

2020年11月11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://bit.ly/326O7iE>
- 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。


OpenRTM-aist
The power to connect

[ホーム](#)
[ダウンロード](#)
[ドキュメント](#)
[コミュニティ](#)
[研究開発](#)
[プロジェクト](#)
[ハードウェア](#)

ホーム » 高度ポリテクセンター「RTミドルウェアによるロボットプログラミング技術」(2020年11月10日～11日)

高度ポリテクセンター「RTミドルウェアによるロボットプログラミング技術」 (2020年11月10日～11日)

Facebookに友達して、最新の「いいね!」を見てみよう。

Table of contents

- 開催概要
- 日時・場所
- プログラム
 - Raspberry Piマウス
 - インストールするソフトウェア
- 資料
 - 1. コース概要
 - 2. プログラミングの基礎
 - 3. RTMによるプログラミング
 - 4. ロボットの運動学と制御の基礎
 - 5. 総合演習



開催概要

2020年11月10日(火)～11月11日(水)の日程で、高度ポリテクセンターの能力開発セミナーの一環として、「RTミドルウェアによるロボットプログラミング技術」講習会を開催いたします。

● コース案内: <https://www.apc-jeed.or.jp/zaishoku/2020/E0771.html>

日時・場所

● 日時: 2019年11月13日(火), 14日(水), 10時00分～16時45分

● 場所: 高度ポリテクセンター

● 交通アクセス

● 主催: 高度ポリテクセンター

● 参加費: 20,000円

プログラム

11月10日(火)	
10:00 - 10:50	1. コース概要 (1) ロボットシステムプログラミングの現状 (2) ロボットOS・ミドルウェア (3) RTミドルウェア(RTM)を用いたロボット開発 資料: 201110-01.pdf

概要

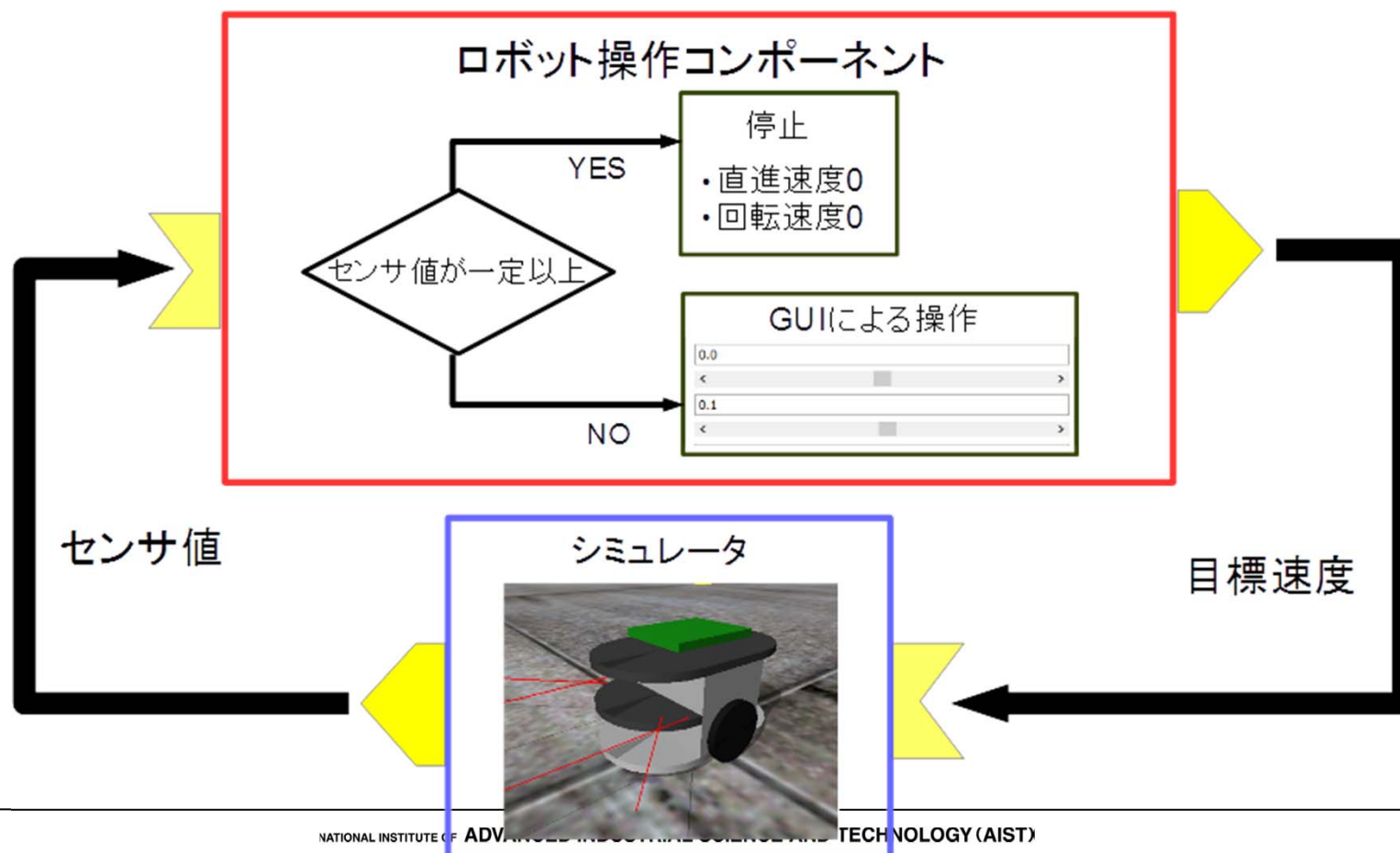
1. ロボット操作コンポーネントの作成
2. シミュレータロボットと接続してテスト
3. 実機ロボットと接続してテスト

目標：

1. センサ入力を利用して動作を制御するRTCを開発する。
2. シミュレータ内ロボットRTCに接続して動作させる方法を学ぶ。
3. 実機ロボットRTCに接続してシミュレータ同様に動作可能であることを確認する。

実習内容（１）

- シミュレータ上の車輪型移動ロボット(Raspberry Piマウス)の操作を行うコンポーネントの作成
 - GUIにより目標速度入力
 - センサ値が一定以上の場合に停止



チュートリアル

- 本日のコースのページから以下のページに移動
 - チュートリアル (RaspberryPiマウスシミュレータ)

11月11日 (水)	
10:00 -11:45	4. 総合演習(1) (1) ロボットシステムの設計 (2) ロボット制御プログラムの作成 ・チュートリアル (Raspberry Pi Mouseシミュレータ、Windows編) ・チュートリアル (RaspberryPiマウス) 資料: 201111-04.pdf
11:45 -12:30	昼食
12:30 -16:45	5. 総合演習'(2) (1) SLAMについて (2) SLAMにより地図作成・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: 201111-05.pdf

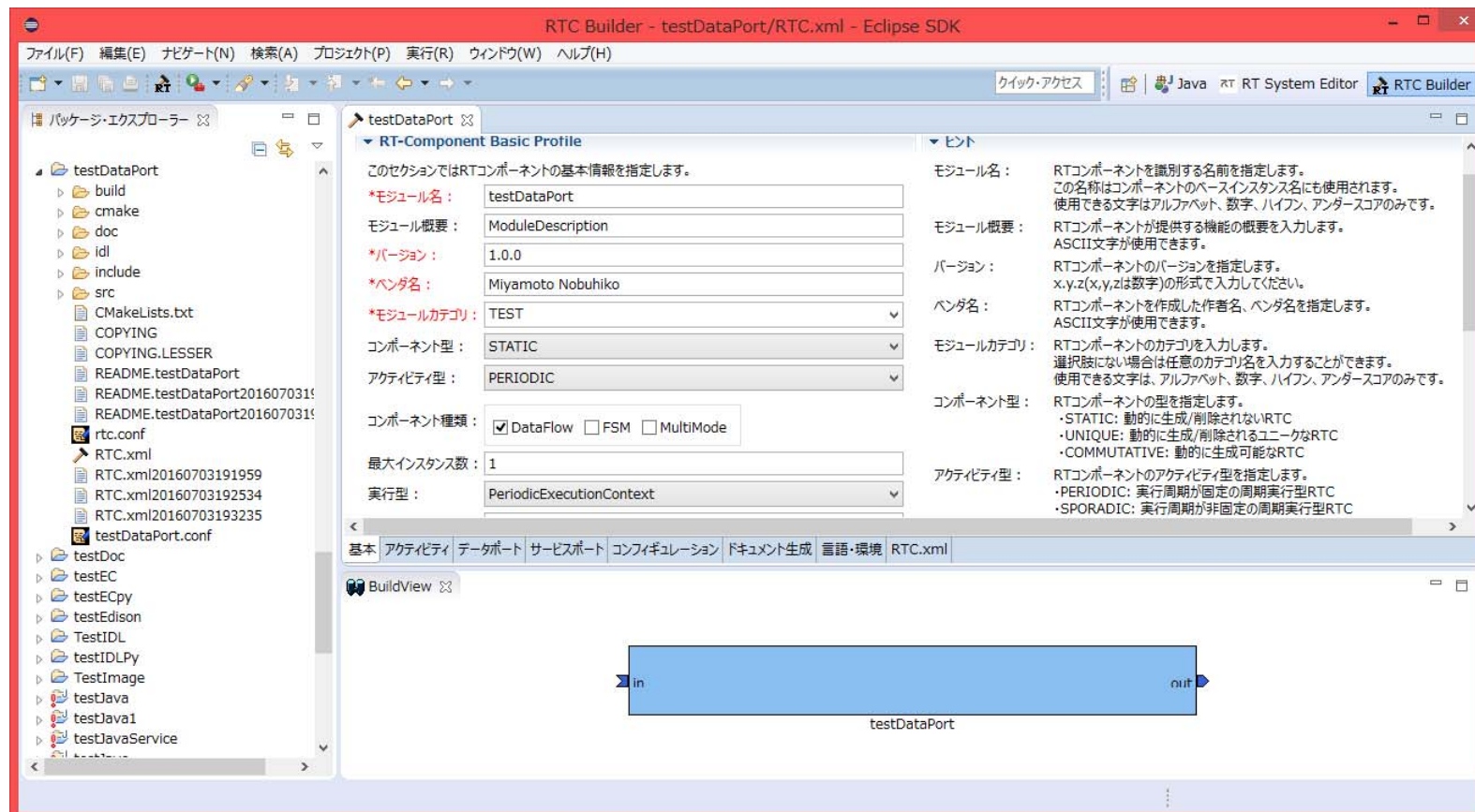
全体の手順

- RTC Builderによるソースコード等のひな型の作成
- ソースコードの編集、ビルド
 - ビルドに必要な各種ファイルを生成
 - CMakeLists.txtの編集
 - CMakeにより各種ファイル生成
 - ソースコードの編集
 - RobotController.h、 RobotController.cppの編集
 - ビルド
 - Visual Studio、 Code::Blocks
- RTシステムエディタによるRTシステム作成、動作確認
 - RTシステム作成
 - データポート接続、コンフィギュレーションパラメータ設定

コンポーネント開発ツール RTC Builderについて (おさらい)

RTCBuilder

- コンポーネントのプロファイル情報を入力し、ソースコード等のひな型を生成するツール
 - C++、Python、Javaのソースコードを出力



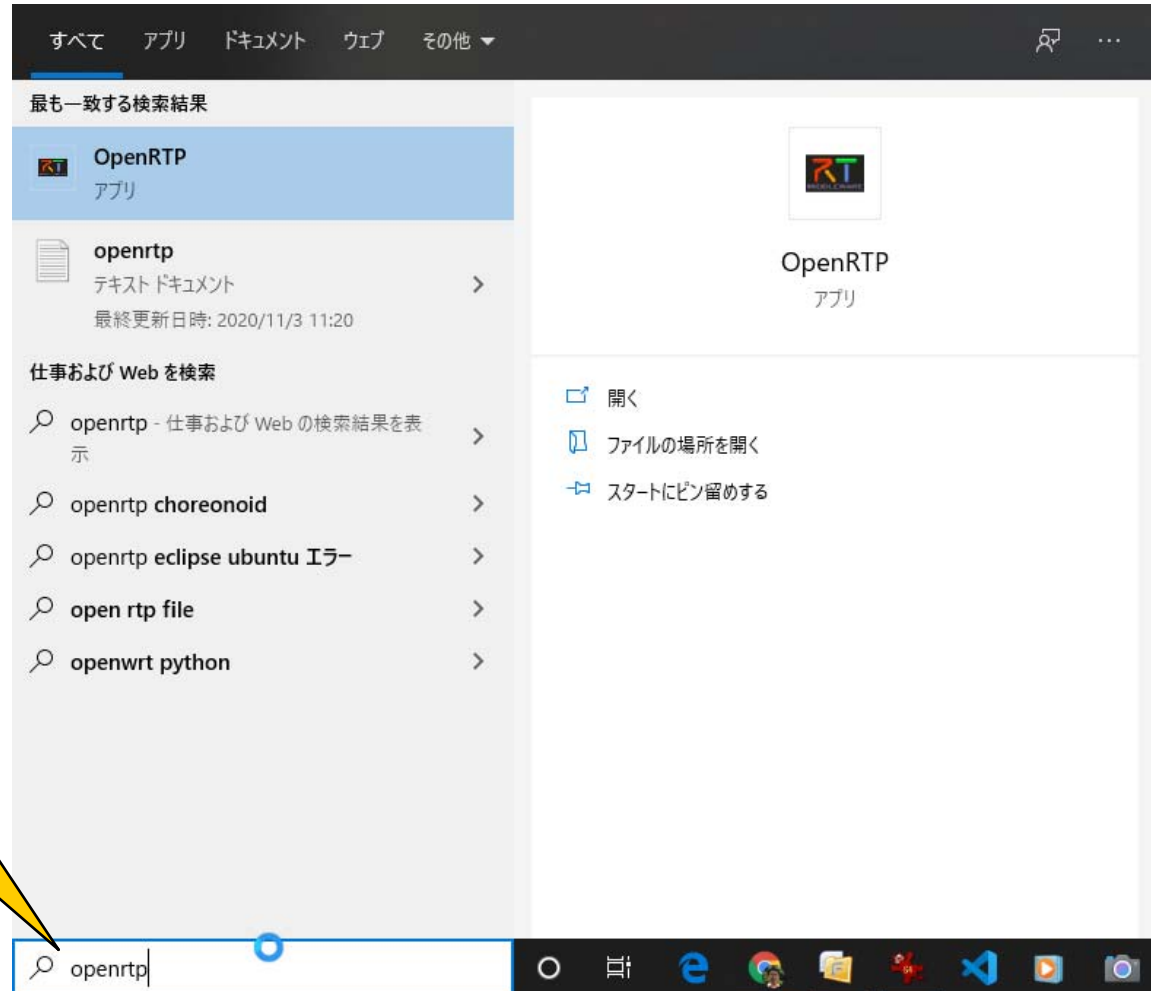
RTC Builderの起動

- 起動する手順
 - Windows 10
 - 左下検索窓に「OpenRTP」を入力
 - Ubuntu
 - 以下のコマンドを入力
 - \$./openrtp

RTC Builderの起動

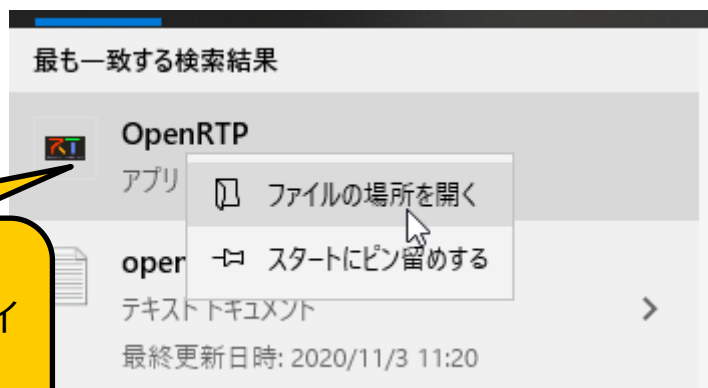
- Windows 10

左下の検索窓に“openrtp”と入力すると“OpenRTP”が候補として出てくる

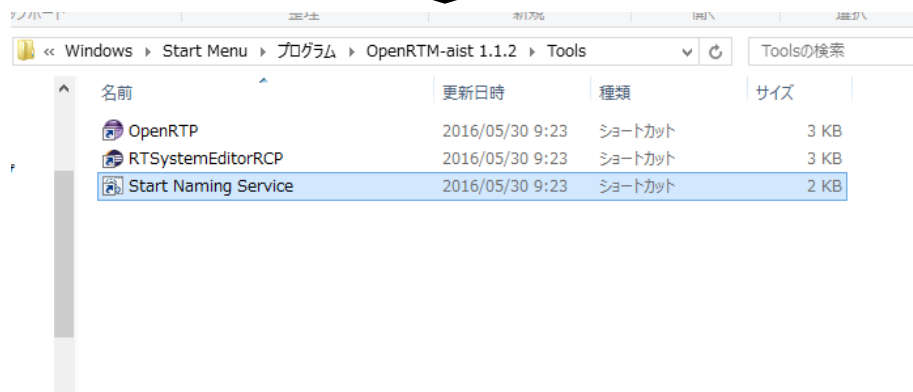


RTC Builderの起動

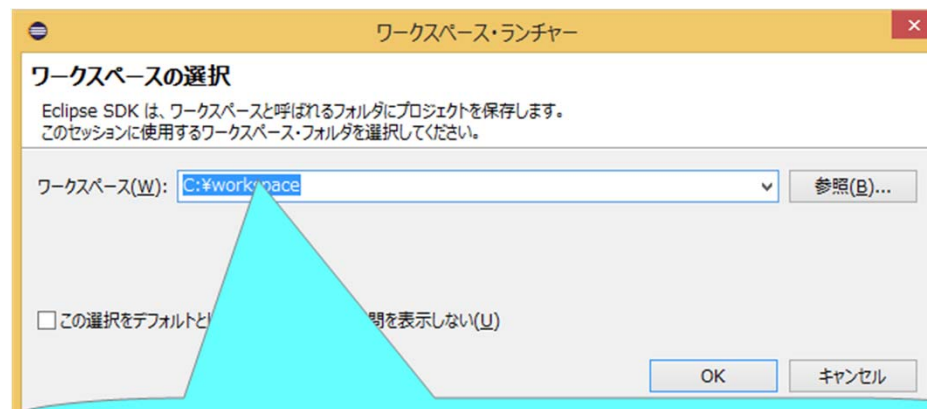
- いちいちアプリビューから起動するのは非常に手間がかかるため、以下の作業をしてスタートメニューのフォルダを開いておくことをお勧めします。



ここを右クリックして「ファイルの場所を開く」を選択



RTC Builderの起動

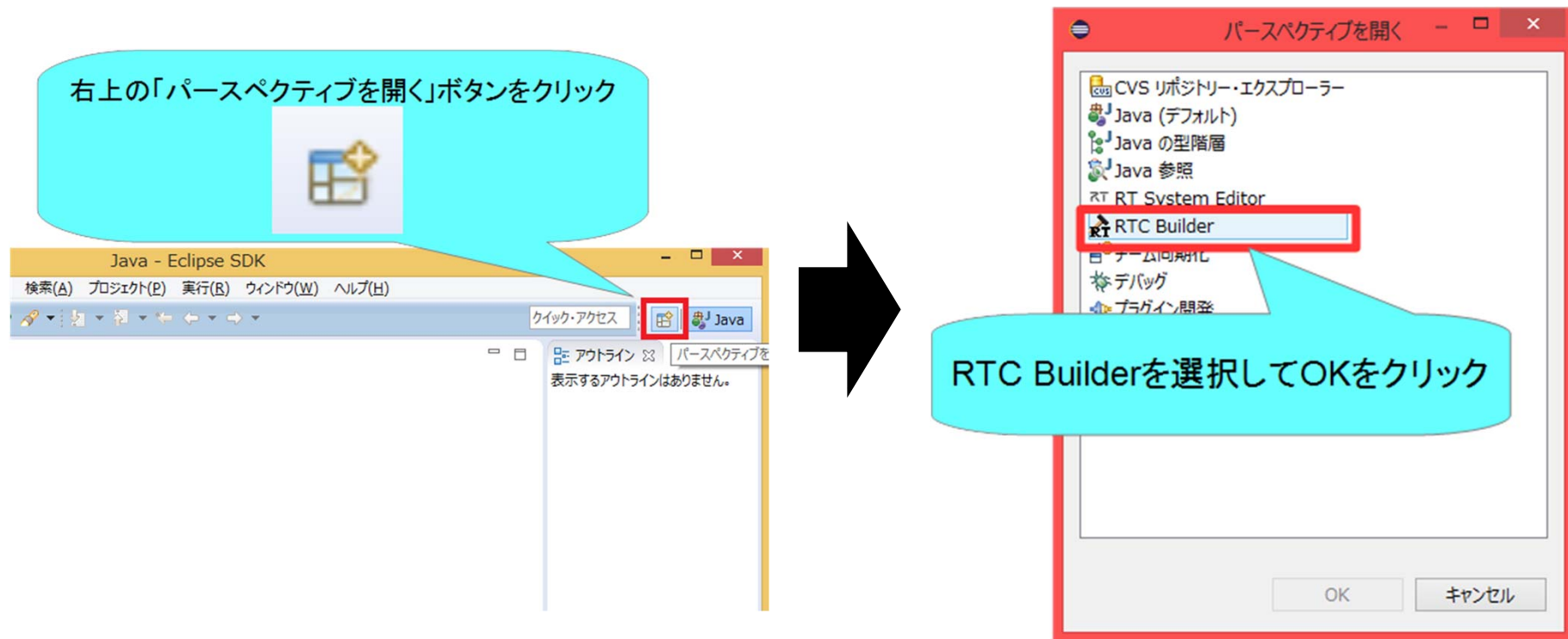


ワークスペースに適切な場所を指定してOKをクリックする
※指定したフォルダをエクスプローラで開いておくことをお勧めします

最初に起動したときはWelcomeページが開くので×を押して閉じる

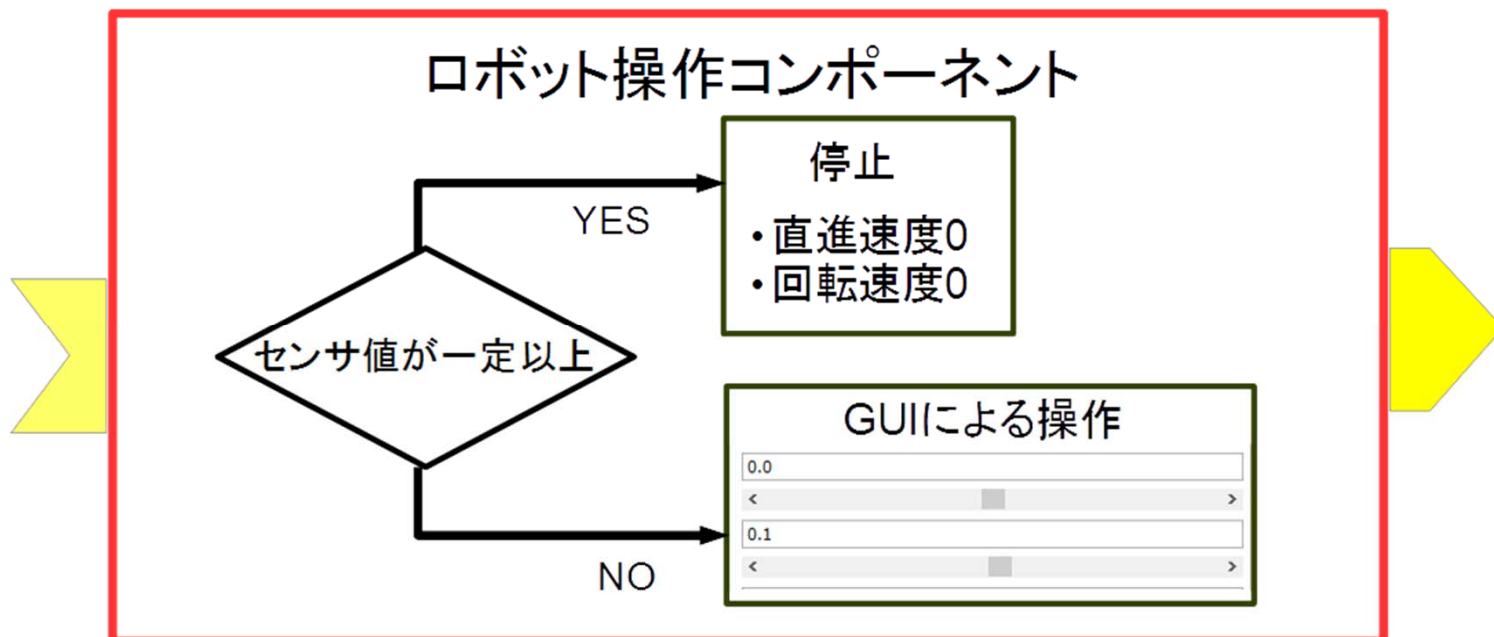


RTC Builderの起動

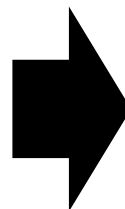
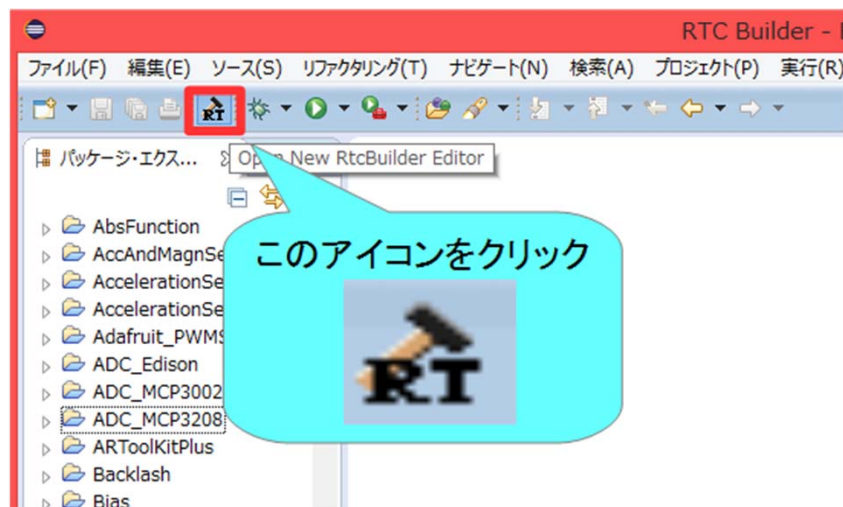


プロジェクト作成

- RobotControllerコンポーネントのひな形コードを作成する。
 - － 車輪型移動ロボット操作コンポーネント
 - GUIでロボットを操作
 - センサ値が一定以上の場合に停止



プロジェクト作成



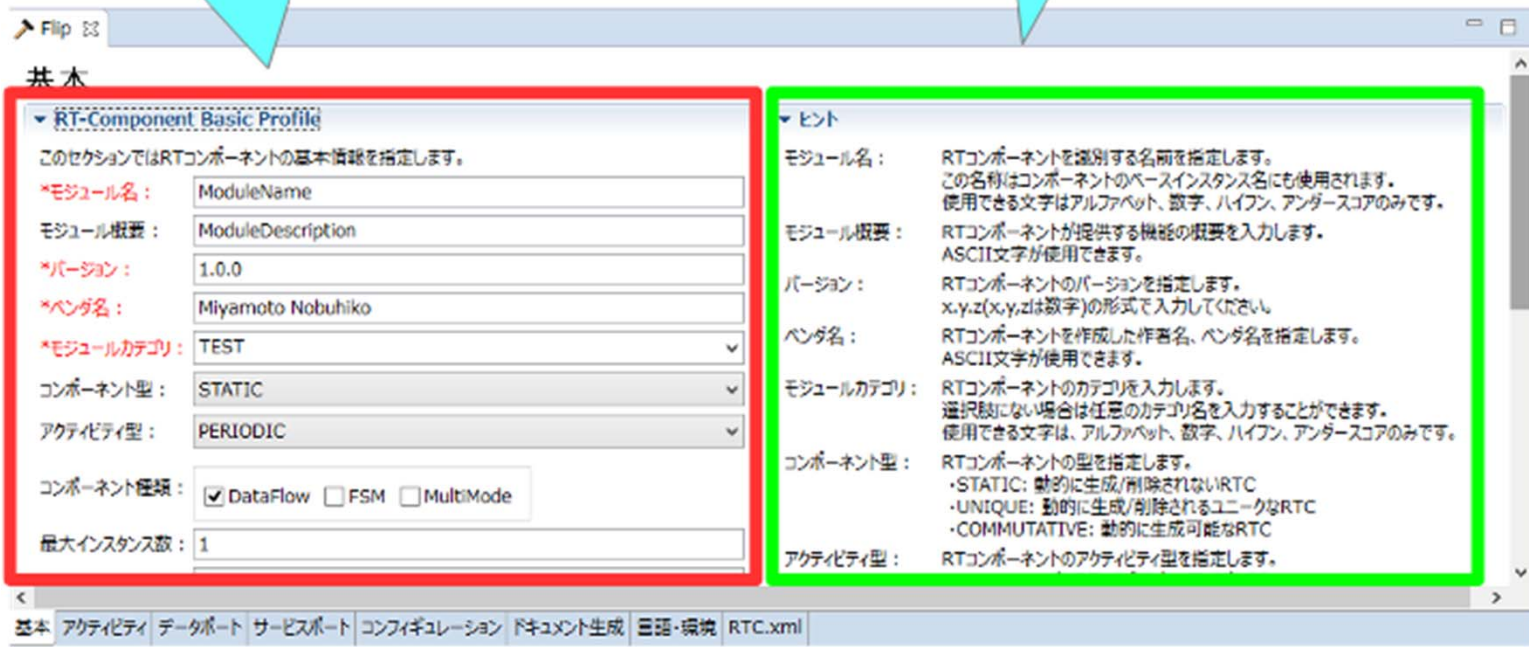
- Eclipse起動時にワークスペースに指定したディレクトリに「RobotController」というフォルダが作成される
 - この時点では「RTC.xml」と「.project」のみが生成されている
- 以下の項目が設定する
 - 基本プロファイル
 - アクティビティ・プロファイル
 - データポート・プロファイル
 - サービスポート・プロファイル
 - コンフィギュレーション
 - ドキュメント
 - 言語環境
 - RTC.xml

基本プロフィールの入力

- RTコンポーネントのプロファイル情報など、コンポーネントの基本情報を設定。
- コード生成、インポート/エクスポート、パッケージング処理を実行

RTCプロフィールエディタ
ここに各項目を入力する

ヒント



基本

▼ 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。

*モジュール名: ModuleName

モジュール概要: ModuleDescription

*バージョン: 1.0.0

*ベンダ名: Miyamoto Nobuhiko

*モジュールカテゴリ: TEST

コンポーネント型: STATIC

アクティビティ型: PERIODIC

コンポーネント種類: ☒ DataFlow ☐ FSM ☐ MultiMode

最大インスタンス数: 1

▼ ヒント

モジュール名: RTコンポーネントを識別する名前を指定します。この名称はコンポーネントのベースインスタンス名にも使用されます。使用できる文字はアルファベット、数字、ハイフン、アンダースコアのみです。

モジュール概要: RTコンポーネントが提供する機能の概要を入力します。ASCII文字が使用できます。

バージョン: RTコンポーネントのバージョンを指定します。x.y.z(x,y,zは数字)の形式で入力してください。

ベンダ名: RTコンポーネントを作成した作者名、ベンダ名を指定します。ASCII文字が使用できます。

モジュールカテゴリ: RTコンポーネントのカテゴリを入力します。選択されていない場合は任意のカテゴリ名を入力することができます。使用できる文字は、アルファベット、数字、ハイフン、アンダースコアのみです。

コンポーネント型: RTコンポーネントの型を指定します。
・STATIC: 静的に生成/削除されないRTC
・UNIQUE: 動的に生成/削除されるユニークなRTC
・COMMUTATIVE: 動的に生成可能なRTC

アクティビティ型: RTコンポーネントのアクティビティ型を指定します。

基本 アクティビティ データポート サービスポート コンフィギュレーション ドキュメント生成 言語・環境 RTC.xml

「基本」タブを選択

基本プロファイルの入力

- **モジュール名**
 - **RobotController**
- モジュール概要
 - 任意(Robot Controller Component)
- バージョン
 - 任意(1.0.0)
- ベンダ名
 - 任意
- モジュールカテゴリ
 - 任意(Controller)
- コンポーネント型
 - STATIC
- アクティビティ型
 - PERIODIC
- コンポーネントの種類
 - DataFlow
- 最大インスタンス数
 - 1
- 実行型
 - PeriodicExecutionContext
- 実行周期
 - 1000.0
- 概要
 - 任意

基本

▼ 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。

*モジュール名 :

RobotController

モジュール概要 :

Robot Controller Component

*バージョン :

1.0.0

*ベンダ名 :

AIST

*モジュールカテゴリ :

Controller ▼

コンポーネント型 :

STATIC ▼

アクティビティ型 :

PERIODIC ▼

コンポーネント種類 :

☒ DataFlow
 ☐ FSM
 ☐ MultiMode

最大インスタンス数 :

1

実行型 :

PeriodicExecutionContext ▼

実行周期 :

1000.0

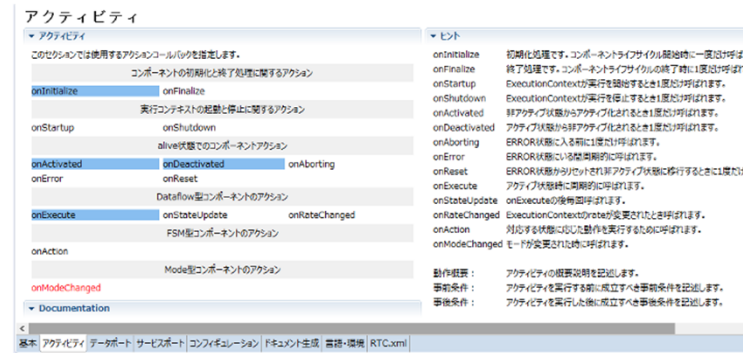
概要 :

講習会用Raspberry Piマウス操作コンポーネント

RTC Type :

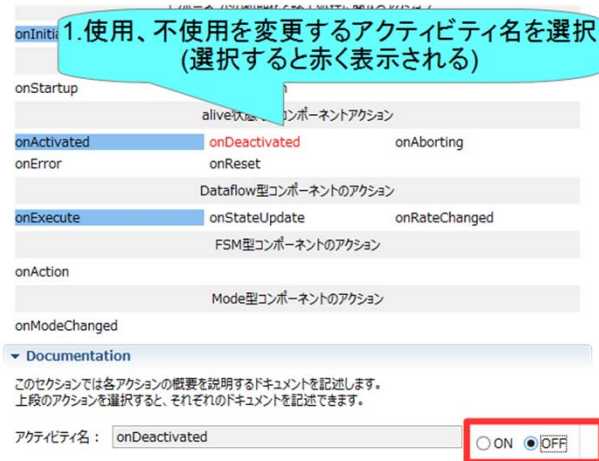
アクティビティの設定

- 使用するアクティビティを設定する

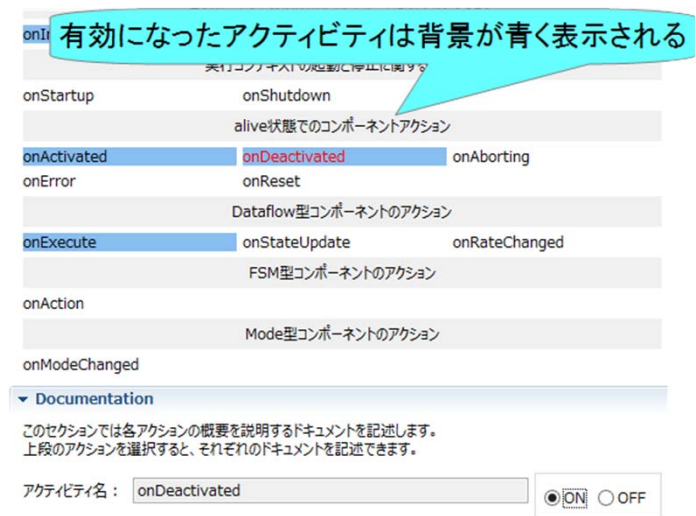


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順



2. アクティビティ名の選択後、ON・OFFを選択する



アクティビティの設定

コールバック関数	処理
onInitialize	初期化处理
onActivated	アクティブ化されるとき1度だけ呼ばれる
onExecute	アクティブ状態時に周期的に呼ばれる
onDeactivated	非アクティブ化されるとき1度だけ呼ばれる
onAborting	ERROR状態に入る前に1度だけ呼ばれる
onReset	resetされる時に1度だけ呼ばれる
onError	ERROR状態のときに周期的に呼ばれる
onFinalize	終了時に1度だけ呼ばれる
onStateUpdate	onExecuteの後毎回呼ばれる
onRateChanged	ExecutionContextのrateが変更されたとき1度だけ呼ばれる
onStartup	ExecutionContextが実行を開始するとき1度だけ呼ばれる
onShutdown	ExecutionContextが実行を停止するとき1度だけ呼ばれる

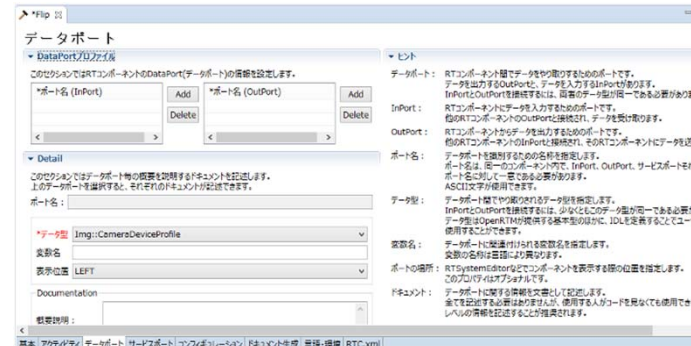
アクティビティの設定

- 以下のアクティビティを有効にする
 - onInitialize
 - **onActivated**
 - **onDeactivated**
 - **onExecute**
- Documentationは適当に書いておいてください
 - 空白でも大丈夫です



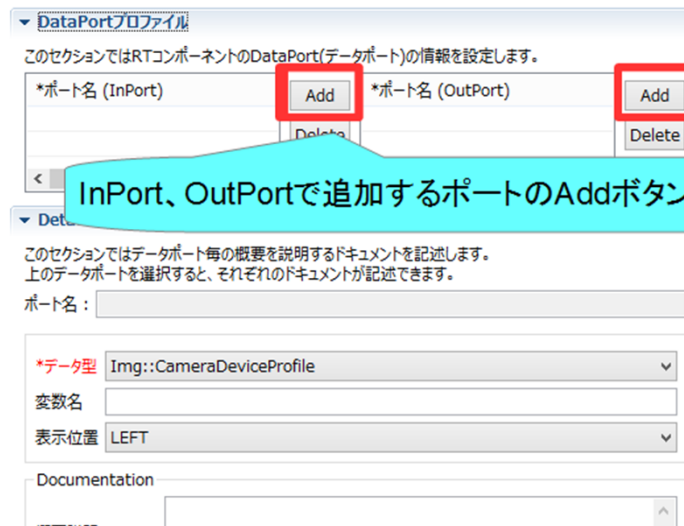
データポートの設定

- InPort、OutPortの追加、設定を行う

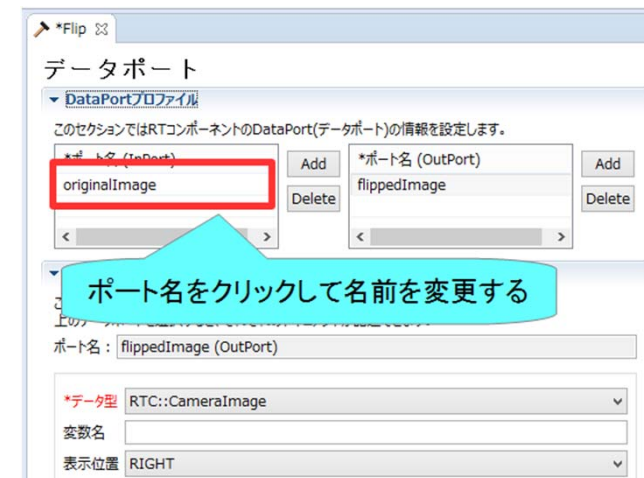


「データポート」タブを選択

- データポートを追加する手順



InPort、OutPortで追加するポートのAddボタンをクリック



ポート名をクリックして名前を変更する

各項目を設定する

データポートの設定

- 以下のOutPortを設定する
 - **out**
 - データ型 : **RTC::TimedVelocity2D**
 - 他の項目は任意
- 以下のInPortを設定する
 - **in**
 - データ型 : **RTC::TimedShortSeq**
 - 他の項目は任意

▼ DataPortプロフィール

このセクションではRTコンポーネントのDataPort(データポート)の情報を設定します。

*ポート名 (InPort)	Add	*ポート名 (OutPort)	Add
in	Delete	out	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。

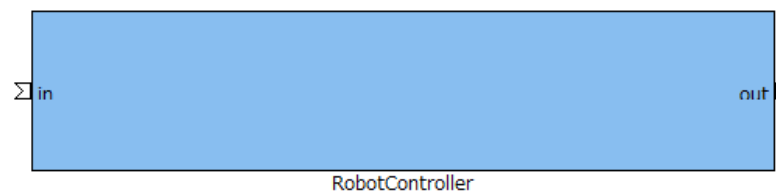
ポート名 : out (OutPort)

*データ型: RTC::TimedVelocity2D

変数名:

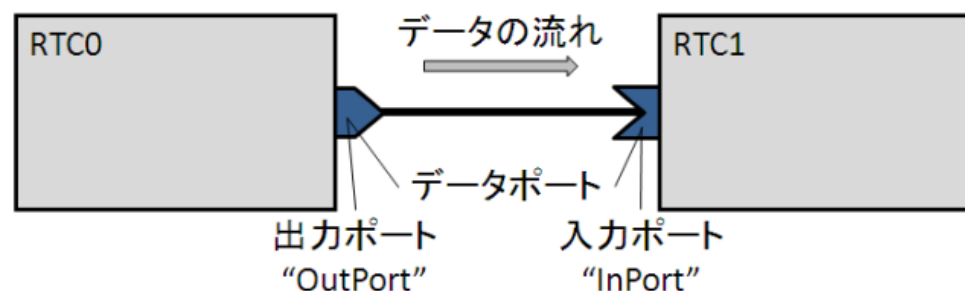
表示位置: RIGHT

Documentation: 目標速度出力

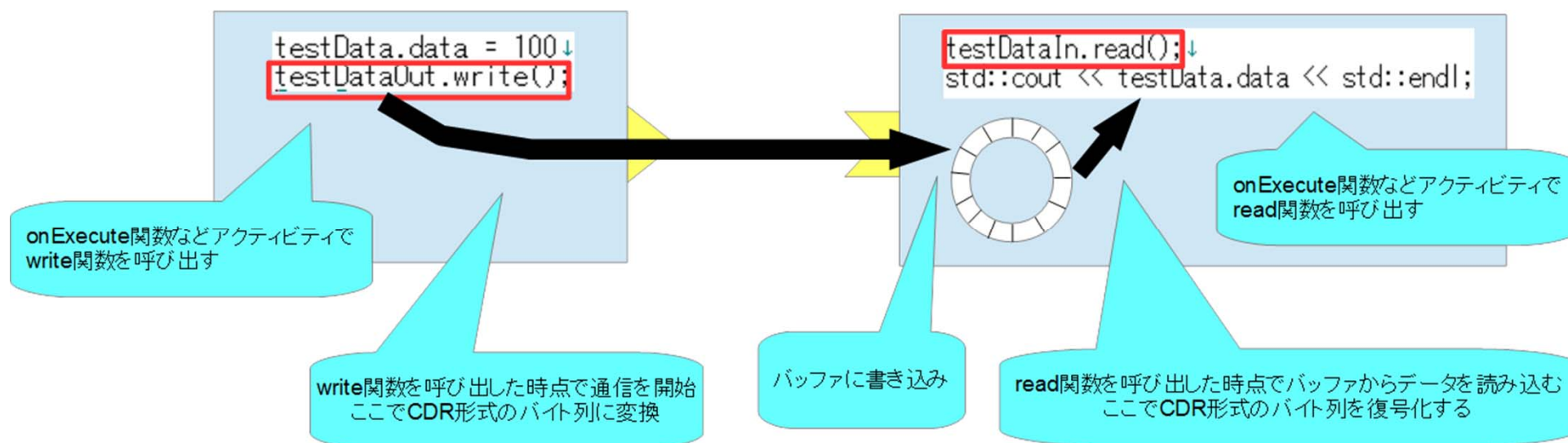


データポートについて

- 連続したデータを通信するためのポート

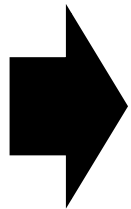


- 以下の例はデータフロー型がpush、サブスクリプション型がflush、インターフェース型がcorba_cdrの場合



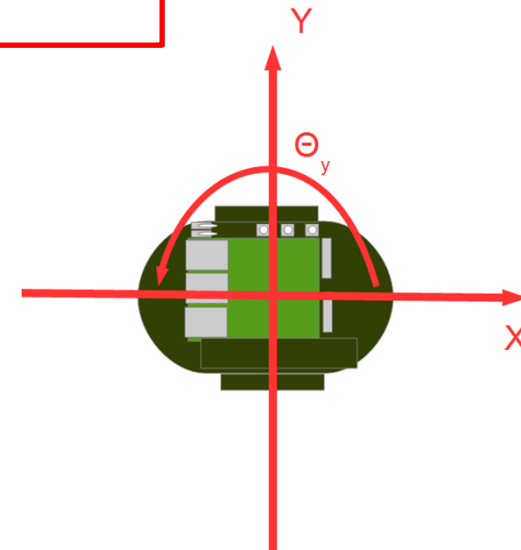
RTC::TimedVelocity2D型について

- ExtendedDataTypes.idlで定義されている**移動ロボットの速度**を表現するためのデータ型
 - vx**: X軸方向の速度
 - vy**: Y軸方向の速度(車輪が横滑りしないと仮定すると0)
 - va**: Z軸周りの角速度



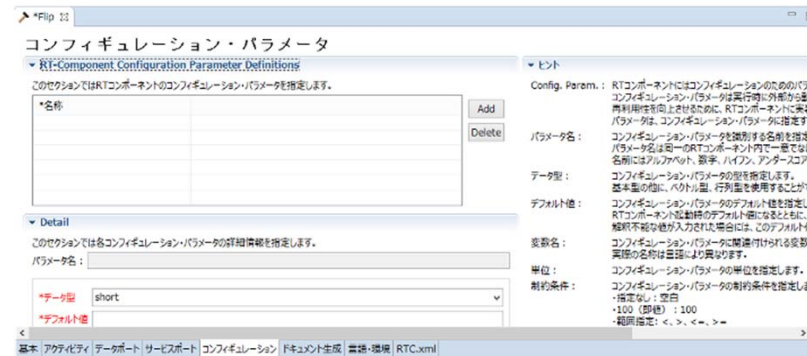
vxで直進速度、**va**で回転速度を設定

```
struct Velocity2D↓
{↓
  /// Velocity along the x axis in metres per second.↓
  double vx;↓
  /// Velocity along the y axis in metres per second.↓
  double vy;↓
  /// Yaw velocity in radians per second.↓
  double va;↓
};↓
```

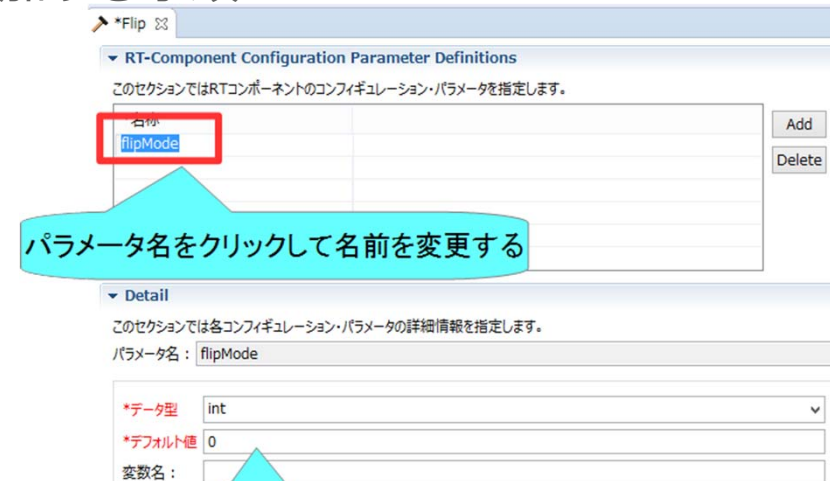


コンフィギュレーションの設定

- コンフィギュレーションパラメータの追加、設定を行う



- コンフィギュレーションパラメータを追加する手順



コンフィギュレーションの設定

- 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_x

- データ型 : double
- デフォルト値 : 0.0
- 制約条件 : $-1.5 < x < 1.5$
- Widget : slider
- Step: 0.01
- 他の項目は任意

– speed_r

- データ型 : double
- デフォルト値 : 0.0
- 制約条件 : $-2.0 < x < 2.0$
- Widget : slider
- Step: 0.01
- 他の項目は任意

RT-Component Configuration Parameter Definitions

このセクションではRTコンポーネントのコンフィギュレーション・パラメータを指定します。

*名称	
speed_x	
speed_r	
stop_d	

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 : speed_x

*データ型 : double

*デフォルト値 : 0.0

変数名 :

単位 : m/s

制約条件 : $-1.5 < x < 1.5$

Widget : slider

Step : 0.01

GUI(スライダー)による移動ロボットの操作ができるように

0.0

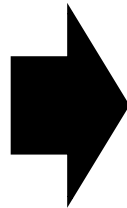
< >

0.1

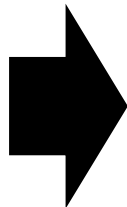
< >

- RT System Editorでコンフィギュレーションパラメータを編集する際にGUIを表示する

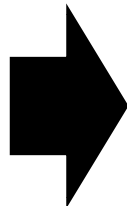
- Widget : text



- 制約条件 :
 $0 \leq x \leq 100$
- Widget : spin
- Step : 10



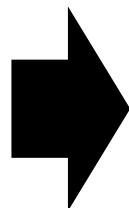
- 制約条件 :
 $0 \leq x \leq 100$
- Widget : slider
- Step : 10



<
>

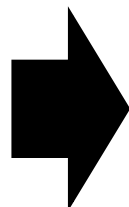
コンフィギュレーションパラメータ の制約、Widgetの設定

- 制約条件 : (0,1,2,3)
- Widget : radio



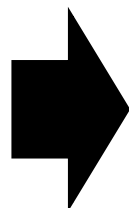
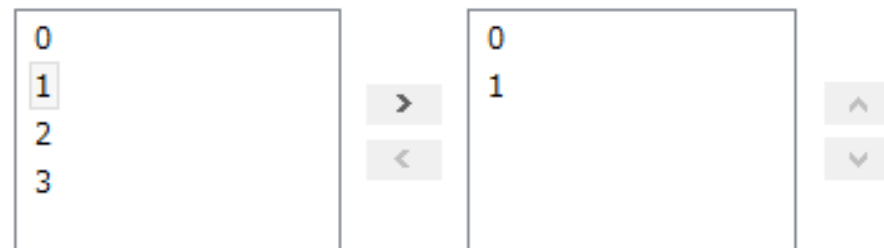

○ 0 ○ 1 ● 2
○ 3

- 制約条件 : (0,1,2,3)
- Widget : checkbox




☑ 0 ☐ 1 ☑ 2
☐ 3

- 制約条件 : (0,1,2,3)
- Widget : ordered_list

0
1
2
3

> <

0
1

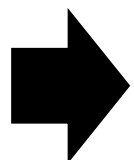
^ v

コンフィギュレーションの設定

- 以下のコンフィギュレーションパラメータを追加

– stop_d

- データ型 : int
- デフォルト値 : 30
- 他の項目は任意



センサ値がこの値以上の場合に停止

▼ RT-Component Configuration Parameter Definitions

このセクションではRTコンポーネントのコンフィギュレーション・パラメータを指定します。

*名称		Add Delete
speed_x		
speed_r		
stop_d		

▼ 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 : stop_d

*データ型

double

*デフォルト値

30

変数名 :

単位 :

制約条件 :

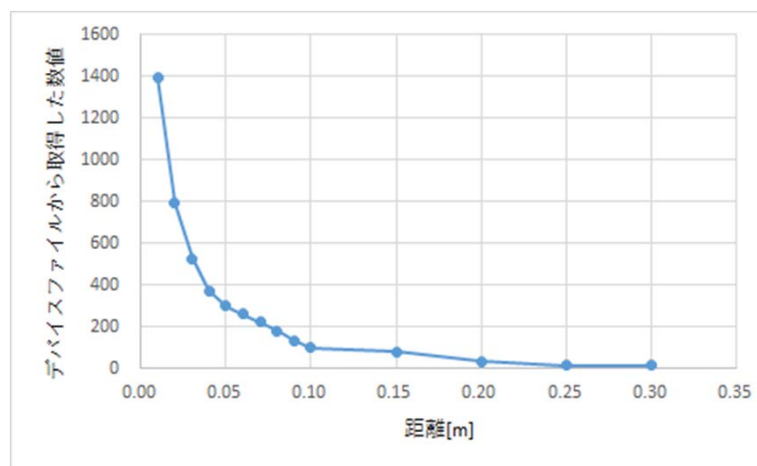
Widget:

text

Step:

Raspberry Piマウスの距離センサ

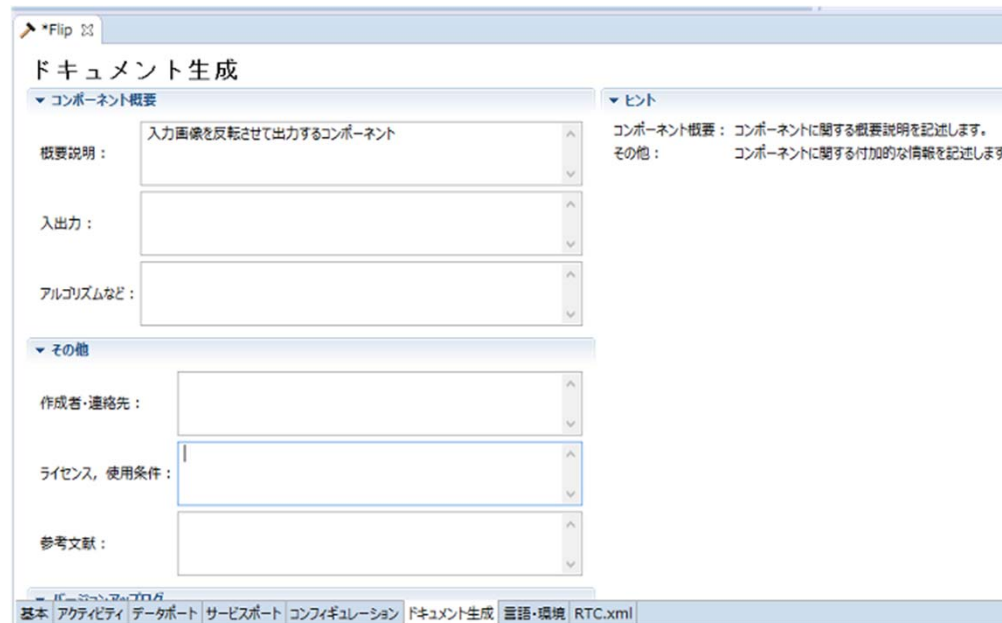
- Raspberry Piマウス実機には距離センサが搭載されている
 - 計測した値は物体までの**距離が近いほど大きな値**となる



- シミュレータでもこのデータに近い値を計算して出力している

ドキュメントの設定

- 各種ドキュメント情報を設定

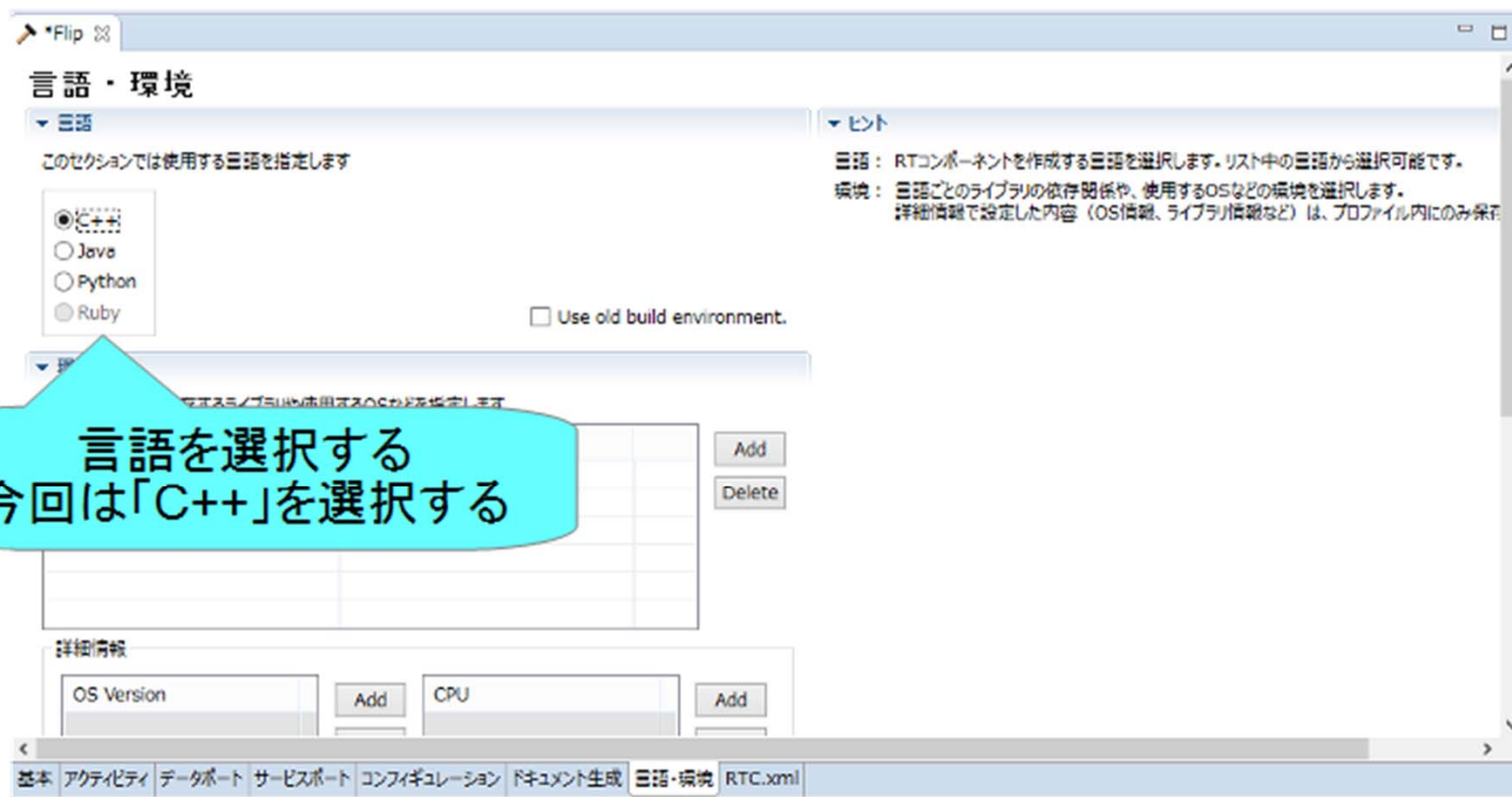


「ドキュメント生成」タブを選択

- 今回は適当に設定しておいてください。
 - 空白でも大丈夫です

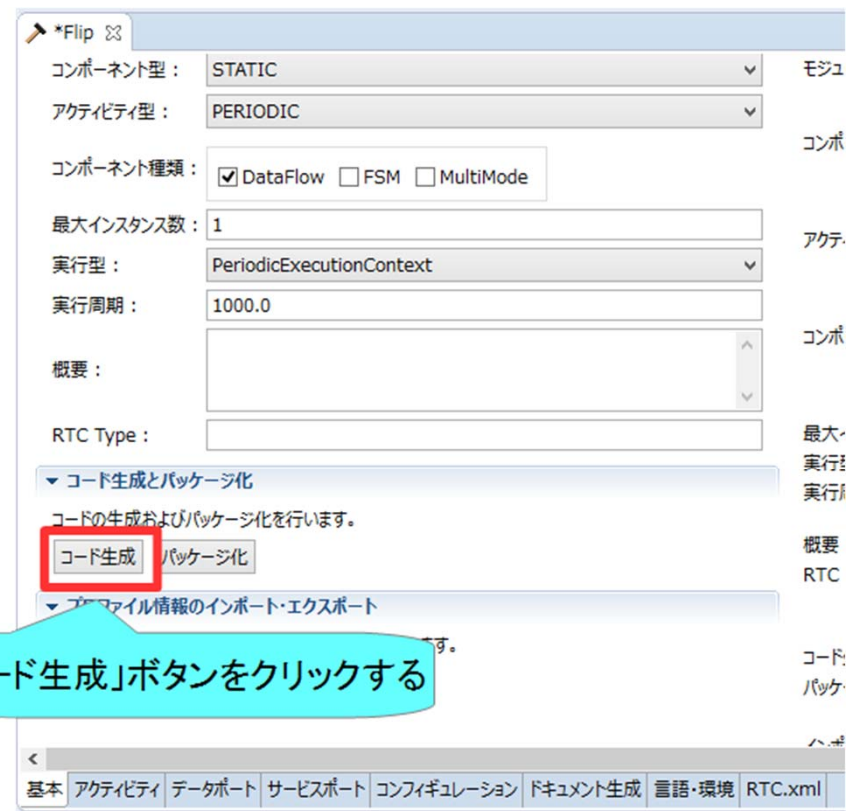
言語の設定

- 実装する言語，動作環境に関する情報を設定



ひな形コードの生成

- 基本タブからコード生成ボタンを押すことでひな形コードが生成される
 - Workspace¥RobotController以下に生成
 - ソースコード
 - C++ソースファイル(.cpp)
 - ヘッダーファイル(.h)
 - » このソースコードにロボットを操作する処理を記述する
 - CMakeの設定ファイル
 - CMakeLists.txt
 - rtc.conf、RobotController.conf
 - 以下略
 - ファイルが生成できているかを確認してください



ソースコードの編集、RTCのビルド

手順

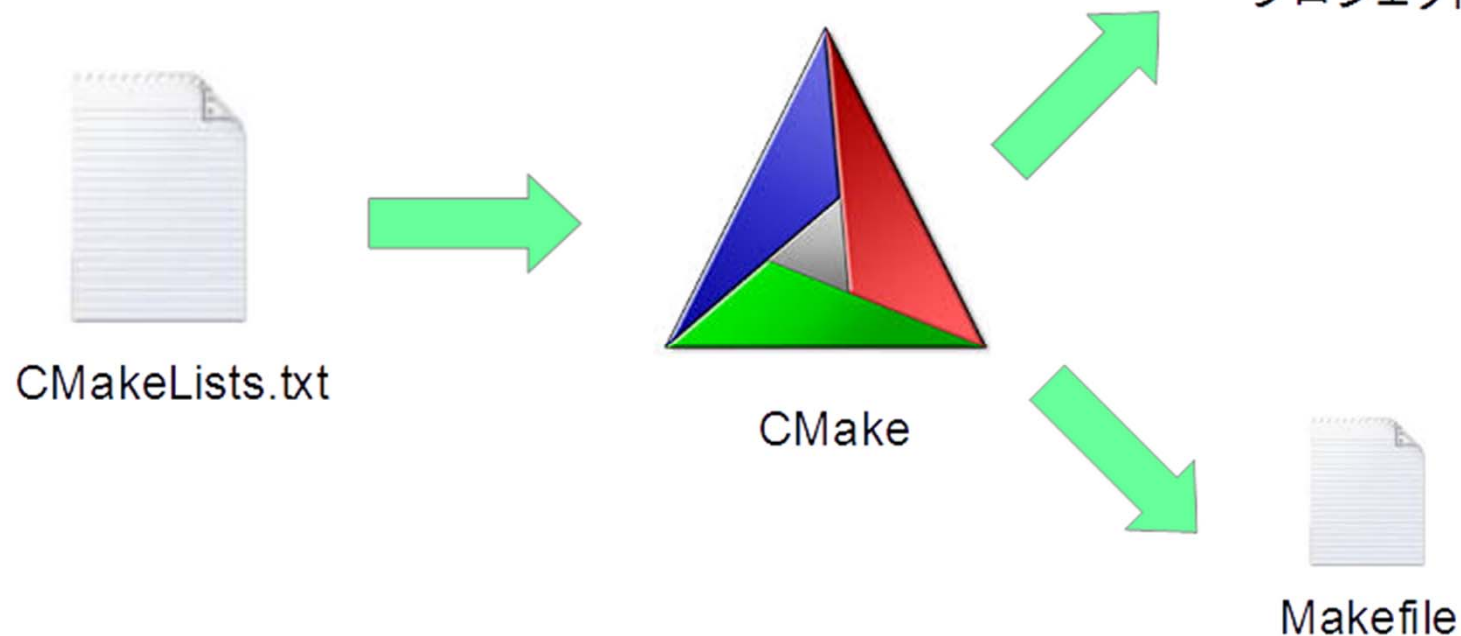
- ビルドに必要な各種ファイルを生成
 - CMakeLists.txtの編集
 - CMakeにより各種ファイル生成
- ソースコードの編集
 - RobotController.hの編集
 - RobotController.cppの編集
- ビルド
 - Windows : Visual Studio
 - Ubuntu : Code::Blocks

CMake

- ビルドに必要な各種ファイルを生成
 - CMakeLists.txtに設定を記述
 - RTC Builderでスケルトンコードを作成した時にCMakeLists.txtも生成され

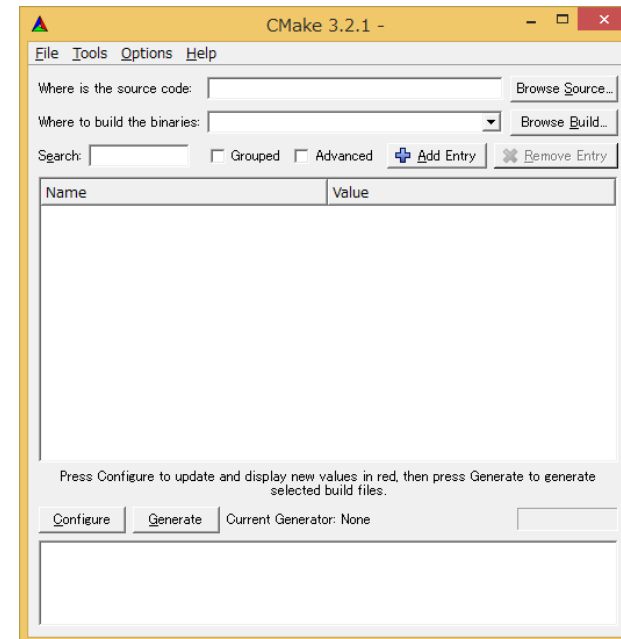


Visual Studio
(ソリューションファイル、
プロジェクトファイル等)



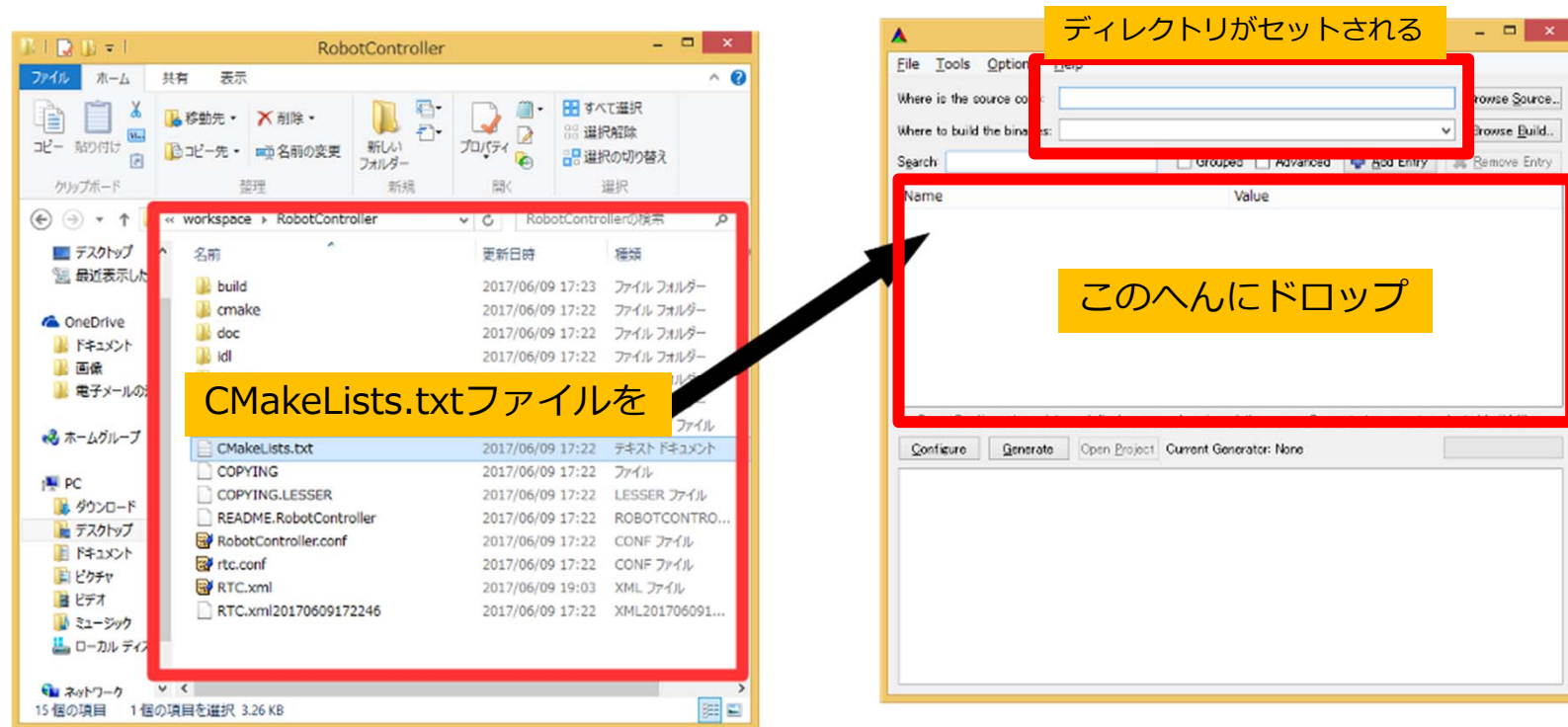
ビルドに必要なファイルの生成

- CMakeを使用する
- Windows10
 - 検索窓で cmakeと検索
 - CMake (cmake-gui) をクリック
 - Ubuntu
 - コマンドで「cmake-gui」を入力



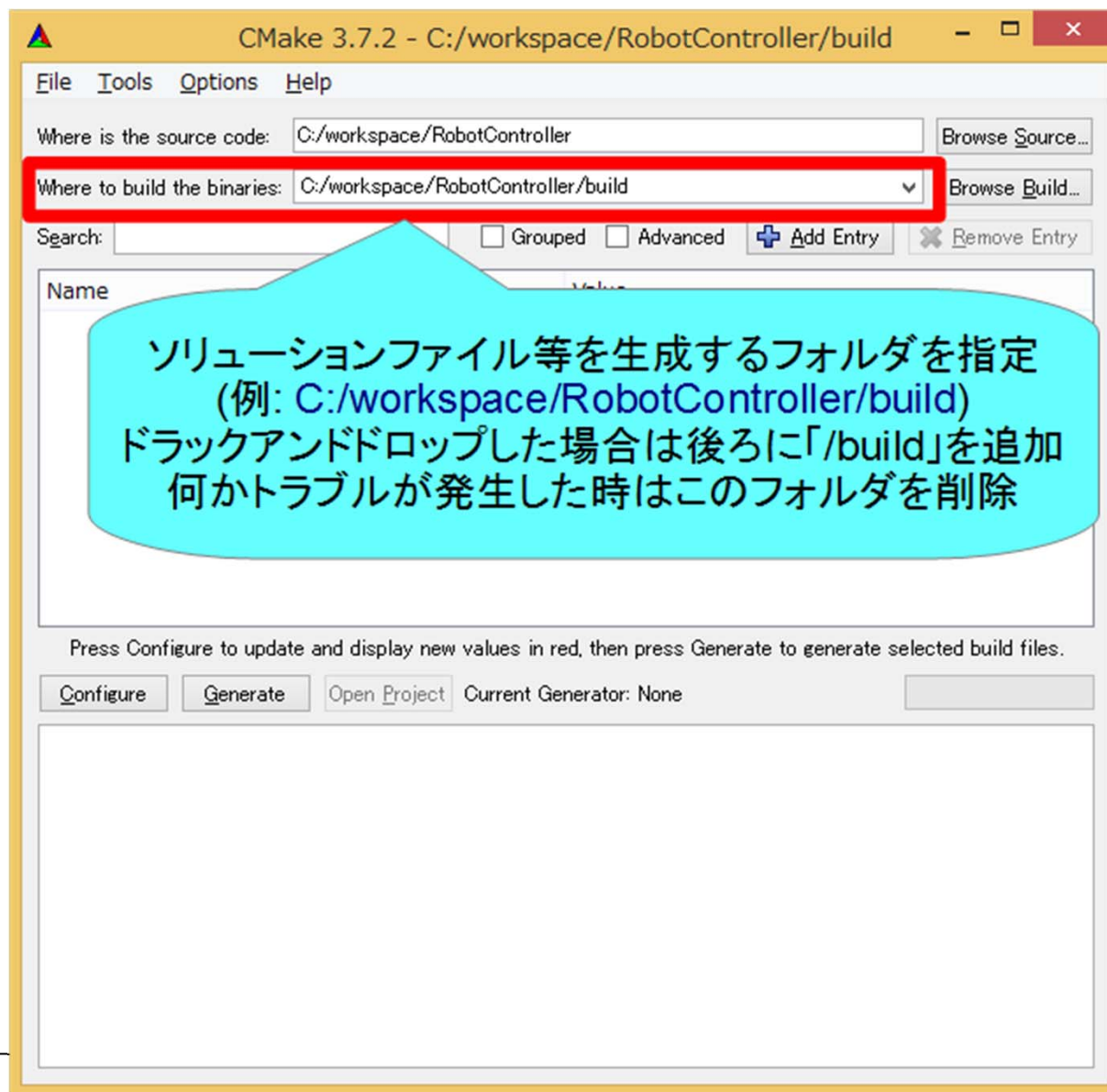
ビルドに必要なファイルの生成

- **CMakeLists.txt**をcmake-guiにドラックアンドドロップ
 - CMakeLists.txtはRTC Builderで生成したプロジェクトのフォルダ
(例： C:¥workspace¥RobotController)

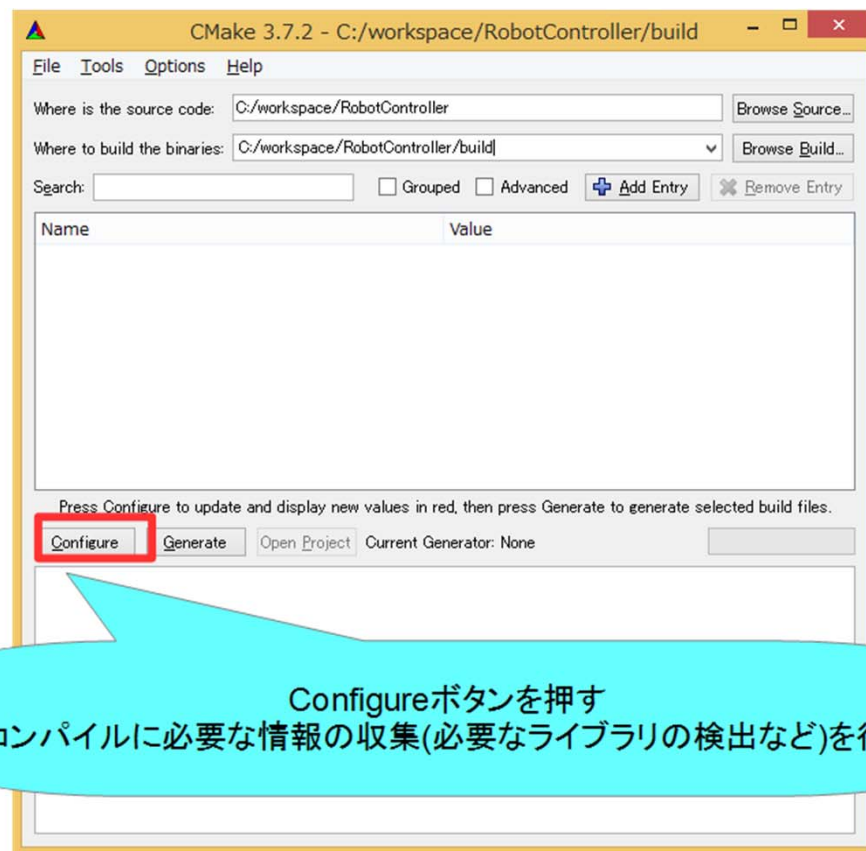


“Where is the source code” および “Where to build the binaries” の窓
に CMakeLists.txtのディレクトリパスがセットされる

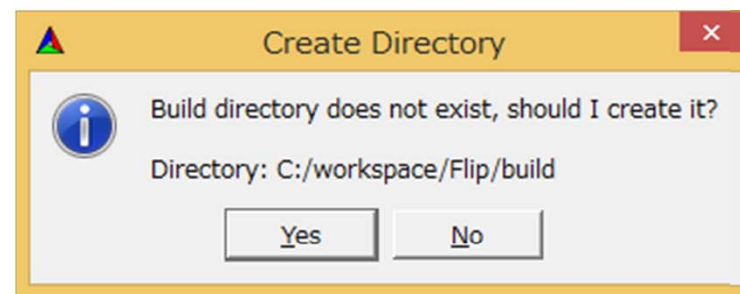
ビルドに必要なファイルの生成



ビルドに必要なファイルの生成



Configureボタンを押す
コンパイルに必要な情報の収集(必要なライブラリの検出など)を行う

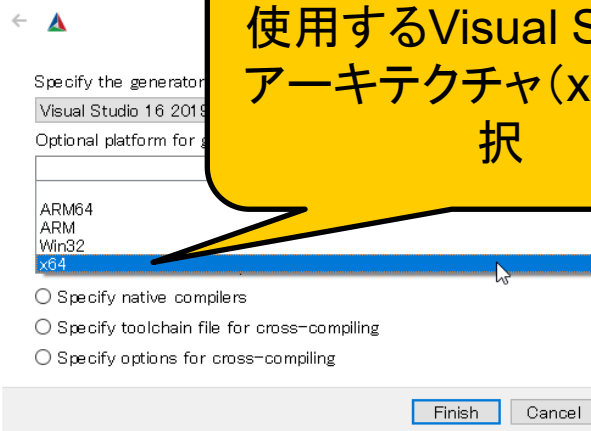


buildフォルダが存在しない場合は
作成するかどうかきかれるため「Yes」を選択

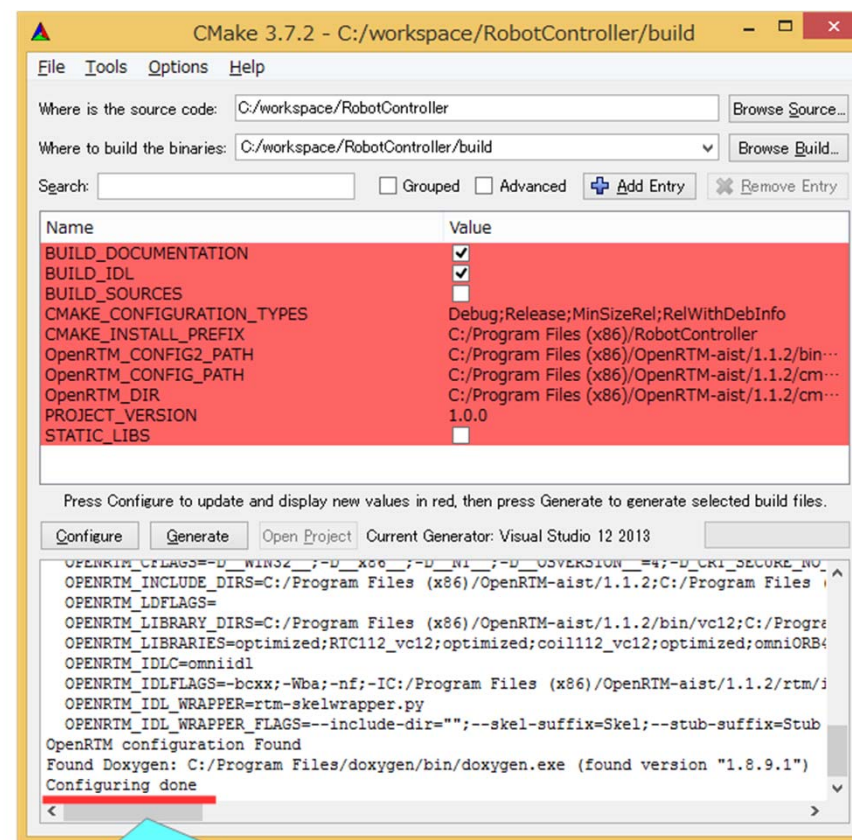
ビルドに必要なファイルの生成



使用するVisual Studioのバージョン(2019)を選択

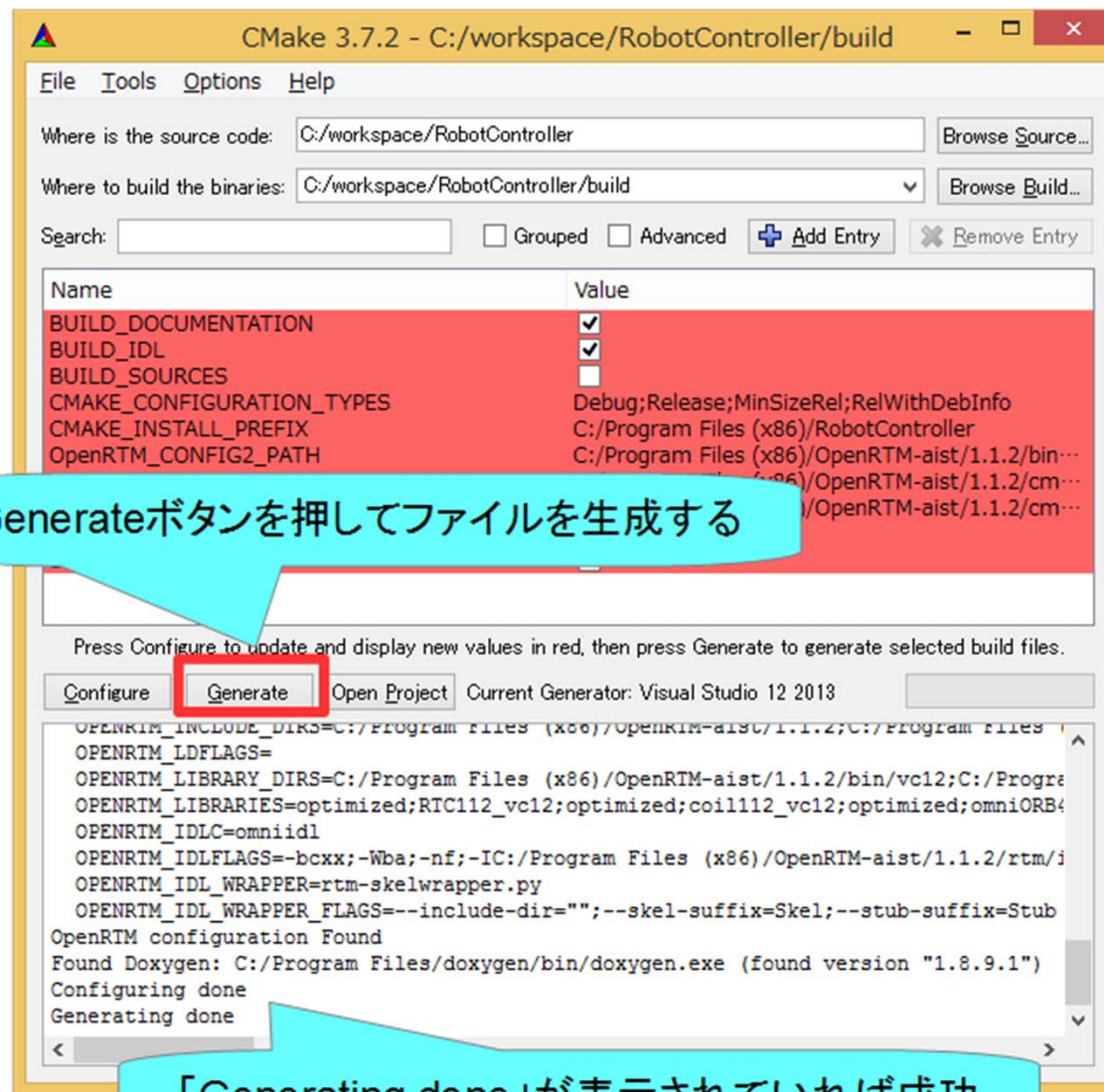


使用するVisual Studioのアーキテクチャ(x64)を選択



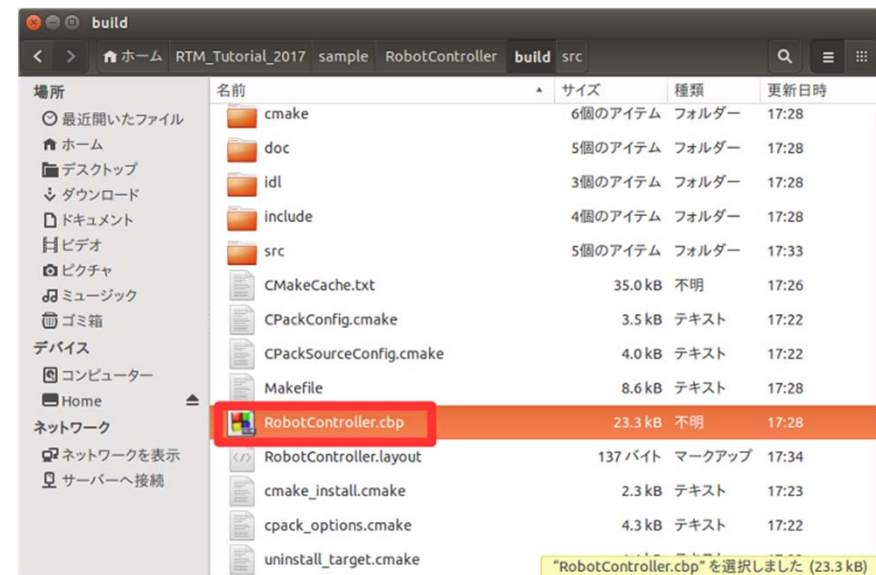
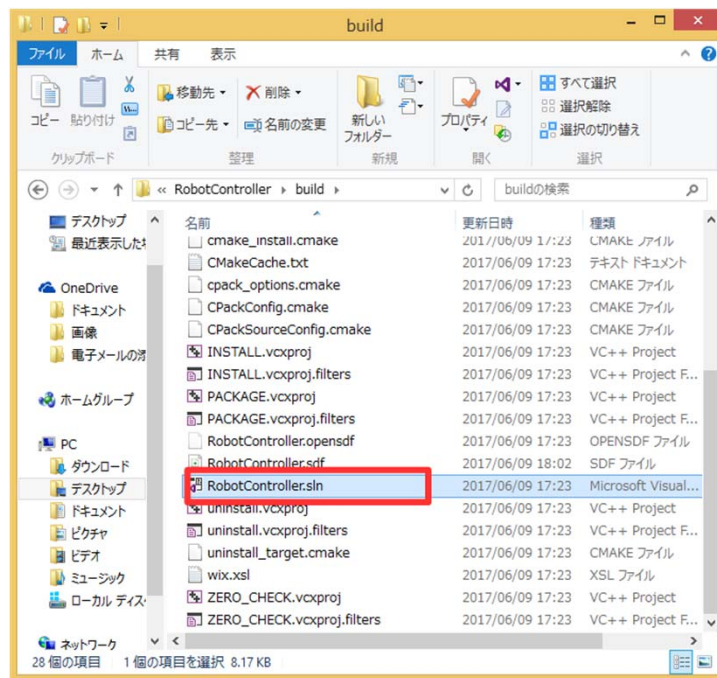
「Configure done」が表示されていれば成功

ビルドに必要なファイルの生成



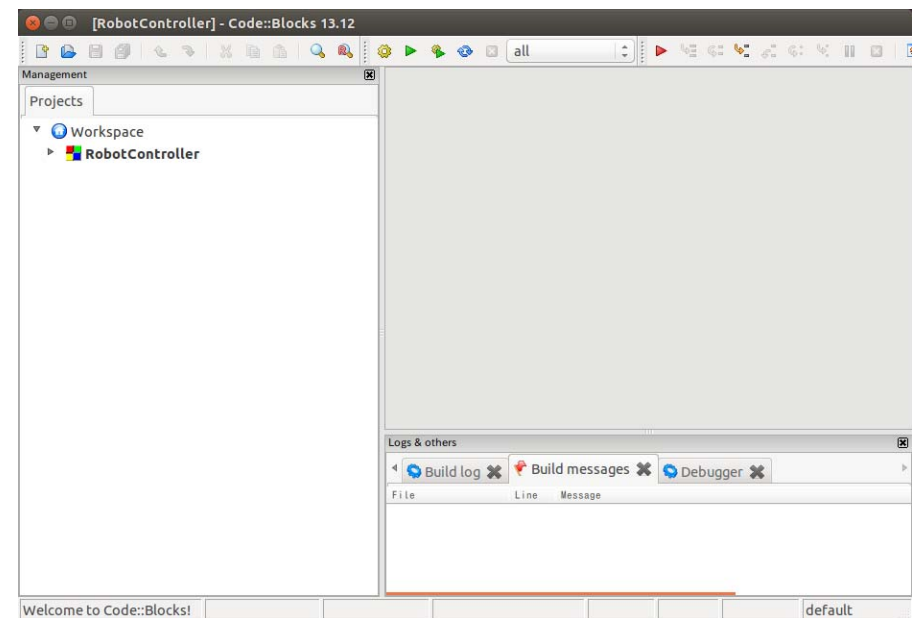
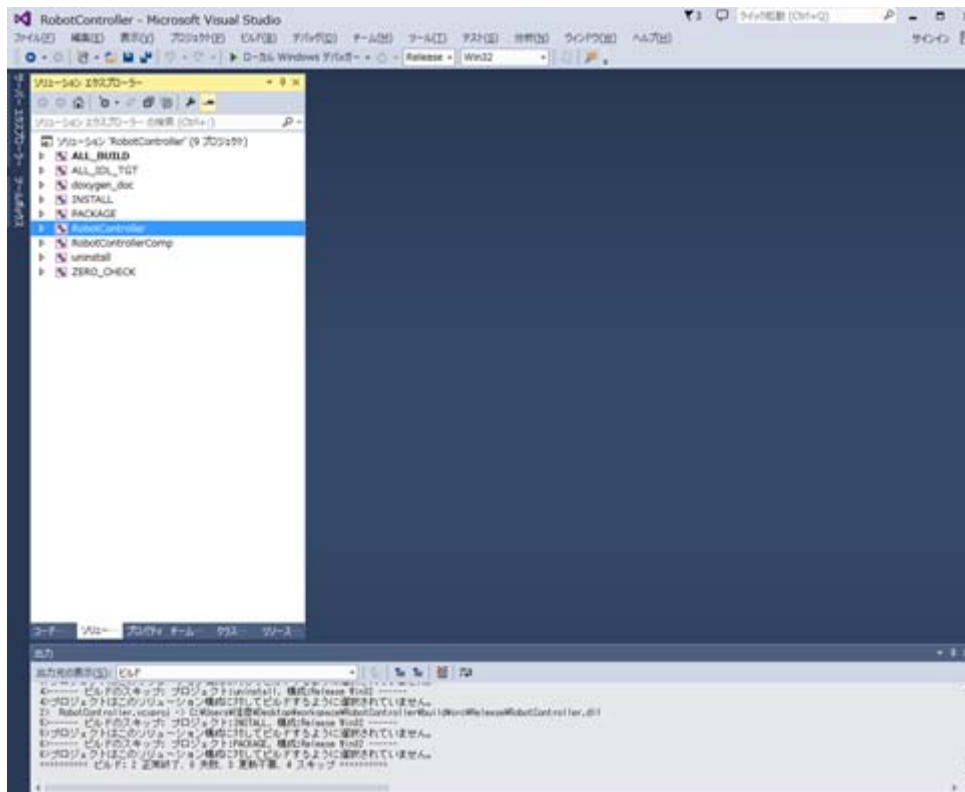
ソースコードの編集

- Windows
 - buildフォルダの「RobotController.sln」をダブルクリックして開く
- Ubuntu
 - buildフォルダの「RobotController.cbp」をダブルクリックして開く



ソースコードの編集

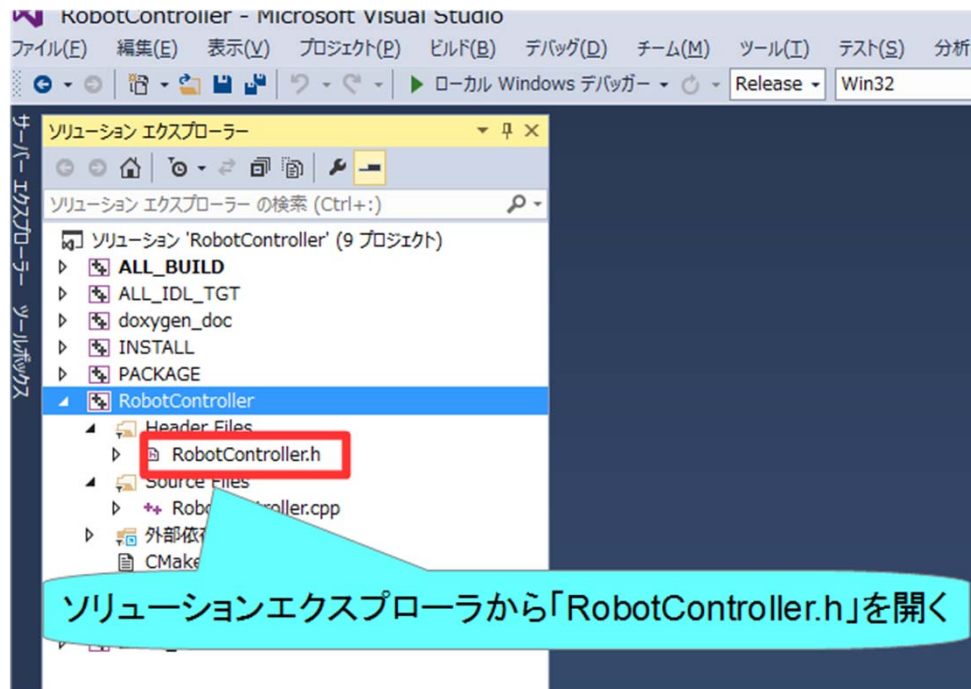
- Windows
 - Visual Studioが起動
- Ubuntu
 - Code::Blocksが起動



ソースコードの編集

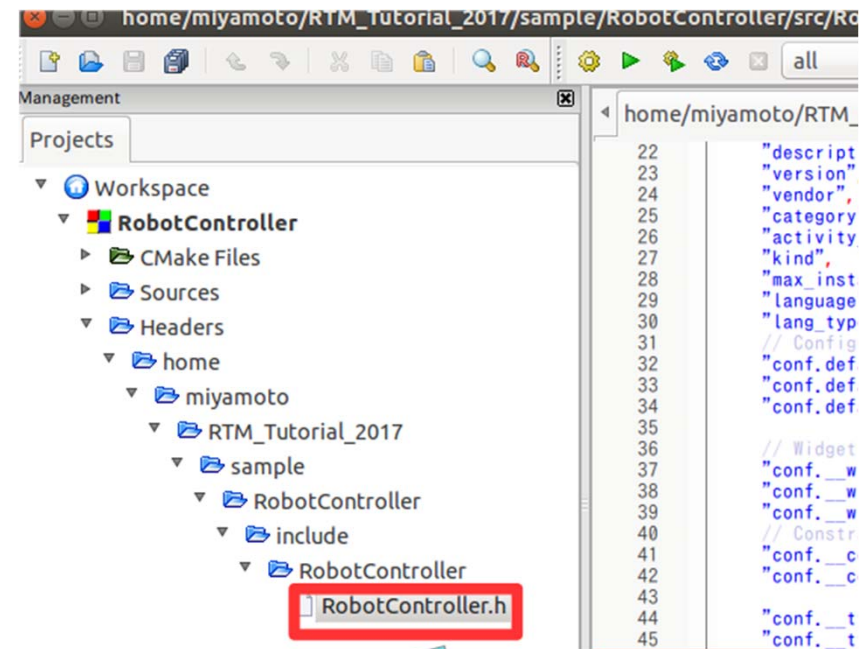
- RobotController.hの編集

Visual Studio



ソリューションエクスプローラから「RobotController.h」を開く

Code::Blocks



ProjectsからRobotController.hを開く

ソースコードの編集

- RobotController.hの編集

センサ値を一時格納する変数の宣言
int sensor_data[4];

```

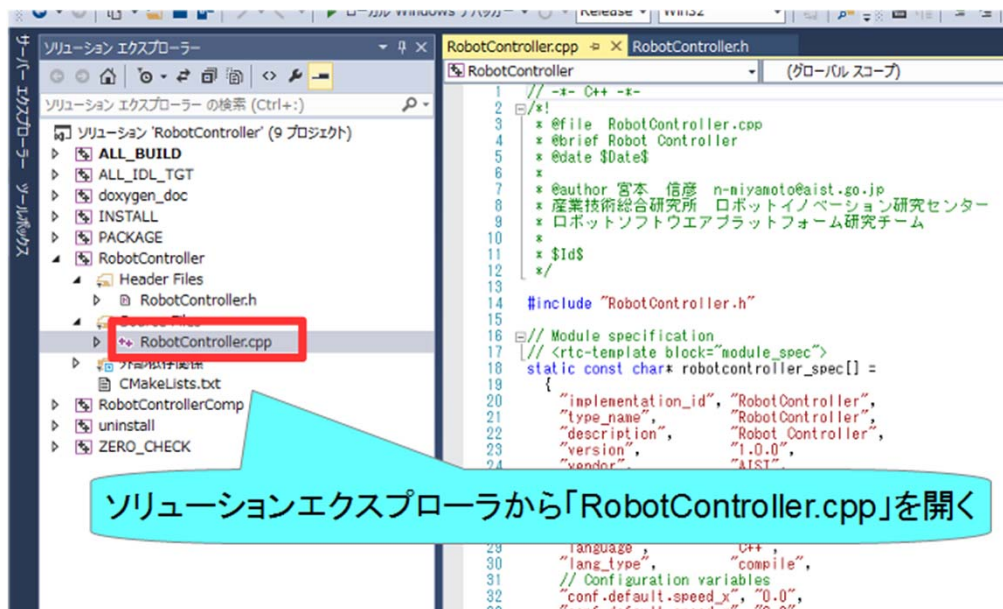
297
298 // <rtc-template block="private">
299
300 // </rtc-template>
301
302 private:
303     int sensor_data[4]; //センサ値を一時格納する変数
304
305 // <rtc-template block="private_attribute">
306
307 // </rtc-template>
308
309 // <rtc-template block="private_operation">
310
311 // </rtc-template>
312
313 };
314
315
316 extern "C"
317 {

```

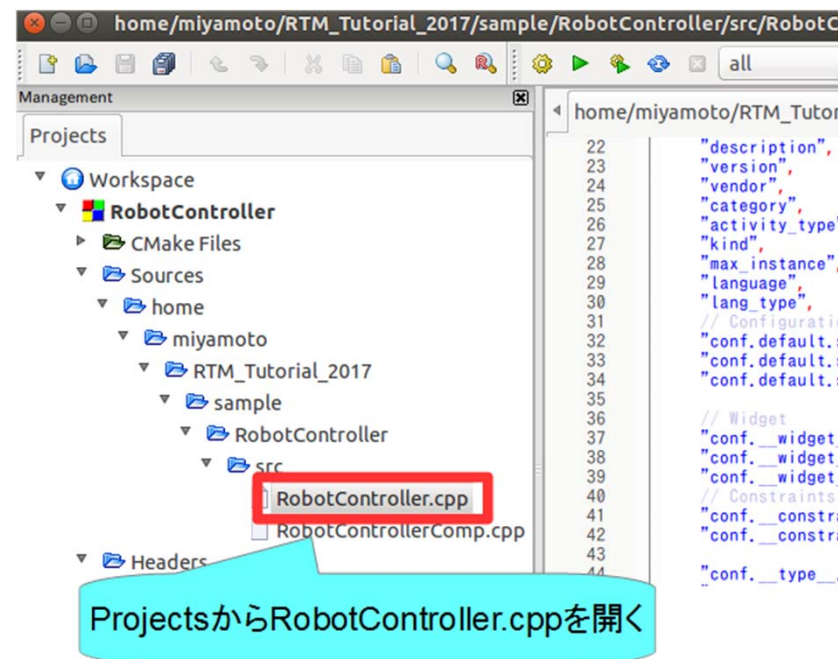
ソースコードの編集

- RobotController.cppの編集
 - 詳細はUSBメモリの資料を参考にしてください

Visual Studio



Code::Blocks



ソースコードの編集

- RobotController.cppの編集

```

125  RTC::ReturnCode_t RobotController::onActivated(RTC::UniqueId ec_id)
126  {
127
128      //センサ値初期化
129      for (int i = 0; i < 4; i++)
130      {
131          sensor_data[i] = 0;
132      }
133
134      return RTC::RTC_OK;
135  }

```

onActivateに追加

```

138  RTC::ReturnCode_t RobotController::onDeactivated(RTC::UniqueId ec_id)
139  {
140
141      //ロボットを停止する
142      m_out.data.vx = 0;
143      m_out.data.va = 0;
144      m_outOut.write();
145      return RTC::RTC_OK;
146  }

```

onDeactivateに追加

ソースコードの編集

- RobotController.cppの編集

```

157 RTC::ReturnCode_t RobotController::onExecute(RTC::UniqueId ec_id)
158 {
159     //入力データの存在確認
160     if (m_inIn.isNew())
161     {
162         //入力データ読み込み
163         m_inIn.read();
164         //この時点で入力データがm_inに格納される
165         for (int i = 0; i < m_in.data.length(); i++)
166         {
167             //入力データを別変数に格納
168             if (i < 4)
169             {
170                 sensor_data[i] = m_in.data[i];
171             }
172         }
173     }
174
175     //前進するときのみ停止するかを判定
176     if (m_speed_x > 0)
177     {
178         for (int i = 0; i < 4; i++)
179         {
180             //センサ値が設定値以上か判定
181             if (sensor_data[i] > m_stop_d)
182             {
183                 //センサ値が設定値以上の場合は停止
184                 m_out.data.vx = 0;
185                 m_out.data.va = 0;
186                 m_outOut.write();
187                 return RTC::RTC_OK;
188             }
189         }
190     }
191     //設定値以上の値のセンサが無い場合はコンフィギュレーションパラメータの値で操作
192     m_out.data.vx = m_speed_x;
193     m_out.data.va = m_speed_r;
194     m_outOut.write();
195     return RTC::RTC_OK;
196 }

```

onExecuteに追加

ソースコードの編集

- データを読み込む手順

isNew関数で新規に書き込まれたデータが存在するかを確認

//入力データ存在確認

```
if (m_inIn.isNew())
```

```
{
```

//入力データ読み込み

```
m_inIn.read();
```

//この時点で入力データがm_inに格納される

```
for (int i = 0; i < m_in.data.length(); i++)
```

```
{
```

//入力データを別変数に格納

```
if (i < 4)
```

```
{
```

```
sensor_data[i] = m_in.data[i];
```

```
}
```

```
}
```

```
}
```

read関数でデータの読み込み

read関数を呼び出した時点で
変数m_inにデータが格納される

補足: TimedShortSeq型は配列のように
複数のデータを保持している。

ソースコードの編集

- データを書き込む手順

```

175 //前進するときのみ停止するかを判定
176 if (m_speed_x > 0)
177 {
178     for (int i = 0; i < 4; i++)
179     {
180         //センサ値が設定値以上か判定
181         if (sensor_data[i] > m_stop_d)
182         {
183             //センサ値が設定値以上の場合は停止
184             //変数m_outにデータを格納する
185             //TimedVelocity2D型のため、vxに直進速度、
186             //vaに回転速度を格納する。
187             m_out.data.vx = m_speed_x;
188             m_out.data.va = m_speed_r;
189         }
190     }
191     //設定値以上の値のセンサが無い場合はコンフィギュレーションパラメータの値で操作
192     m_out.data.vx = m_speed_x;
193     m_out.data.va = m_speed_r;
194     m_outOut.write();

```

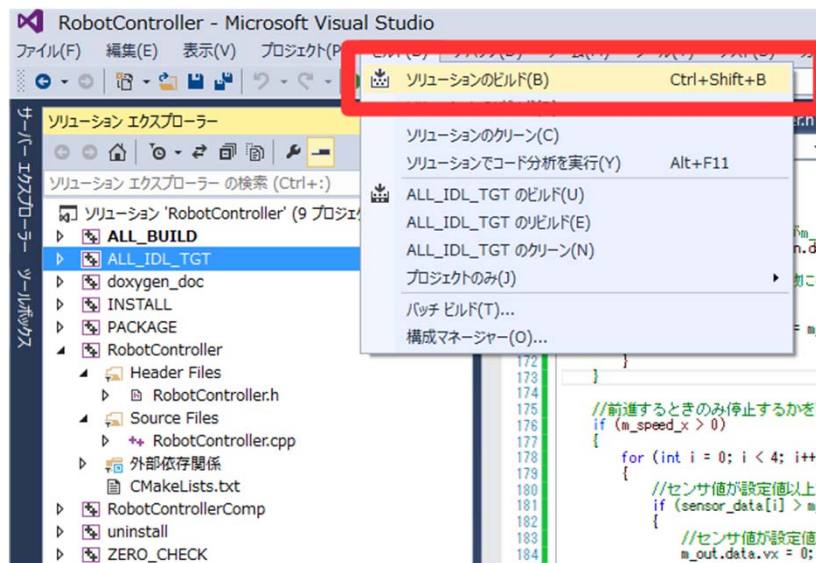
変数m_outにデータを格納する
TimedVelocity2D型のため、vxに直進速度、
vaに回転速度を格納する。

write関数でデータの書き込み

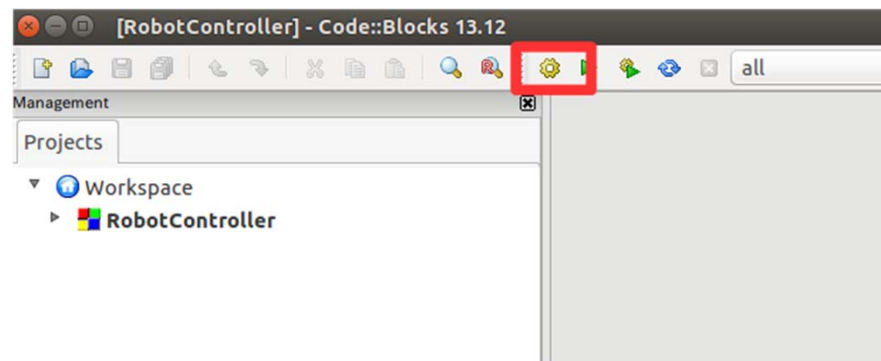
補足:コンフィギュレーションパラメータは変更すると
対応する変数(m_speed_x、m_speed_r、m_stop_d)
に値が格納される

ソースコードのコンパイル

Visual Studio



Code::Blocks

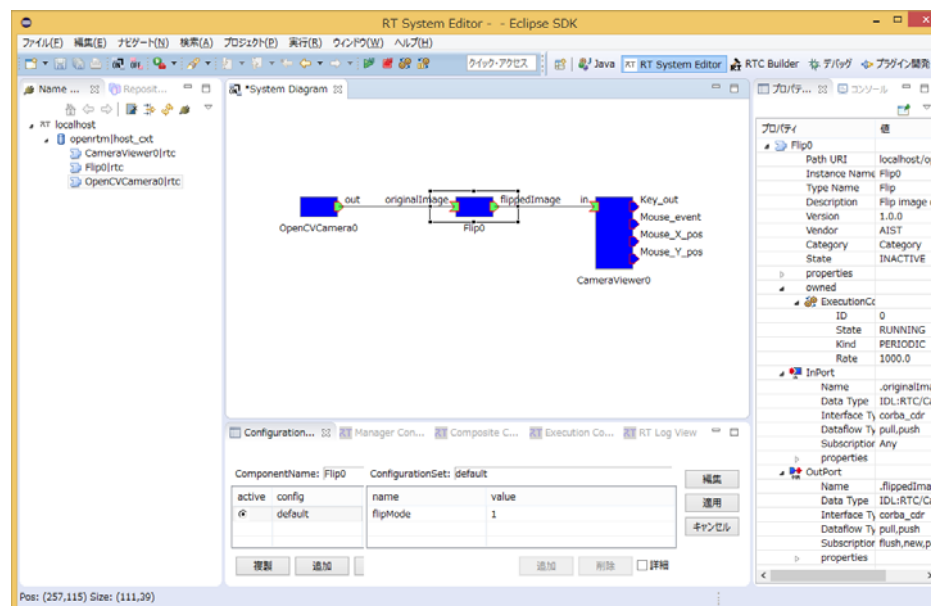


- 成功した場合、実行ファイルが生成される
 - Windows
 - build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内に RobotControllerComp.exeが生成される
 - Ubuntu
 - build/src**フォルダに RobotControllerCompが生成される

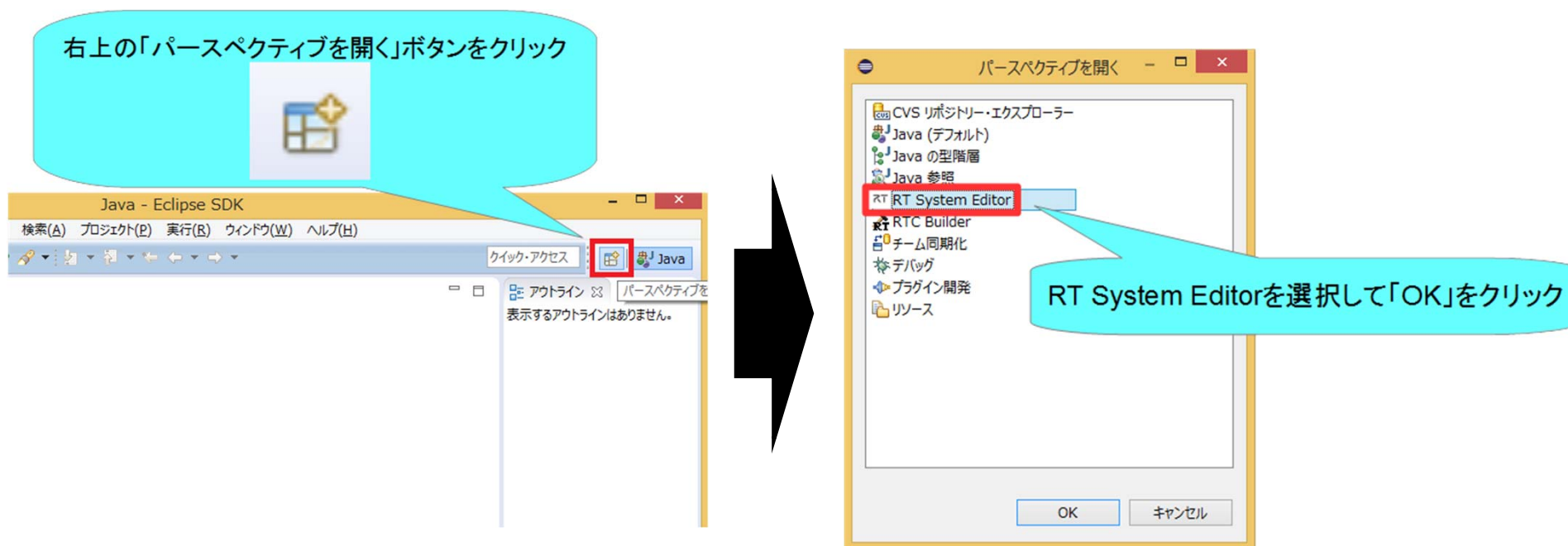
システム構築支援ツール RTSystemEditorについて

RTSystemEditor

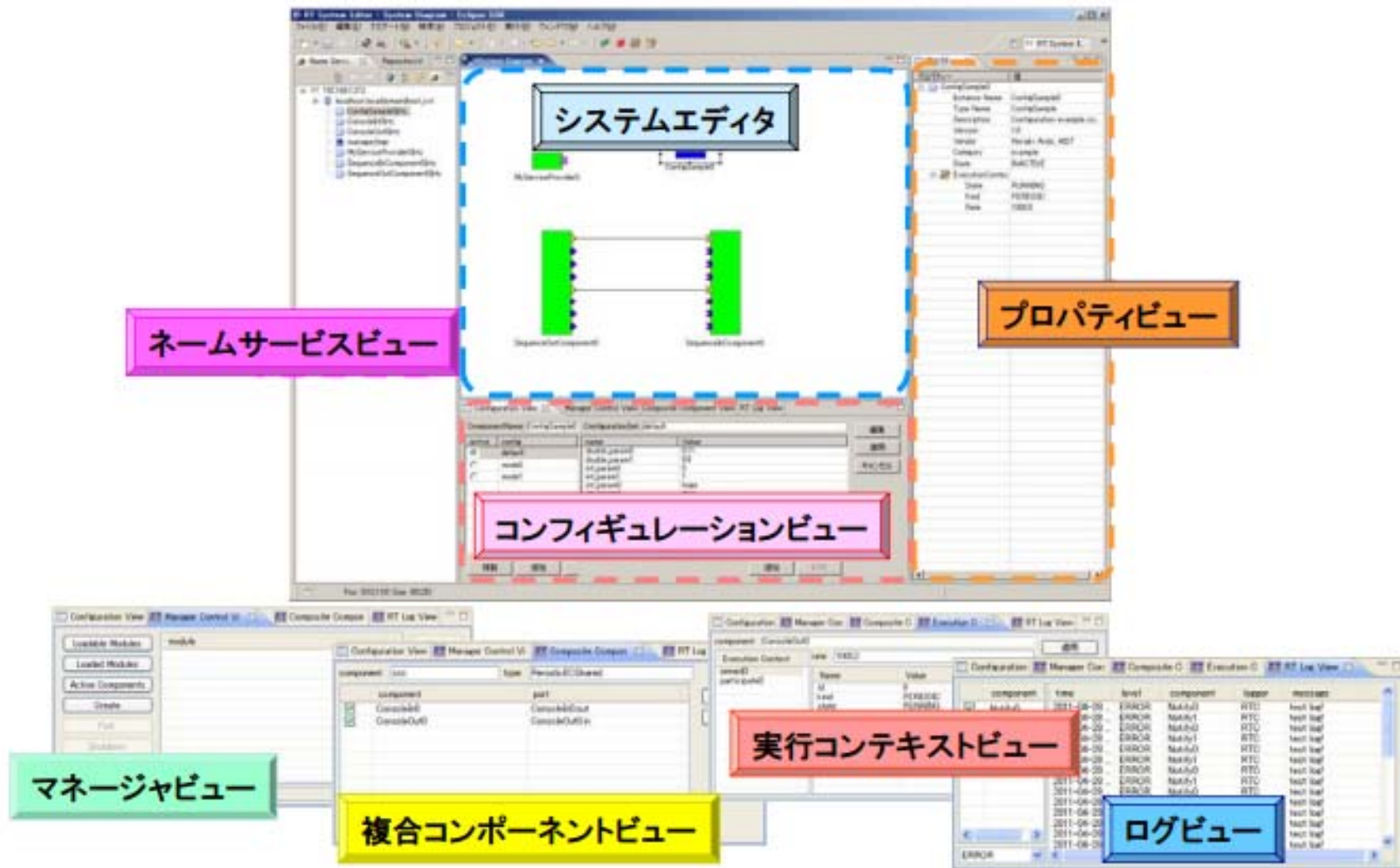
- RTCをGUIで操作するためのツール
 - データポート、サービスポートの接続
 - アクティブ化、非アクティブ化、リセット、終了
 - コンフィギュレーションパラメータの操作
 - 実行コンテキストの操作
 - 実行周期変更
 - 実行コンテキストの関連付け
 - 複合化
 - マネージャからRTCを起動
 - 作成したRTシステムの保存、復元



RT System Editorの起動

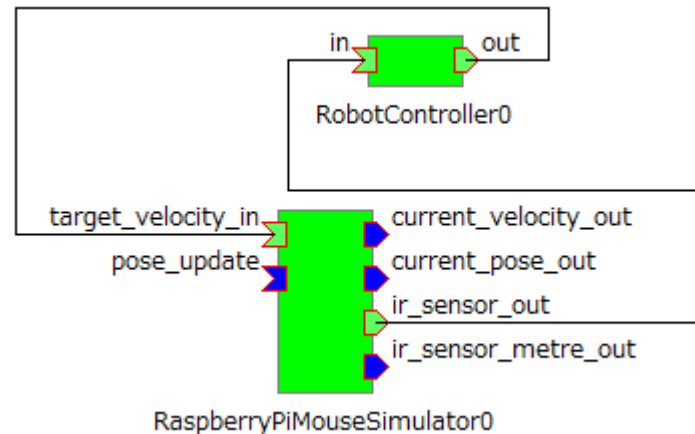


RT System Editorの画面構成



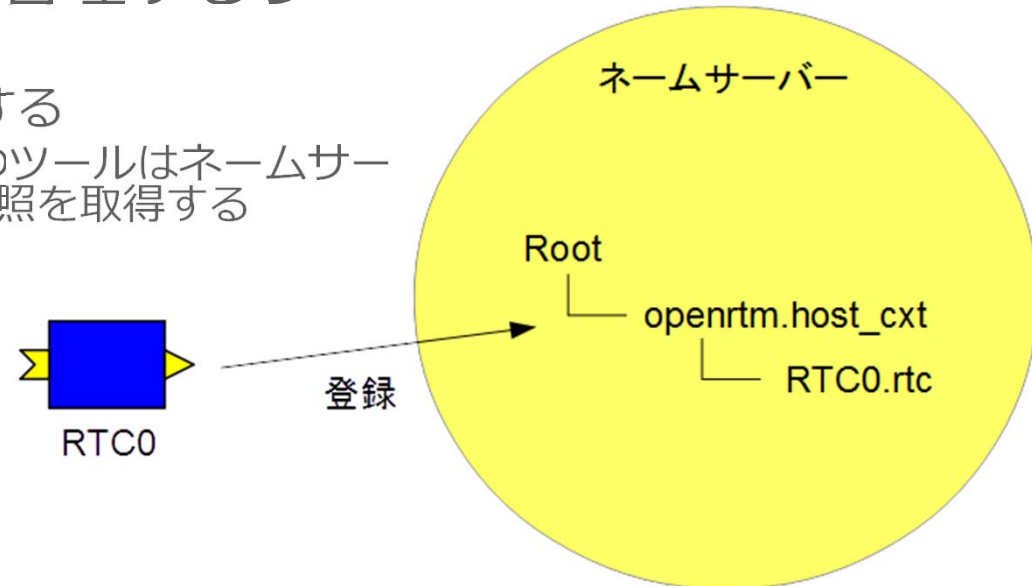
RobotControllerコンポーネントの動作確認

- シミュレータコンポーネントと接続してシミュレータ上のロボットを操作するRTシステムを作成する
 - ネームサーバーを起動する
 - RasPiMouseSimulatorコンポーネントを起動する
 - Windows
 - 配布USBメモリのEXEフォルダ内
「RaspberryPiMouseSimulatorComp.exe」をダブルクリック
 - Ubuntu
 - 配布USBメモリ内のスクリプトでインストール
 - » \$ sh install_raspimouse_simulator_offline.sh
 - RasPiMouseSimulatorRTCに移動して以下のコマンドを実行
 - » \$ build/src/RaspberryPiMouseSimulatorComp
 - RobotControllerコンポーネント起動
 - RasPiMouseSimulatorコンポーネントとRobotControllerコンポーネントを接続して「All Activate」を実行



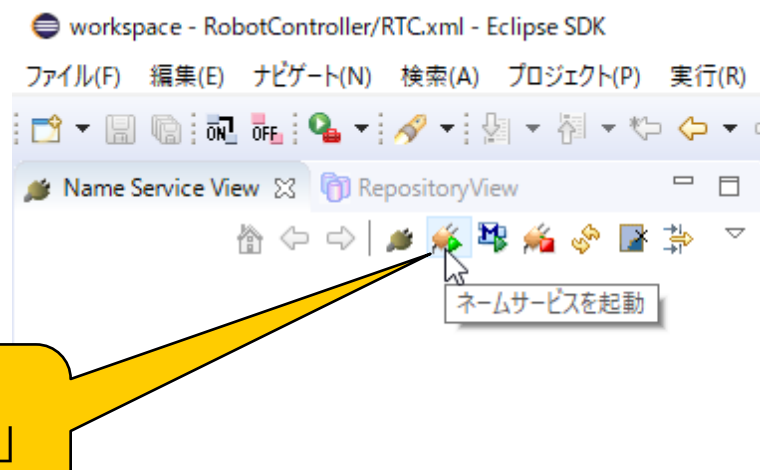
ネームサーバーの起動

- オブジェクトを名前で管理するサービス
 - RTCを一意の名前で登録する
 - RTシステムエディタ等のツールはネームサーバーから名前でRTCの参照を取得する

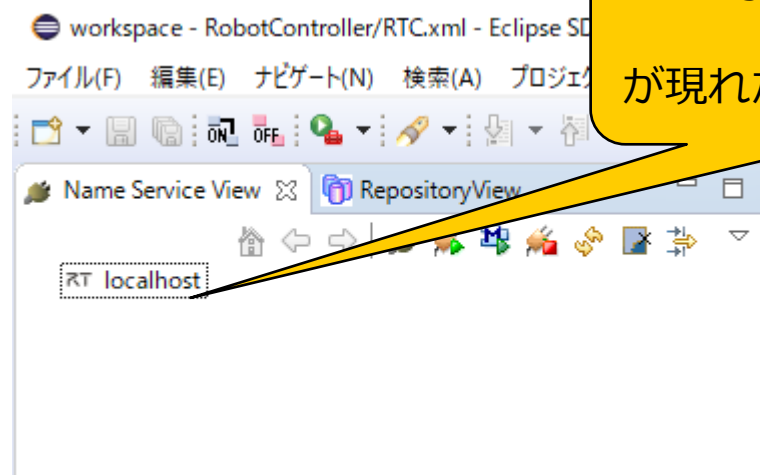


- 起動する手順
 - Windows 7
 - 「スタート」 → 「すべてのプログラム」 → 「OpenRTM-aist 1.1.2」 → 「Tools」 → 「Start Naming Service」
 - Windows 8.1
 - 「スタート」 → 「アプリビュー(右下矢印)」 → 「OpenRTM-aist 1.1.2」 → 「Start Naming Service」
 - Ubuntu
 - \$ rtm-naming

ネームサーバー起動・接続



このアイコン
「ネームサービスを起動」
をクリック

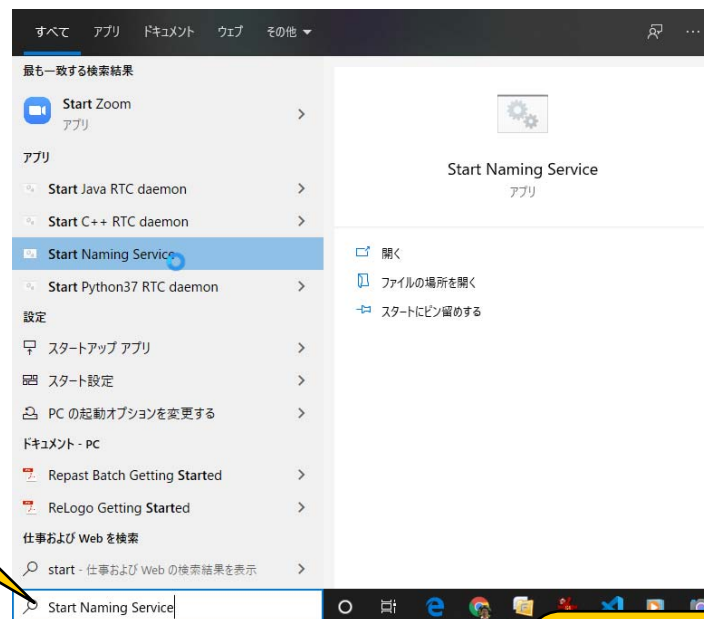


5秒ほど待って
“localhost”
が現れたら起動・接続成功

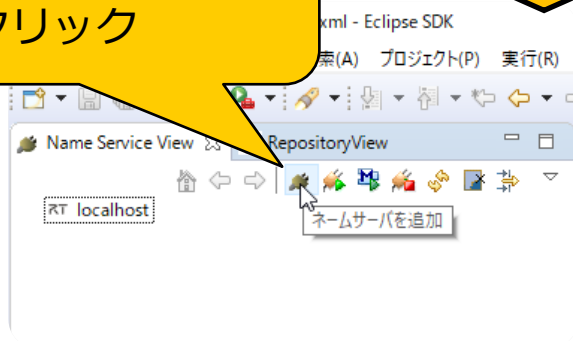
(参考) ネームサーバー起動・接続を個別に行う

- Windows 10

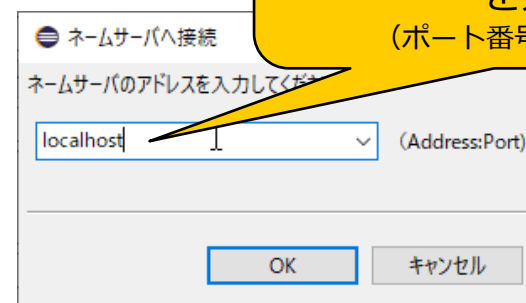
左下の検索窓に"start"と入力すると"Start Naming Service"が候補として出てくる



このアイコン
「ネームサーバを追加」
をクリック



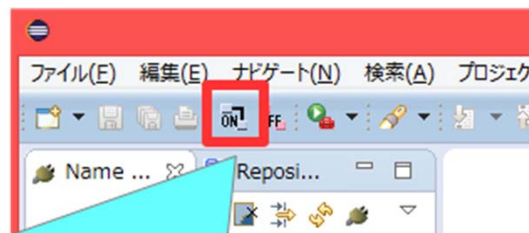
現れるダイアログに
「ホスト名：ポート番号」
を入力
(ポート番号は省略可能)



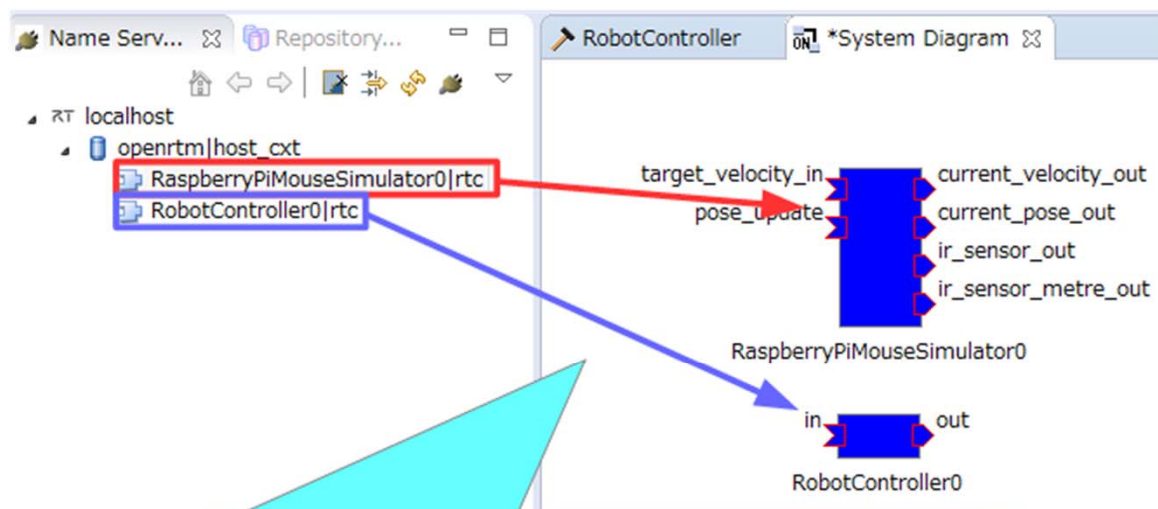
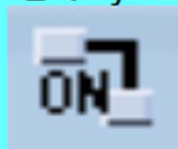
RobotControllerコンポーネントの動作確認

- シミュレータコンポーネントと接続してシミュレータ上のロボットを操作するRTシステムを作成する
 - ネームサーバーを起動する
 - RasPiMouseSimulatorコンポーネントを起動する
 - Windows
 - 配布USBメモリのEXEフォルダ内
「RaspberryPiMouseSimulatorComp.exe」をダブルクリック
 - Ubuntu
 - 配布USBメモリ内のスクリプトでインストール
» `$sudo sh install_raspimouse_simulator.sh`
 - RasPiMouseSimulatorRTCに移動して以下のコマンドを実行
» `build/src/RaspberryPiMouseSimulatorComp`
 - RobotControllerコンポーネント起動
 - Windows
 - **build¥src**フォルダの**Release(もしくはDebug)**フォルダ内に
RobotControllerComp.exeが生成されているためこれを起動する
 - Ubuntu
 - **build/src**フォルダにRobotControllerCompが生成されているためこれを起動する
 - RobotControllerコンポーネント、RasPiMouseSimulatorコンポーネントを接続して「All Activate」を行う

データポートの接続



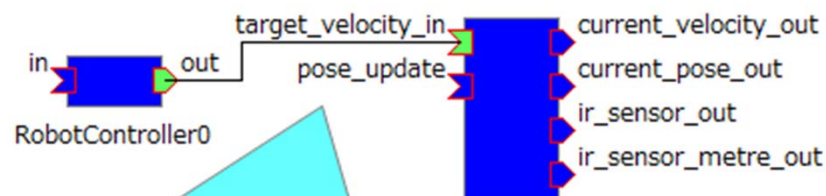
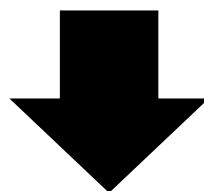
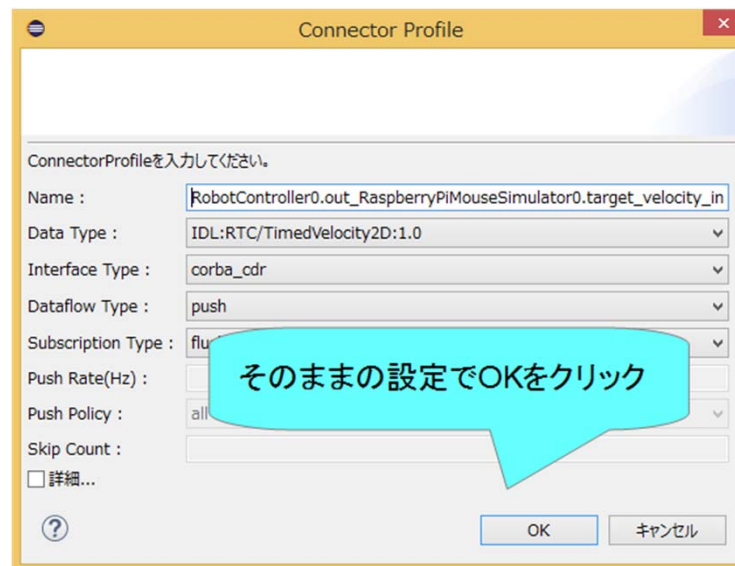
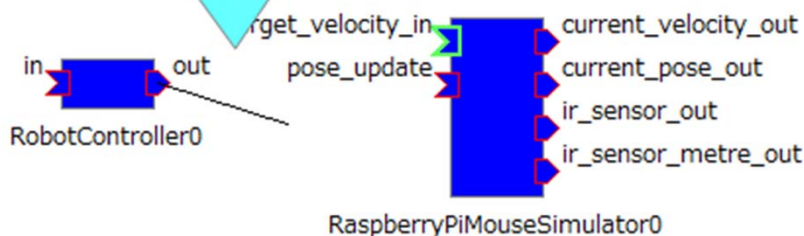
このボタンを押すことでSystem Diagramを表示する



左側のネームサービスビューから
RaspberryPiMouseSimulator0.rtc、RobotController0.rtcを
System Diagramにドラッグアンドドロップ

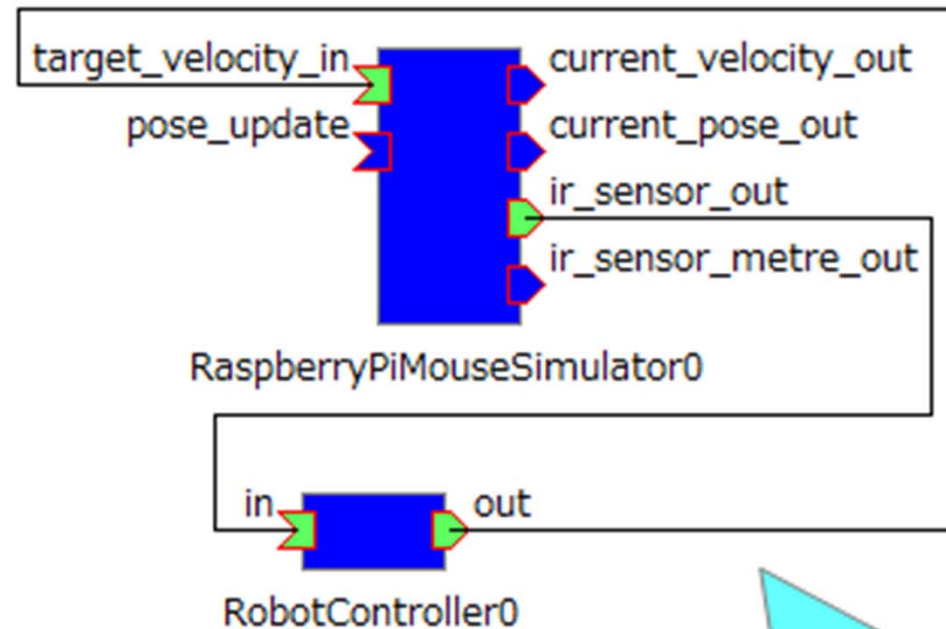
データポートの接続

RobotController0の「out」を選択して、RaspberryPiMouseSimulator0の「target_velocity_in」にドラッグアンドドロップ



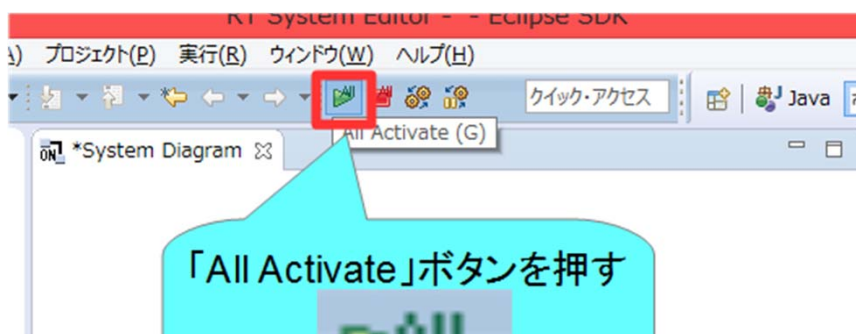
1. ポートの間に線が表示される
2. InPort、OutPortが緑色で表示される

データポートの接続

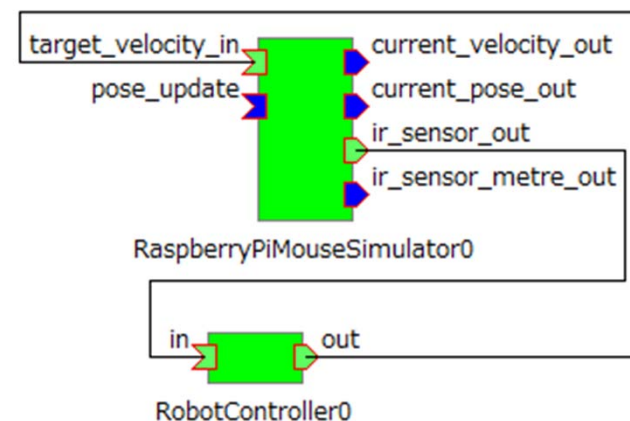
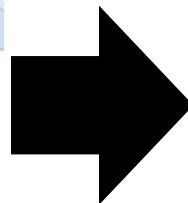


RaspberryPiMouseSimulator0の「ir_sensor_out」と RobotController0の「in」を接続する

アクティブ化



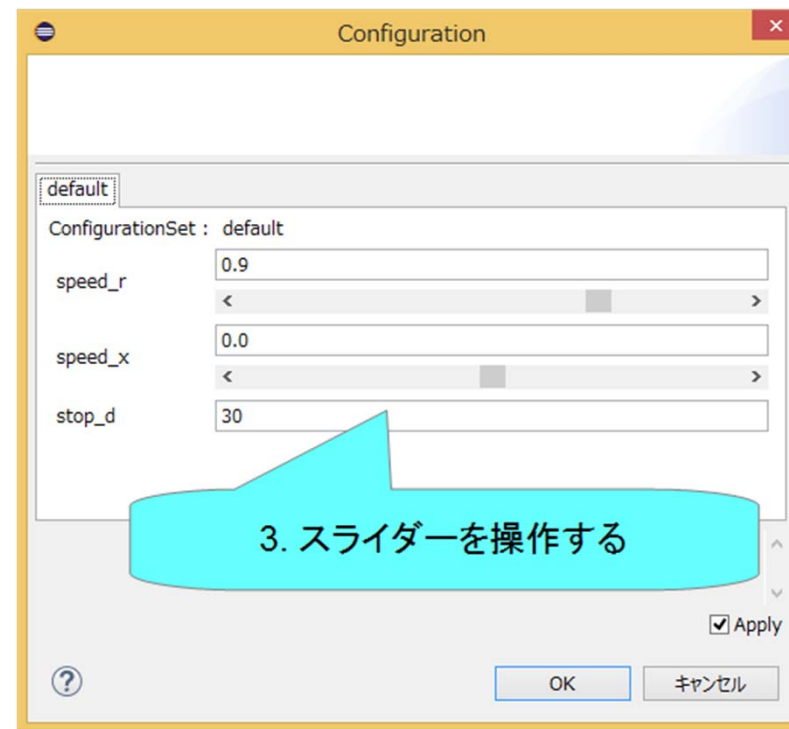
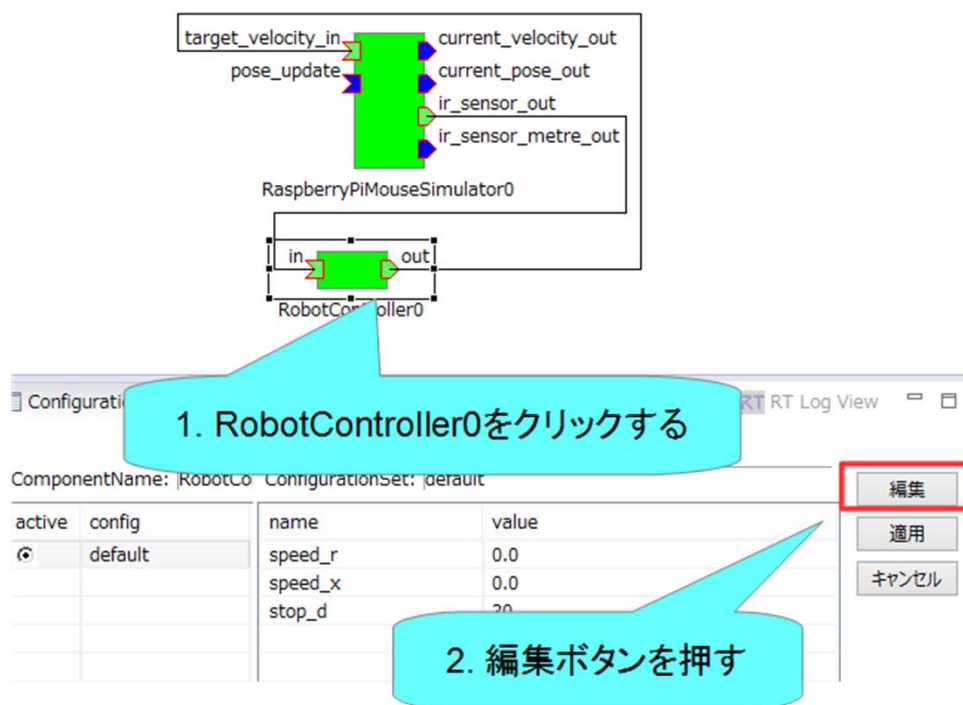
「All Activate」ボタンを押す



RTCが緑色になればアクティブ化成功

コンフィギュレーションパラメータの操作

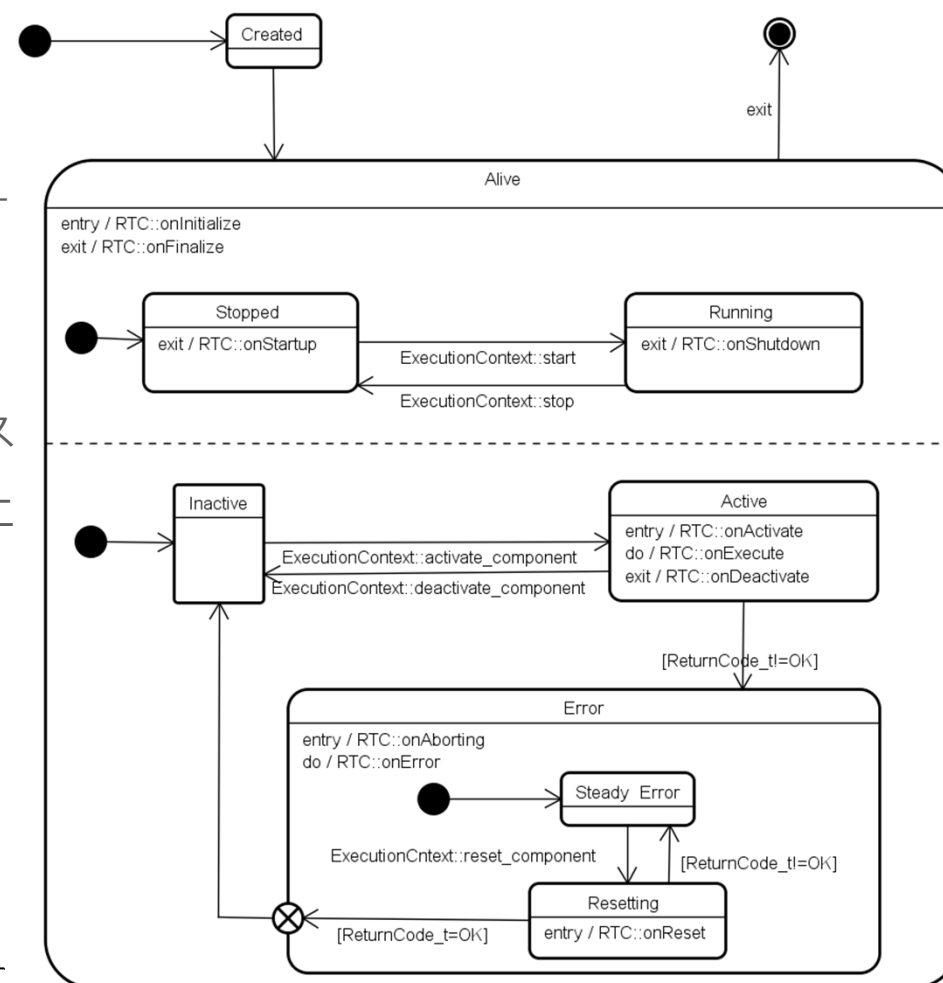
- コンフィギュレーションパラメータをRTシステムエディタから操



- 以下の動作ができるか確認
 - シミュレータ上のロボットがスライダーで操作できるか？
 - ロボットが障害物に近づくと停止するか？

RTコンポーネントの状態遷移

- RTCには以下の状態が存在する
 - Created
 - 生成状態
 - 実行コンテキストを生成し、start()が呼ばれて実行コンテキストのスレッドが実行中(Running)状態になる
 - 自動的にInactive状態に遷移する
 - Inactive
 - 非活性状態
 - activate_componentメソッドを呼び出すと活性状態に遷移する
 - RT System Editor上での表示は青
 - Active
 - 活性状態
 - onExecuteコールバックが実行コンテキストにより実行される
 - リターンコードがRTC_OK以外の場合はエラー状態に遷移する
 - RT System Editor上での表示は緑
 - Error
 - エラー状態
 - onErrorコールバックが実行コンテキストにより実行される
 - reset_componentメソッドを呼び出すと非活性状態に遷移する
 - RT System Editor上での表示は赤
 - 終了状態

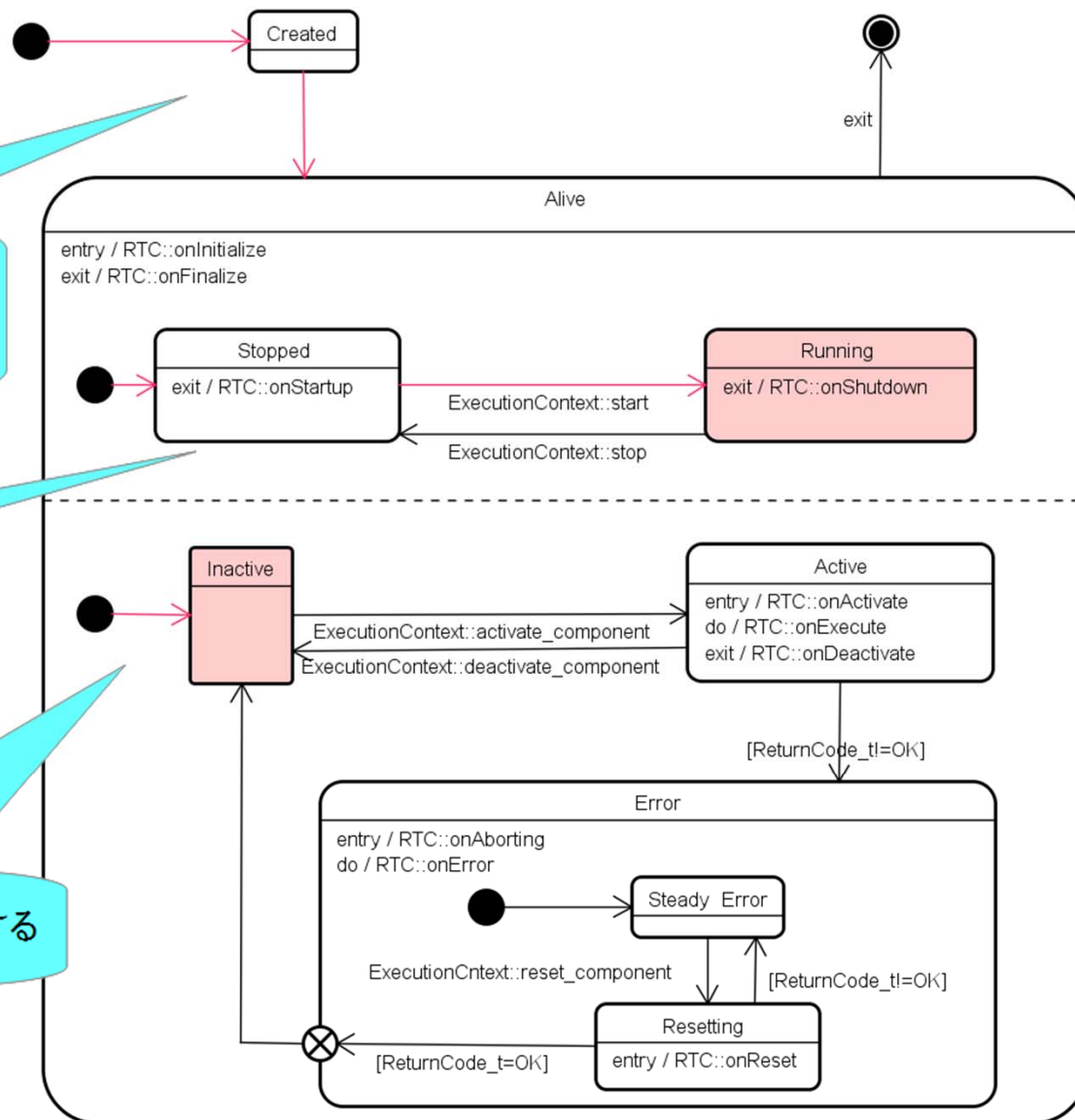


RTコンポーネントの状態遷移(生成直後)

最初にCreated状態に遷移して
実行コンテキストの生成等を行う
この時にonInitializeコールバックを実行する

startメソッドにより実行コンテキストが
Running状態に遷移する
このときonStartupコールバックが
呼び出される

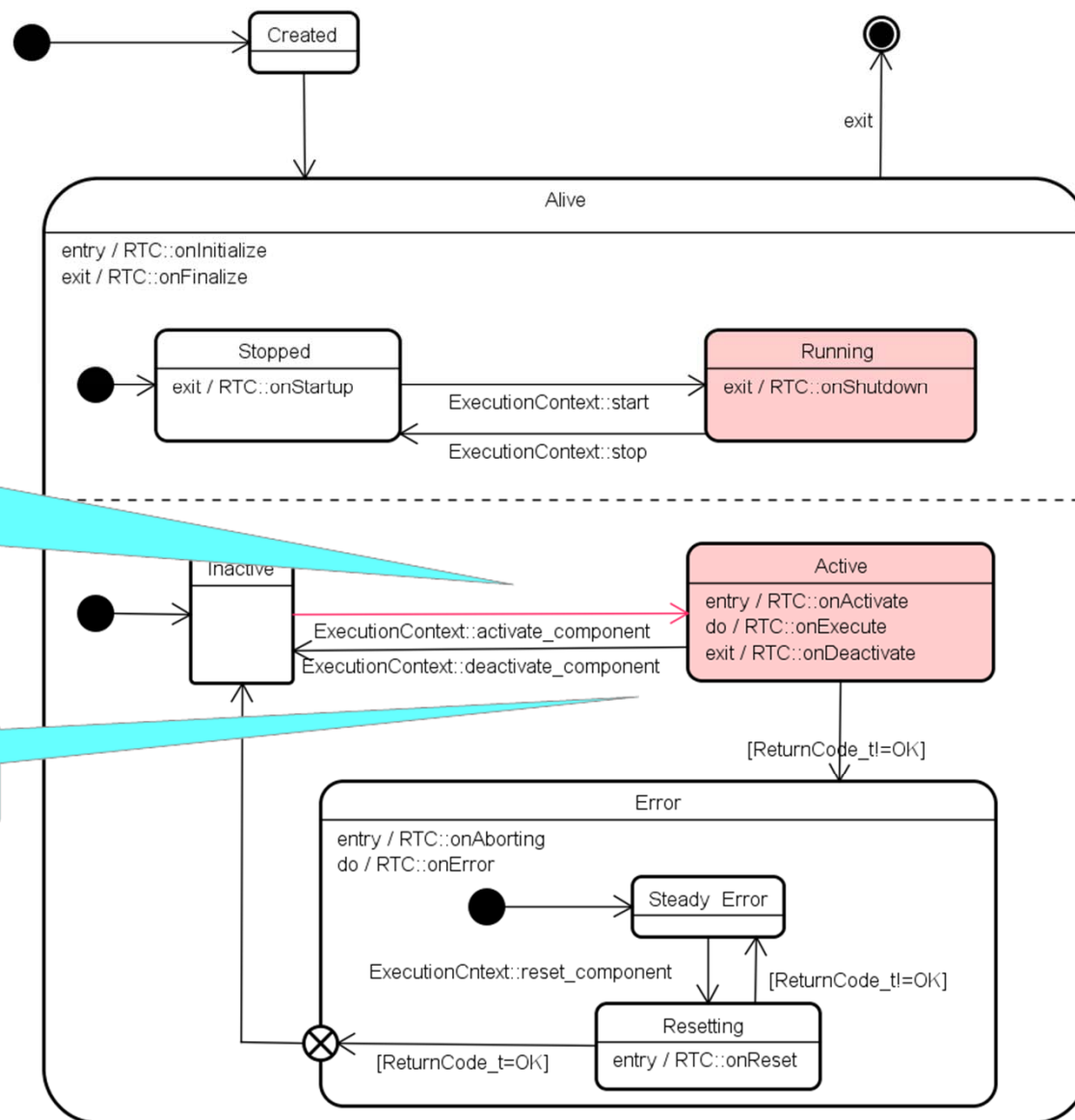
Created状態の次にInactive状態に遷移する



RTコンポーネントの状態遷移(アクティブ化)

RTシステムエディタの操作によりRTコンポーネントのアクティブ化を行うとactivate_componentメソッドが呼び出される。
activate_componentメソッドによりコンポーネントがActive状態に遷移する。
この時onActivatedコールバックが実行される

周期実行の実行コンテキストの場合、onExecuteコールバックが周期的に呼び出される。



システムの保存

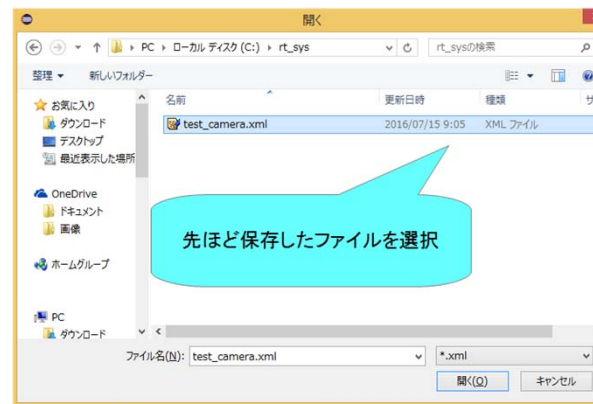
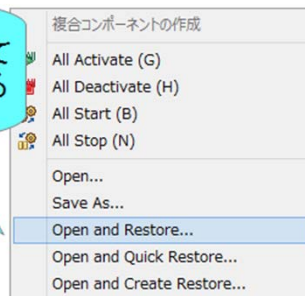
System Diagram上で右クリックして「Save As...」を選択する

ベンダ名、システム名、バージョン、保存ファイル名を入力

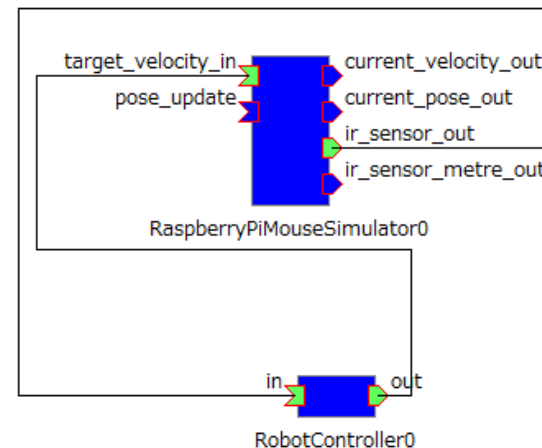
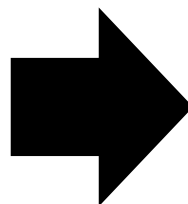
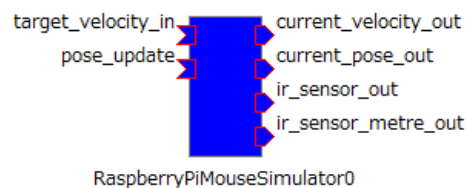
The image illustrates the process of saving a system in the RT Middleware environment. On the left, a System Diagram is shown with a component named 'RobotController0'. A right-click context menu is open over the component, with the 'Save As...' option highlighted by a red rectangle. A blue callout bubble points to this menu with the text 'System Diagram上で右クリックして「Save As...」を選択する'. A large black arrow points from the context menu to the 'Profile Information' dialog box on the right. This dialog box contains fields for 'Vendor' (AIST), 'System Name' (test), 'Version' (0), and 'Path' (C:\work\test.xml). A blue callout bubble points to these fields with the text 'ベンダ名、システム名、バージョン、保存ファイル名を入力'. Below the 'Update Log' section, there are checkboxes for 'RobotController0' and 'RaspberryPiMouseSimulator0', both of which are checked. Buttons for 'すべて選択(S)' (Select All), 'すべて解除(D)' (Deselect All), 'OK', and 'キャンセル' (Cancel) are also visible.

システムの復元

System Diagram上で右クリックして
「Open and Restore...」を選択する



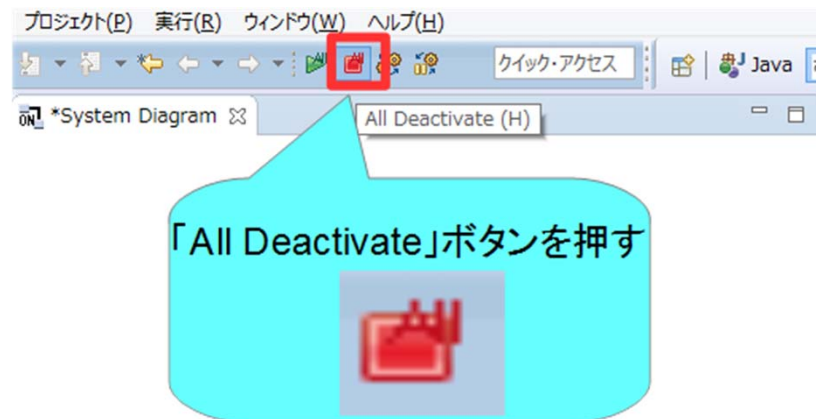
先ほど保存したファイルを選択



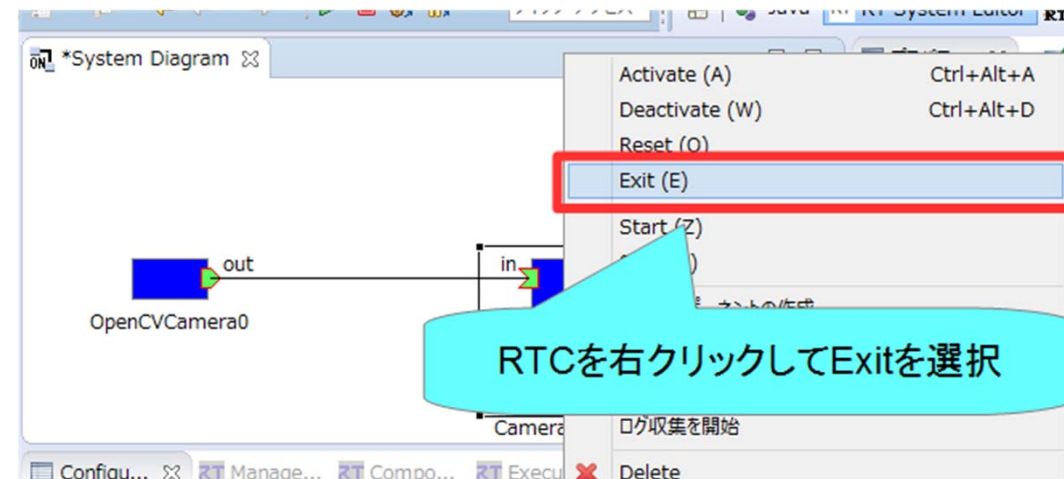
- 以下の内容を復元
 - ポート間の接続
 - コンフィギュレーション
 - 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

非アクティブ化、終了

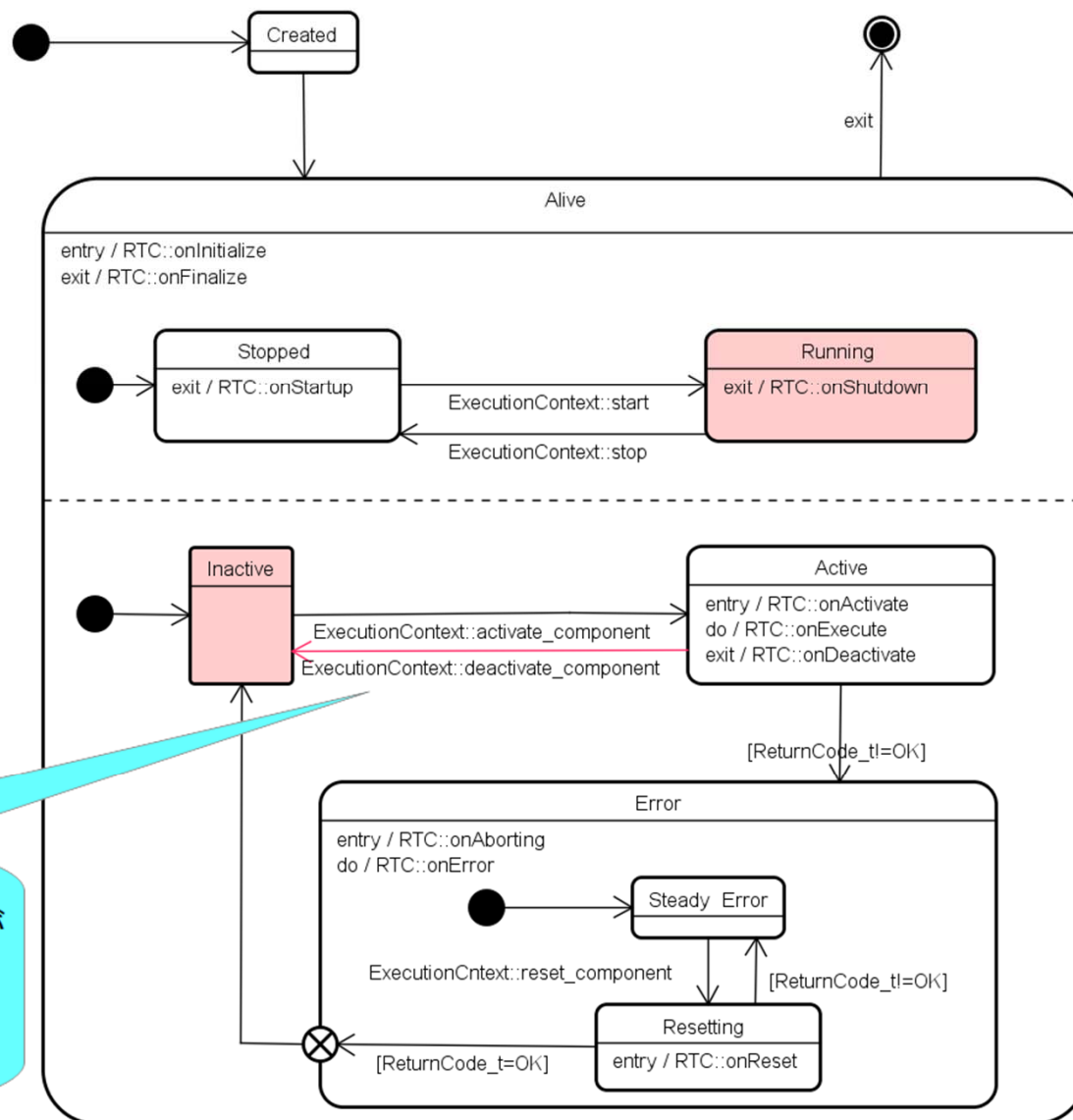
- 非アクティブ化



- 終了



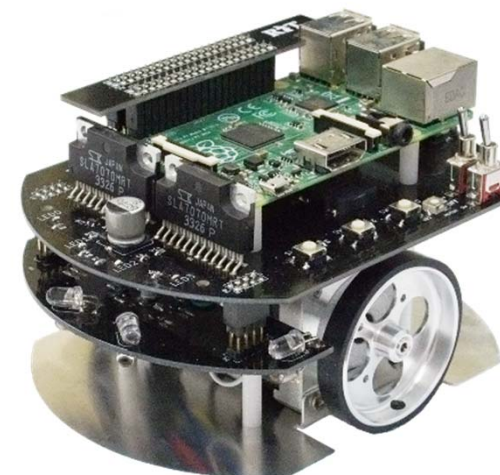
RTコンポーネントの状態遷移(非アクティブ化)



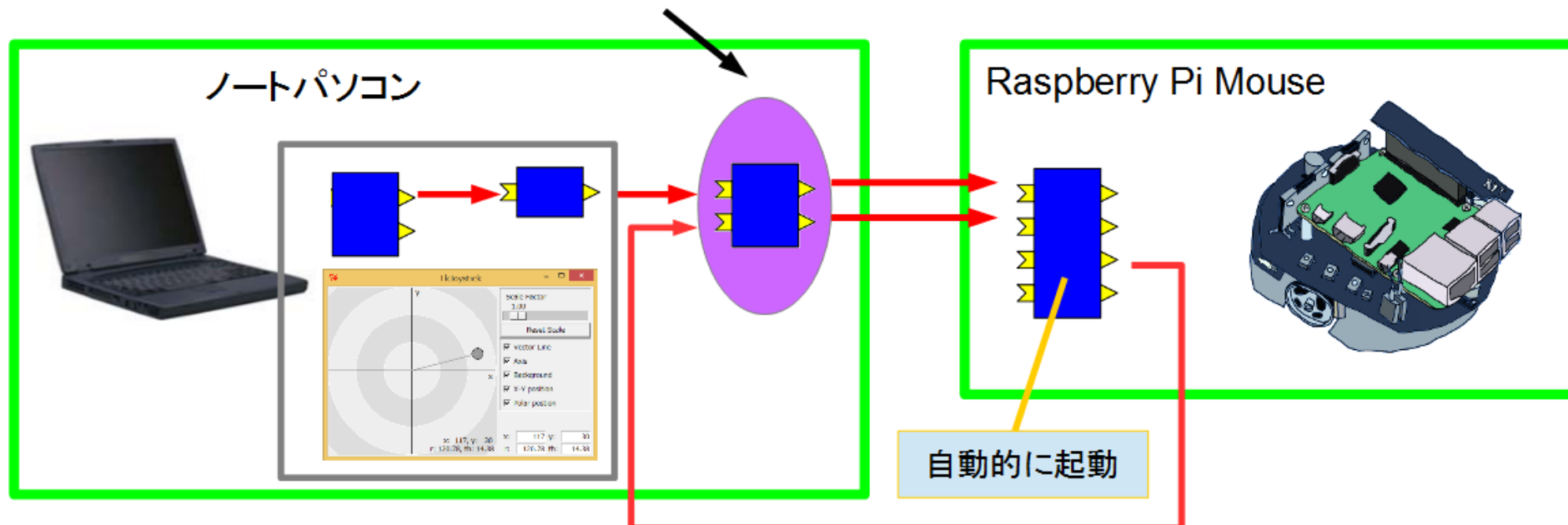
RTシステムエディタの操作によりRTコンポーネントの非アクティブ化を行うとdeactivate_componentメソッドが呼び出される。
deactivate_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onDeactivatedコールバックが実行される

実習内容（２）

- 車輪型移動ロボットを操作するRTシステムの作成
 - Raspberry Pi Mouse を使用
- まずはジョイスティックコンポーネントで動作確認を行う
- 動作確認後、各自で作成したコンポーネントでロボットの操作を行う

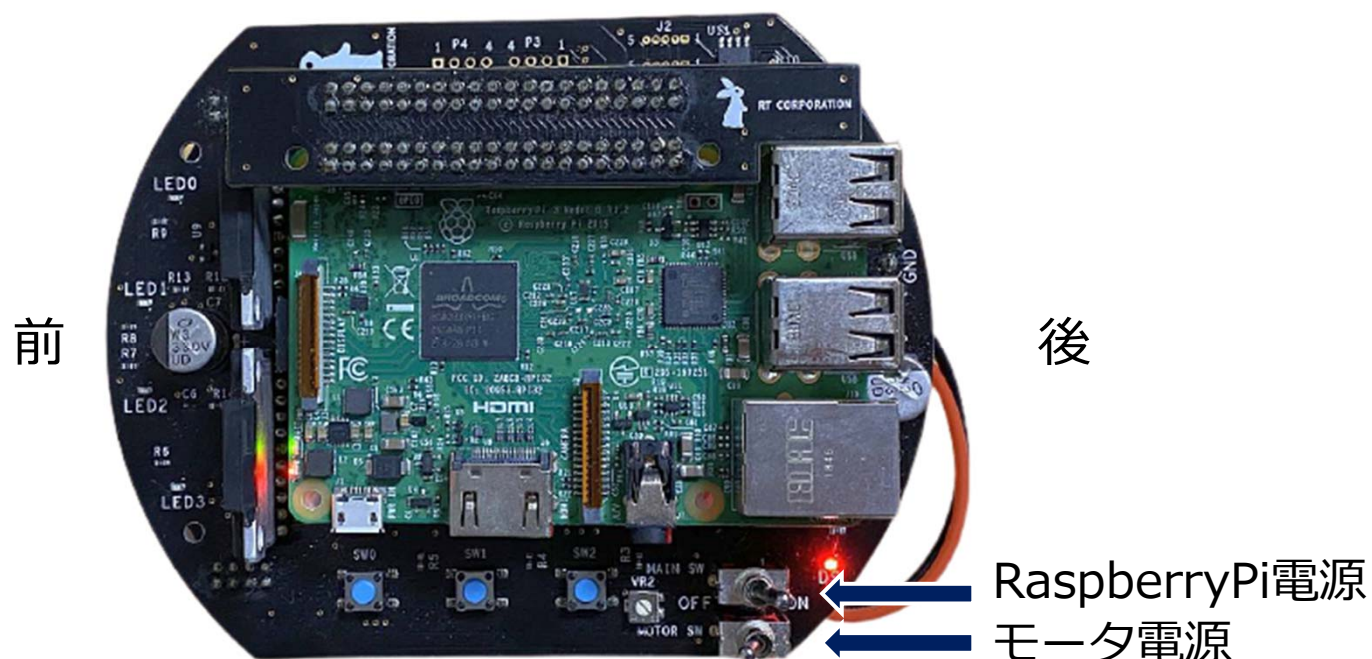


新規作成



RaspberryPi Mouse起動

RaspberryPiの電源を入れます



RaspberryPi Mouseには2つの電源スイッチがあり、内側がコンピュータ（RaspberryPi）のスイッチです。外側のモータスイッチは、走らせるときのみONにします。

実機とのWiFi接続

- 無線LANによりRaspberry Piと接続します
 - Raspberry Piがアクセスポイントとして動作する
 - ノートPCからRasPiMouseのアクセスポイントを探して、WiFi接続する

- 以下のSSIDに接続
 - SSID : RPiMouse<番号>
 - PW : openrtm-aist

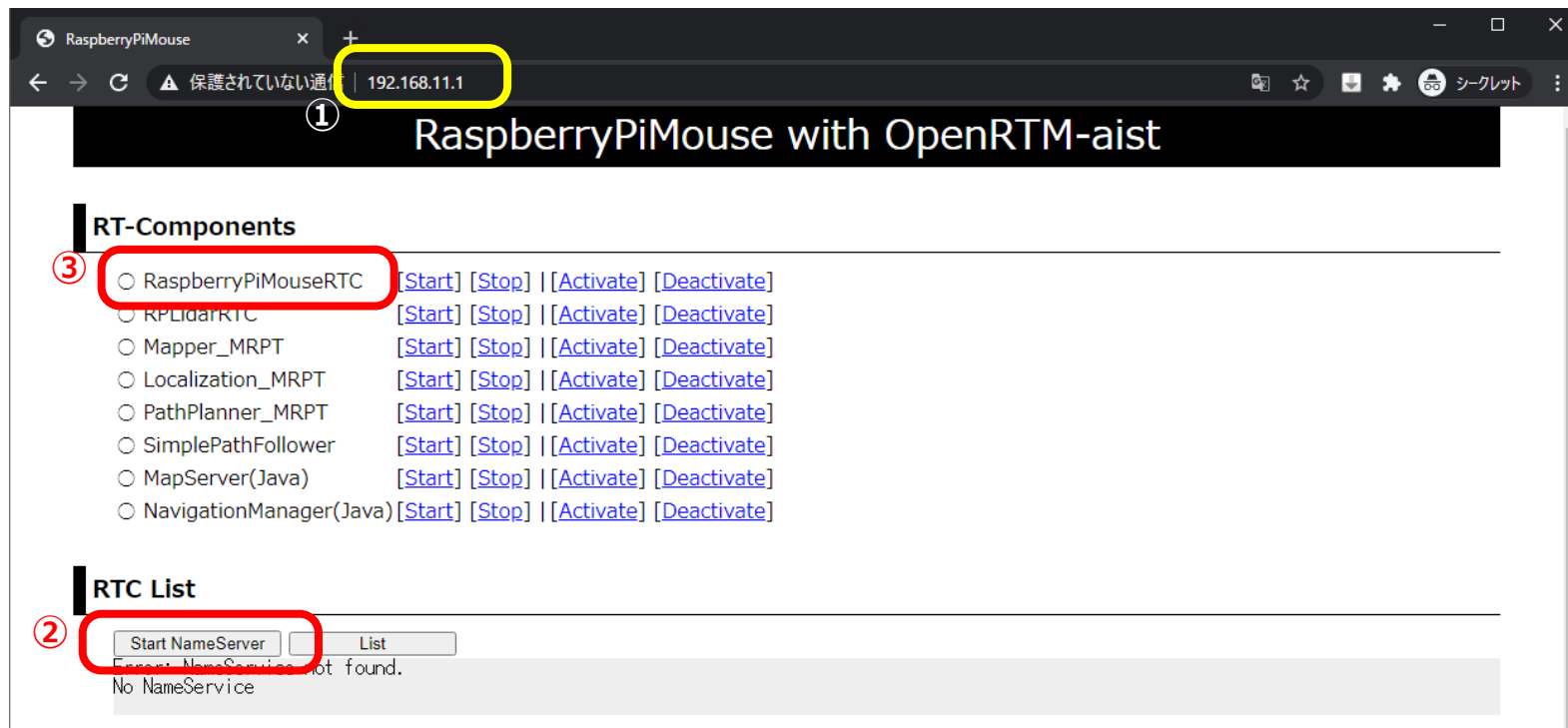


WiFiアクセスポイント

- 注意
 - RasPi MouseのWifiに接続すると、インターネットに接続できなくなります。
 - ネットワークを切り替えた際にはネームサーバー、起動中のRTCを再起動してください。

RasPiMouse RTCの起動

① ノートPCのブラウザで192.168.11.1にアクセス

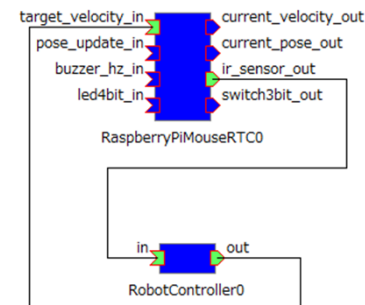
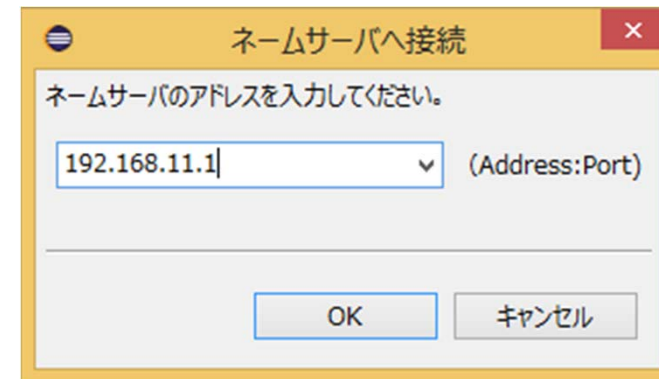
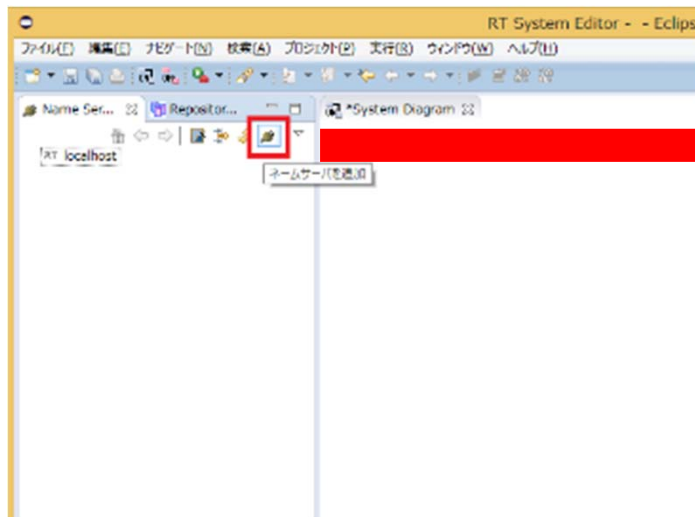


② “Start NameServer”をクリックしてネームサーバを起動

③ “RaspberryPiMouseRTC”をクリックしてRTCを起動

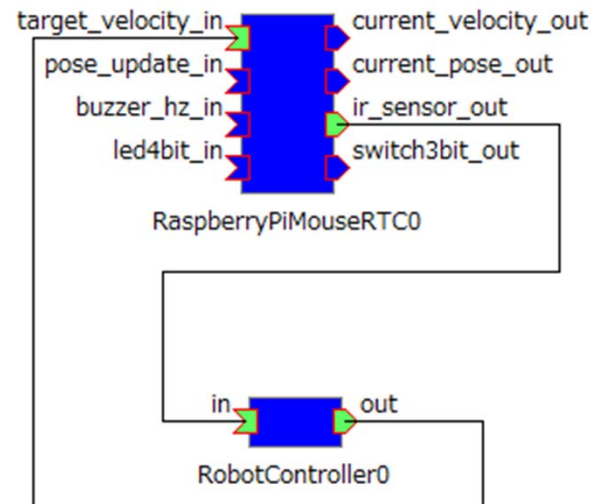
ネームサーバへの接続

- ノートPC上でOpenRTPを起動、RTSystemEditorのパースペクティブを開く
- ネームサービスビューの接続ボタンを押し、RasPiMouseのネームサーバへ接続
 - ダイアログには 192.168.11.1 を入力



RTCの接続と動作確認

- ノートPC上でネームサーバを起動
- ノートPC上でRobotControllerを起動
- RTCを配置
 - RasPiMouse上のRaspberryPoMouseRTCをドラッグアンドドロップ
 - ノートPC上のRobotControllerをドラッグアンドドロップ
- ポートを接続
 - RobotController.out → RasPiMouseRTC.target_velocity_in
 - RasPiMouseRTC.ir_sensor_out → RobotController.in



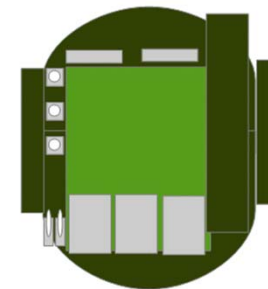
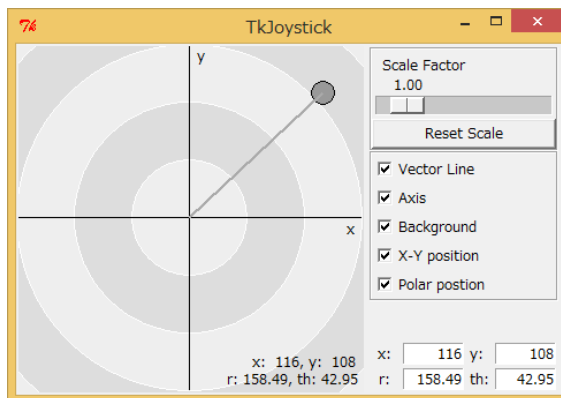
チュートリアル

- おまけのチュートリアル
 - JoystickでRasPiMouseを操作してみる

11月11日 (水)	
10:00 -11:45	4. 総合演習(1) (1) ロボットシステムの設計 (2) ロボット制御プログラムの作成 ・チュートリアル (Raspberry Pi Mouseシミュレータ、Windows編) ・チュートリアル (RaspberryPiマウス) 資料: 201111-04.pdf
11:45 -12:30	昼食
12:30 -16:45	5. 総合演習(2) (1) SLAMについて (2) SLAMにより地図作成・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: 201111-05.pdf

ジョイスティックによる操作

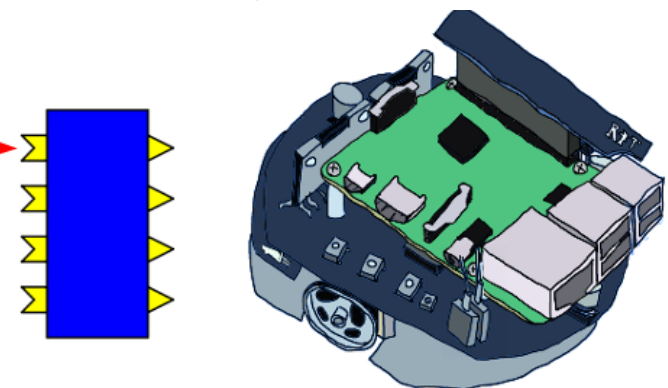
- ジョイスティックでも操作してみる



ノートパソコン



ラズパイマウス



起動するRTC

- ノートPC側
 - TkJoystick
 - OpenRTMのサンプルとして付属
 - 検索窓で"TkJoystickComp"と入力・実行
 - FloatSeqToVelocity
 - ジョイスティックの出力をTimedVelocity2Dに変換するコンポーネント
 - 共有フォルダにて配布
- RasPi Mouse側
 - RaspberryPiMouseRTC
 - 前の実習で使ったもの

おわりに

- 時間があまったら、自作したコンポーネントを改造してみましょう
 - RTCBuilderでポートを追加して、自律移動とJoystickのハイブリッド制御を試みる
 - センサ値を利用して、壁伝いに移動してみる
 - 迷路で壁にぶつからずに走行する
 - 等