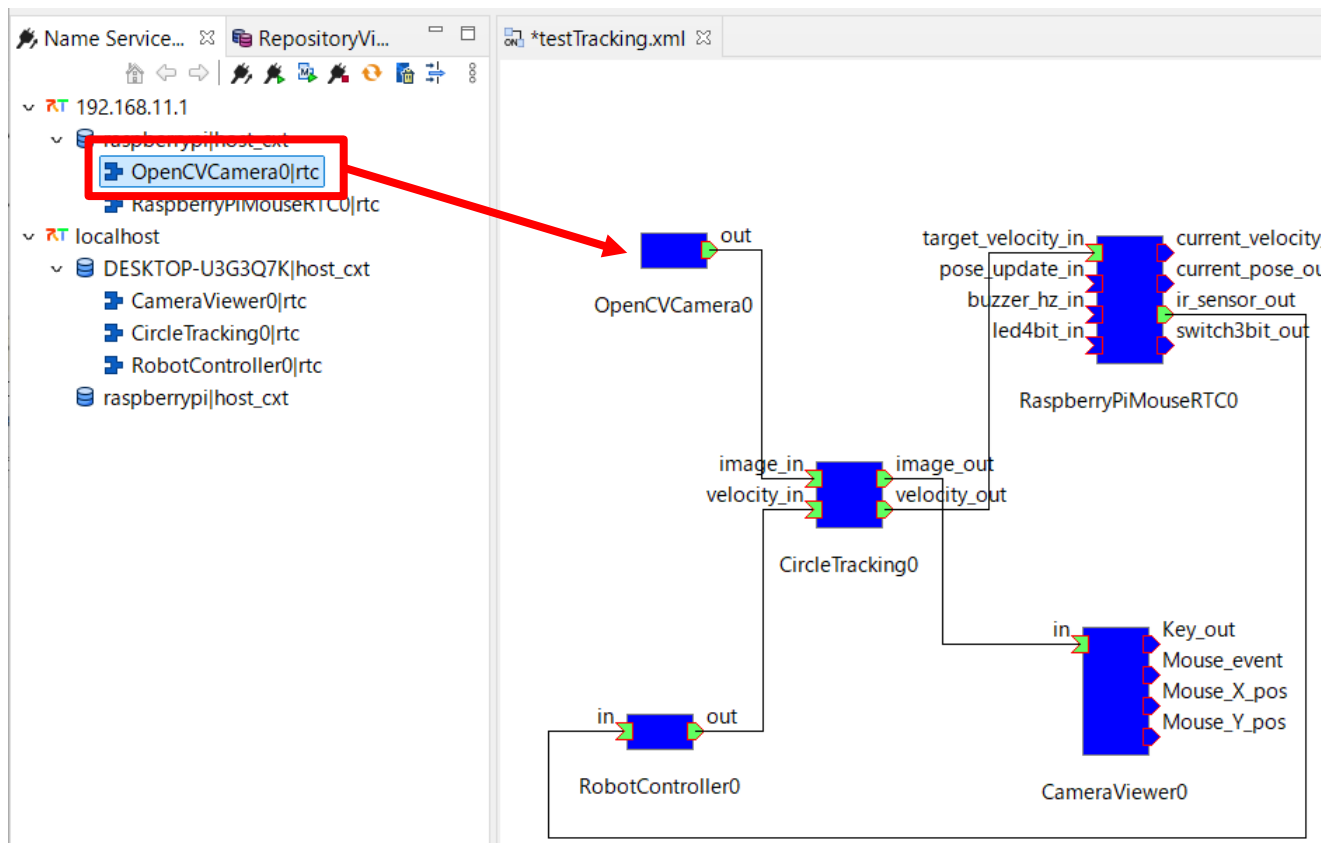
```
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Linux raspberrypi 5.10.103-v7+ #1529 SMP Tue Mar 8 12:21:37 GMT 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Nov 18 10:17:30 2024 from 192.168.11.16
pi@raspberrypi:~ $ /usr/local/share/openrtm-1.2/components/c++/opencv-rtcs/OpenCVCamera/OpenCVCameraComp
```

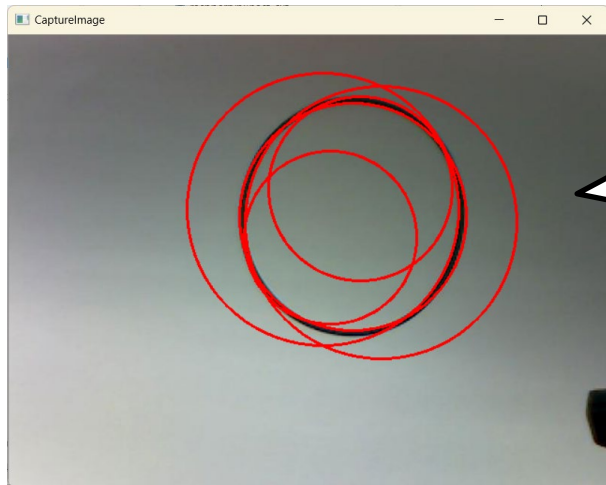
RTシステム構築

- Raspberry PiのネームサーバーのOpenCVCameraコンポーネントを接続する



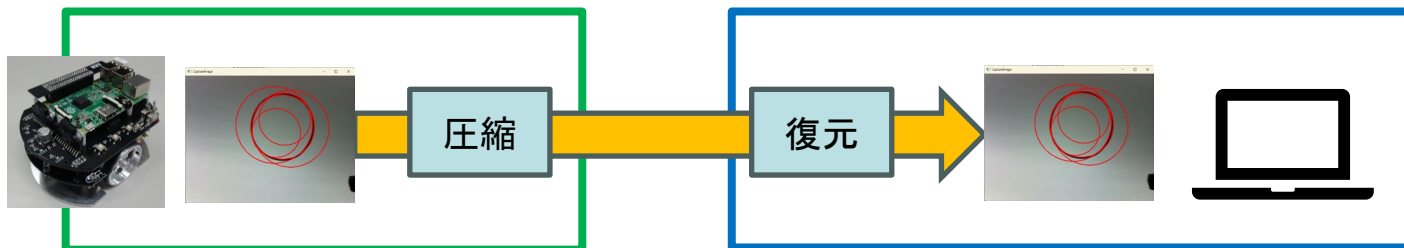
動作確認

- RTCをアクティブ化して動作確認



問題点: カメラ画像の遅延が大きい

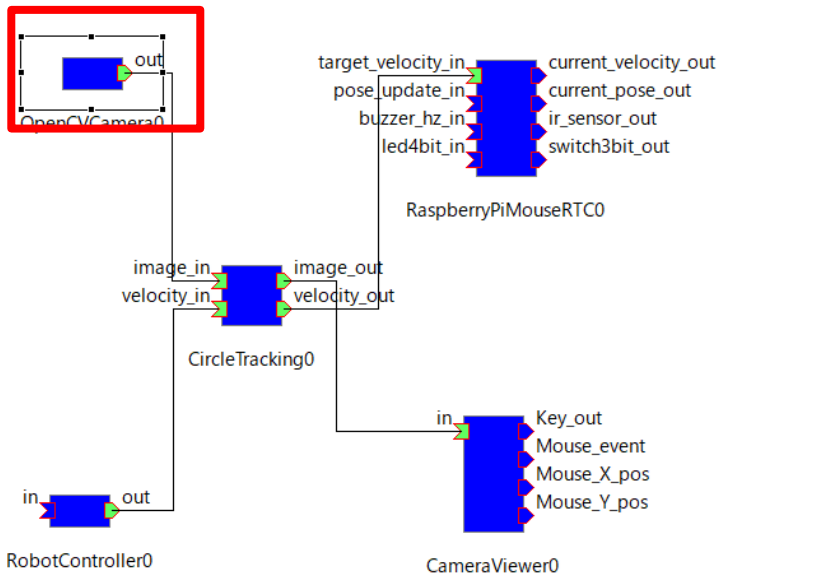
圧縮した画像を送信するように変更する



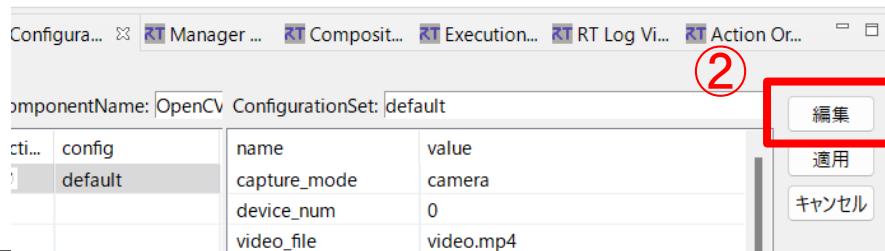
OpenCVCameraの設定

- OpenCVCamera0のコンフィギュレーションパラメータを変更する

①



②



③

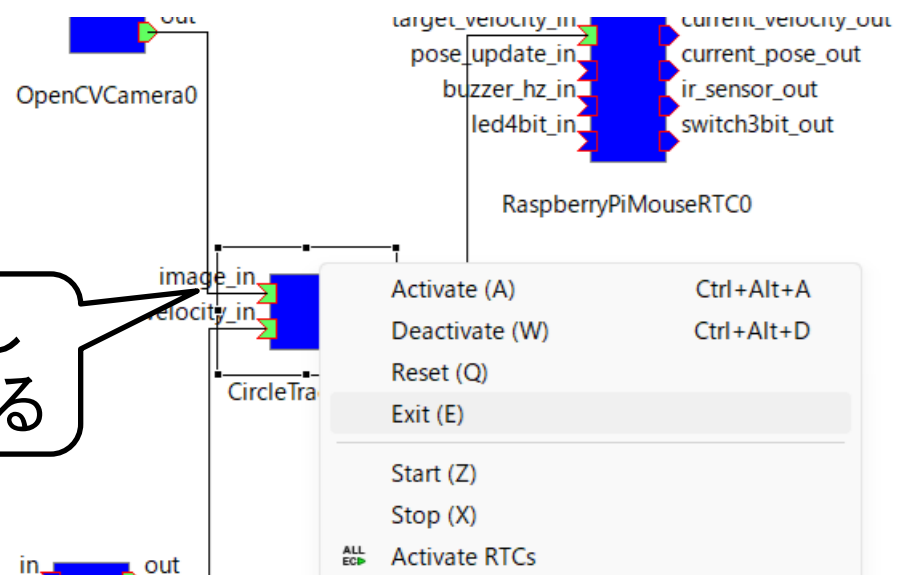


string_encodeを「jpeg」に
設定する

CircleTrackingの変更

- CircleTracking0を一旦終了する

CircleTracking0を右クリックして「Exit」をクリックして終了する



CircleTrackingの変更

- Visual StudioでCircleTracking.cppを編集してonExecute関数に以下のコードを追加する

```
//圧縮のフォーマットにより処理を分岐
std::string format = (const char*)m_image_in.format;
if (format == "jpeg" || format == "png")
{
    std::vector<uchar> buff;
    int len = m_image_in.pixels.length();
    buff.resize(len);
    memcpy(&buff[0], &m_image_in.pixels[0], sizeof(unsigned char) * len);
    m_imageBuff = cv::imdecode(cv::Mat(buff), cv::IMREAD_COLOR);
}
else
{
    std::memcpy(m_imageBuff.data,
                (void*)&(m_image_in.pixels[0]),
                m_image_in.pixels.length());
}
/*
// 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());
*/
```

CameraImage型
構造体のformat変
数に画像の圧縮形
が格納される

m_imageBuffに画像デ
ータをコピーするコードの
上に追加する。この行は
コメントアウトする。

再度動作確認

- Visual Studioでビルド後、CircleTrackingを起動して動作確認する

② RTCをアクティブ化する

① データポートを再接続する。
一度接続して、右クリックで「Save System」でシステムを保存後、「Load System」で復元すると再接続の手間が省ける

