

Processing実習

宮本 信彦

国立研究開発法人産業技術総合研究所
インテリジェントシステム研究部門



資料

- 「WEBページ」フォルダのHTMLファイルを開く
 - Processing 活用事例 _ OpenRTM-aist.html
- もしくは以下のリンク
 - <https://openrtm.org/openrtm/ja/node/7232>



Processing 活用事例

 いいね! Facebookに登録して、友達の「いいね!」を見てみましょう。

Table of contents

- Processingとは?
- 実習概要
- Processing開発環境の起動
- OpenRTM-aist Processing用ライブラリのインストール
- graficaのインストール
- プログラミング
- RTシステムの構築、動作確認

Processingとは?

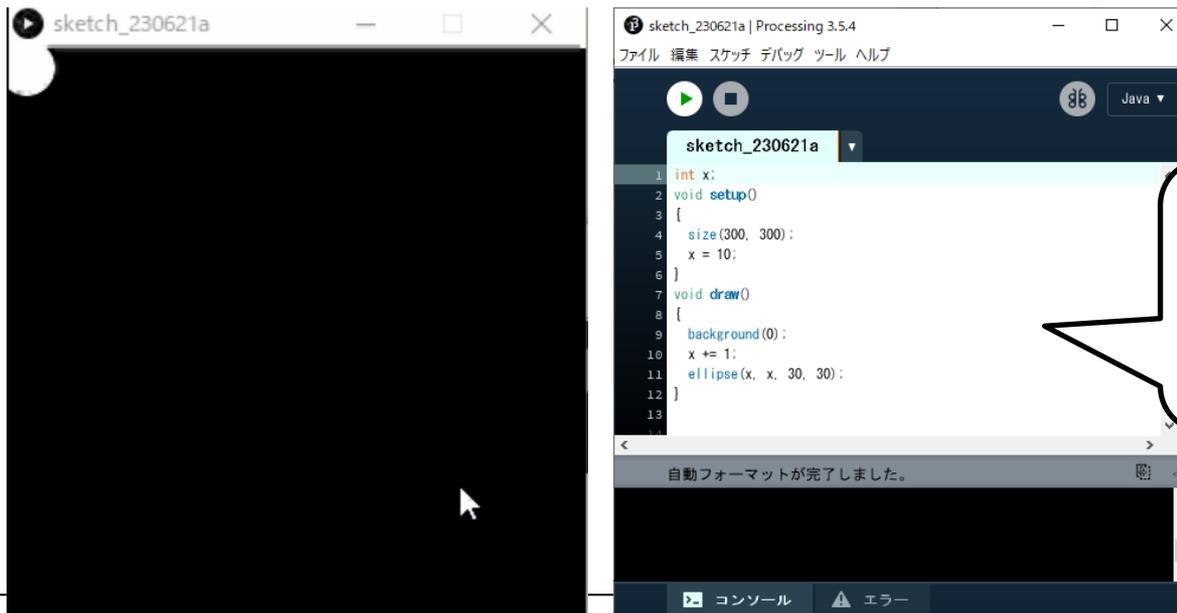
Processingはオープンソースのプログラミング言語で、以下の特長があることから初心者向けであるとされています。

- 視覚的な表現が他の言語と比較して簡単である。(グラフや図形のアニメーションやインタラクション等)
- 開発環境の導入が簡単

実習概要

プログラミング言語Processing

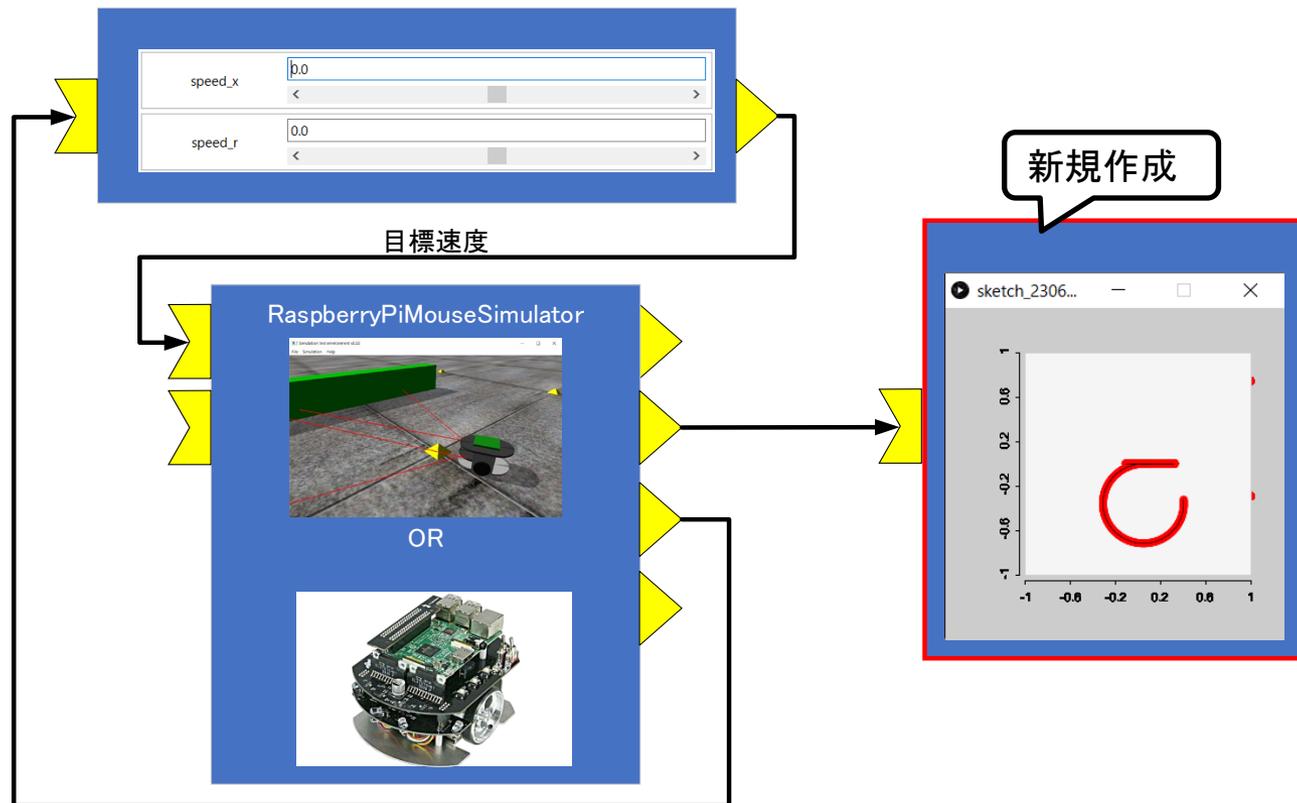
- 以下の特長を持つプログラミング言語
 - 視覚的な表現が他の言語と比較して簡単であり、初心者向け
 - グラフや図形のアニメーション
 - インタラクション等



- 例えば、動画の図形を動かすアニメーションを作成する場合、12行のコードで記述できる。
- 実行する場合は統合開発環境の左上の実行ボタンを押す。

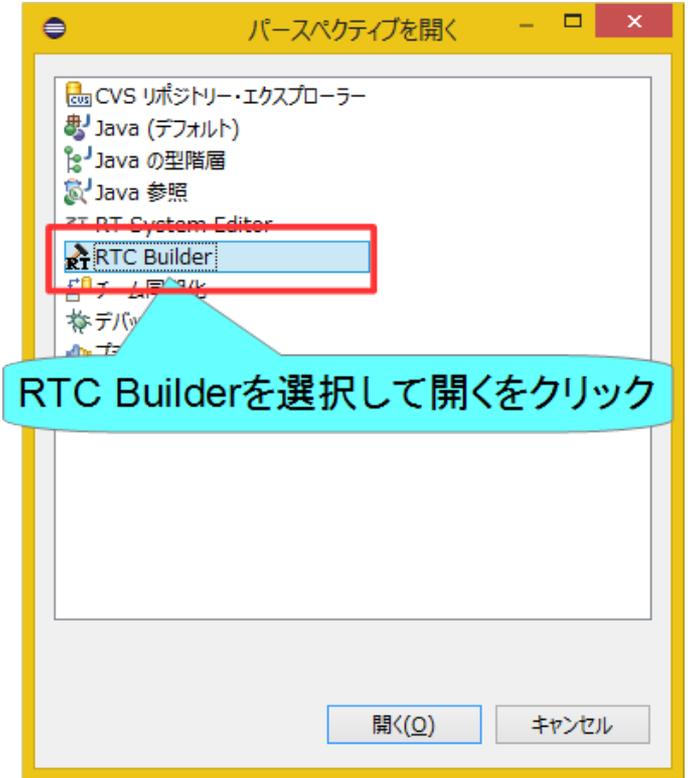
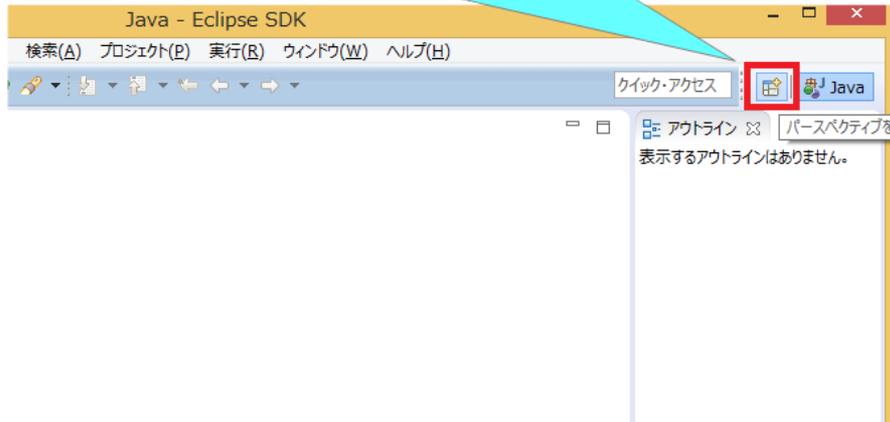
実習の概要

- Raspberry Piマウスの移動経路をグラフに描画するシステムの作成



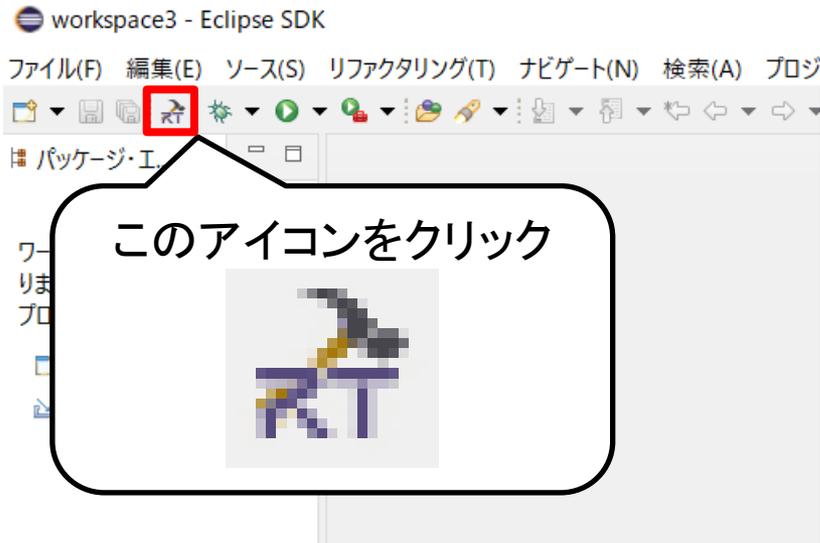
RTC Builderの起動

右上の「パースペクティブを開く」ボタンをクリック

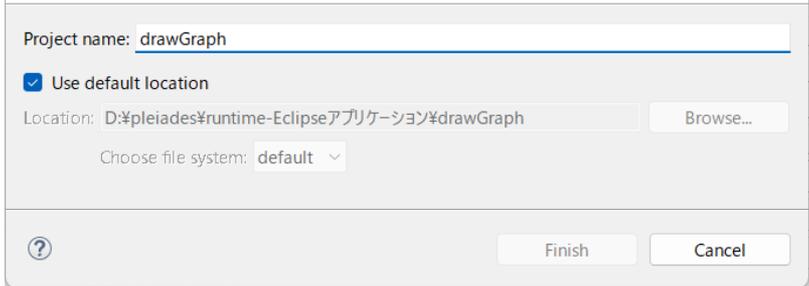


RTC Builderを選択して開くをクリック

RTC Builderプロジェクト作成



プロジェクト名に「drawGraph」と入力して終了をクリックする



- Eclipse起動時にワークスペースに指定したディレクトリに「drawGraph」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

基本プロファイルの入力

- **コンポーネント名**
 - **drawGraph**
- モジュール概要
 - 任意(Draw graph 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(Sample)
- 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

drawGraph ☒

RT-Component Basic Profile

▼ RT-Component Basic Profile ▼ ヒント

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: drawGraph コンポーネ:

概要: Draw graph component 概要:

*バージョン: 1.0.0 バージョン

*ベンダ名: AIST ベンダ名:

*カテゴリ: Sample カテゴリ:

コンポーネント型: STATIC コンポーネ:

アクティビティ型: PERIODIC コンポーネ:

最大インスタンス数: 1 コンポーネ:

実行型: PeriodicExecutionContext コンポーネ:

実行周期: 1000.0 コンポーネ:

概要: 2次元平面グラフ上に移動ロボットの位置を描画するコンポーネント アクティビ

RTC Type: 最大イン

▼ 言語 実行型:

このセクションでは使用する言語を指定します 実行周期

基本 | アクティビティ | FSM | 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 | RTC.xml

「基本」タブを選択

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**
 - **onExecute**
- 今回は練習のため、Documentationは空白でも大丈夫です

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

▼ Documentation

このセクションでは各アクションの概要を説明するドキュメントを記述します。
上段のアクションを選択すると、それぞれのドキュメントを記述できます。

アクティビティ名: ON OFF

動作概要:

事前条件:

基本 | **アクティビティ** | FSM | 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 | RTC.xml

「アクティビティ」タブを選択

データポートの設定

- 以下のInPortを設定する

- in

- データ型：
RTC::TimedPose2D
- 他の項目は任意
- ※TimedPose3D型と間違えないようにしてください。
- ※TimedPoint2D型と間違えないようにしてください。

データポート

▼ 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)		*ポート名 (OutPort)	
in	Add		Add
	Delete		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: in (InPort)

*データ型: Reload

IDLファイル:

変数名:

表示位置:

Documentation

概要説明:

詳細説明:

基本
アクティビティ
FSM
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
RTC.xml

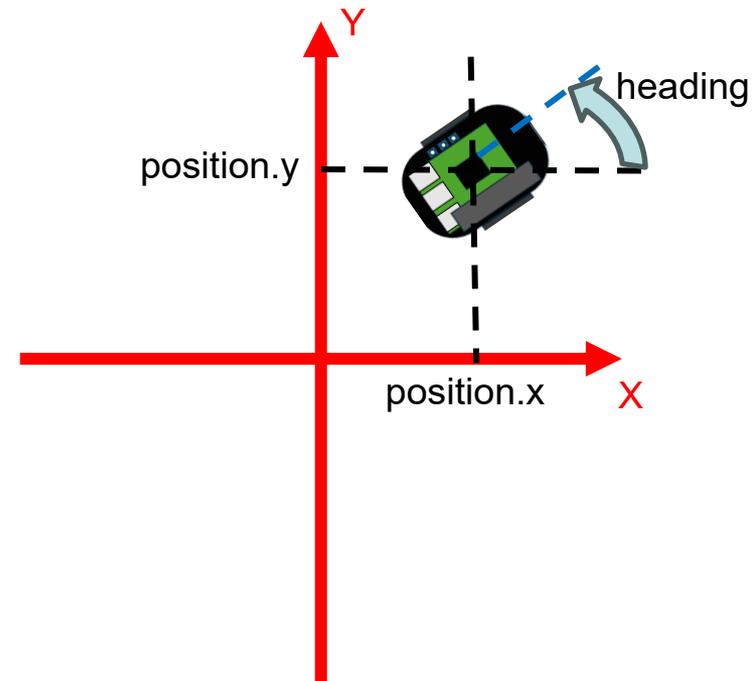
「データポート」タブを選択

RTC::TimedPose2D型について

- ExtendedDataTypes.idlで定義されている**2次元平面状での位置・姿勢**を表現するためのデータ型
 - position.x**: X軸座標
 - position.y**: Y軸座標
 - heading**: Z軸周りの角度

```
struct Pose2D
{
    /// 2D position.
    Point2D position;
    /// Heading in radians.
    double heading;
};
```

```
struct Point2D
{
    /// X coordinate in metres.
    double x;
    /// Y coordinate in metres.
    double y;
};
```



言語の設定

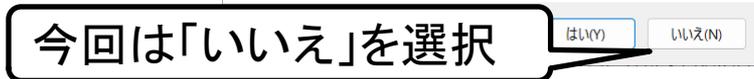
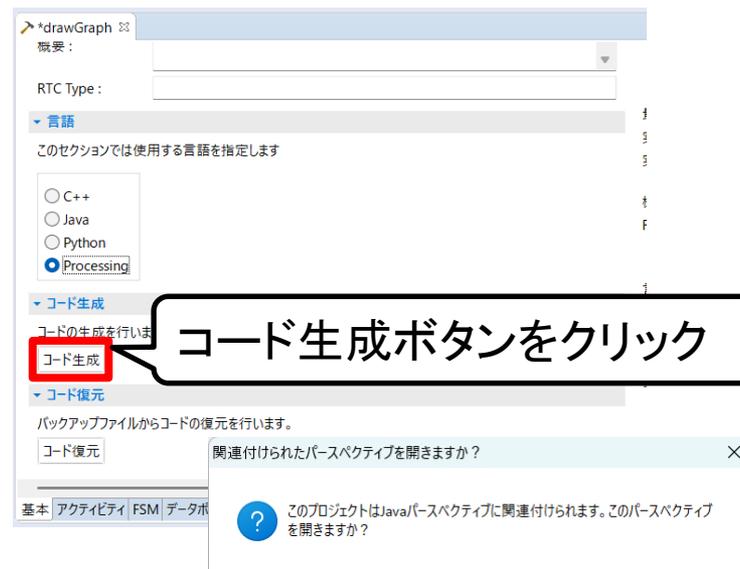
- 実装する言語，動作環境に関する情報を設定



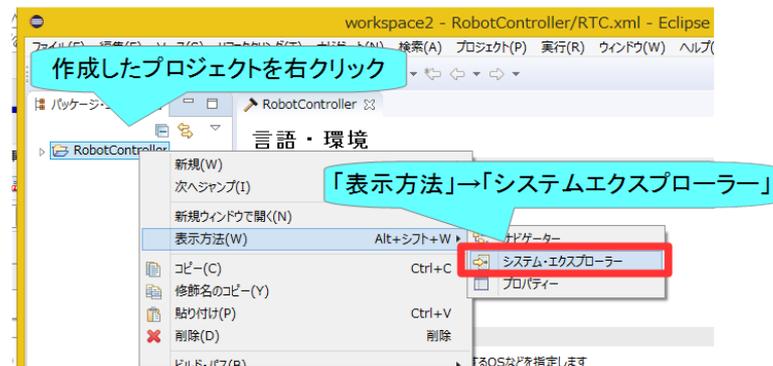
The screenshot shows a configuration page for a project named '*drawGraph'. It includes fields for a summary and RTC Type. The '言語' (Language) section is expanded, showing radio buttons for C++, Java, Python, and Processing. The 'Processing' option is selected and highlighted with a red box. A callout bubble points to this selection with the text: '言語を選択する 今回は「Processing」を選択' (Select the language. This time, select 'Processing'). Below this, the 'コード生成' (Code Generation) section is visible, with a callout bubble pointing to the '基本' (Basic) tab in the bottom navigation bar, containing the text: '「基本」タブを選択' (Select the 'Basic' tab).

スケルトンコードの生成

- 基本タブからコード生成ボタンを押すことでスケルトンコードが生成される
 - Workspace¥drawGraph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、drawGraph.conf
 - 以下略



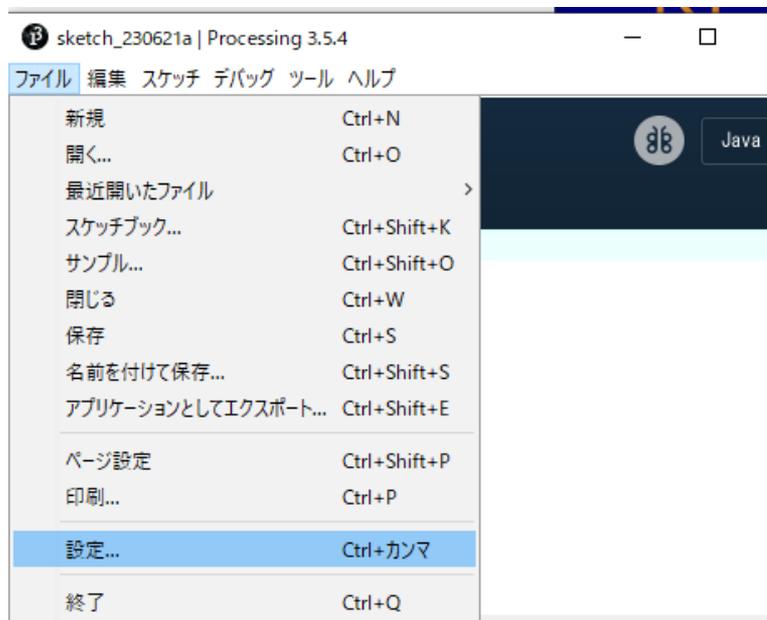
- 生成したファイルの確認
 - 作成したプロジェクトを右クリックして、「表示方法」→「システムエクスプローラー」を選択する
 - エクスプローラーでワークスペースのフォルダが開くため、上記のファイルが存在するかを確認する



Processing開発環境の起動

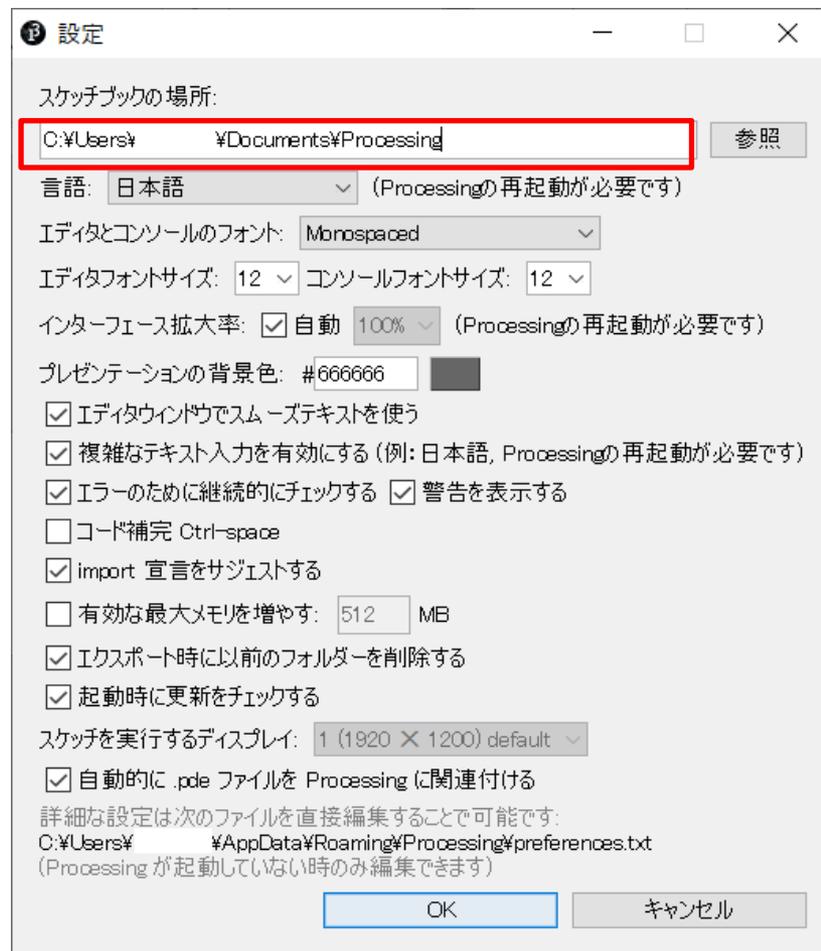
- **Windows : processing.exe**を実行する
 - <https://github.com/processing/processing/releases/download/processing-0270-3.5.4/processing-3.5.4-windows64.zip>
 - USBメモリの**Processing¥processing-3.5.4-windows64**フォルダ内
 - Javaのバージョンの問題で、OpenRTM-aistはProcessing 4.0以降では現状は利用不可
- **Ubuntu : processing**を実行する
 - USBメモリの**Processing¥processing-3.5.4-linux64**フォルダ内

OpenRTM-aist Processing用ライブラリのインストール



自動フォーマットが完了しました。

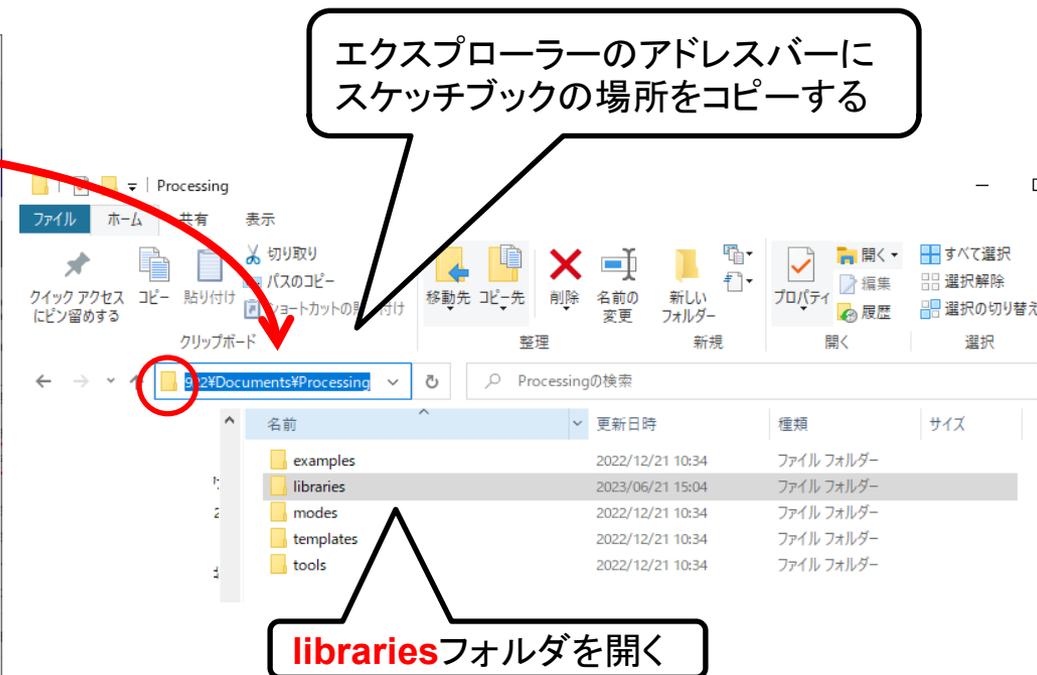
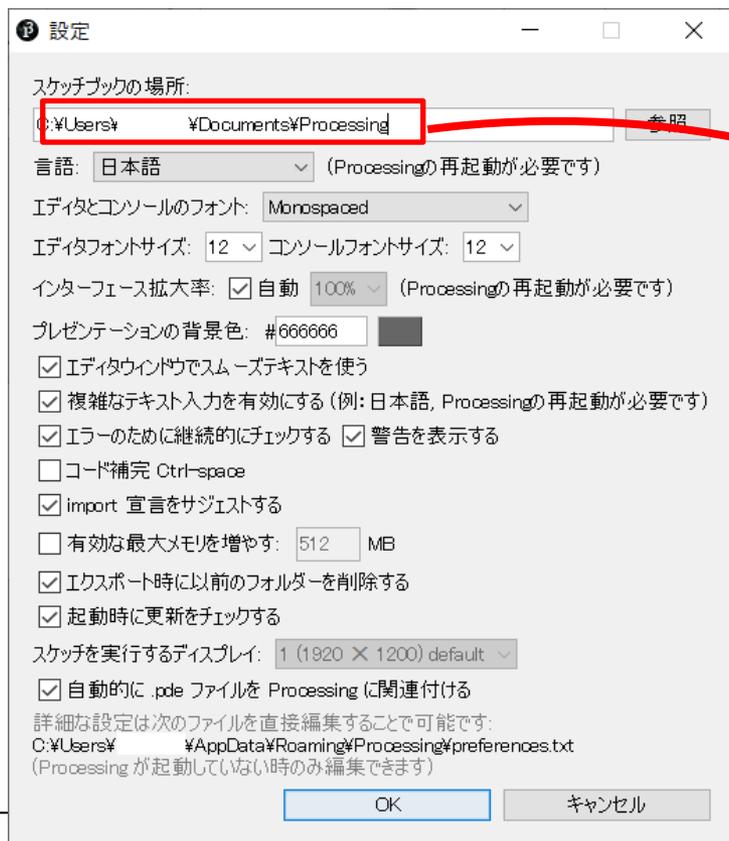
Processing開発環境で、「ファイル」→「設定」をクリックすると、設定画面が表示される



OpenRTM-aist

Processing用ライブラリのインストール

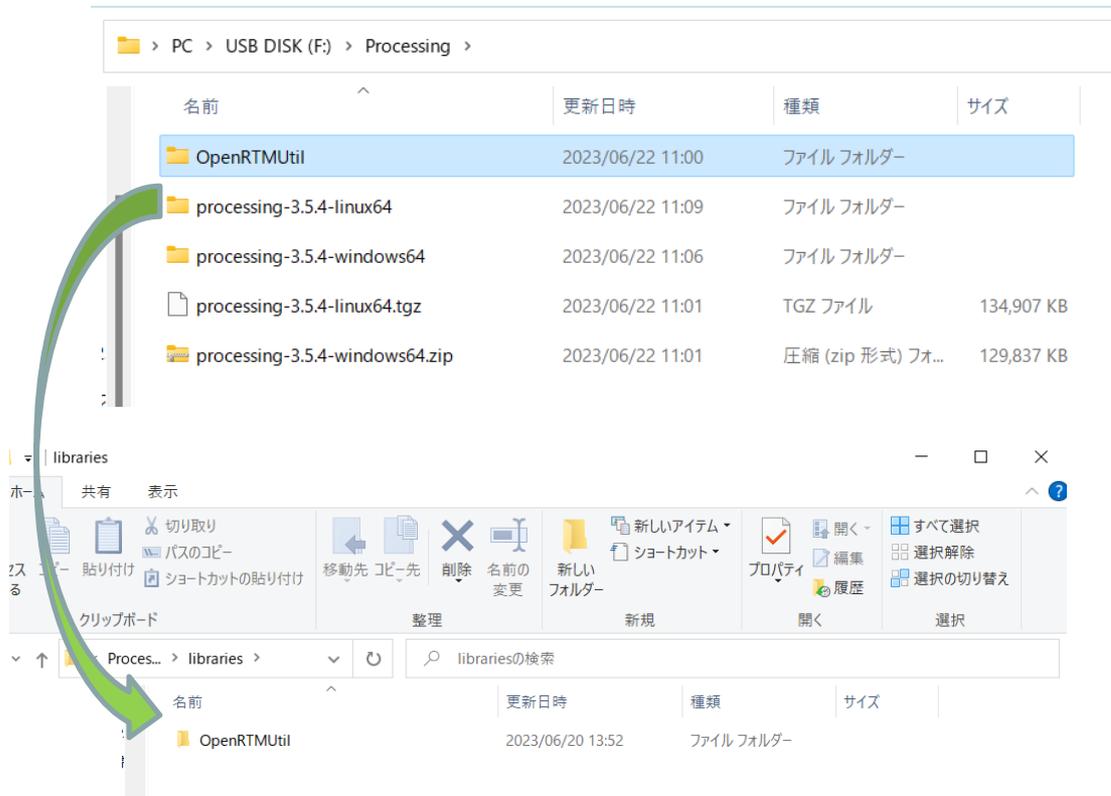
- スケッチブックの場所の **libraries** フォルダをエクスプローラーで開く



OpenRTM-aist

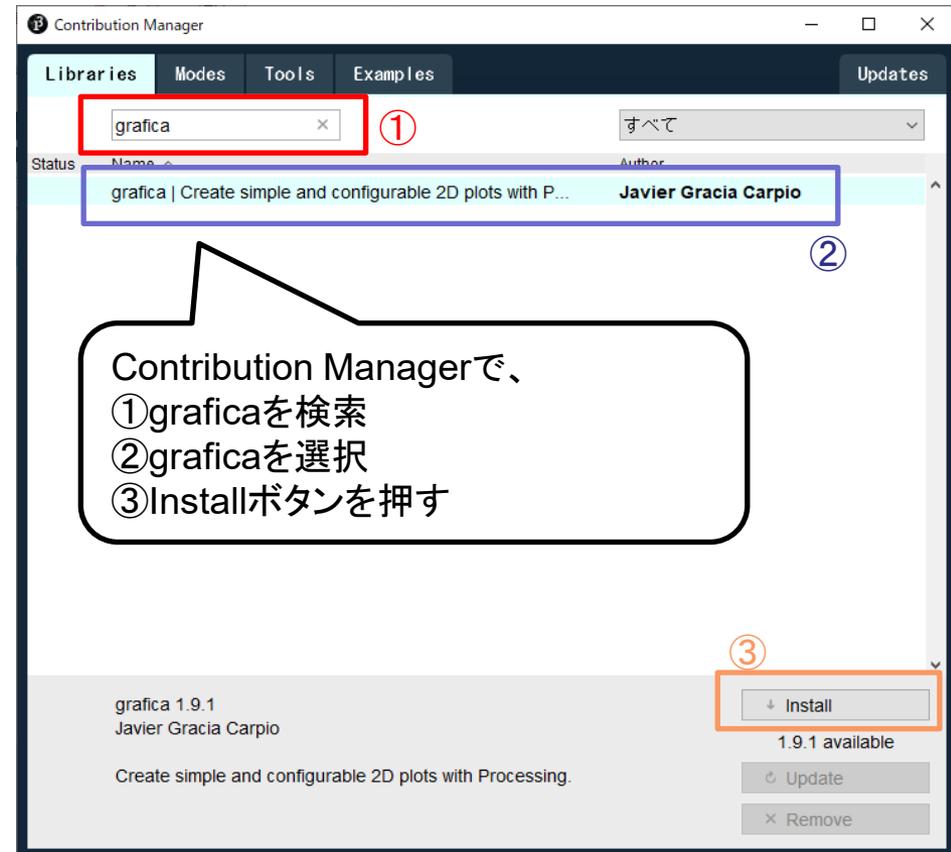
Processing用ライブラリのインストール

- RTMTutorialのProcessing¥OpenRTMUtilフォルダをlibrariesフォルダ内にコピーする



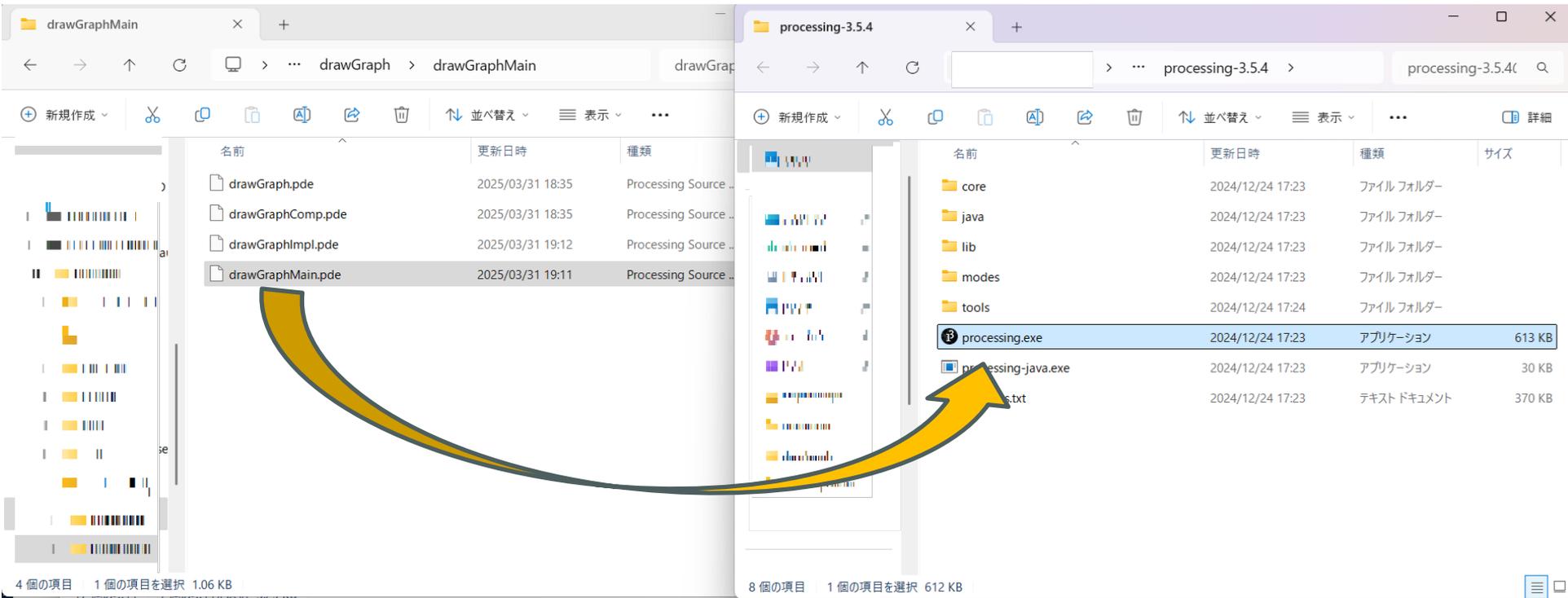
graficaのインストール

- グラフ描画用ライブラリのgraficaをインストールする



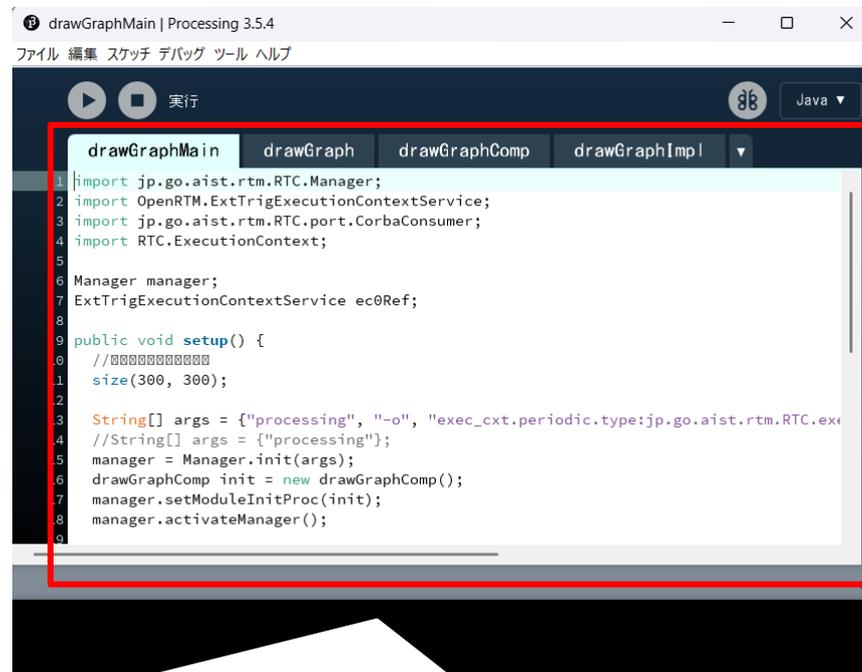
ソースコードの編集

- **drawGraphMain¥drawGraphMain.pde**のファイルを、**processing.exe**にドラッグアンドドロップして開く



ソースコードの編集

- Processing開発環境でコードを編集する



```
drawGraphMain | Processing 3.5.4
ファイル 編集 スケッチ デバッグ ツール ヘルプ

実行

drawGraphMain drawGraph drawGraphComp drawGraphImpl

1 import jp.go.aist.rtm.RTC.Manager;
2 import OpenRTM.ExtTrigExecutionContextService;
3 import jp.go.aist.rtm.RTC.port.CorbaConsumer;
4 import RTC.ExecutionContext;
5
6 Manager manager;
7 ExtTrigExecutionContextService ec0Ref;
8
9 public void setup() {
10 //@@@@@@@@@@@@
11 size(300, 300);
12
13 String[] args = {"processing", "-o", "exec_cxt.periodic.type:jp.go.aist.rtm.RTC.ext
14 //String[] args = {"processing"};
15 manager = Manager.init(args);
16 drawGraphComp init = new drawGraphComp();
17 manager.setModuleInitProc(init);
18 manager.activateManager();
19
```

スライド20、21、22ページに記載のソースコードを入力する。
USBメモリ内のsample¥drawGraph¥drawGraphMainの中身をコピーしてもいいです。

ソースコードの編集

- drawGraphMain.pdeを編集する
 - 画面サイズ、フレームレートを設定する

```
9 public void setup() {  
10     //ウィンドウサイズを設定  
11     size(300, 300); //追加  
12     frameRate(10); //追加
```

Processingでsetup関数は実行開始時に1度だけ呼ばれる。
setup関数内でRTCの生成を行っている。

RT System Editorが固まった場合は、Processingの停止ボタンを押して中断後に、frameRateを変更してみてください。

- drawGraphImpl.pdeを編集する
 - ライブラリのインポート

```
23 import RTC.ReturnCode_t;  
24  
25 import grafica.*; //追加  
26
```

ソースコードの編集

- drawGraphImpl.pdeを編集する
 - 変数の宣言

```
283 protected InPort<TimedPose2D> m_inIn;
284
285 //グラフに描画する点のデータを格納する配列を宣言
286 GPointsArray data; //追加
```

- onActivated関数の編集

```
124 @Override
125 protected ReturnCode_t onActivated(int ec_id) {
126     //配列dataの初期化
127     data = new GPointsArray();
128     return super.onActivated(ec_id);
129 }
```

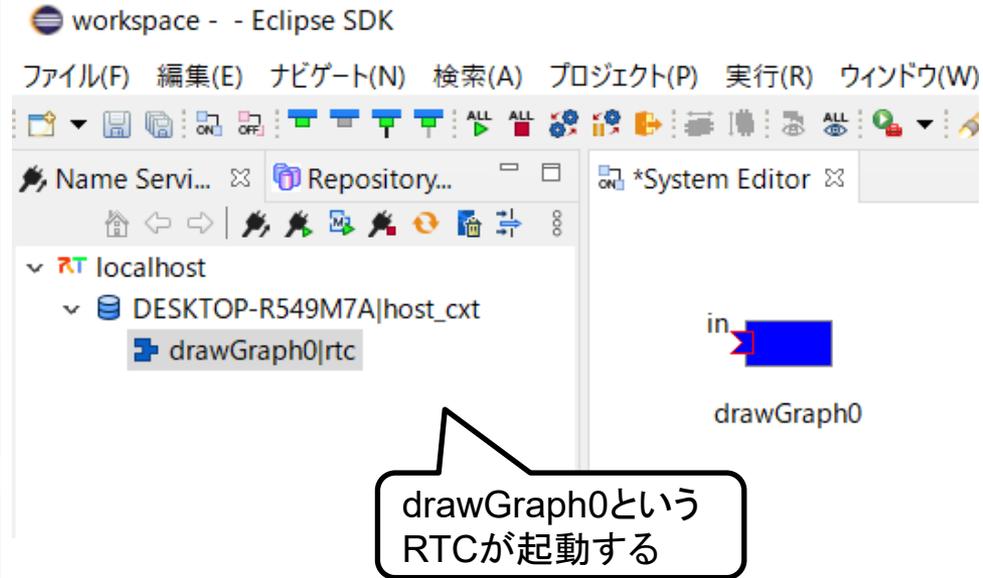
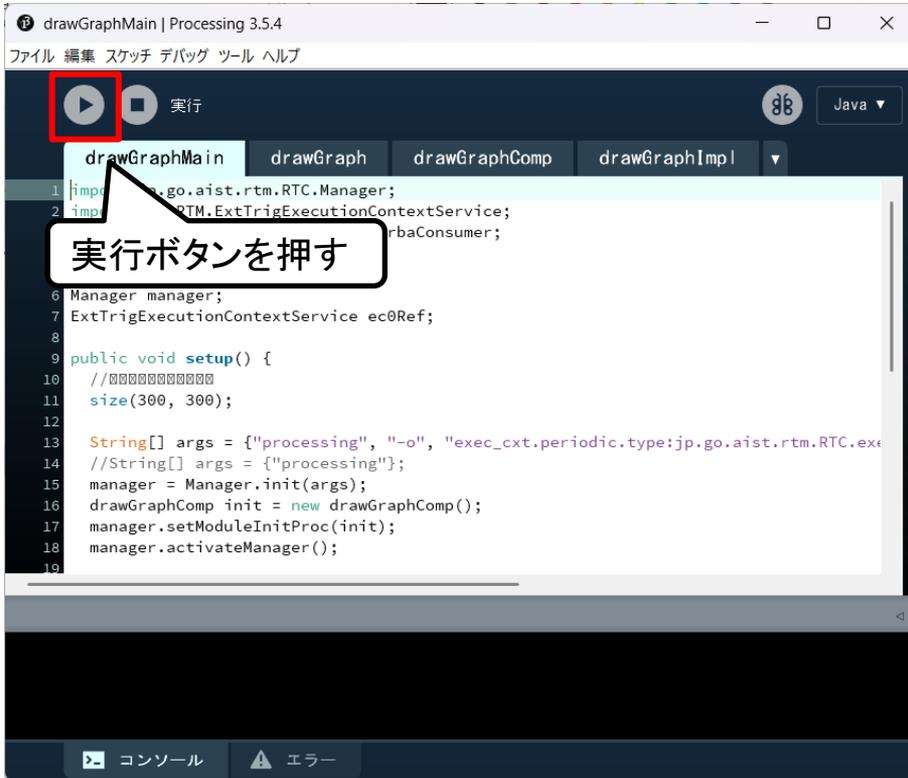
ソースコードの編集

- drawGraphImpl.pdeを編集する
 - onExecute関数の編集

```
156 @Override
157 protected ReturnCode_t onExecute(int ec_id) {
158     //InPortでデータを受信した時の処理
159     if (m_inIn.isNew())
160     {
161         //受信データの読み込み
162         m_inIn.read();
163         //配列dataに取得した位置を追加する
164         data.add((float)m_in.v.data.position.x,
165                 (float)m_in.v.data.position.y);
166
167         //配列の大きさが1000を超えた場合、古いデータは捨てる
168         if (data.getNPoints() > 1000)
169         {
170             data.remove(0);
171         }
172     }
```

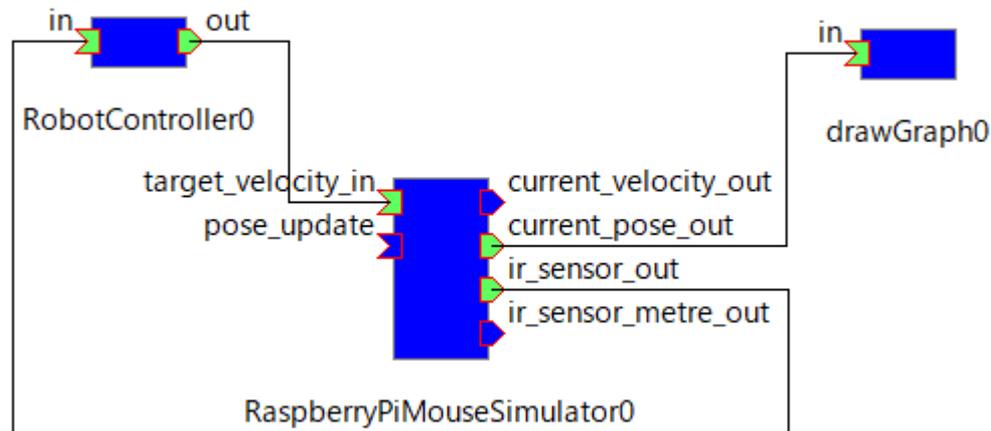
```
173     //グラフをウィンドウの(0,0)から(300,300)の範囲に描画する
174     GPlot plot = new GPlot(m_applet, 0, 0, 300, 300);
175     //グラフの縦軸、横軸の上限、下限を設定する
176     plot.setXLim(-1.0, 1.0);
177     plot.setYLim(-1.0, 1.0);
178     plot.setFixedXLim(true);
179     plot.setFixedYLim(true);
180     //配列dataをグラフに設定する
181     plot.addPoints(data);
182     //グラフの描画を開始する
183     plot.beginDraw();
184     //グラフに外枠、座標、折れ線、縦軸、横軸を描画する
185     plot.drawBox();
186     plot.drawPoints();
187     plot.drawLines();
188     plot.drawXAxis();
189     plot.drawYAxis();
190     //グラフの描画を終了する
191     plot.endDraw();
192     return super.onExecute(ec_id);
193 }
```

RTシステム構築



RTシステム構築

- 以下のようにポートを接続して、RTCをアクティブ化する
 - シミュレータ(RaspberryPiMouseSimulator0)、実機(RaspberryPiMouseRTC0)のどちらでも可



RTシステム実行

