RTコンポーネント操作マニュアル

効率的な
RTシステム開発および運用のための
汎用ビューワコンポーネント

平成20年11月28日版

東京大学生産技術研究所

橋本研究室

佐々木 毅

更新履歴

2008 年 10 月 7 日版	第 1 版。
2008 年 11 月 11 日版	FAQ の記述の見直し。
2008 年 11 月 12 日版	動的入出力ポートを使用したサンプルとして動的入力ポートテストコンポーネント (DynamicInPortTest) を作成。それに伴い DynamicInPortTest に関する記述を追加。
2008 年 11 月 28 日版	3.2.1 節に “gpSeparatePos による区切り位置の指定” を追加。誤字や表現の一部を修正。

目 次

1 本コンポーネント群の概要

1.1 開発の背景

RT コンポーネントによるシステム開発においては、それぞれのコンポーネントの動作確認が必要であり、また運用段階においても、センサデータやその処理結果などの表示、異常発生時の原因の早期発見などが重要となります。したがって、効率的な RT システムの開発および運用には、コンポーネントの出力を視覚化するビューワコンポーネントが有用であると考えられます。ビューワコンポーネントの開発にあたっては、様々なデータを様々な形式で表示できる汎用性や、使用法がわかりやすいことなどが望まれます。そこで、グラフ描画ツールとして広く用いられている gnuplot (gnuplot homepage: http://www.gnuplot.info/) を利用したビューワコンポーネントを開発することといたしました。

コンポーネントの設計指針等につきましては、

> 佐々木毅，橋本秀紀, “効率的な RT システム開発および運用のための汎用ビューワコンポーネント”, 第 9 回計測自動制御学会システムインテグレーション部門講演会, 2008.

に詳細がありますのでそちらもご参照いただけましたら幸いです。

1.2 開発環境

開発環境は以下の通りです。

- OS: Ubuntu Linux 8.04 LTS
- RT ミドルウエア: OpenRTM-aist-0.4.2-RELEASE
- コンパイラ: gcc 4.2.3
- CORBA: omniORB 4.1.1-2
- ACE (The ADAPTIVE Communication Environment): ACE 5.4.7-13
- Eclipse: Eclipse 3.2.0
- Java 実行環境: Sun Java 1.6.0-06

1.3 開発したコンポーネント群

ビューワコンポーネントとそのツールとして、以下のコンポーネント群を開発しました。

- ビューワコンポーネント (GnuplotViewer)
  gnuplot を用いた汎用ビューワ。
- コンソール文字列入力コンポーネント (ConsoleInString)
  コンソールから入力した文字列を OutPort に出力する。

また、GnuplotViewer の使用手順と実用例を示すために、以下のコンポーネント群を開発しました。

- 正弦 · 余弦関数出力コンポーネント (SinCosFunction)
  正弦波、余弦波を出力する。
- レーザレンジファインダコンポーネント (LRFComponent)
  北陽電機株式会社製のレーザレンジファインダ URG-04LX を RT コンポーネント化したもの。

- 移動体トラッキングコンポーネント (SimpleTracker)

  レーザレンジファインダのスキャンデータから移動物体の位置を出力する。

さらに、GnuplotViewer の機能として、入出力ポートの動的追加 · 削除機能を実現しています。この機能の使用方法を示すため、以下のコンポーネントを作成しました。

- 動的入力ポートテストコンポーネント (DynamicInPortTest)

  動的入力ポートのプログラム例および動作テスト用コンポーネント。

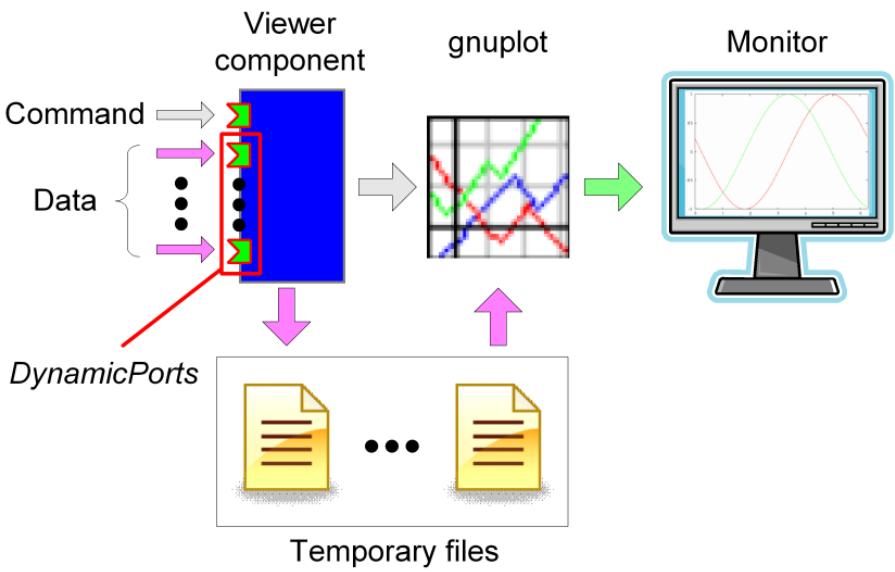
それぞれのコンポーネントの詳細については 2 章を、GnuplotViewer の使用方法については 3 章を参照してください。また、動的入出力ポートの使用方法については 4 章を参照してください。

2　各コンポーネントの説明

2.1　ビューワおよびそのツールコンポーネント

2.1.1　ビューワコンポーネント (GnuplotViewer)

GnuplotViewer は、gnuplot を用いた汎用ビューワです。そのため、このコンポーネントを使用するには gnuplot がインストールされている必要があります。コンポーネントの動作概要を下図に示します。ビューワコンポーネントはまず、Configuration の p{Short, Long, Float, Double}InPortNum の値を設定してアクティブ化すると、対応する入力ポートがその数だけ作成されるとともに gnuplot を起動します。このとき、Configuration の gpOpenOption に gnuplot の起動オプションが与えられていれば、そのオプションにしたがって gnuplot をオープンします。アクティブ化後は他のコンポーネントから入力データを受け取ると、それを gnuplot が扱える形式で一時ファイルに出力します。さらに、コンポーネントに新しいデータが入力される度に Configuration の gpPeriodicExecCommand に記入されたコマンドを実行します。InPort から入力されたデータは、例えば 3 つ目の short 型ポートへのデータであれば “short[2]”、最初の long 型ポートへのデータであれば “long[0]” で参照できます。また InPort の Command から gnuplot へコマンドを送ることも可能です。詳しい使い方に関しては 3 章を参照してください。



- InPort

名称	型	説明
Command	TimedString	gnuplot に送るコマンド。
ShortSeqData	DynamicInPort<TimedShortSeq>	プロットする short 型整数値列データ。
LongSeqData	DynamicInPort<TimedLongSeq>	プロットする long 型整数値列データ。
FloatSeqData	DynamicInPort<TimedFloatSeq>	プロットする float 型実数値列データ。
DoubleSeqData	DynamicInPort<TimedDoubleSeq>	プロットする double 型実数値列データ。

- OutPort

  なし

- Configuration 変数

名称	型	デフォルト値	説明
gpOpenOption	string	*	gnuplot を開く際のオプション。アクティブ状態での変更は無効（反映されない）。スペースを入れたい場合は _を、\を入れたい場合は\\を代わりに使うこと。
gpSeparatePos	string	1	データの区切り位置のリスト。Short, Long, Float, Double のポートの順に各ポートに対しコロン (:) を区切り文字として指定する。詳しくは 3.2.1 節 “gpSeparatePos による区切り位置の指定” を参照。指定されなかったポートの値は 1 となる。
gpPeriodicExecCommand	string	*	gnuplot が周期実行するコマンド。コンポーネントに新しいデータが入力される度に実行される。スペースを入れたい場合は _を、\を入れたい場合は\\を代わりに使うこと。
pShortInPortNum	int	0	TimedShortSeq 型 InPort のポート数。アクティブ状態での変更は無効（反映されない）。
pLongInPortNum	int	0	TimedLongSeq 型 InPort のポート数。アクティブ状態での変更は無効（反映されない）。
pFloatInPortNum	int	0	TimedFloatSeq 型 InPort のポート数。アクティブ状態での変更は無効（反映されない）。
pDoubleInPortNum	int	0	TimedDoubleSeq 型 InPort のポート数。アクティブ状態での変更は無効（反映されない）。

- サービスポート

  なし

2.1.2 コンソール文字列入力コンポーネント (ConsoleInString)

ConsoleInString は、OpenRTM-aist のホームページ (http://www.is.aist.go.jp/rt/OpenRTM-aist/) にサンプルとして取り上げられている ConsoleIn コンポーネントの出力の型を TimedString としたものです。アクティブ化すると入力を促すメッセージがコンソールに表示されます。コンソールから入力を行うと、その文字列が OutPort に出力されます。

- InPort

  なし

- OutPort

名称	型	説明
OutString	TimedString	コンソールから入力された文字列。

- Configuration 変数

  なし

- サービスポート

  なし

2.2 GnuplotViewer の使用手順の説明で用いるコンポーネント

2.2.1 正弦 · 余弦関数出力コンポーネント (SinCosFunction)

SinCosFunction は、正弦波、余弦波を出力するコンポーネントです。アクティブ化を行うと、$t = 0$ に初期化され、$x, \sin(k_s(x-c_st)), \cos(k_c(x-c_ct))$ $(0 \leq x < 2\pi)$ のデータが 1 次元配列として OutPort に出力されます。出力形式は、xData[0], sinData[0], cosData[0], ···, xData[SampleNum−1], sinData[SampleNum−1], cosData[SampleNum−1] の順です。つまり、$0, \sin(k_s(0-c_st)), \cos(k_c(0-c_ct)), \cdots, 2\pi\dfrac{\mathrm{SampleNum}-1}{\mathrm{SampleNum}}, \sin\left(k_s\left(2\pi\dfrac{\mathrm{SampleNum}-1}{\mathrm{SampleNum}} - c_st\right)\right)$, $\cos\left(k_c\left(2\pi\dfrac{\mathrm{SampleNum}-1}{\mathrm{SampleNum}} - c_ct\right)\right)$ です。

- InPort

  なし

- OutPort

名称	型	説明
SinCosData	TimedDoubleSeq	$x, \sin(k_s(x-c_st)), \cos(k_c(x-c_ct))$ の値。配列長は 3×SampleNum。

- Configuration 変数

名称	型	デフォルト値	説明
sWaveNum	double	1.0	正弦波の波数 $k_s(= 2\pi/\lambda_s)$。単位は rad/m。
sPhaseVel	double	1.0	正弦波の位相速度 $c_s(= \omega_s/k_s)$。単位は m/s。
cWaveNum	double	1.0	余弦波の波数 $k_c(= 2\pi/\lambda_c)$。単位は rad/m。
cPhaseVel	double	1.0	余弦波の位相速度 $c_c(= \omega_c/k_c)$。単位は m/s。
SampleNum	int	200	データ数。[0,2π) の範囲をこのデータ数でサンプルする。

- サービスポート

  なし

2.3 GnuplotViewer の実用例で用いるコンポーネント

2.2.1 レーザレンジファインダコンポーネント (LRFComponent)

LRFComponent は北陽電機株式会社製のレーザレンジファインダ URG-04LX を RT コンポーネント化したものです。アクティブ化すると距離データと計測範囲の情報を OutPort に出力します。距離データの単位は URG04-LX の出力形式と同じ mm です。

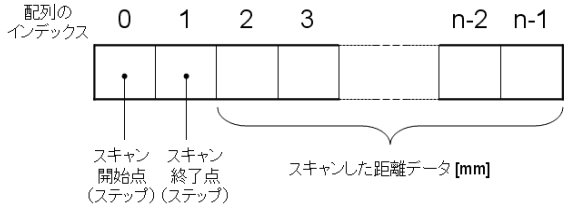
また、測定パラメータを設定するための Configuration 変数と、最大計測距離などのセンサ情報を他のコンポーネントに提供するためのサービスポートが用意されています。

- InPort

  なし

- OutPort

名称	型	説明
ScanData	TimedShortSeq	センサが獲得した距離データ。単位は mm。最初の 2 つのデータは計測範囲の情報（下図）。ステップについては URG の仕様書を参照。



- Configuration 変数

名称	型	デフォルト値	説明
DeviceName	string	/dev/ttyACM0	デバイスが接続されたポートの名前。詳しくは URG の仕様書を参照。アクティブ状態での変更は無効（反映されない）。
ScanRate	int	19200	通信速度。単位は bps。詳しくは URG の仕様書を参照。アクティブ状態での変更は無効（反映されない）。
ScanStart	int	0	スキャンの開始点（ステップ）。有効範囲は [0,768]。詳しくは URG の仕様書を参照。
ScanEnd	int	768	スキャンの終了点（ステップ）。有効範囲は [0,768] で、ScanStart よりも大きい値。詳しくは URG の仕様書を参照。
ScanStep	int	1	ScanStart と ScanEnd の間のスキャンのステップ数。有効範囲は [1,99]。詳しくは URG の仕様書を参照。

- サービスポート

  InfoPort (provider)

関数名	引数	戻り値の型	説明
getMaxRange	なし	short	センサの最大計測可能距離を返す。単位は mm。
getMinAngle	なし	short	最小ステップ（ステップ 0）に対応する角度を返す。単位は deg。
getMaxAngle	なし	short	最大ステップに対応する角度を返す。単位は deg。
getMaxStep	なし	short	最大ステップを返す。

2.3.2 移動体トラッキングコンポーネント (SimpleTracker)

SimpleTracker は、レーザレンジファインダのスキャンデータから移動物体の位置を出力するものです。アクティブ状態のときに LRFComponent から観測データを受け取ると、まず背景情報（静止物体領域の情報）を獲得します。そのため、アクティブ化を行うときは観測領域に人間などの移動物体が入らないように注意してください。背景情報の獲得後は、受信した距離データから背景と異なる部分を見つけ出し、その物体のスキャン平面上

での位置を出力します。観測領域内に複数の移動物体がある場合には最大の大きさを持つ物体の位置を出力します。出力の単位は m です。（URG04-LX の出力形式が mm ですので、移動体位置の出力に際して 1/1000 倍しています。したがって、cm や m で出力するセンサを接続した場合は m にはなりません。）非アクティブ化すると背景情報等は失われ、アクティブ化すると再び背景情報の獲得から処理が行われます。

なお、このコンポーネントでは、アクティブ状態で InPort の ScanData のデータ長、StartPoint および EndPoint の値が変化した場合、移動体トラッキング処理を中断します。LRFComponent の測定パラメータを変化させたい場合には、このコンポーネントを非アクティブ状態にしてから行ってください。

- InPort

名称	型	説明
ScanData	TimedShortSeq	LRFComponent が出力した距離データ。単位は mm を想定。最初の 2 つのデータは計測範囲の情報。詳しくは 2.3.1 節レーザレンジファインダコンポーネント (LRFComponent) の OutPort を参照。

- OutPort

名称	型	説明
Position2D	TimedDoubleSeq	観測領域内で最大の大きさをもつ移動物体の 2 次元位置 (x, y)。単位は m を想定（Scan Data の 1/1000）。

- Configuration 変数

  なし

- サービスポート

  InfoPort (consumer)

  提供される関数については 2.3.1 節レーザレンジファインダコンポーネント (LRFComponent) のサービスポートを参照。

2.4 動的入出力ポートのサンプルコンポーネント

2.4.1 動的入力ポートテストコンポーネント (DynamicInPortTest)

DynamicInPortTest は、動的入力ポートのプログラム例の提示と動作テスト用のコンポーネントです。アクティブ状態で入力ポートの InPortManip に文字列が入力されると、それに応じて動的入力ポートの追加と削除を行います。入力ポートの InPortManip に入力された文字列が数字のみの場合、その数字の番号の入力ポートがあればそのポートを削除します。入力ポートの InPortManip に入力された文字列に数字以外の文字が含まれていた場合、その文字数だけポートを追加します。動的入力ポート StringData に入力があった場合、そのポートの名前、番号と文字列をコンソールに出力します。非アクティブ化するとすべての動的入力ポートを削除します。

- InPort

名称	型	説明
InPortManip	TimedString	動的入力ポートの追加・削除コマンド。
StringData	DynamicInPort<TimedString>	コンソールに出力する文字列データ。

- OutPort

  なし

- Configuration 変数

  なし

- サービスポート

  なし

3 GnuplotViewer の使用方法

3.1 RTSystemEditor や RtcLink によるシステム構築の手順

RTSystemEditor や RTCLink を用いたシステム構築は通常、以下の手順で行われます。

RTSystemEditor や RTCLink を用いたシステム構築の手順

1. ネームサーバの起動
2. RTSystemEditor または RTCLink の起動
3. ネームサーバへの接続
4. コンポーネントの起動
5. システムの構築と実行

これらの手順はどのようなシステムでも共通ですので、操作方法は OpenRTM-aist のホームページ (http://www.is.aist.go.jp/rt/OpenRTM-aist/) を参照してください。以下の節では手順 5 に関して説明します。

3.2 GnuplotViewer の使用手順

GnuplotViewer の使用手順をまとめると以下のようになります。

GnuplotViewer の使用手順

1. Configuration を設定する
   (a) p{Short, Long, Float, Double}InPortNum から使用するポートの数を設定する
   (b) gpSeparatePos から 1 次元ベクトルデータの区切り位置を設定する
   (c) gpPeriodicExecCommand からプロットコマンドを設定する
2. コンポーネントをアクティブ化する
3. 出力を表示したいコンポーネントと接続する
4. 適宜 Command ポートからプロット設定などのコマンドを入力する

手順 1(b) および 1(c) は手順 2 のアクティブ化の後に行うことも可能です。以下の節では具体的な例を挙げながら GnuplotViewer の使い方を説明します。

3.2.1 基本的な使い方 – 1 次元配列データのプロット機能の利用 –

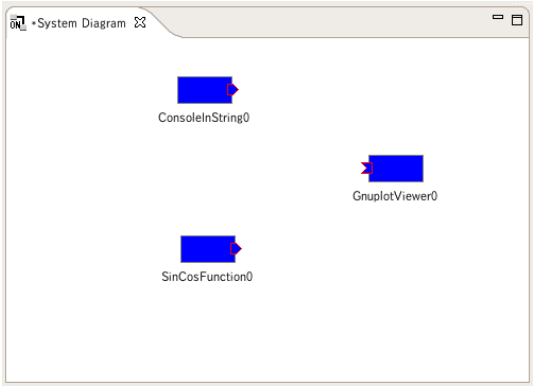
まず、GnuplotViewer の最も基本的な使用方法である 1 次元配列データのプロット機能について説明します。使用するコンポーネントの種類とその数は

- ビューワコンポーネント (GnuplotViewer) · · · · · · 1 つ
- コンソール文字列入力コンポーネント (ConsoleInString) · · · · · · 1 つ
- 正弦 · 余弦関数出力コンポーネント (SinCosFunction) · · · · · · 1 つ

の計 3 つです。以下に使用手順を示します。

(1) コンポーネントの配置

コンポーネントを起動し、システムエディタ上に GnuplotViewer, ConsoleInString, SinCosFunction を配置してください。



(2) Configuration 変数の設定

GnuplotViewer を選択し、ConfigurationView からパラメータを設定します。ここでは、SinCosFunction の OutPort から出力されるデータのうち、x の値と sin() の値をプロットすることを考えます。

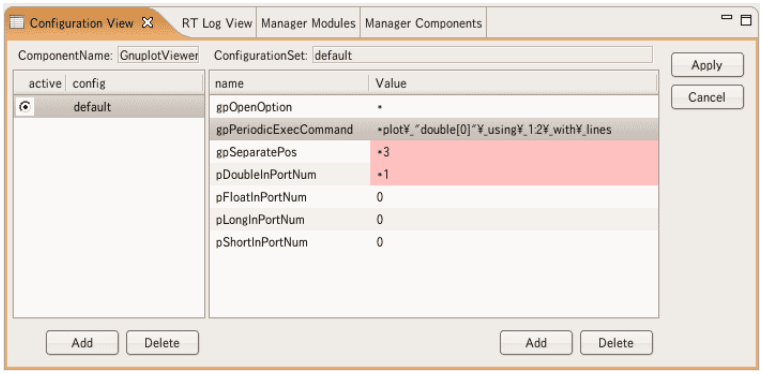
まず、使用するポートの種類と数の設定を行います。SinCosFunction の OutPort は TimedDoubleSeq 型なので、pDoubleInPortNum の値を 1 にします。次に、1 次元データ配列の区切り位置を設定します。SinCosFunction の OutPort は xData[0], sinData[0], cosData[0], xData[1], sinData[1], cosData[1], · · · というように 3 つごとに同種のデータを出力するので gpSeparatePos の値を 3 にします。なお、gpSeparatePos の指定についての詳細は次頁の囲み枠内 “gpSeparatePos による区切り位置の指定” を参照してください。続いて gnuplot が周期実行するコマンドを指定します。ここでは、唯一の（0 番目の）double 型のポートの 1 列目 (x) と 2 列目 (sin()) の値をプロットするので、

plot ”double[0]” using 1:2 with lines

というコマンドを実行させたいと思います（上記のコマンドは p ”double[0]” u 1:2 w l などと省略することも可能です)。しかし、現在の RTSystemEditor や RtcLink の ConfigurationView ではスペースを扱えない（スペース以降の文字列は無視される）ので、コマンドにスペースを入れたい場合は_を、\を入れたい場合は\\を代わりに使用します。したがって、実際に gpPeriodicExecCommand に指定するコマンドは以下のようになります。

plot_”double[0]”_using_1:2_with_lines

また、gpOpenOption は gnuplot を開く際のオプションですが、ここでは特に指定する必要はないと思います。なお、ここではすべての設定をアクティブ化前に行いましたが、前述の通り gpSeparatePos と gpPeriodicExecCommand はアクティブ状態にしてからの設定も可能です。設定が終了したら変更を確定するために Apply をクリックします。



gpSeparatePos による区切り位置の指定

GnuplotViewer の Configuration 変数である gpSeparatePos は、1 次元配列データの区切り位置を指定し、2 次元プロットや 3 次元プロットを簡単に行えるようにするための値です。例えば、元の 1 次元配列データが 1 2 3 4 5 6 7 8 9 10 であるとき、gpSeparatePos の値が 3 であれば、このデータは

1 列目	2 列目	3 列目
1	2	3
4	5	6
7	8	9
10		

のように整列しているものと見なされます。これにより、gnuplot の using オプションで例えば 2:3 のように指定すれば、2 列目と 3 列目のデータの 2 次元プロットなどが可能となります。

また、複数ポートがある場合には、Short, Long, Float, Double のポートの順にコロン (:) を区切り文字としてそれぞれの区切り位置を指定します。例えば、ShortSeq 型のポートが 2 つ、FloatSeq 型のポートが 1 つあり、それぞれ区切り位置を 1,2,3 としたいときには 1:2:3 と入力してください。

(3) コンポーネントのアクティブ化

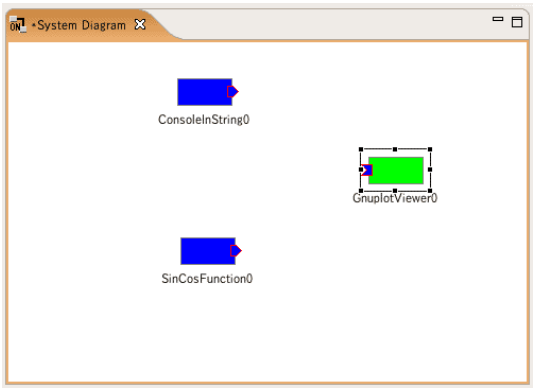
GnuplotViewer を選択し、右クリックメニューの Activate をクリックしてコンポーネントをアクティブ化します。GnuplotViewer のコンソールに

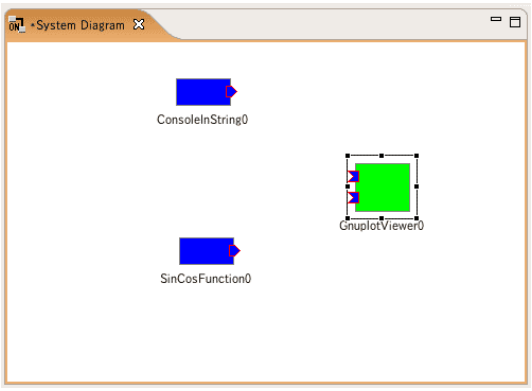
```
Set separate position list.
3
Set command: plot ”double[0]” using 1:2 with lines
```

と表示されれば gnuplot のオープンとプロット設定の初期化は完了です*。

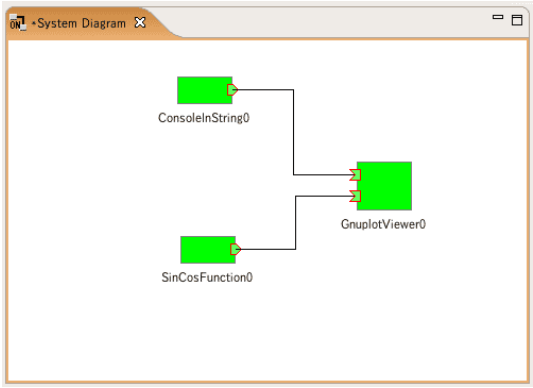
このとき、システムエディタの表示内容の更新が行われないため、システムエディタでの見た目上は入力ポートが追加されていません。GnuplotViewer をドラッグして移動させるなどすることで表示内容が更新され、追加したポートが現れます。

*エラーが発生する場合には 5 章の内容をご確認ください。





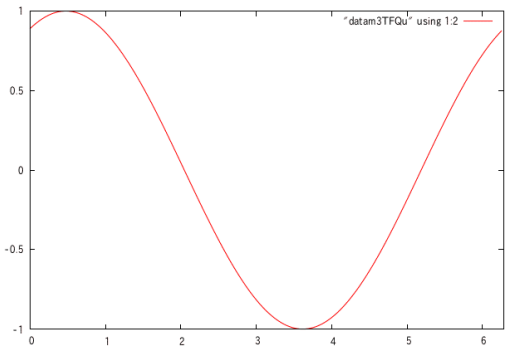
ConsoleInString の OutString ポートと GnuplotViewer の Command ポート、SinCosFunction の SinCosData ポートと GnuplotViewer の DoubleSeqData ポートをそれぞれ接続し、ConsoleInString と SinCosFunction もアクティブ化します。



すると、SinCosFunction からデータが送られてきたことによって GnuplotViewer の gpPeriodicExecCommand が実行され、gnuplot のウインドウ内に x 軸方向に移動する正弦波が表示されます。なお、ConsoleInString のコンソールから

```
set xrange [0:2*pi]
set yrange [-1:1]
```

と入力して x 軸、y 軸の表示範囲を変更するとより見やすくなります。



なお、ConsoleInString から送るコマンドが長い場合や、gpPeriodicExecCommand に設定したいコマンドが複数行にわたる場合などは、そのコマンドを予めファイルに保存しておき、gnuplot の load コマンドや call コマンド（こちらだと引数を与えられる）によって呼び出しを行うと便利です。

この他にも、ConsoleInString から送るコマンドや各コンポーネントの Configuration の設定を変更することで様々な表示が可能です。以下に例を示します。なお、ここでは GnuplotViewer の gpPeriodicExecCommand はスペースの都合上、2 行にわたってしまっているものもありますが、実際には通常通り 1 行で（改行を無視して）入力を行ってください。

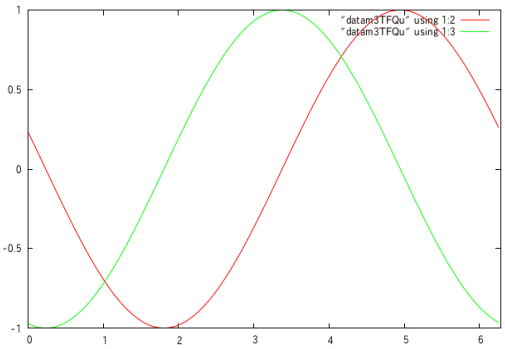
2 つの系列を同時に表示（第 2 系列と第 3 系列を同じグラフ上に表示 $y = \sin(x), y = \cos(x)$）

- ConsoleInString から送るコマンド

  set xrange [0:2*pi]
  set yrange [-1:1]

- GnuplotViewer の gpPeriodicExecCommand

  p_”double[0]”_u_1:2_w_l,
  ”double[0]”_u_1:3_w_l



演算子の適用、系列同士の演算（第 2 系列と第 3 系列の積を 2 倍して表示 $y = 2\sin(x)\cos(x) = \sin(2x)$）

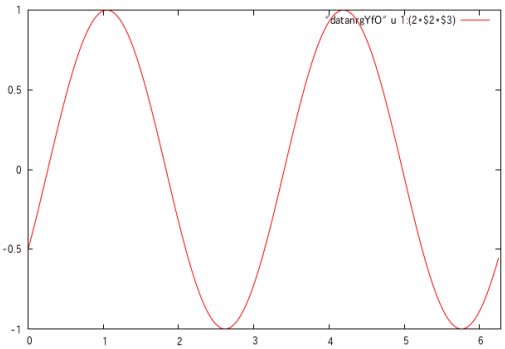
- ConsoleInString から送るコマンド

  set xrange [0:2*pi]
  set yrange [-1:1]

- GnuplotViewer の gpPeriodicExecCommand

  p_”double[0]”_u_1:(2*$2*$3)_w_l

  * この他にも対数 (log) や正の平方根 (sqrt) などが使用できます。



極座標プロット（第 2 系列を極座標表示 $r = \sin(5t)$）

- ConsoleInString から送るコマンド

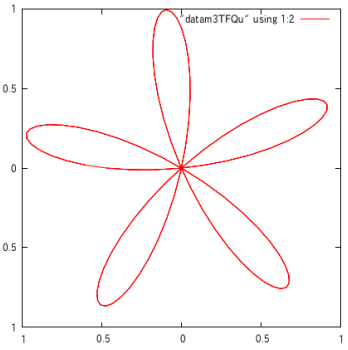
  set size square
  set xrange [-1:1]
  set yrange [-1:1]
  set polar

- SinCosFunction の Configuration

  sWaveNum 5

- GnuplotViewer の gpPeriodicExecCommand

  p_”double[0]”_u_1:2_w_l



3.2.2　発展的な使い方 – テキスト処理ツールの利用 –

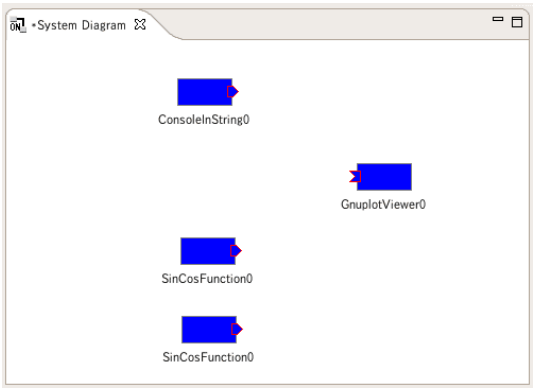
テキスト処理を行える Unix コマンドや、sed、AWK などのツールを用いることで、1 次元配列データのプロット機能だけでは困難な様々なプロット機能を実現できます。ここではその例として、2 つの異なるコンポーネントから得られた出力から 1 系列ずつをそれぞれ x,y の値として 2 次元プロットを行います。使用するコンポーネントの種類とその数は

- ビューワコンポーネント (GnuplotViewer) · · · · · · 1 つ
- コンソール文字列入力コンポーネント (ConsoleInString) · · · · · · 1 つ
- 正弦 · 余弦関数出力コンポーネント (SinCosFunction) · · · · · · 2 つ

の計 4 つです。以下に使用手順を示します。

(1) コンポーネントの配置

コンポーネントを起動し、システムエディタ上に GnuplotViewer, ConsoleInString, SinCosFunction を配置してください。



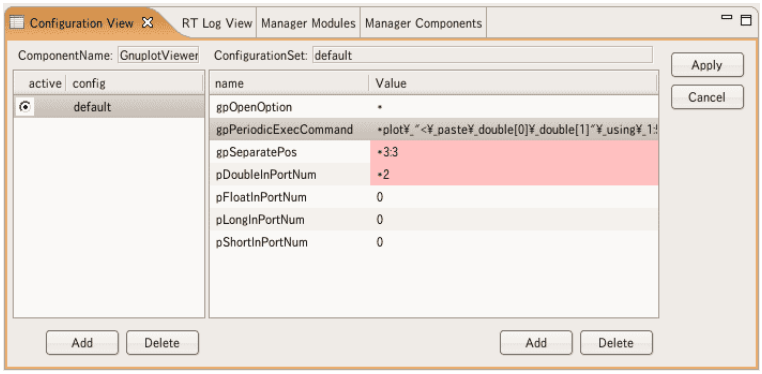
(2) Configuration 変数の設定

GnuplotViewer を選択し、ConfigurationView からパラメータを設定します。ここでは、Unix コマンドの paste を用いて 2 つの異なるポートから入力されたデータを結合し、1 つめの SinCosFunction の OutPort から出力されるデータの x の値と、2 つめの SinCosFunction の OutPort から出力されるデータの sin() の値をプロットすることを考えます。

まず、使用するポートの種類と数の設定を行います。SinCosFunction の OutPort は TimedDoubleSeq 型なので、pDoubleInPortNum の値を 2 にします。次に、1 次元データ配列の区切り位置を設定します。SinCosFunction の OutPort は xData[0], sinData[0], cosData[0], xData[1], sinData[1], cosData[1], · · · というように 3 つごとに同種のデータを出力し、それが 2 つあるので gpSeparatePos の値を 3:3 にします。続いて gnuplot が周期実行するコマンドを指定します。ここでは、2 つの（0 番目と 1 番目の）double 型のポートのそれぞれ 1 列目 (x) と 2 列目 (sin()) の値を paste コマンドにより結合してプロットするので、gpPeriodicExecCommand に指定するコマンドは以下のようになります。

plot_”<_paste_double[0]_double[1]”_using_1:5_with_lines

ここで、using の設定が 1:5 なのは、結合を行うと 1 つめのコンポーネントの 3 列が挿入されることにより 2 つめのコンポーネントの 2 列目が 3 + 2 = 5 列目になるためです。また、gpOpenOption は gnuplot を開く際のオプションですが、ここでは特に指定する必要はないと思います。なお、ここではすべての設定をアクティブ化前に行いましたが、前述の通り gpSeparatePos と gpPeriodicExecCommand はアクティブ状態にしてからの設定も可能です。設定が終了したら変更を確定するために Apply をクリックします。



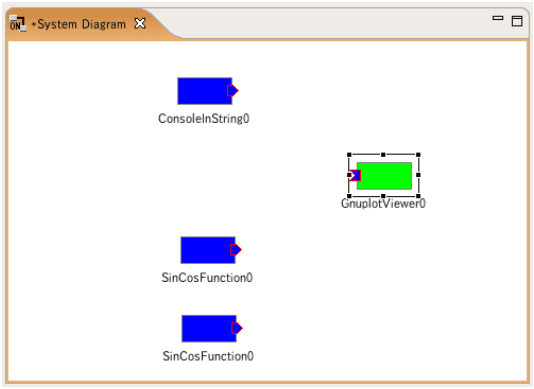
(3) コンポーネントのアクティブ化

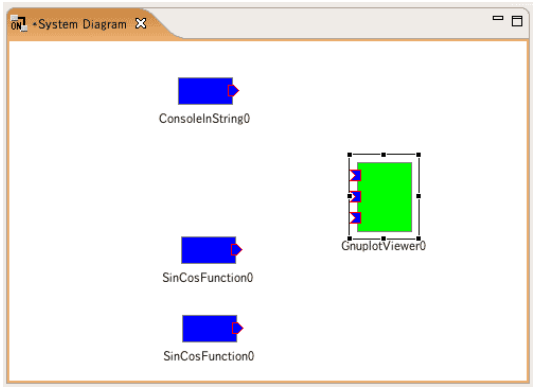
GnuplotViewer を選択し、右クリックメニューの Activate をクリックしてコンポーネントをアクティブ化します。GnuplotViewer のコンソールに

Set separate position list.
3:3
Set command: plot ”< paste double[0] double[1]” using 1:5 with lines

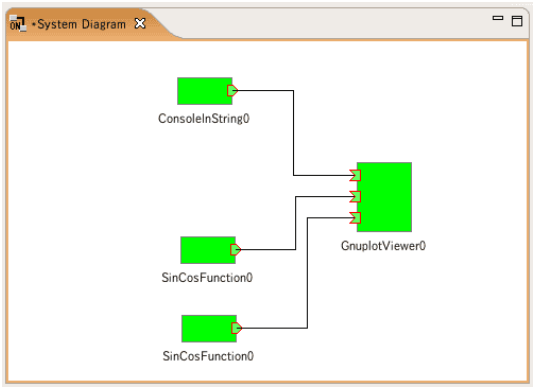
と表示されれば gnuplot のオープンとプロット設定の初期化は完了です。

このとき、システムエディタの表示内容の更新が行われないため、システムエディタでの見た目上は入力ポートが追加されていません。GnuplotViewer をドラッグして移動させるなどすることで表示内容が更新され、追加したポートが現れます。





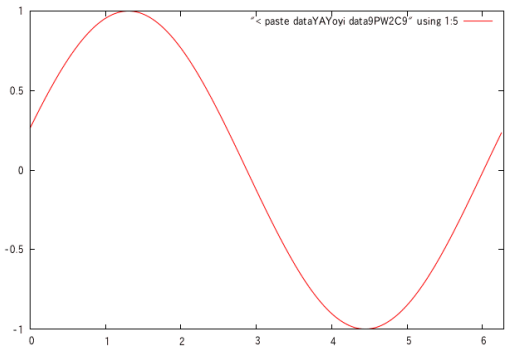
ConsoleInString の OutString ポートと GnuplotViewer の Command ポート、SinCosFunction の SinCosData ポートと GnuplotViewer の DoubleSeqData ポートをそれぞれ接続し、ConsoleInString と 2 つの SinCosFunction もアクティブ化します。



すると、2 つの SinCosFunction からデータが送られてきたことによって GnuplotViewer の gpPeriodicExecCommand が実行され、前節と同様 gnuplot のウインドウ内に x 軸方向に移動する正弦波が表示されます。なお、ConsoleInString のコンソールから

```
set xrange [0:2*pi]
set yrange [-1:1]
```

と入力して x 軸、y 軸の表示範囲を変更するとより見やすくなります。



3.3 GnuplotViewer の実用例

本節では、北陽電機株式会社製のレーザレンジファインダ URG-04LX を用いた人間トラッキングを取り上げ、GnuplotViewer の実用例を示します。使用するコンポーネントの種類とその数は

- ビューワコンポーネント (GnuplotViewer) · · · · · · 1 つ
- コンソール文字列入力コンポーネント (ConsoleInString) · · · · · · 1 つ
- レーザレンジファインダコンポーネント (LRFComponent) · · · · · · 1 つ
- 移動体トラッキングコンポーネント (SimpleTracker) · · · · · · 1 つ

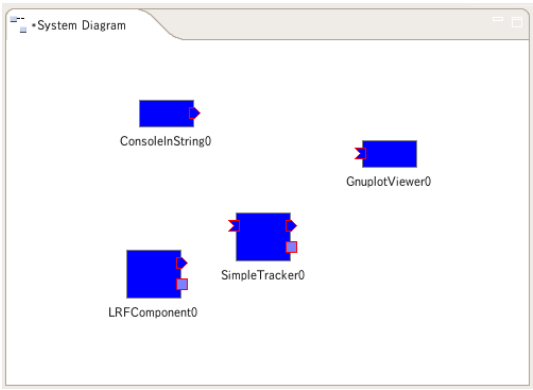
の計 4 つです。

まず LRFComponent が出力する距離データをプロットする例を示し、続いて SimpleTracker が出力する位置データをプロットする例を示します。

3.3.1 距離データの表示

(1) コンポーネントの配置

コンポーネントを起動し、システムエディタ上に GnuplotViewer, ConsoleInString, LRFComponent, SimpleTracker を配置してください。



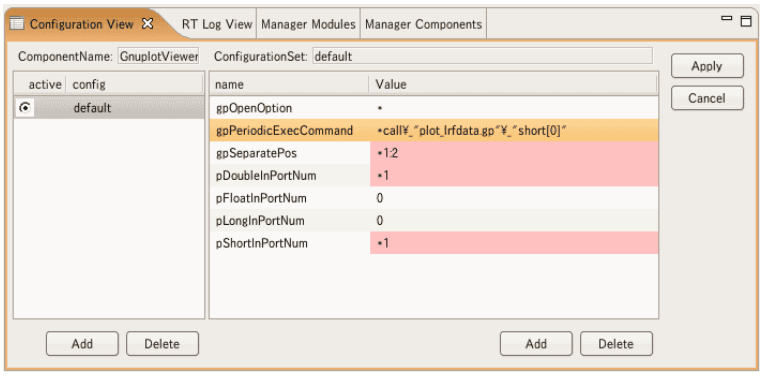
(2) Configuration 変数の設定

GnuplotViewer を選択し、ConfigurationView からパラメータを設定します。

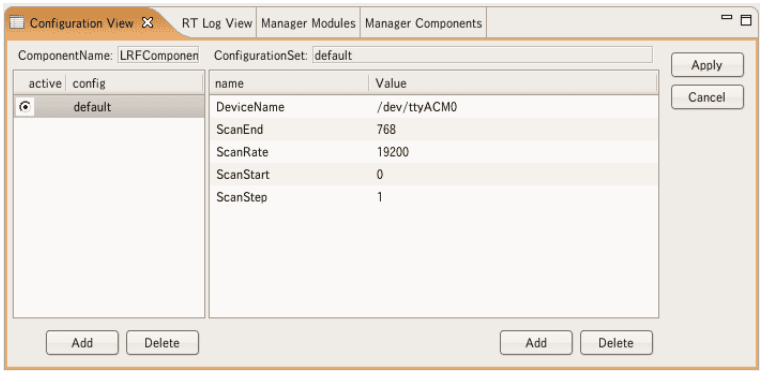
まず、使用するポートの種類と数の設定を行います。ここでは、TimedShortSeq 型の OutPort をもつ LRFComponent と TimedDoubleSeq 型の OutPort をもつ SimpleTracker の出力データを表示させるため、pShortInPortNum と pDoubleInPortNum の値を 1 にします。次に、1 次元データ配列の区切り位置を設定します。LRFComponent の OutPort は距離データのみを出力し、SimpleTracker の出力は xData[0], yData[0], xData[1], yData[1], ··· というように 2 つごとに同種のデータを出力するので gpSeparatePos の値を 1:2 にします。続いて gnuplot が周期実行するコマンドを指定します。前述のとおり最初は LRFComponent が出力する距離データをプロットします。そのための AWK を用いたプロット設定ファイルがすでに用意されていますので、それを call します。gpPeriodicExecCommand に以下のコマンドを指定してください。

call_"plot_lrfdata.gp"_"short[0]"

また、gpOpenOption は gnuplot を開く際のオプションですが、ここでは特に指定する必要はないと思います。なお、ここではすべての設定をアクティブ化前に行いましたが、前述の通り gpSeparatePos と gpPeriodicExecCommand はアクティブ状態にしてからの設定も可能です。設定が終了したら変更を確定するために Apply をクリックします。



さらに、LRFComponent を選択し、ConfigurationView から測定パラメータ等を設定してください。通常はデフォルトのままでよいと思いますが、必要に応じて DeviceName を変更してください。



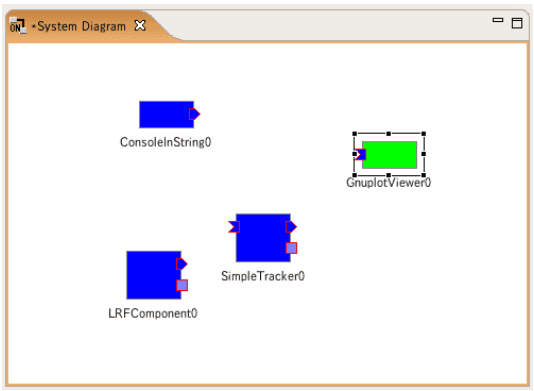
(3) コンポーネントのアクティブ化

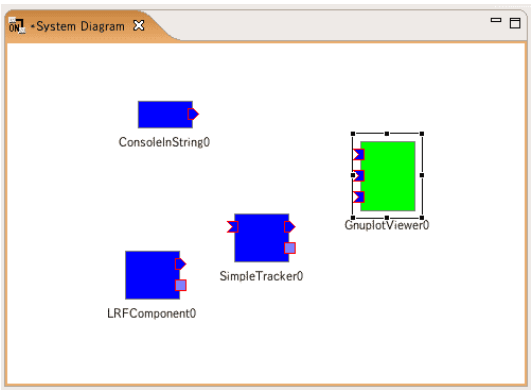
GnuplotViewer を選択し、右クリックメニューの Activate をクリックしてコンポーネントをアクティブ化します。GnuplotViewer のコンソールに

Set separate position list.
1:2
Set command: call "plot_lrfdata.gp" "short[0]"

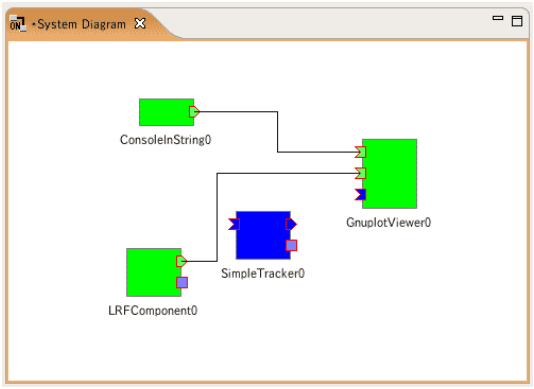
と表示されれば gnuplot のオープンとプロット設定の初期化は完了です。

このとき、システムエディタの表示内容の更新が行われないため、システムエディタでの見た目上は入力ポートが追加されていません。GnuplotViewer をドラッグして移動させるなどすることで表示内容が更新され、追加したポートが現れます。

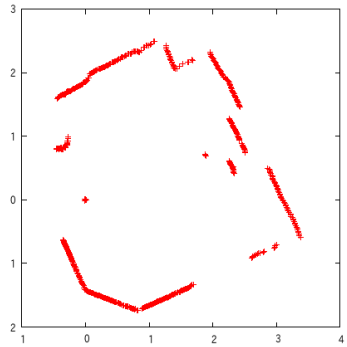




ConsoleInString の OutString ポートと GnuplotViewer の Command ポート、LRFComponent の ScanData ポートと GnuplotViewer の ShortSeqData ポートをそれぞれ接続し、ConsoleInString と LRFComponent もアクティブ化します。



すると、LRFComponent からデータが送られてきたことによって GnuplotViewer の gpPeriodicExecCommand が実行され、gnuplot のウインドウ内にスキャン結果が表示されます。なお、ConsoleInString のコンソールから x 軸、y 軸の表示範囲を変更するとより見やすくなります。

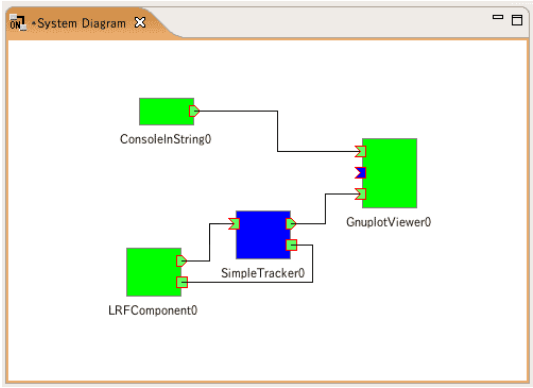


3.3.2 位置データの表示

前節に続き、今度は SimpleTracker が出力する位置データをプロットします。

(1) コンポーネントの接続の変更

LRFComponent の ScanData ポートの接続を SimpleTracker の ScanData ポートに変更し、またこれらのコンポーネントのサービスポートを接続します。さらに、SimpleTracker の Position2D ポートと GnuplotViewer の DoubleSeqData ポートを接続します。

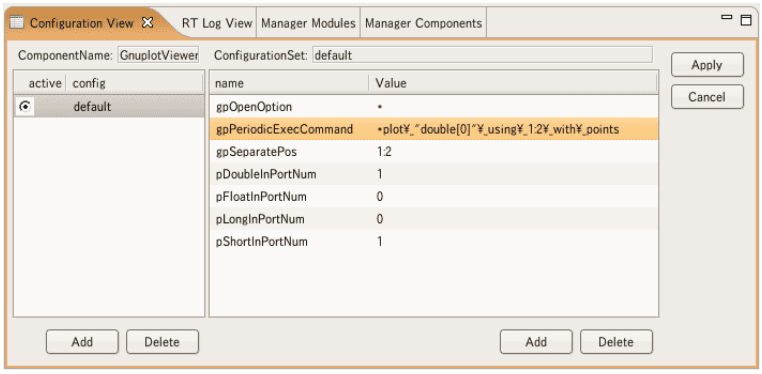


(2) Configuration 変数の設定変更

GnuplotViewer を選択し、ConfigurationView から gnuplot が周期実行するコマンドを変更します。gpPeriodicExecCommand に以下のコマンドを指定してください。

plot_"double[0]"_using_1:2_with_points

設定が終了したら変更を確定するために Apply をクリックします。



GnuplotViewer のコンソールに

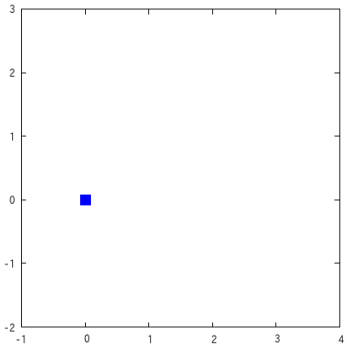
Set command: plot "double[0]" using 1:2 with points

と表示されれば gnuplot の周期実行コマンドの変更は完了です。

さらに、ここではトラッキング結果の履歴を点列として表示させることを考えます。そのための設定ファイルが用意されていますので、ConsoleInString から以下のコマンドを送信し、ロードを行います。

load "tracking_lrf.conf"

GnuplotViewer のコンソールにも同様のコマンドが表示されれば送信は完了です。するとレーザの位置 (0,0) を示したグラフが表示されます。なお、プロット範囲等は先ほどの距離データの表示結果を基に tracking_lrf.conf の内容を編集し、変更を行っておくとより見やすくなります。編集を行ったらファイルを保存後もう一度ロードを行ってください。



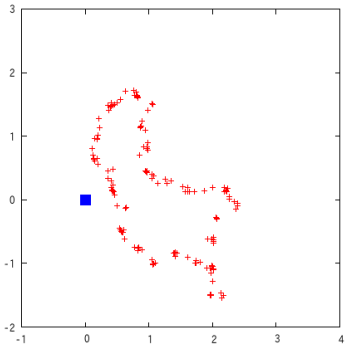
(3) コンポーネントのアクティブ化

SimpleTracker を選択し、右クリックメニューの Activate をクリックしてコンポーネントをアクティブ化します。SimpleTracker は、最初に背景情報（静止物体領域の情報）を獲得するので、アクティブ化を行うときは観測領域に人間などの移動物体が入らないように注意してください。SimpleTracker のコンソールに

Learning background...done

と表示されれば背景情報の獲得は完了です。

この状態で移動物体がレーザレンジファインダの観測領域内を移動すると gnuplot のウインドウ内にトラッキング結果が表示されます。表示をリセットしたい場合は、もう一度 ConsoleInString から tracking_lrf.conf のロードを行ってください。



4　動的入出力ポートの使用方法

GnuplotViewer に付加されている動的な入出力ポートの追加 · 削除の機能はクラス化されており、他のコンポーネントでも簡単に利用することができます。また、GnuplotViewer ではアクティブ化時に Configuration からポートの数を読み込み、ポートの生成を行っていましたが、このクラスはアクティブ状態のままポートの追加や削除を行うことも可能です。本章では、動的入出力ポートクラスである `DynamicInPort`, `DynamicOutPort` に関して説明します。

4.1　動的入出力ポート `DynamicInPort`, `DynamicOutPort` の公開メンバ

ここでは、動的入出力ポート `DynamicInPort`, `DynamicOutPort` の公開メンバ変数、メンバ関数に関して概説します。これらの具体的な使用方法 · 手順については 4.2 節を参照してください。

4.1.1 メンバ変数

公開メンバ変数は以下のとおりです。変数名は DynamicInPort, DynamicOutPort で共通ですが、変数 m_port の型はそれぞれ InPortVect<*DataType, Buffer*>, OutPortVect<*DataType, Buffer*>となります。

変数名	型	説明
m_data	PortDataVect<*DataType, Buffer*>	獲得したデータもしくは出力したいデータを格納する変数の動的配列。個々の要素が通常の入出力ポートの m_*PortName* に対応する。
m_port	InPortVect<*DataType, Buffer*> OutPortVect<*DataType, Buffer*>	入出力ポートの動的配列。個々の要素が通常の入出力ポートの m_*PortName*In や m_*PortName*Out に対応する。

4.1.2 メンバ関数

コンストラクタ、デストラクタを除く公開メンバ関数は以下のとおりです。関数名、機能は DynamicInPort, DynamicOutPort で共通です。

関数名	引数	戻り値の型	説明
addPort	なし	int	入出力ポートを 1 つ追加する。追加に成功すれば 0 を、エラーの時はそれ以外の値を返す。実際にこのポートを利用するには RTM::RtcBase::registerInPort() / RTM::RtcBase::registerOutPort() によるポートの登録が必要。
deletePort	int	int	引数で指定されたポート番号の入出力ポートを削除する。引数が指定されない場合は最後のポートを削除する。削除に成功すれば 0 を、エラーの時はそれ以外の値を返す。ポートを削除するとそれより番号の大きいポートのポート番号は 1 つずつ詰められる。ポートを削除する前には RTM::RtcBase::deletePortByName() によるポートの登録解除が必要。
getSize	なし	unsigned int	追加されているポートの数を返す。ポート番号は 0 番から順に付けられるので、利用可能なポートは 0 番から getSize()-1 番までである。
getName	int	const char*	引数で指定されたポート番号のポート名を返す。ポート名はポートの登録・登録解除の際に利用する。

4.2 動的入出力ポートの使用手順

動的入出力ポートを使用する手順を以下に示します。4.2.1 節, 4.2.2 節はコンポーネントの.h ファイルに追加する処理、4.2.3 節から 4.2.6 節は.cpp ファイルに追加する処理です。

4.2.1 ファイルのインクルード

コンポーネントの.h ファイルの冒頭部で dynamic_port.h ファイルをインクルードします。

ファイルのインクルード

```
#include "dynamic_port.h"
```

なお、インクルードパスは適宜変更するか Makefile に追加してください。

4.2.2 変数の宣言

• 動的 InPort を宣言する場合:

コンポーネントの.h ファイルの<rtc-template block="inport_declare">内にて宣言を行います。

動的 InPort の宣言

```
DynamicInPort<DataType,Buffer> VariableName;
```

〈例〉m_ShortDataIn という変数名の TimedShortSeq 型のデータを受け取る動的 InPort の宣言

```
DynamicInPort<TimedShortSeq,RTC::RingBuffer> m_ShortDataIn;
```

ここで、*DataType* はやりとりするデータの型、*Buffer* はバッファの種類、*VariableName* は変数名です。変数名は自由につけてかまいませんが、OpenRTM-aist の通常の入出力ポートの名前の付け方にならい、m_*PortName*In としておくとわかりやすいと思います。

• 動的 OutPort を宣言する場合:

コンポーネントの.h ファイルの<rtc-template block="outport_declare">内にて宣言を行います。DynamicInPort が DynamicOutPort になることを除いては、動的 InPort の場合と同様です。

動的 OutPort の宣言

```
DynamicOutPort<DataType,Buffer> VariableName;
```

〈例〉m_ShortDataOut という変数名の TimedShortSeq 型のデータを出力する動的 OutPort の宣言

```
DynamicOutPort<TimedShortSeq> m_ShortDataOut;
```

ここでは InPort と OutPort のそれぞれについて記述しましたが、基本的に InPort と OutPort で手順は変わりませんので、以下では InPort に対してのみ説明します。OutPort に対しては In の部分を Out に変更してください。

4.2.3 コンストラクタによる初期化

コンポーネントの.cpp ファイルの<rtc-template block="initializer">内にてコンストラクタに引数を与えて初期化を行います。

コンストラクタによる初期化

```
VariableName(PortName),
```

〈例〉ShortDataIn というポート名で変数 m_ShortDataIn を初期化

```
m_ShortDataIn("ShortDataIn"),
```

ここで、*VariableName* は 4.2.2 節で宣言した変数名、*PortName* は RTSystemEditor や RtcLink 上で表示されるポートの名前です。ポートを追加すると、ポートの名前が順に *PortName*0、*PortName*1、*PortName*2、... となります。ポート名も自由に付けることができますが、前述のとおり変数名を m_*PortName*In としておき、*PortName* を設定する（もしくは In の部分まで *PortName* に含める）とわかりやすくなると思います。

4.2.4 ポートの追加

ポートを追加したい時には以下のようにポートの追加と登録を行います。ポートの追加には動的入出力ポートのメンバ関数である *VariableName*.addPort() を使います。ポートを複数個追加したい場合は以下の処理を追加したいポートの数だけ繰り返してください。

動的 InPort の追加

VariableName.addPort();
registerInPort(*VariableName*.getName(*VariableName*.getSize()-1),
VariableName.m_port[*VariableName*.getSize()-1]);

〈例〉変数 m_ShortDataIn に InPort を 1 つ追加し、それを登録

```
m_ShortDataIn.addPort();
registerInPort(m_ShortDataIn.getName(m_ShortDataIn.getSize()-1),
               m_ShortDataIn.m_port[m_ShortDataIn.getSize()-1]);
```

4.2.5 ポートへのデータ入出力、取得したデータの参照

OpenRTM-aist では、ポートからのデータの入力、ポートへのデータの出力等は m_*PortName*In や m_*PortName*Out という変数のメンバ関数を用いて行い、獲得したデータや出力したいデータは m_*PortName* という変数に格納されます[†]。4.1.2 節で述べたように、動的入出力ポートの場合にはこれらがそれぞれ *VariableName*.m_port[*PortNumber*], *VariableName*.m_data[*PortNumber*] となります。具体的な使用例を以下に示します。

通常の入出力ポートと動的入出力ポートの対応関係

	通常の入出力ポート	動的入出力ポート
入出力ポート変数	m_*PortName*In / m_*PortName*Out	*VariableName*.m_port[*PortNumber*]
データ格納変数	m_*PortName*	*VariableName*.m_data[*PortNumber*]

〈例〉

変数 m_ShortDataIn の i 番目のポートからデータの読み込み

```
m_ShortDataIn.m_port[i].read()
```

変数 m_ShortDataIn の i 番目のポートに新しいデータが送られてきたかどうかチェック

```
m_ShortDataIn.m_port[i].isNew()
```

変数 m_ShortDataIn の i 番目のポートから得た可変長配列データの長さを獲得

```
m_ShortDataIn.m_data[i].data.length()
```

変数 m_ShortDataIn の i 番目のポートから得た可変長配列データの j 番目のデータを参照

```
m_ShortDataIn.m_data[i].data[j]
```

4.2.6 ポートの削除

ポートを削除したい時にはまずポートの登録を解除し、次に削除を行います。ポートの削除には動的入出力ポートのメンバ関数である *VariableName*.deletePort(*PortNumber*) を使います。*PortNumber* は削除したいポートの番号です。ポート番号はその動的入出力ポート変数が現在保持するポートに対して順に 0,1,2,... と付けられています。つまり、ポートを削除するとそれより番号の大きいポートのポート番号は 1 つずつ詰められます。例えば 3 番と 5 番のポートを削除したい場合、先に 3 番のポートを削除すると、5 番のポートは 1 つ分番号が詰められて 4 番になるため注意が必要です。もちろん、先に 5 番のポートを削除して次に 3 番のポートを削除すれば問題はありません。

[†]RtcTemplate を用いるか、RTCBuilder を用いて Var Name の設定を省略した場合の変数名。RTCBuilder を用いて Var Name を指定した場合にはそれがデータを格納する変数の名前となる。

動的 InPort の削除

```
deletePortByName(m_ShortDataIn.getName(PortNumber));
m_ShortDataIn.deletePort(PortNumber);
```

〈例〉変数 m_ShortDataIn のすべてのポートを登録解除し、削除

```
while(m_ShortDataIn.getSize() > 0){
   deletePortByName(m_ShortDataIn.getName(m_ShortDataIn.getSize()-1));}
   m_ShortDataIn.deletePort(m_ShortDataIn.getSize()-1);}
}
```

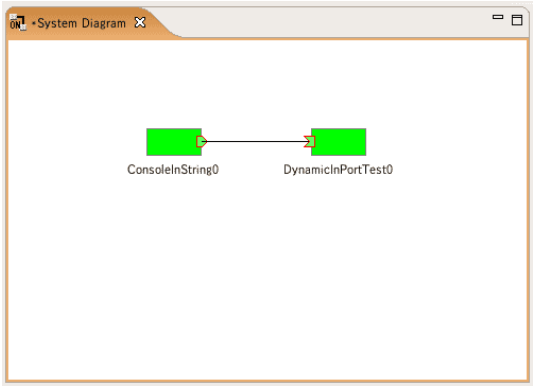
4.3 動的入出力ポート動作テスト用サンプルコンポーネント DynamicInPortTest

動的入出力ポートのポートの追加・削除機能の動作テストサンプルとして DynamicInPortTest コンポーネントが用意されています。この動作テストに使用するコンポーネントの種類とその数は

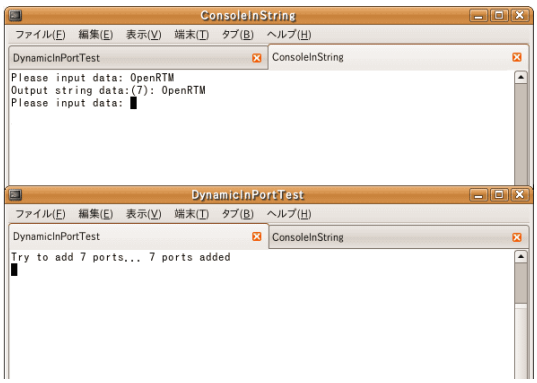
- 動的入力ポートテストコンポーネント (DynamicInPortTest) ・・・・・・ 1 つ
- コンソール文字列入力コンポーネント (ConsoleInString) ・・・・・・ 1 つ

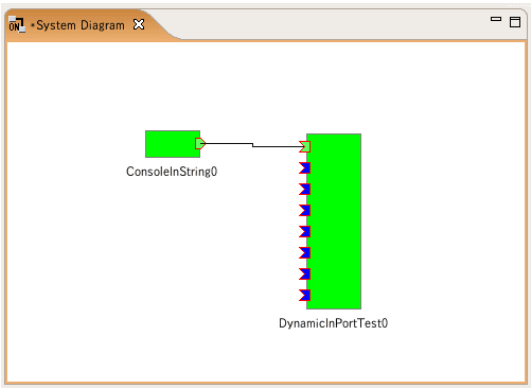
の計 2 つです。

まず、ConsoleInString の OutString ポートと DynamicInPortTest の InPortManip ポートを接続し、アクティブ化します。

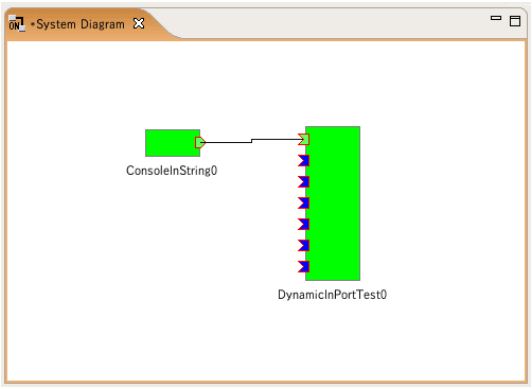


この状態で、数字以外の文字を含む文字列を ConsoleInString のコンソールから入力すると、その数だけ TimedString 型の入力ポートが追加されます。このとき、システムエディタの表示内容の更新が行われないため、システムエディタでの見た目上は入力ポートが追加されていません。DynamicInPortTest をドラッグして移動させるなどすることで表示内容が更新され、追加したポートが現れます。削除の際も同様です。

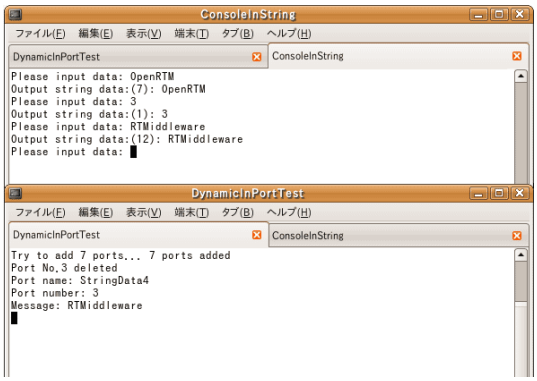


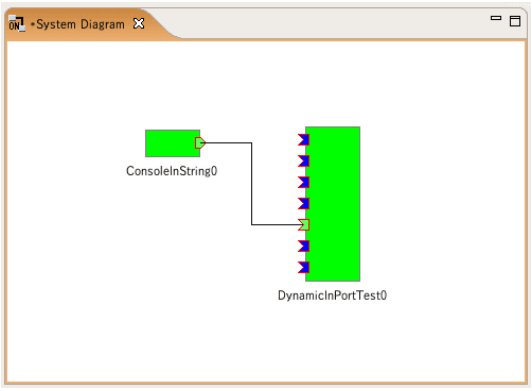


また、数字のみの文字列を ConsoleInString のコンソールから入力すると、そのポート番号の入力ポートが削除されます。ポート番号は動的入力ポートの生成されたものから順に 0,1,2,･･･と付けられ、削除された番号は詰められます（4.2.6 節を参照）。対応する番号のポートが存在しない場合には何も行われません。



さらに、生成された動的入力ポートに対し入力を行うとそのポートの名前（StringData#, ここで#は削除されたポートを含め累積何番目に生成されたのかを表す数）、ポート番号と入力された文字列がコンソールに出力されます。





5 FAQ

Q1: GnuplotViewer をアクティブ化しようとすると Error in Gnuplot::initialize(): Cannot open gnuplot と表示され、エラー状態になりアクティブ化できない。

Ans.: これには主に 2 つの原因が考えられます。

1. gnuplot がインストールされていない
2. gnuplot へのパスが通っていない

まずは gnuplot がインストールされているかを確認してください。次に、カレントフォルダで gnuplot と入力し、gnuplot が起動できるかどうか確認してください。gnuplot がインストールされているにもかかわらず gnuplot が起動できない場合には、環境変数 PATH に gnuplot の実行ファイルのパスを指定してください。

Q2: GnuplotViewer をアクティブ化しようとすると Error in Gnuplot::addData(): Cannot open file と表示され、エラー状態になりアクティブ化できない。

Ans.: GnuplotViewer が利用する一時ファイルが、そのフォルダへの書き込み許可がない等の理由により作成できないことが原因であると考えられます。デフォルトではカレントフォルダが一時ファイルの保存場所となっていますので、フォルダのパーミッションの変更を行ってください。デフォルトの一時ファイルの保

存場所を変更したいという場合には、ソースファイルからこの設定を変更します。GnuplotViewer.h の 396 行目の Gnuplot クラスの変数宣言部分で一時ファイルの保存場所を指定してください。Gnuplot クラスのコンストラクタの引数が一時ファイルの保存場所です。

〈例〉

（変更前）Gnuplot gp;

（変更後）Gnuplot gp(/tmp/);

このとき、最後の/までを含めて指定してください。変更後はファイルを保存しビルドを行ってください。

Q3: Sequence 型データ以外の型や独自のデータ型をもったデータをプロットしたい。

Ans.: 可視化を行うデータは通常、単一の数値よりも数値列であると考えられるため、現状では効率の面から整数および実数値の可変長 1 次元配列データ型である TimedShortSeq, TimedLongSeq, TimedFloatSeq, TimedDoubleSeq のみをサポートしています。要望がありましたら他の型へのサポートも行いますが、現状ではこのような場合にはデータ変換用の別のコンポーネントを用意する必要があります。

謝辞

レーザレンジファインダコンポーネント (LRFComponent) の開発に当たりましては、東京大学生産技術研究所橋本研究室（当時）の川路浩平氏、Dražen Brščić 氏、佐々木毅が作成した C++ UrgLaser クラスのコードの一部を利用しております。川路浩平氏、Dražen Brščić 氏の両名に感謝申し上げます。また、レーザレンジファインダコンポーネント (LRFComponent)、移動体トラッキングコンポーネント (SimpleTracker) は昨年度の RT ミドルウエアコンテストへの出品作品を開発成果プレゼンテーションやコンポーネントを利用していただいた方からのご指摘を基に改良したものです。アドバイスをいただきました皆様に感謝申し上げます。