

中級者向け講習会課題 1

会津大学 RTミドルウェア講習会

超音波センサーで障害物を検知し、それを避けて進むシステムを作成する

目次

1	課題.....	1
1.1	課題説明.....	1
1.2	コンポーネント概要.....	2
1.2.1	入出力ポート.....	2
1.2.2	超音波センサーのデータ型.....	2
1.2.3	サンプルソース.....	3
2	EV3 の起動.....	4
3	超音波センサーの仕様.....	6
3.1	超音波センサーについて.....	6
3.2	超音波センサーの性能.....	6
3.3	超音波センサーの値の取り方.....	6
4	速度の与え方.....	7
4.1	データ型.....	7
4.2	値.....	7
4.3	値の制限.....	8
5	作成のヒント.....	9
5.1	システムのアルゴリズム.....	9
6	雛型の作成.....	10
6.1	RTCBuilder の起動.....	10
6.1.1	ようこそ画面.....	10
6.1.2	パースペクティブを開く.....	10
6.1.3	新規プロジェクトの作成.....	12
6.1.4	プロファイル情報入力とコードの生成.....	14
6.1.5	コード生成.....	18
7	EV3 の前進とコンポーネントの作成.....	19
7.1	EV3 を前進させるサンプルソース (Python).....	19
7.2	前進コンポーネントの作成.....	20
7.2.1	コード編集.....	20
7.2.2	動作確認.....	21
7.2.3	vx の値を変えて動かす.....	26
8	旋回をするコンポーネントの作成.....	28
8.1	EV3 を旋回させるサンプルソース (Python).....	28
8.2	旋回コンポーネントの作成.....	28
8.2.1	コード編集.....	28
8.2.2	動作確認.....	29
8.2.3	値を変えて動かす.....	29
9	数秒前進後数秒旋回するコンポーネントの作成.....	30
9.1	2 秒間前進、2 秒旋回の手順.....	30

9.2	2秒前進2秒間旋回するサンプルソース	31
9.2.1	サンプルソース (Python)	31
9.2.2	動作確認	32
9.2.3	va の値を変えて 90 度旋回を行う	32
10	障害物を検知し旋回後前進するコンポーネントの作成.....	34
10.1	①超音波センサーの値を InPort で取得する手順.....	34
10.2	②障害物検知の手順.....	35
10.3	障害物を検知し旋回するサンプルソース	36
10.3.1	サンプルソース (Python)	36
10.3.2	動作確認.....	37

※ 文中の「x.y」や「x.y.z」の表記は使用環境の OpenRTM-aist のバージョンに読み替えてください。

当ドキュメントは下記ページを参考にしています。

・移動ロボット Kobuki の制御

http://www.openrtm.org/openrtm/ja/content/raspberrypi_kobuki_control

(2016/1/20 アクセス)

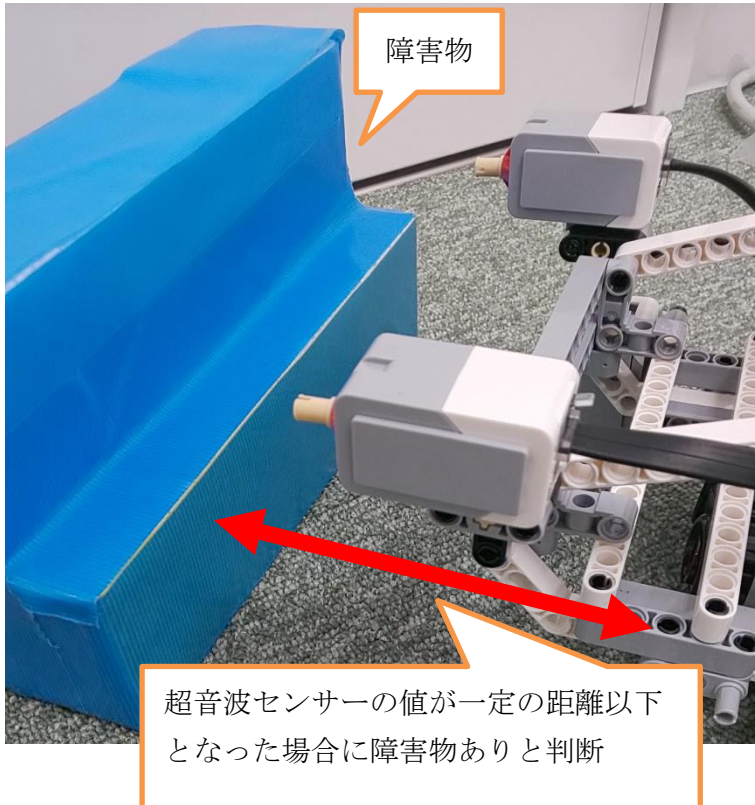
・LEGO Mindstorms EV3 活用事例

http://www.openrtm.org/openrtm/ja/casestudy/lego_mindstorm_ev3 (2016/1/20 アクセス)

1 課題

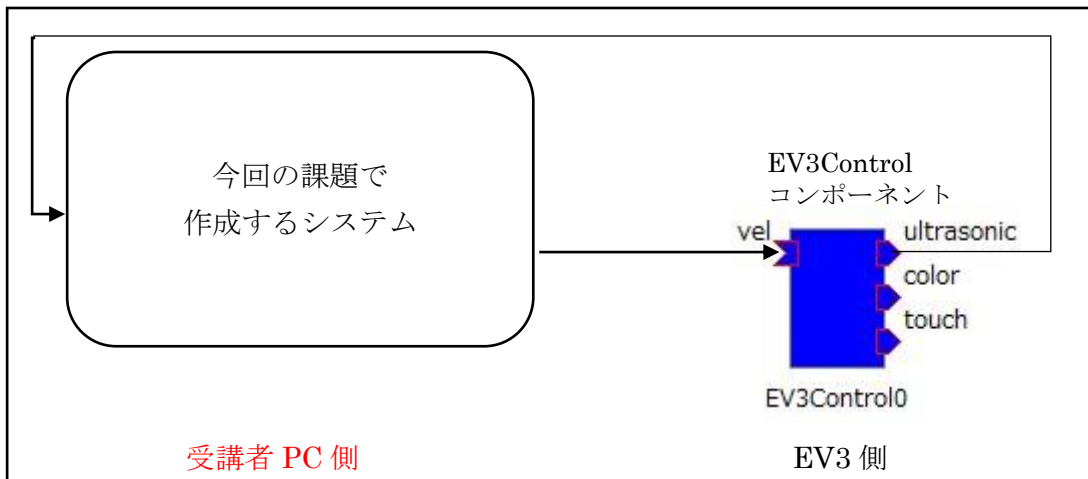
1.1 課題説明

超音波センサーにて障害物の検知を行い、障害物に衝突する前に右旋回させることで障害物を避けて進むシステムを作成してください。



1.2 コンポーネント概要

下記 EV3Control コンポーネントは、EV3 を制御するためのコンポーネントです。機能は、接続されたセンサーにて取得した値を各 OutPort から出力し、EV3 のモータを InPort からの入力値で制御します。今回の課題では、センサーにて取得した値を受け取り、その値を基に EV3 の速度を決定して出力を行います。



1.2.1 入出力ポート

EV3Control コンポーネントの InPort と OutPort は以下の内容になります。

ポート種別	ポート名	データの型	説明
InPort	vel	RTC.TimedVelocity2D	EV3 の速度の値
OutPort	ultrasonic	RTC.RangeData	超音波センサーの値
	color	RTC.TimedString	カラーセンサーの値
	touch	RTC.TimedBooleanSeq	タッチセンサーの値

この実習では EV3 の速度と超音波センサーの値を使用します。

1.2.2 超音波センサーのデータ型

超音波センサーは前方の障害物までの距離を m(メートル)単位で出力します。

EV3Control の超音波センサーのデータ型は以下のようになります。

【RTC.RangeData】

型	変数名	意味
sequence<double> RangeList	ranges	距離の値(m)
RangerGeometry	geometry	スキャンデータが測定された時点のレンジャーの形状 ※今回は使用しません
RangerConfig	config	スキャンデータが測定されたときのレンジャーの設定 ※今回は使用しません
RTC.Time	tm	タイムスタンプ ※今回は使用しません

上記の変数の `ranges` は `double` 型の配列です。配列の長さ（要素数）は 1 で 0 番目（先頭）の要素に超音波センサーの値が入ります。単位は `m`(メートル)です。

※配列の要素とは、配列中（同一の型のデータを一行に並べたもの）の各データのことをいい、要素は 0 番目を先頭として 0, 1, 2, 3,... と順に番号がついていて、最後の要素は (N-1) 番目。

OpenRTM で使用される変数の情報は以下のページに記載されています。

https://tmp.openrtm.org/doc/id/1.1/idreference_ja/structRTC_1_1RangeData.html

1.2.3 サンプルソース

超音波センサーの値取得のサンプルソース (Python)

```
# 超音波センサーの値が更新されている場合
if self._ultrasonicIn.isNew():
    # 超音波センサーの値を読み込み
    self._d_ultrasonic = self._ultrasonicIn.read()
    # 超音波センサーの値を取得
    distance = self._d_ultrasonic.ranges[0]
```

2 EV3 の起動

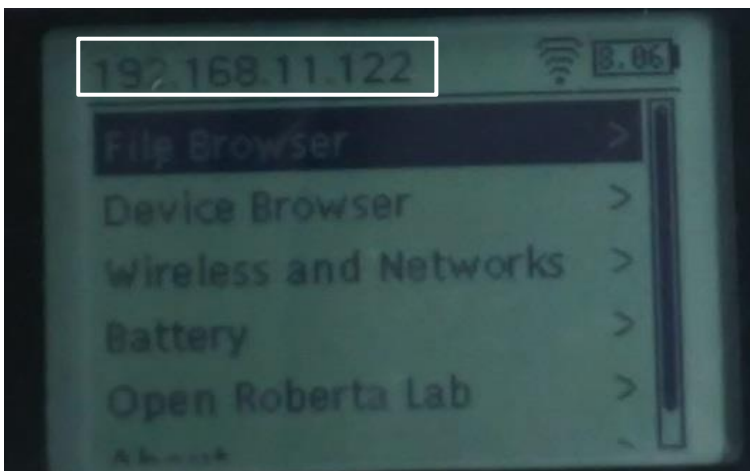
EV3 を起動させます。以下の本体写真の「決定」ボタンを押下してください。



しばらくすると ev3dev が起動して画面にメニューが表示されます。表示されずにずっと boot の状態が続くときは「戻る」ボタン「決定」ボタン「左」ボタンを同時に数秒間押し続けてください。強制的に再起動になります。

メニューの左上に IP アドレスが表示されるのを確認してください。

繋がっている場合は画面左上に IP アドレス[192.168.11.xxx]と表示されます。



※EV3 が Wi-Fi に接続されない場合は、以下の手順で Wi-Fi 接続を行ってください。

① 初期画面の状態から、「十字」キーで、「Wireless and Networks」を選択し、

中級者向け講習会課題 1

「決定」ボタンを押下します。

- ② 接続方法が表示されるので「Wi-Fi」を選択し、「決定」ボタンを押下します。
- ③ 「Powered」と表示されるので「決定」ボタンを押下します。
- ④ 現在使用可能な Wi-Fi が表示されるので接続したい ID を選択し、「決定」ボタンを押下します。
- ⑤ 「Connect」と「Network Connection」が表示されるので「Connect」を選択し、「決定」ボタンを押下します。
- ⑥ ID のパスワードを入力するようにダイアログが表示されます。再度、「決定」ボタンを押下すると、キーボードが現れるので、パスワードを入力します。
- ⑦ パスワードを入力後、「Accept」を選択し「決定」ボタンを押下すると、先ほどのダイアログにパスワードが入力された状態で表示されるので、再度「Accept」を選択し、「決定」ボタンを押下します。
- ⑧ しばらくするとネットワークに繋がります。左上に割り当てられた IP アドレスが表示されていれば設定完了です。

3 超音波センサーの仕様

3.1 超音波センサーについて



超音波センサーとは超音波を対象物に向け発信し、その反射波を受信することにより、対象物の有無や対象物までの距離を測定するセンサーです。EV3の超音波センサーは左図となります。この超音波センサーをEV3に接続することにより超音波センサーを使用することが出来るようになります。

3.2 超音波センサーの性能

計測可能範囲	3cm～250cm
計測角度	約 20°
計測精度	+/-1cm

3.3 超音波センサーの値の取り方



- (1) 電源が入っていないければ、「2 EV3の起動」の手順でEV3の電源を入れます。
- (2) 「十字」キーと「中央」ボタンで「Device Browser」→「Sensors」→「lego-ev3-us at in3」を選択します。
※「lego-ev3-us at in3」の at in3 はポート番号を
- (3) 選択後「下」ボタンを押下し、「Watch values」を選択してください。
超音波センサーで現在取得している値を確認することができます。EV3に接続された超音波センサーの前に手をかざすと

4 速度の与え方

ここでは EV3 への速度の与え方について説明します。

4.1 データ型

今回使用する EV3 制御用コンポーネントでは二次元速度ベクトル(RTC.TimedVelocity2D)を使用しています。二次元速度ベクトルとは、OpenRTM のデータの型で以下のようなデータ構造をしています。

【RTC.TimedVelocity2D 型】

型	変数名	意味
RTC.Velocity2D	data	速度データ
RTC.Time	Tm	タイムスタンプ※今回は使用しません

【RTC.Velocity2D 型】

型	変数名	意味
Double	vx	並進速度 (前方) [m/s]
Double	vy	並進速度 (横方) [m/s]
Double	va	角速度[rad/s]

【RTC.Time 型】

型	変数名	意味
int	sec	秒
int	nsec	ナノ秒

この型に値を与えて EV3 のモータを動かします。

4.2 値

上記の型を使用して EV3 を動かします。実際に使用するデータは va と vx のみで対向 2 輪型では vy は常に 0 となります。

EV3 を前進させたい場合は、

$vx = 0.1, \quad va = 0.0, \quad vy = 0.0$
--

後退させたい場合は、

$vx = -0.1, \quad va = 0.0, \quad vy = 0.0$

となり、va が 0 かつ vx が+(プラス)の時に各モータは前方に回転し、va が 0 かつ vx が-(マイナス)の時に後方に回転します。

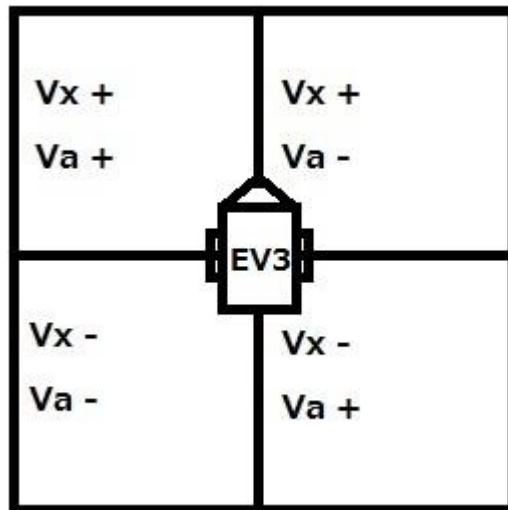
中級者向け講習会課題 1

EV3 を旋回させたい場合は

$$v_x = 0.0, \quad v_a = 0.1, \quad v_y = 0.0$$

となり、 v_x が 0 かつ v_a が+(プラス)の時に左のモータが後方、右のモータが前方に回転するので左旋回、 v_x が 0 かつ v_a が-(マイナス)の時は逆に左モータが前方、右のモータが後方に回転するので右旋回します。

二次元速度ベクトルの値と進行方向の関係は図のようになります。



例えば、 v_x が+(プラス)で v_a が+(プラス)だと右モータの値が大きくなるので左に旋回します。

4.3 値の制限

EV3 に与えられる速度にはモータの性能の関係上限りがあります。与えられる値の最大・最小値は以下となります。

変数名	最小値	最大値
v_x	-0.5m/s	0.5m/s
v_a	-8.5m/s	8.5m/s

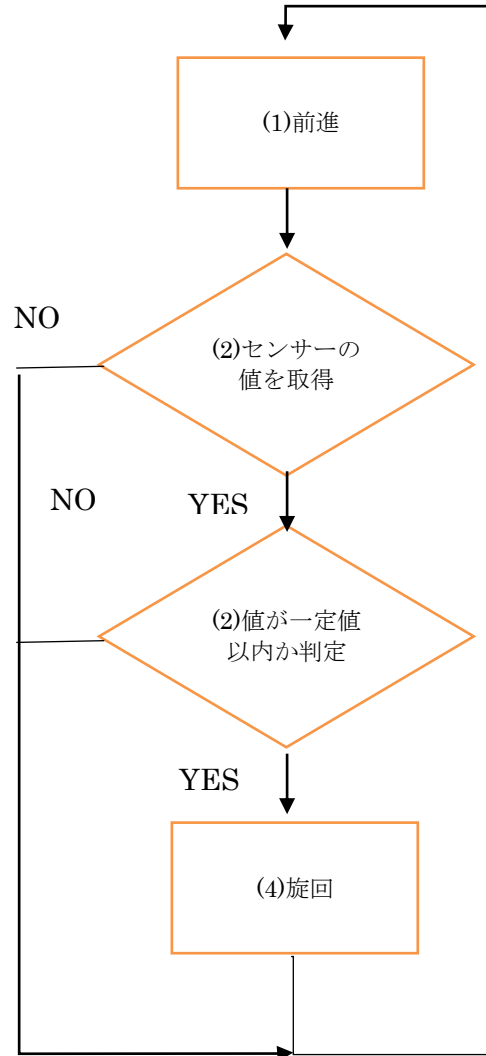
この範囲以外の値を与えてもモータの動きには反映されません。

これらのことを参考に EV3 に速度を与えてみてください。

5 作成のヒント

5.1 システムのアルゴリズム

作成するシステムの一例を示します。今回のシステムの動きをフローチャートにすると以下ようになります。



- (1) 前進の処理。前進処理終了後(2) センサーの値を取得できたか判定。
- (2) センサーの値を取得できたか判定。NO なら(1)前進の処理に移行。YES なら(3)の処理に移行。
- (3) 取得した値が一定値以内か判定。NO なら(1)前進の処理に移行。YES なら(4)旋回の処理に移行。
- (4) 90° 旋回の処理実行。旋回処理終了後(1)の判定に戻る。

旋回の処理に関しては速度を一回与えた後 `sleep` 関数を使用して一定時間コンポーネントを停止させる方法などがあります。

6 雛型の作成

超音波センサーを使ったコンポーネントを作成します。以下の手順に従って作成します。

6.1 RTCBuilder の起動

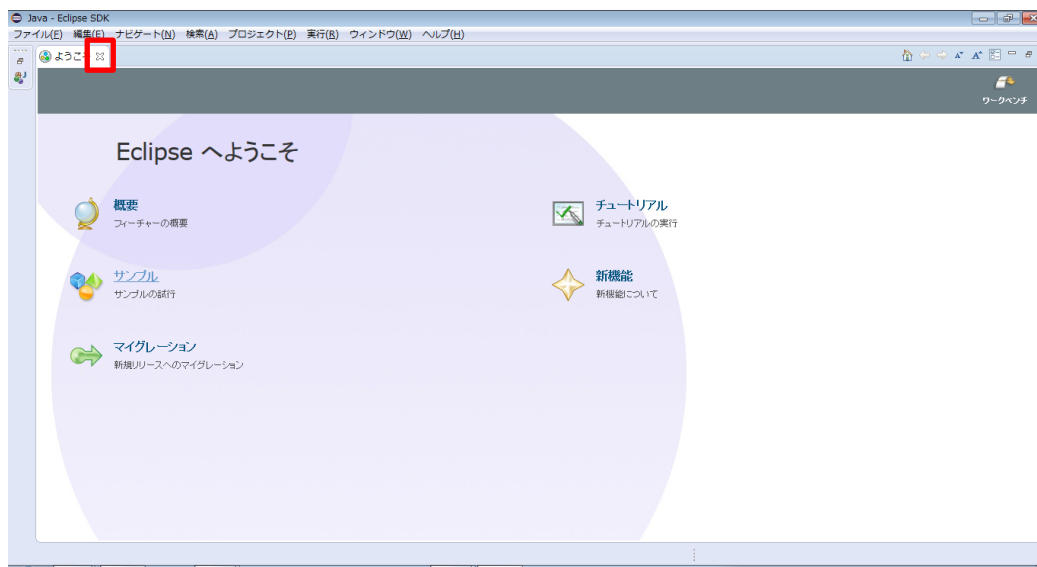
OpenRTP を起動させると作成物を保存するディレクトリを指定します。

ここでは、下記フォルダに保存します。

C:\¥rtcws

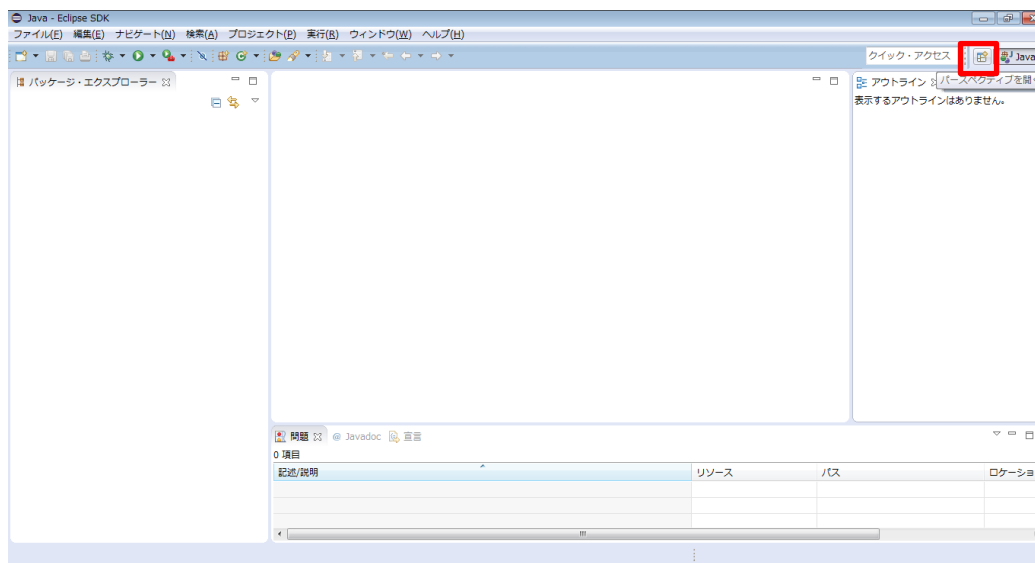
6.1.1 ようこそ画面

OpenRTP を初めて起動した際は下記画面が表示されます。この画面は使用しないので、左上の（「ようこそ」タブの右にある）「×」ボタンを押下します。



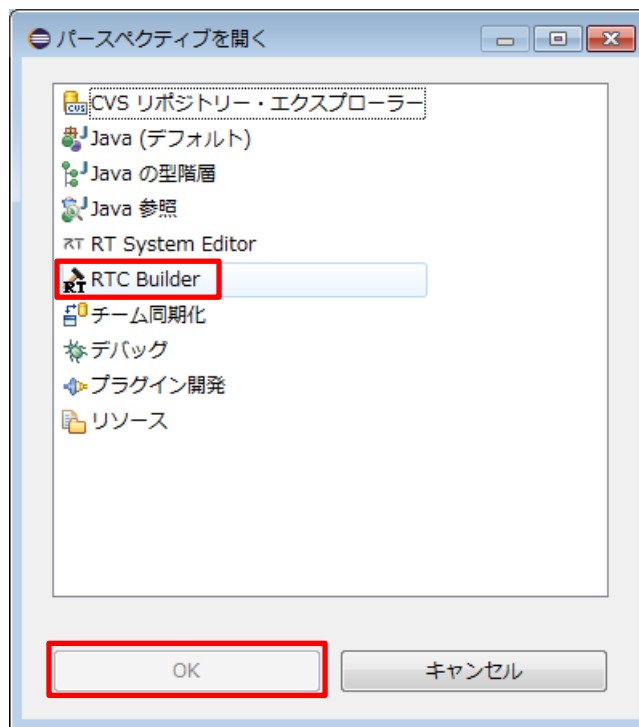
6.1.2 パースペクティブを開く

「×」ボタンを押下すると下記画面が表示されます。右上の「パースペクティブを開く」を左クリックしてください。

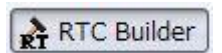


中級者向け講習会課題 1

下記画面が表示されるので「RTC Builder」を選択し、「OK」ボタンを左クリックします。

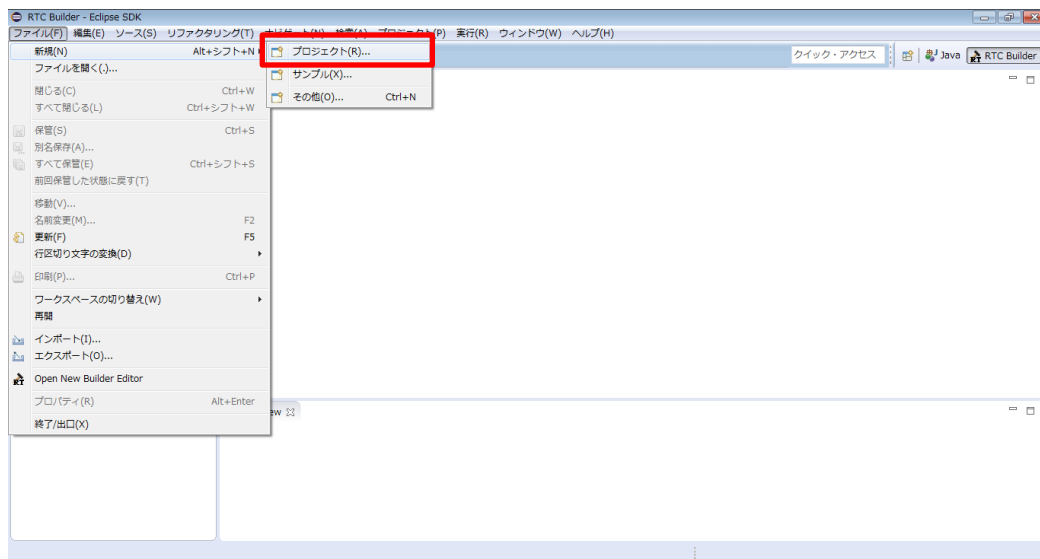


「RTC Builder」を選択することで、RTC Builder が起動します。メニューバーに
のアイコンが表示されたら完了です。

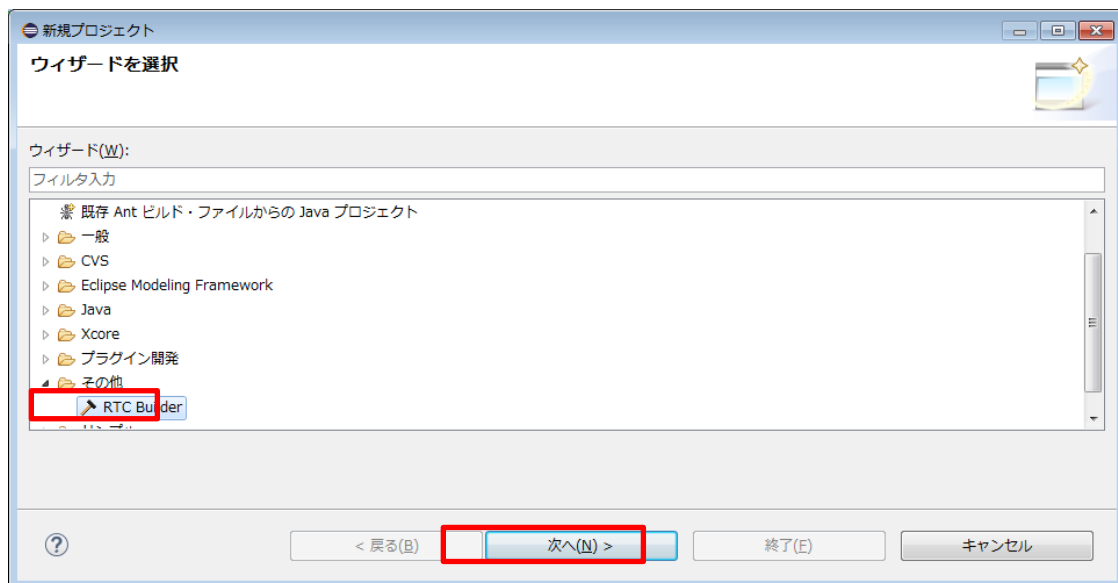


6.1.3 新規プロジェクトの作成

画面上部のメニューから「ファイル」→「新規」→「プロジェクト」を選択します。

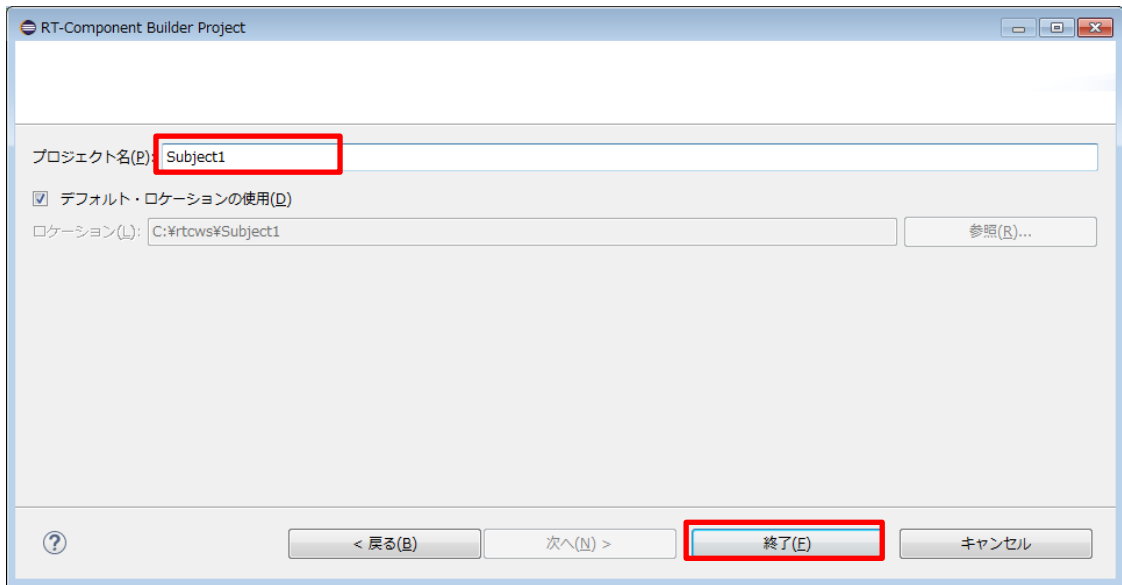


「新規プロジェクト」画面で「その他」→「RTC Builder」を選択し、「次へ」ボタンを左クリックします。

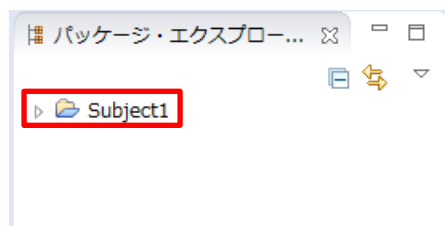


中級者向け講習会課題 1

「プロジェクト名」欄に作成するプロジェクト名（ここでは **Subject1**）を入力して「終了」ボタンを左クリックします。



下図のようにパッケージ・エクスプローラー内に「Subject1」というプロジェクトが追加されたら完了です。

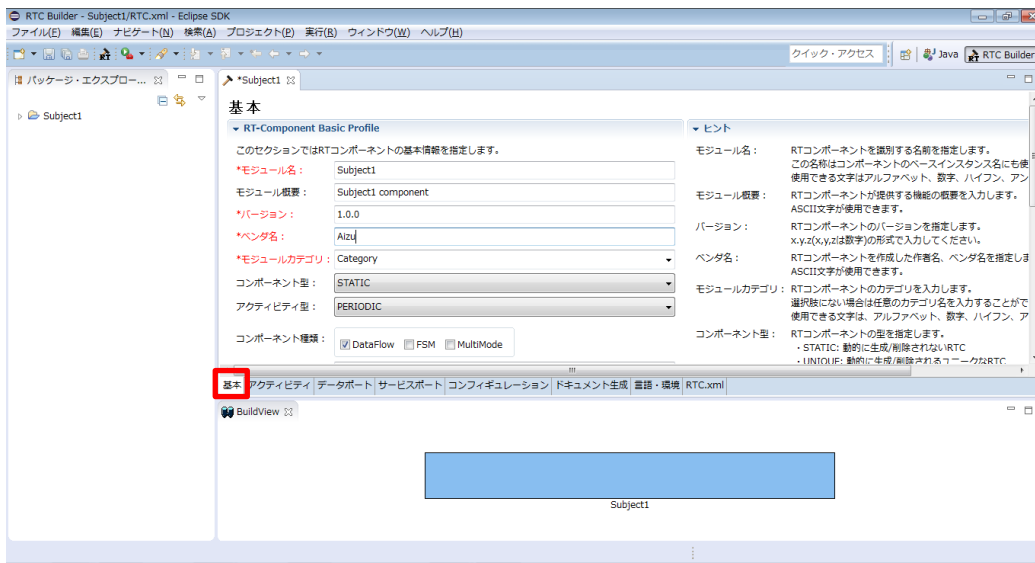


6.1.4 プロファイル情報入力とコードの生成

6.1.4.1 基本

一番左の「基本」タブを選択し、基本情報を設定します。コンポーネントの名前や概要などを記入します。ラベルが赤字の項目は必須項目です。その他はデフォルトの状態では変更は不要となります。

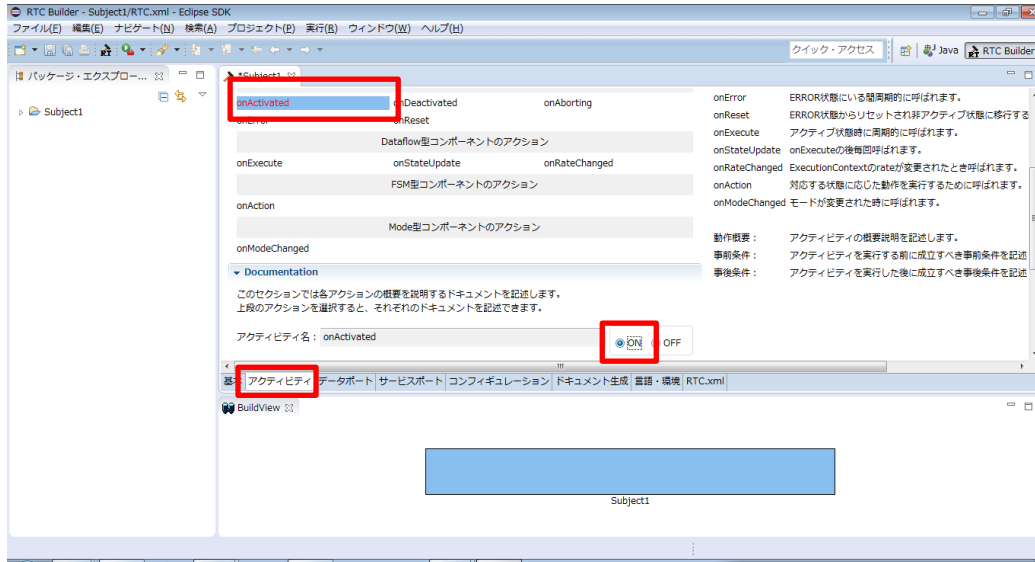
モジュール名 : Subject1
モジュール概要 : Subject1 component
バージョン : 1.0.0
ベンダ名 : Aizu
モジュールカテゴリ : Category
コンポーネント型 : STATIC
アクティビティ型 : PERIODIC
コンポーネント種類 : DataFlowComponent
最大インスタンス数 : 1
実行型 : PeriodicExecutionContext
実行周期 : 1000.



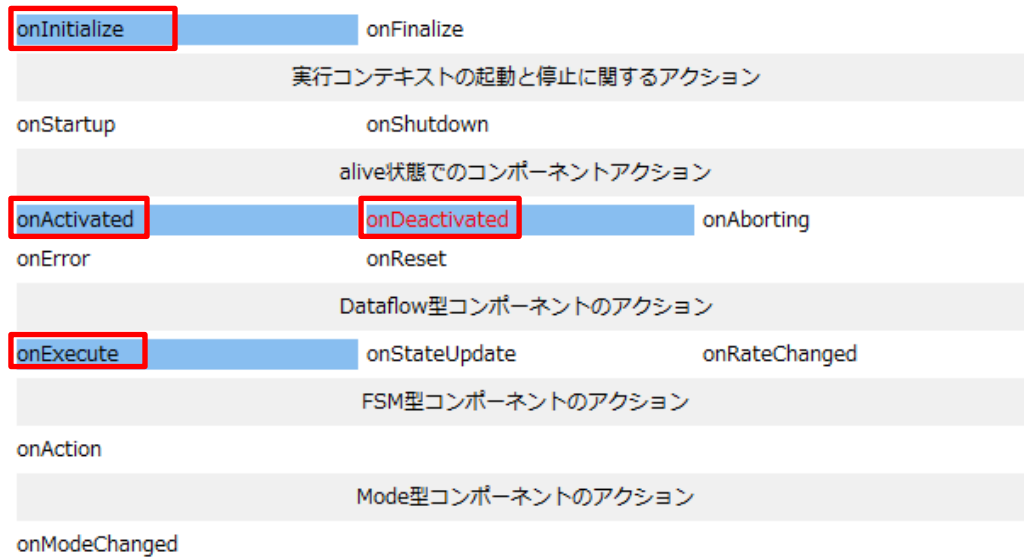
中級者向け講習会課題 1

6.1.4.2 アクティビティ

次に、「アクティビティ」タブを選択し、使用するアクションコールバックを指定します。Subject1 コンポーネントでは、onActivated, onDeactivated, onExecute コールバックを使用します。下図のように赤枠の①onActivated をクリック後に②赤枠のラジオボタンにて"on"にチェックを入れます。onDeactivated, onExecute についても同様に設定します。



最終的に下図のようになります。

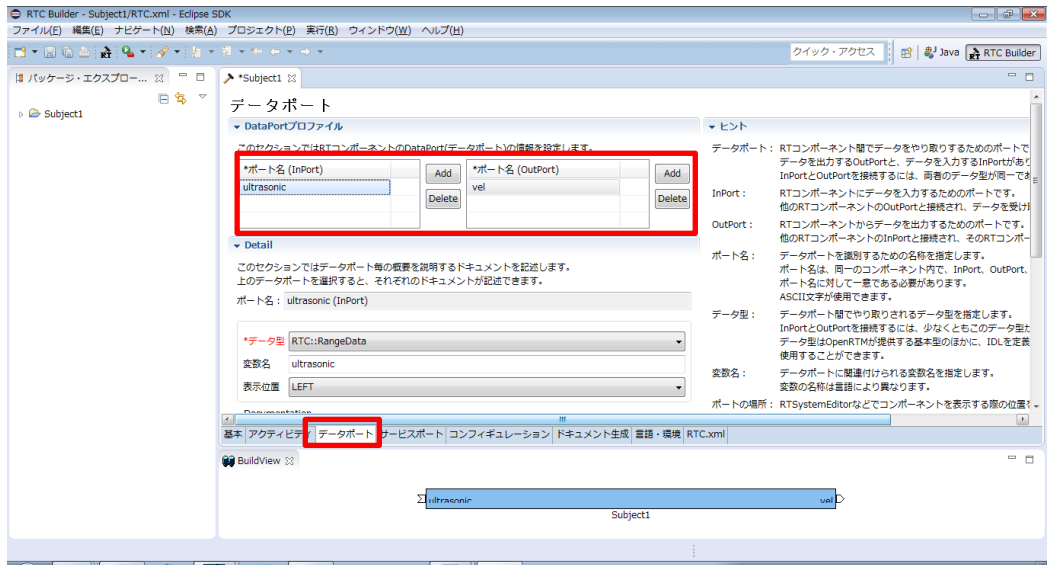


6.1.4.3 データポート

「データポート」タブを選択し、データポートの情報を入力します。以下のように入力します。「Add」ボタンを押下して新しいデータポートを追加します。

• InPort
 ポート名: ultrasonic
 データ型: RTC::RangeData
 変数名: ultrasonic
 表示位置: left

• OutPort
 ポート名: vel
 データ型: RTC::TimedVelocity2D
 変数名: vel
 表示位置: right



ここで設定した情報は Python プログラム上では以下の様に反映されます。

InPort ポート名	self._ポート名 In
InPort 変数名	self._d_変数名
OutPort ポート名	self._ポート名 Out
OutPort 変数名	self._d_変数名

このコンポーネントでは以下の様になります。

InPort ultrasonic	self._ultrasonicIn
InPort ultrasonic	self._d_ultrasonic
OutPort vel	self._velOut
OutPort vel	self._d_vel

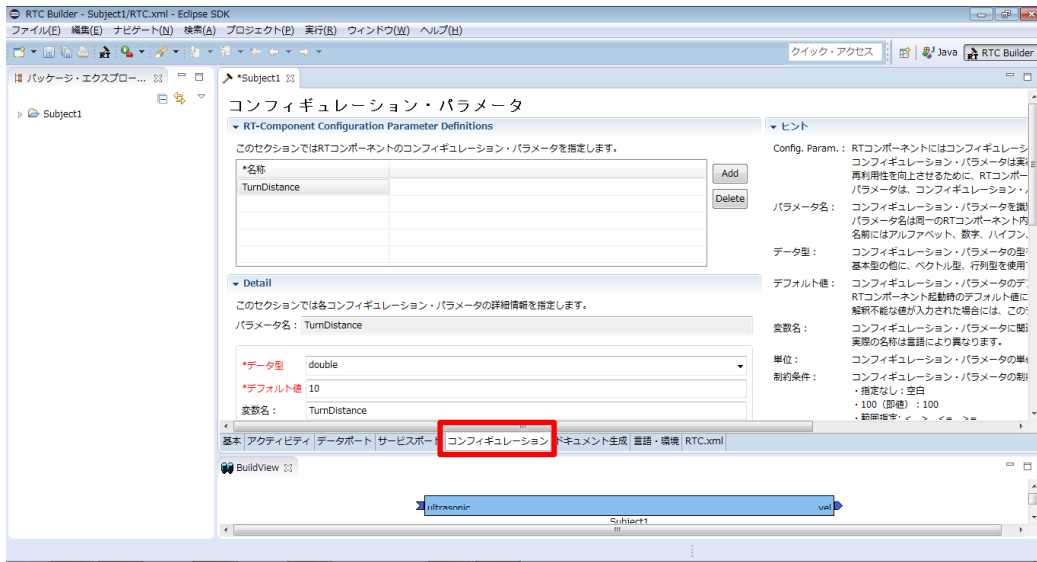
6.1.4.4 コンフィギュレーション

「コンフィギュレーション」タブを選択し、Configuration の情報を入力します。制約条件および Widget とは、RTSystemEditor でコンポーネントのコンフィギュレーション・パラメータを表示する際に GUI で値の変更を行うための形式を表すものです。

「Add」ボタンを押下して新しいコンフィギュレーションを追加します。

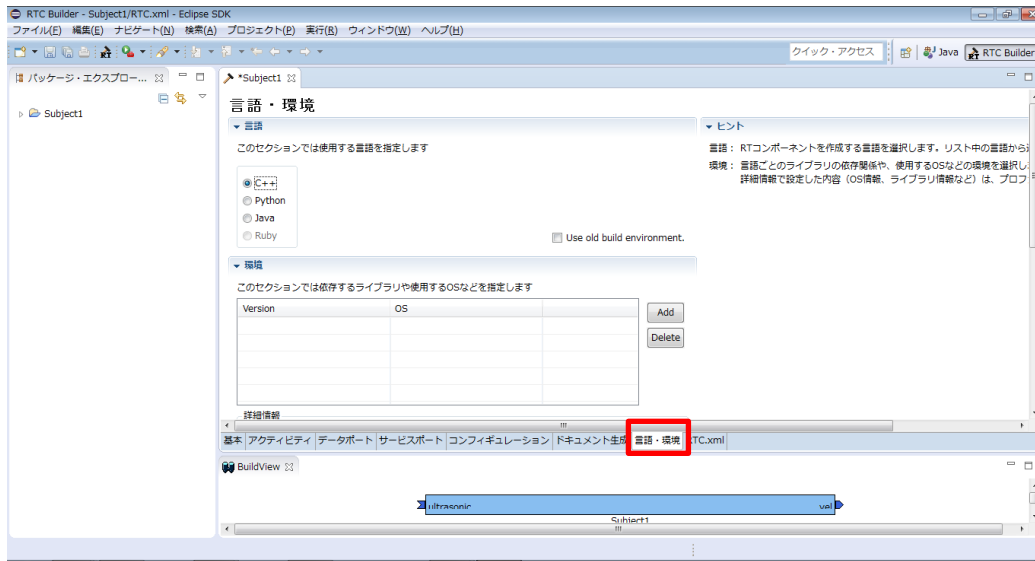
ここでは例として超音波センサーの距離が何 m で回避動作を行うかの値を追加します。後で調整したい値を適宜、追加しておくといよいでしょう。

名称: TurnDistance
データ型: double
デフォルト値: 0.1
変数名: TurnDistance
制約条件: $0.03 \leq x \leq 2.5$
Widget: slider



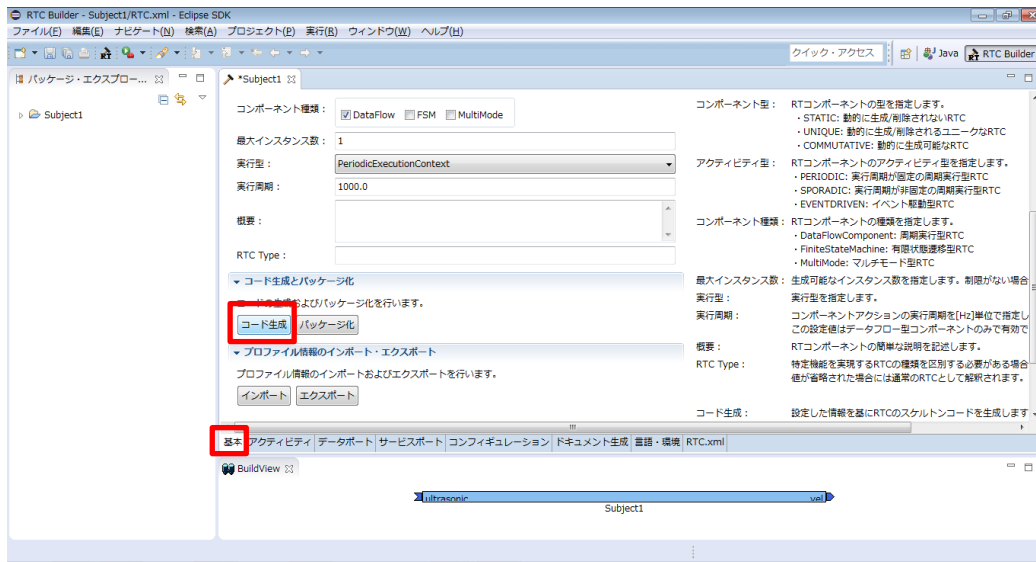
6.1.4.5 言語・環境

「言語・環境」タブを選択し、プログラミング言語を選択します。ここでは”Python”を選択します。言語・環境はデフォルトでは設定されていないので、指定し忘れるとコード生成時にエラーとなりますので、必ず言語の指定を行うようにしてください。



6.1.5 コード生成

全ての設定が完了したら「基本」タブに戻り「コード生成」ボタンを左クリックします。問題がなければコンポーネントの雛型が生成されます。



7 EV3 の前進とコンポーネントの作成

作成した雛型をもとにコンポーネントを作成します。

EV3 を前進させるコンポーネントを作成します。

「4 速度の与え方」で説明した通り、EV3 を前進させるには

```
vx = 0.04,    va = 0.0,    vy = 0.0
```

というように vx にのみプラスの値を設定し、va, vy には 0 を設定します。

これをプログラムに直すと以下のようになります。

7.1 EV3 を前進させるサンプルソース (Python)

```
#変数に前進の値を代入
self._d_vel.data.vx = 0.04
self._d_vel.data.va = 0.0
self._d_vel.data.vy = 0.0
# アウトポート ( vel ) に書込み
self._velOut.write()
```

[write()]は OutPort から値を出力する関数

変数に値を設定し、write()でデータを OutPort から送信します。

7.2 前進コンポーネントの作成

サンプルソースを参考に、EV3 に前進の値を送信するコンポーネントを作成してみましょう。

7.2.1 コード編集

Subject1.py を開き「def onExecute(self, ec_id):」内にサンプルソースを貼り付けてください。

Subject1.py は下記フォルダにあります。

C:\¥rtcws¥Subject1¥Subject1.py

貼り付けを行うと以下のようになります。

```
def onExecute(self, ec_id):  
  
    #変数に前進の値を代入  
    self._d_vel.data.vx = 0.04  
    self._d_vel.data.va = 0.0  
    self._d_vel.data.vy = 0.0  
    # アウトポート ( vel ) に書込み  
    self._velOut.write()  
    return RTC.RTC_OK
```

そして「def __init__(self, manager):」内を以下のように修正します。

```
修正前 : self._d_vel = RTC.TimedVelocity2D(*vel_arg)  
修正後 : self._d_vel = RTC.TimedVelocity2D(RTC.Time(0, 0),  
RTC.Velocity2D(0.0, 0.0, 0.0))
```

Subject1.py を保存し、Subject1.py をダブルクリックしてコンポーネントを起動します。

7.2.2 動作確認

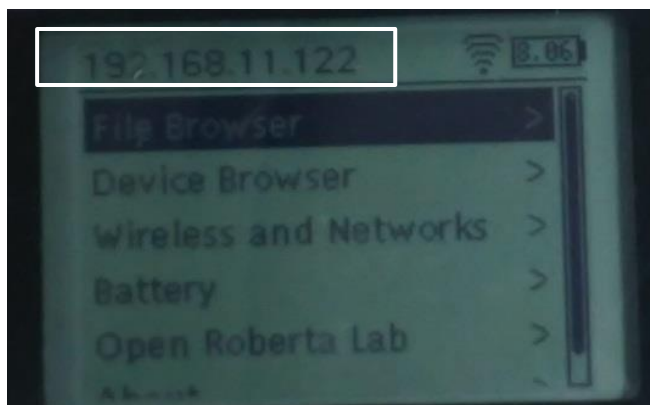
EV3 内の EV3 制御用コンポーネントとネームサーバを起動します。

この講習会では事前に OpenRTM のインストールと RTC-Library-Fukushima の EV3 制御用コンポーネント (<https://rtc-fukushima.jp/component/1703/>) の配置を行っています。

7.2.2.1 EV3 制御用コンポーネントとネームサーバの起動

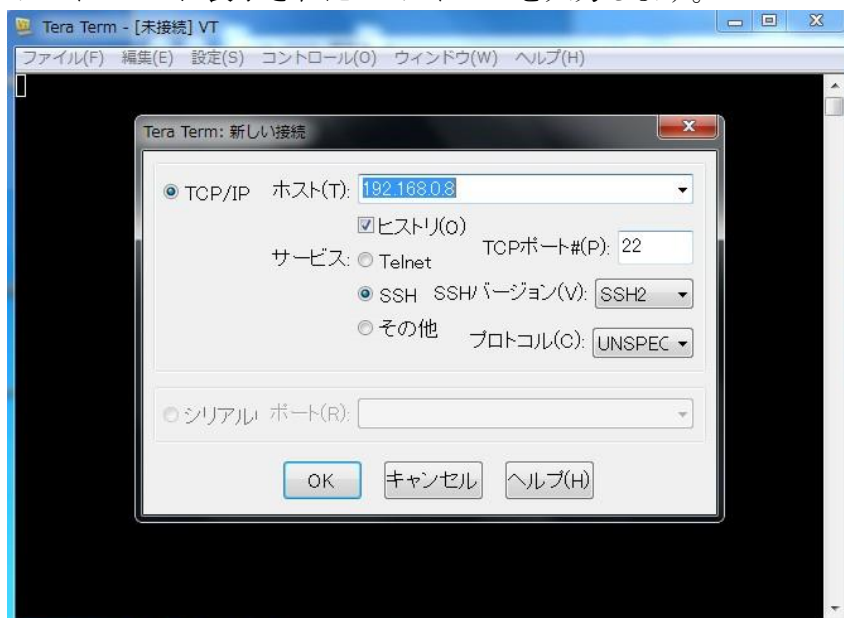
EV3 制御用コンポーネントとネームサーバを起動します。

EV3 の画面に表示される IP アドレスを再度確認してください。



Tera Term で EV3 に接続します。PC の Tera Term を起動してください。

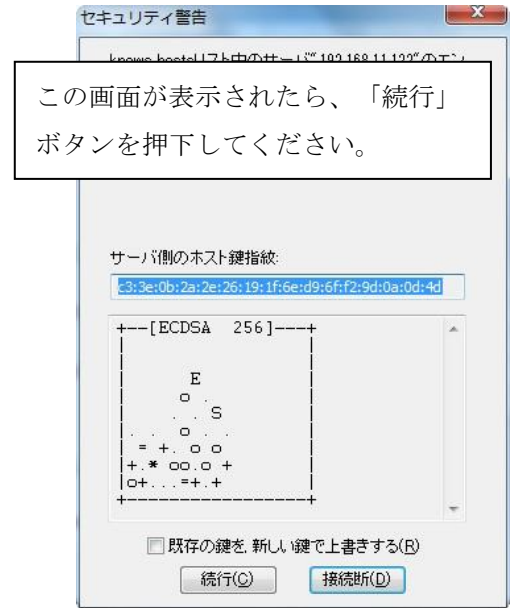
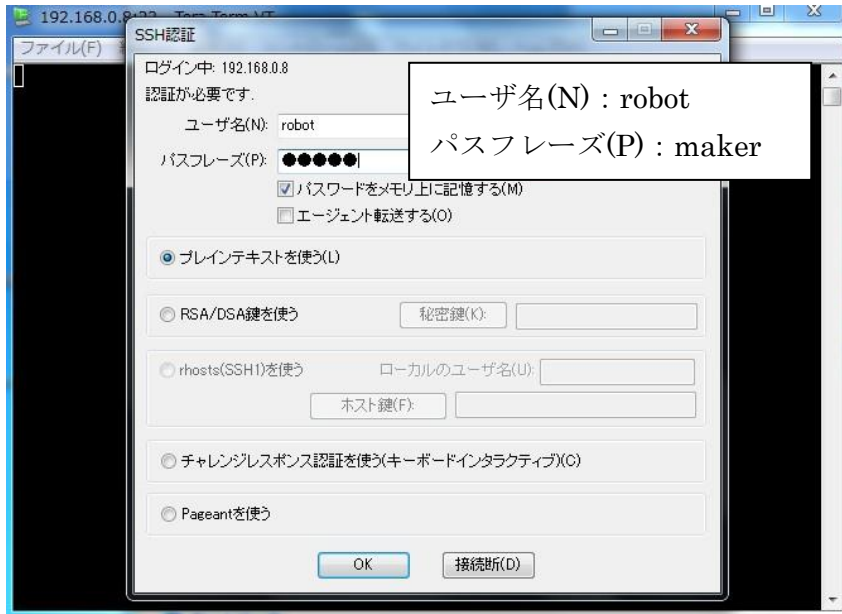
ホスト : EV3 に表示された IP アドレスを入力します。



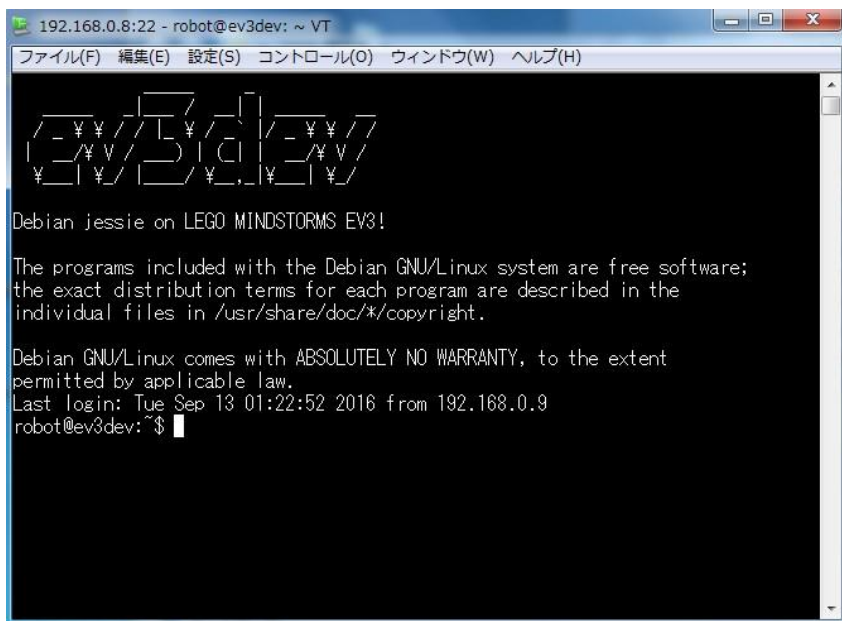
中級者向け講習会課題 1

ユーザ名 : robot

パスワード : maker を入力します。



この画面が表示されたらログイン完了です。



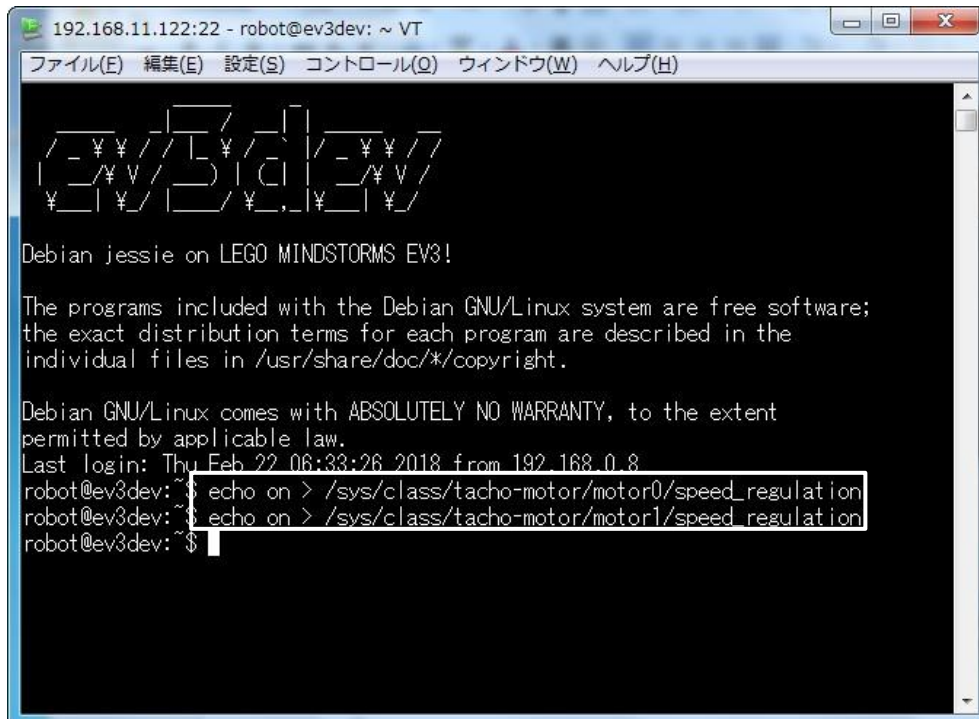
7.2.2.2 デバイスファイルの設定

今回 EV3 用コンポーネントで速度制御を行うために、各モータのデバイスファイルの設定を変更する必要があります。変更するファイルは `speed_regulation` です。これはデフォルトでは `off` になっており、`on` に変更することにより速度制御が行えるようになります。

以下のコマンドによりモータのデバイスファイルに書き込むことができます。

```
$ echo on > /sys/class/tacho-motor/motor0/speed_regulation
$ echo on > /sys/class/tacho-motor/motor1/speed_regulation
```

echo : 引数に指定された文字列や変数の内容を表示する。



The image shows a terminal window titled "192.168.11.122:22 - robot@ev3dev: ~ VT". The window contains the following text:

```
192.168.11.122:22 - robot@ev3dev: ~ VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)

Debian jessie on LEGO MINDSTORMS EV3!

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: Thu Feb 22 06:33:26 2018 from 192.168.0.8
robot@ev3dev:~$ echo on > /sys/class/tacho-motor/motor0/speed_regulation
robot@ev3dev:~$ echo on > /sys/class/tacho-motor/motor1/speed_regulation
robot@ev3dev:~$
```

7.2.2.3 EV3 のネームサーバと EV3 制御用コンポーネントの起動

① ネームサーバを起動します。

以下のコマンドでネームサーバが起動します。

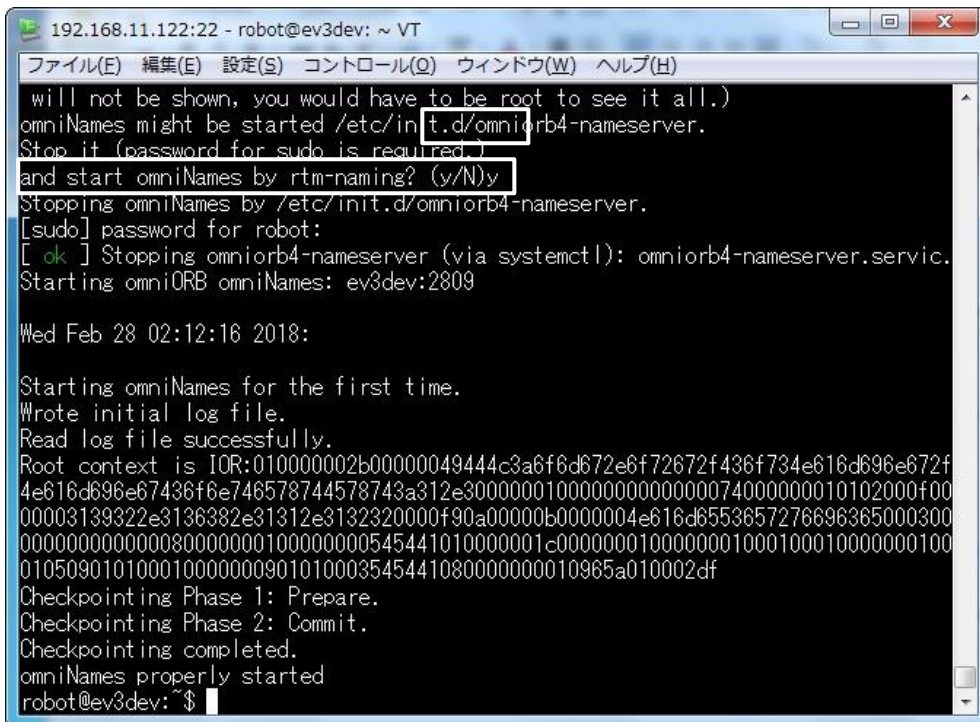
```
$ rtm-naming
```

途中 (y/n)の入力を求められますので「y」を入力します。

また、[sudo] password for robot: の文言が表示されパスワードの入力を求められます。

その際は EV3 のパスワード「maker」を入力してください。パスワードの入力のとき文字の表示などはなくブランク状態になっています。

コマンドを入力すると下図のようになります。



```
192.168.11.122:22 - robot@ev3dev: ~ VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
will not be shown, you would have to be root to see it all.)
omniNames might be started /etc/init.d/omniorb4-nameserver.
Stop it (password for sudo is required.)
and start omniNames by rtm-naming? (y/N)y
Stopping omniNames by /etc/init.d/omniorb4-nameserver.
[sudo] password for robot:
[ok] Stopping omniorb4-nameserver (via systemctl): omniorb4-nameserver.servic.
Starting omniORB omniNames: ev3dev:2809

Wed Feb 28 02:12:16 2018:

Starting omniNames for the first time.
Wrote initial log file.
Read log file successfully.
Root context is IOR:010000002b00000049444c3a6f6d672e6f72672f436f734e616d696e672f
4e616d696e67436f6e746578744578743a312e30000001000000000000074000000010102000f00
00003139322e3136382e31312e3132320000f90a00000b0000004e616d6553657276696365000300
00000000000080000000100000000545441010000001c000000010000000100010001000000100
010509010100010000000901010003545441080000000010965a010002df
Checkpointing Phase 1: Prepare.
Checkpointing Phase 2: Commit.
Checkpointing completed.
omniNames properly started
robot@ev3dev:~$
```

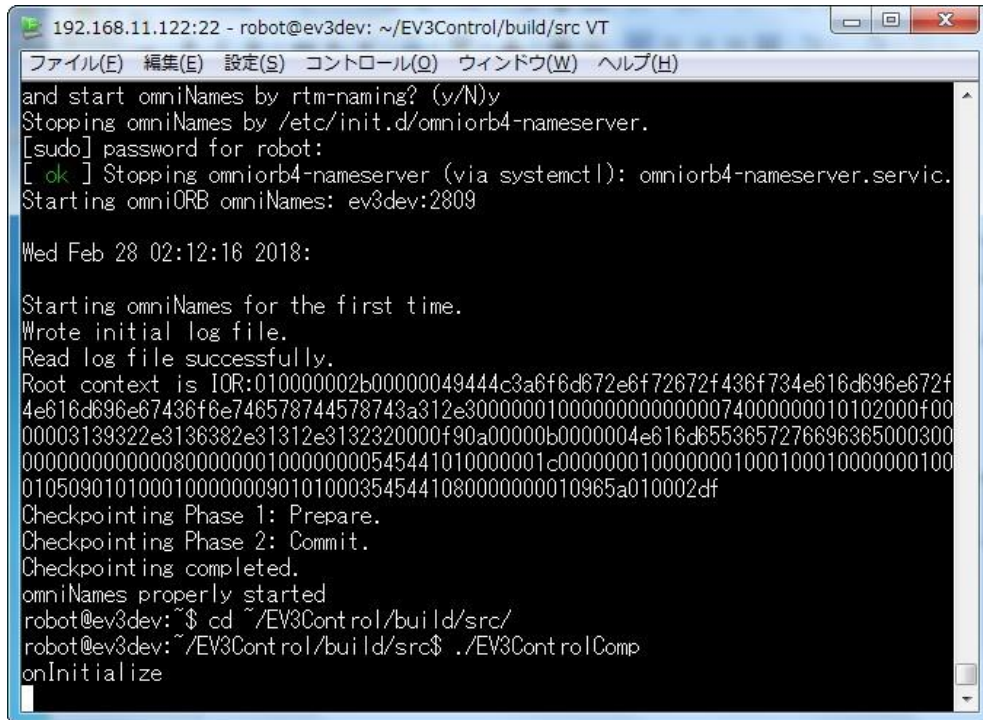
中級者向け講習会課題 1

② プログラムを起動します。

以下のコマンドで事前に配置されている EV3 側のコンポーネントを起動します。

```
$ cd ~/EV3Control/build/src/  
$ ./EV3ControlComp
```


コマンドを入力すると下図のようになります。



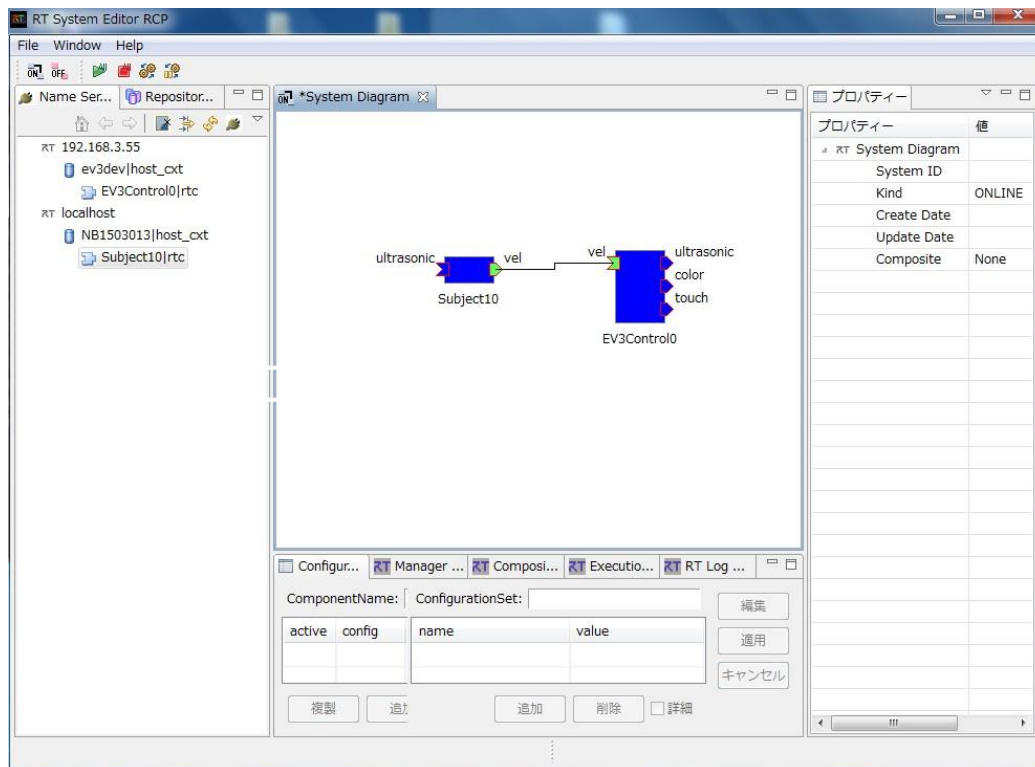
```
192.168.11.122:22 - robot@ev3dev: ~/EV3Control/build/src VT  
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)  
and start omniNames by rtm-naming? (y/N)y  
Stopping omniNames by /etc/init.d/omniorb4-nameserver.  
[sudo] password for robot:  
[ ok ] Stopping omniorb4-nameserver (via systemctl): omniorb4-nameserver.servic.  
Starting omniORB omniNames: ev3dev:2809  
  
Wed Feb 28 02:12:16 2018:  
  
Starting omniNames for the first time.  
Wrote initial log file.  
Read log file successfully.  
Root context is IOR:010000002b00000049444c3a6f6d872e8f72672f436f734e616d696e672f  
4e616d696e67436f6e746578744578743a312e300000010000000000000074000000010102000f00  
00003139322e3136382e31312e3132320000f90a00000b0000004e616d6553657276696365000300  
00000000000080000000100000000545441010000001c0000000100000001000100010000000100  
010509010100010000000901010003545441080000000010965a010002df  
Checkpointing Phase 1: Prepare.  
Checkpointing Phase 2: Commit.  
Checkpointing completed.  
omniNames properly started  
robot@ev3dev:~$ cd ~/EV3Control/build/src/  
robot@ev3dev:~/EV3Control/build/src$ ./EV3ControlComp  
onInitialize
```

7.2.2.4 PC のネームサーバの起動と RTSystemEditor の起動

作成したコンポーネントを起動したら、「スタートメニュー」→「すべてのプログラム」→「OpenRTM-aist x.y.z」→「Tools」より「Start Naming Service」と「RTSystemEditorRCP」を起動します。

 を左クリックすると「ネームサーバへ接続」画面が表示されるので、EV3 の IP アドレスを入力し「OK」ボタンを押下します。

「Name Service View」に表示されたコンポーネントをドラッグ&ドロップし、下図のようにポートを接続します。



ポート接続後、All Active(緑のアイコン)を押下してコンポーネントをアクティベート化します。これで EV3 が前進します。

前進することを確認したら、All Deactive (赤のアイコン) を押してコンポーネントをディアクティベート化します。これで EV3 が止まります。

EV3 が止まりましたら、前進コンポーネントを右クリックし、メニューを表示します。メニュー内の「Exit (E)」をクリックするとコンポーネントが終了します。

7.2.3 vx の値を変えて動かす

現在 vx の値は 0.04 になっています。自分で vx の値を変更して速度を色々変更してみてください。

中級者向け講習会課題 1

v_x の値の範囲は以下になります。

	最小値	最大値
v_x	-0.5	0.5

$v_x > 0$ の場合前進、 $v_x < 0$ の場合後退します。 $v_x = 0$ の場合は停止します。

値を変えて色々動かしてみてください。

8 旋回をするコンポーネントの作成

前進のコンポーネントのにコード追加して前進後 EV3 がその場で旋回するコンポーネントを作成します。

このコンポーネントを作成するには、 V_x の値を 0 にして、 V_a に値を入れると旋回します。

```
vx = 0,    va = 0.5,    vy=0
```

8.1 EV3 を旋回させるサンプルソース (Python)

```
# 変数に旋回の値を代入
self._d_vel.data.vx = 0.0
self._d_vel.data.va = 0.5
self._d_vel.data.vy = 0.0
# アウトポート (vel) に書込み
self._velOut.write()
```

8.2 旋回コンポーネントの作成

サンプルソースを参考に、EV3 に旋回の値を送信するコンポーネントを作成してみましょう。

8.2.1 コード編集

先ほど開いた Subject1.py を再度開き [def onExecute(self, ec_id):] 内にサンプルソースを貼り付けてください。

Subject1.py は以下の場所にあります。

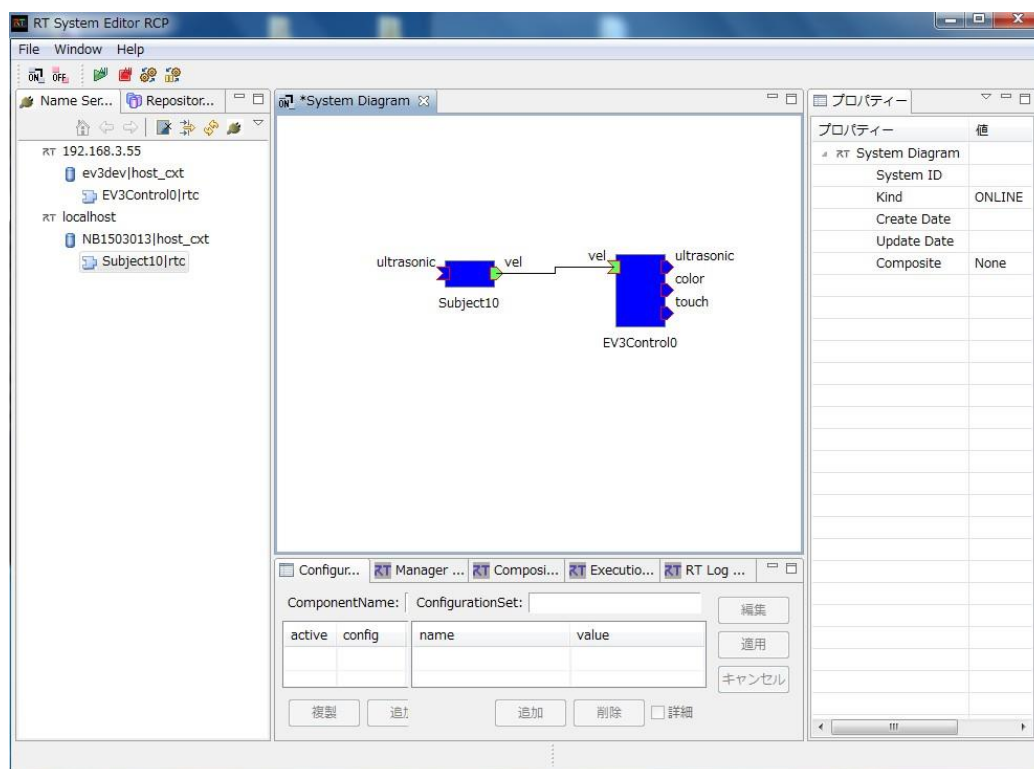
C:\rtcws\Subject1\Subject1.py


```
def onExecute(self, ec_id):

    # 変数に旋回の色を代入
    self._d_vel.data.vx = 0.0
    self._d_vel.data.va = 0.5
    self._d_vel.data.vy = 0.0
    # アウトポート (vel) に書き込み
    self._velOut.write()
    return RTC.RTC_OK
```

編集が完了したら Subject1.py を保存して Subject1.py をダブルクリックしてコンポーネントを起動します。

8.2.2 動作確認



コンポーネントを EV3 用コンポーネントに接続して動きを確かめてください。

8.2.3 値を変えて動かす。

va の値の範囲は以下になります。

	最小値	最大値
va	-8.5	8.5

va>0 の場合左旋回、va<0 の場合後右旋回します。

値を変えて色々動かしてみてください。

9 数秒前進後数秒旋回するコンポーネントの作成

旋回のコンポーネントにコードを追加して、2秒前進 2秒間旋回させるコンポーネントを作成します。

9.1 2秒間前進、2秒旋回の手順

2秒前進、2秒間旋回とは前進のデータを送信後 2秒後に旋回のデータを送信して 2秒後前進のデータを送信することで表すことができます。ここでは `sleep` 関数を使用して 2秒後に前進のデータを送信します。

手順としては以下になります。

- ① 前進のデータを送信
- ② 2秒間コンポーネントを停止(`sleep`)
- ③ 旋回のデータを送信
- ④ 2秒間コンポーネントを停止(`sleep`)

`sleep` 関数は指定した秒数だけプログラムを停止させます。EV3 制御用コンポーネントは速度を一度与えるとその速度で EV3 を動かし続けます。従って、値を与えず続ける必要はありませんので、`sleep` を使っても問題ありません。

プログラムで表すと以下のようにになります。

・2秒間前進するサンプルソース (Python)

```
# ①前進指令
self._d_vel.data.vx = 0.04
self._d_vel.data.va = 0.0
self._d_vel.data.vy = 0.0
self._velOut.write()
# ②2秒間コンポーネントを停止
time.sleep(2)
```

・2秒間旋回するサンプルソース (Python)

```
# ③旋回指令
self._d_vel.data.vx = 0.0
self._d_vel.data.va = 0.5
self._d_vel.data.vy = 0.0
self._velOut.write()
# ④2秒間コンポーネントを停止
time.sleep(2)
```

以上のことを踏まえて、2秒前進後、2秒間旋回するコンポーネントを作成します。

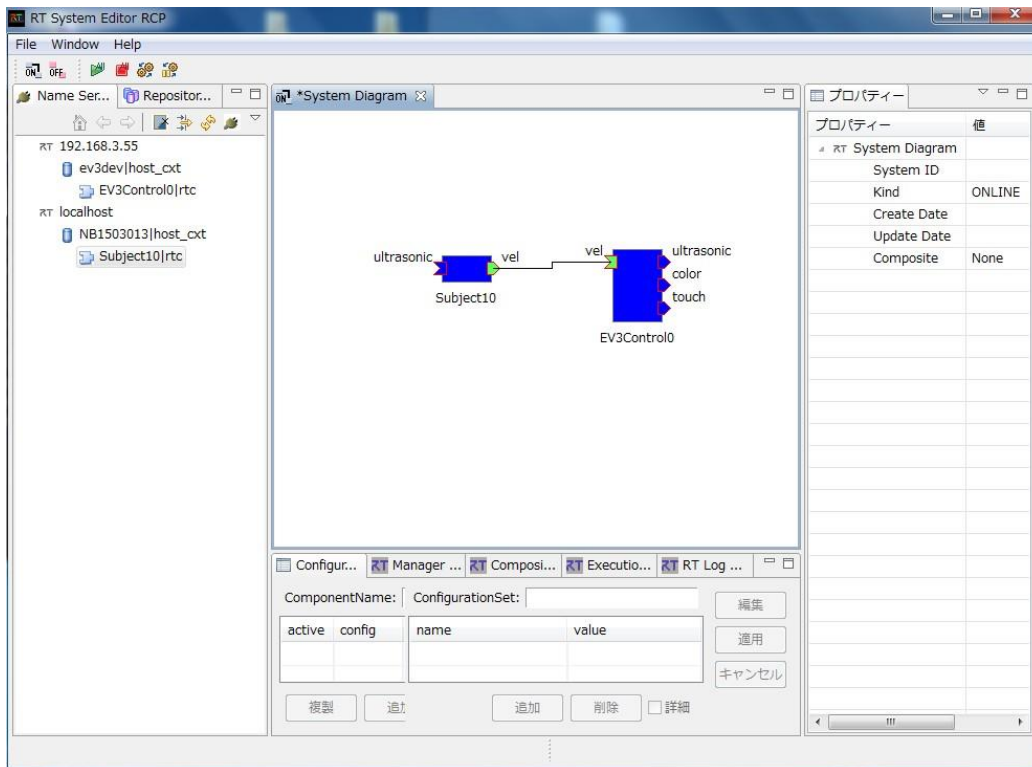
9.2 2秒前進 2秒間旋回するサンプルソース

9.2.1 サンプルソース (Python)

```
def onExecute(self, ec_id):
    # ①前進指令
    self._d_vel.data.vx = 0.04
    self._d_vel.data.va = 0.0
    self._d_vel.data.vy = 0.0
    self._velOut.write()
    # ②2秒間コンポーネントを停止
    time.sleep(2)
    # ③旋回指令
    self._d_vel.data.vx = 0.0
    self._d_vel.data.va = 0.5
    self._d_vel.data.vy = 0.0
    self._velOut.write()
    # ④2秒間コンポーネントを停止
    time.sleep(2)
    return RTC.RTC_OK
```

9.2.2 動作確認

上記のソースコードを使用して実際に試してみましょう。



9.2.3 va の値を変えて 90 度回転を行う

現状 va の値が 0.5 の場合 2 秒では 90 度回転は出来ません。
va の値を色々変更して 90 度回転を実現してみてください。

値を変更して目的の値を見つけるには 2 分探索をすると見つけやすいです。

・ 2 分探索

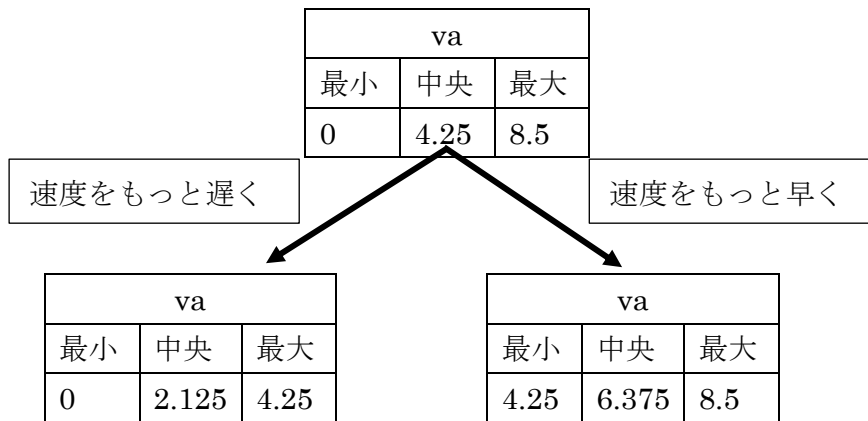
中央値の値を見て、目的の値がその値より大きいかわり小さいかを判断して探索を進めていく方法です。

この方法を使用して回転の値を探索すると、探索の仕方は以下になります。

最小値：0 と最大値：8.5 と中央値：4.25 の値でそれぞれの EV3 の回転速度をみます。そして目的の速度は中央値での速度を比較して遅いか早いかをみます。仮に目的の速度が

中級者向け講習会課題 1

中央値の速度より遅い場合は、中央値：4.25 と最小値：0 の中央値：2.125 で速度をみて。早いか遅いかを判断します。



この様にして少しずつ目的の値に近づいていきます。

10 障害物を検知し旋回後前進するコンポーネントの作成

先ほど作成したコンポーネントにコードを追加して、目の前の障害物を検知し 2 秒間旋回させるコンポーネントを作成します。

このコンポーネントを作成するには、超音波センサーを使用します。①超音波センサーの値を InPort で取得し、②障害物との距離が一定以内になったら、③旋回の値を OutPort から送信する流れになります。

10.1 ①超音波センサーの値を InPort で取得する手順

超音波センサーの値を InPort からの取得の手順は以下になります。

- ① 新しいデータを受信しているか確認
- ② データがある場合、その後データを読み込む
- ③ 読み込んだデータを変数に代入

プログラムで表すと以下ようになります。

・ InPort のデータを読み込むサンプルソース (Python)

```
# ①超音波センサーの値が更新されている場合
if self._ultrasonicIn.isNew():
    # ②超音波センサーの値を読み込み
    self._d_ultrasonic = self._ultrasonicIn.read()
    # ③超音波センサーの値を取得
    distance = self._d_ultrasonic.ranges[0]
```

[isNew()]は新しい値を受信しているか確認する関数

[read()]新しい値を読み込む関数

データがない時に「read()」をした場合、空データを読み込もうとしてコンポーネントがエラーになります。従って、データの有無の確認のために「isNew()」が必要になります。

10.2②障害物検知の手順

前の項目で障害物までの距離を取得しました。

次に距離が一定以内にあるかの確認をします。

一定以内にある場合旋回処理をするという判定はプログラムだと if 文を使用した条件分になります。

従ってプログラムは以下の様になります。

- ・ 距離が一定以内にあるかの判定 (Python)

```
if distance <= self._TurnDistance[0]:#距離が一定以内の場合
    #旋回の処理を行う#
```

[self._TurnDistance[0]]はプログラム実行中でも値が変更できるコンフィギュレーションというものです。現在通常では 0.1m に設定されています。

つまり、このプログラムは取得した距離が 0.1m 以下なら旋回の処理を行うという意味になります。

コンポーネントを作成します。

10.3 障害物を検知し旋回するサンプルソース

旋回のプログラムはすでに行っているので[10.1]と[10.2]と旋回のプログラムを組み合わせると以下のプログラムになります。

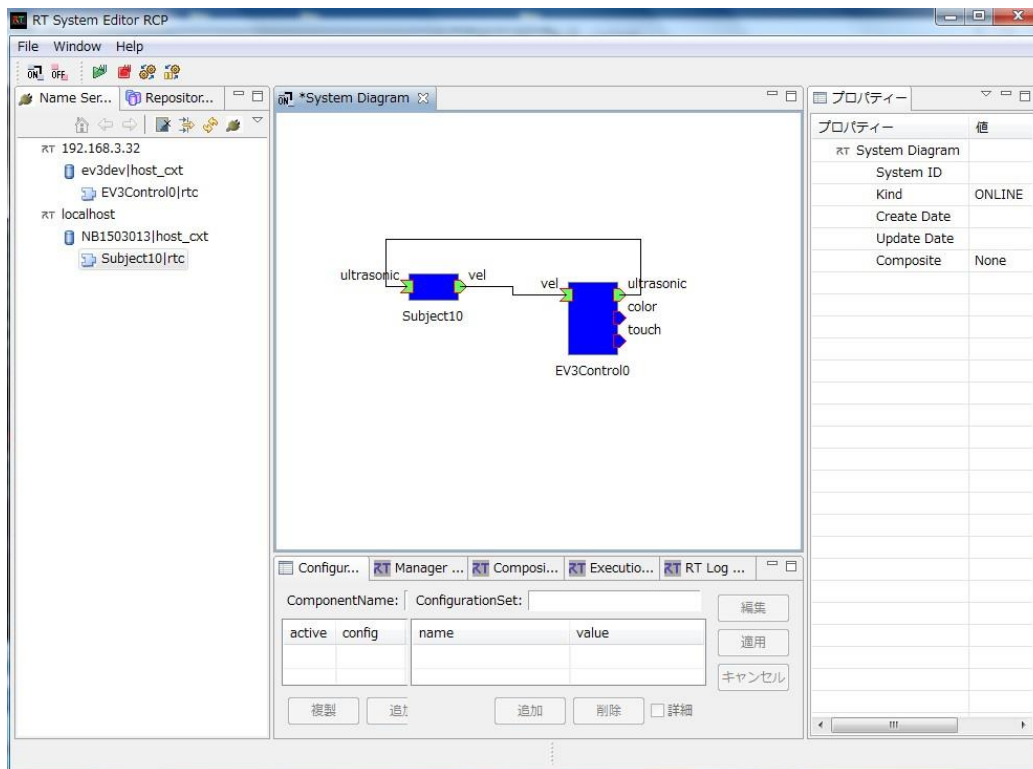
10.3.1 サンプルソース (Python)

```
def onExecute(self, ec_id):
    #前進指令
    self._d_vel.data.vx = 0.04
    self._d_vel.data.va = 0.0
    self._d_vel.data.vy = 0.0
    self._velOut.write()
    #①超音波センサーの値を InPort で取得
    if self._ultrasonicIn.isNew():
        self._d_ultrasonic = self._ultrasonicIn.read()
        distance = self._d_ultrasonic.ranges[0]
        #②距離が一定以内
        if distance <= self._TurnDistance[0]:
            #旋回指令
            self._d_vel.data.vx = 0.0
            self._d_vel.data.va = 0.5
            self._d_vel.data.vy = 0.0
            self._velOut.write()
            time.sleep(2)
    return RTC.RTC_OK
```


10.3.2 動作確認

上記のソースコードを使用して実際に試してみましょう。

今回は超音波センサーの値を取得するので、EV3Control コンポーネントの超音波センサー (OutPort)と Subject1 コンポーネントの超音波センサー(InPort)を接続し動作確認します。



コンフィギュレーションを使用すると旋回を開始する障害物までの距離が変化します。

