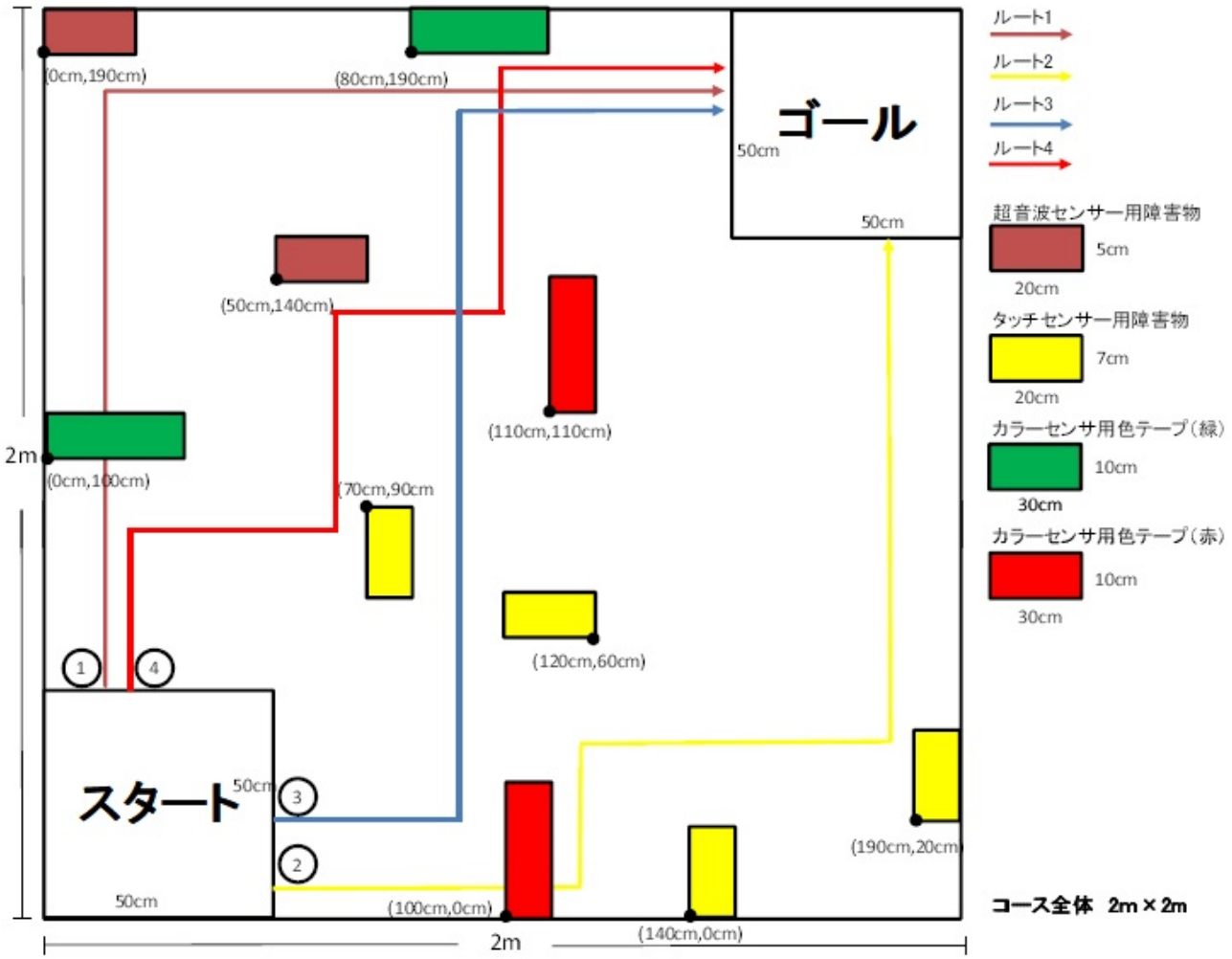
中級者向け講習会課題 1

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

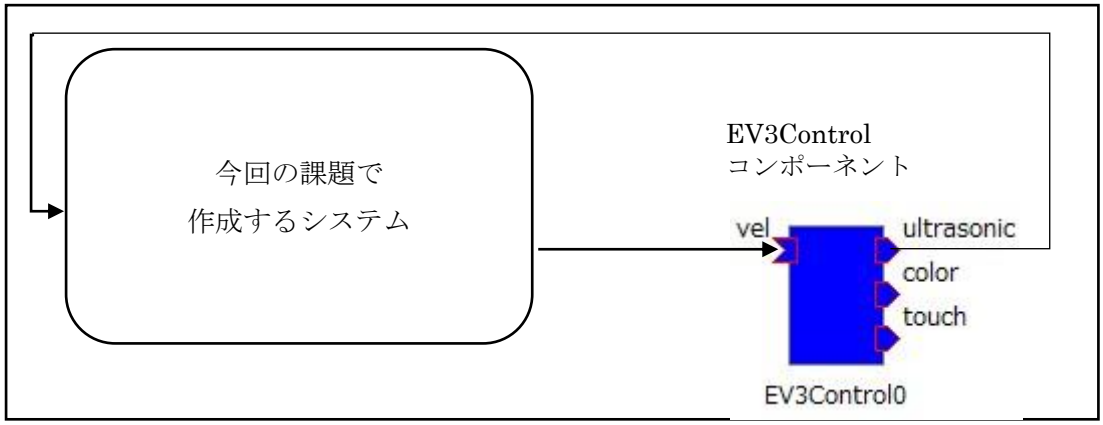
内容

1. システム概要

コース上の配置された障害物を EV3 に接続された超音波センサーで検知し、少し下がり右旋回して障害物を避けて進むシステムを作成してください。通るルートは下記図のルート①です。



下記 EV3Control コンポーネントは EV3 制御用コンポーネントです。機能は、接続されたセンサーの値を各 OutPort から出力し、EV3 のモータを Inport からの入力の値で制御します。センサーの値を受け取りその値を元に EV3 の速度を決定して出力するのが今回の課題です。



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

ポート種類	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

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。

```
struct RangeData
{
    Time tm;
    RangeList ranges;
    RangerGeometry geometry;
    RangerConfig config;
};
```

上記の変数の ranges は double 型の配列です。長さは 1 で 0 番目に超音波センサーの値が入ります。
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。
http://openrtm.org/doc/idl/1.1/idlreference_ja/annotated.html

超音波センサーの値取得のサンプルソース（C++）

```
double ranges;

// 超音波センサーの値が更新された場合
if (m_ultrasonicIn.isNew())
{
        // 超音波センサーの値 読み込み
        m_ultrasonicIn.read();
        // 超音波センサーの値 取得
        ranges = m_ultrasonic.ranges[0];
}
```

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 読み込み
        self._d_ultrasonic = self._ultrasonicIn.read()
        # 超音波センサーの値 取得
        distance = self._d_ultrasonic.ranges[0]
```

2. 超音波センサーの仕様

① 超音波線センサーについて

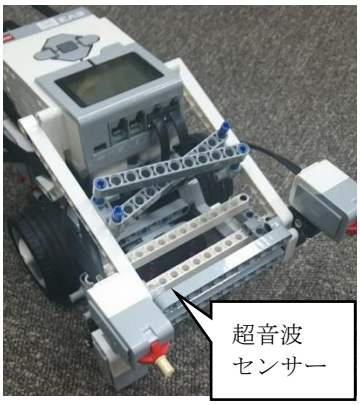
超音波センサーとは超音波を対象物に向け発信し、その反射波を受信することにより、対象物の有無や対象物までの距離を検出する機器です。EV3 の超音波センサーは左の写真の物になります。この超音波センサーを EV3 に接続することにより超音波センサーを使用することが出来る様になります。

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 度
計測制度	+/-1cm

③ 超音波センサーの値の取り方

※EV3 の起動方法については EV3 起動終了手順.docx を参照



EV3 に接続された超音波センサーはどの様に値を取っているのか確認してみましょう。

1)EV3 の電源を入れ、しばらくすると初期画面が表示されます。
2)その後十字キーと中央ボタンで[Device Browser]- [Sensors] - [lego-ev3-us at in3]と選択します。
※[lego-ev3-us at in3]の at in3 はポート番号をさしているので異なる場合があります。
3)選択後下ボタンを押し、[Watch values]を選択してください。
4)超音波センサーで現在取得している値が確認できます。
5)EV3 に接続された超音波センサーの前に手をかざすと値が変化します。

3. 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。
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。

```
struct Velocity2D
 {
    double va; //  角速度  [rad/s]
    double vx; //  並進速度(前方) [m/s]
    double vy; //  並進速度(横方向) [m/s]
 };

struct TimedVelocity2D
{
   Time tm;
   Velocity2D data;
};
```

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 になります。

EV3 を直進させたい場合は、

```
vx = 0.1 ,      va = 0,     vy= 0
```

後進させたい場合は、

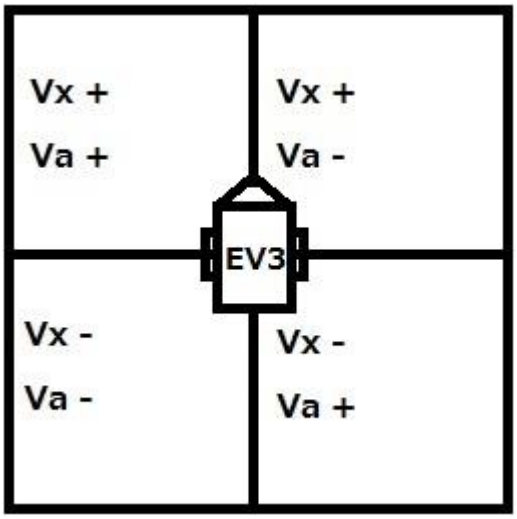
```
vx = -0.1 ,      va = 0,     vy= 0
```

となり、va が 0 の時、vx が＋(プラス)の時各モータは前方に回転し、-(マイナス)の時に後方に回転します。
EV3 を旋回させたい場合は

```
vx = 0 ,      va = 0.1,     vy= 0
```

となり、vx が 0 の時、va は＋(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、-(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をします。

二次元速度ベクトルの値と進行方向の関係は図の様になります。



例えば、vx が+(プラス)で va が+(プラス)だと右モータの値が大きくなるので左に曲がります。

EV3 に与える速度にはモータの性能の関係上値に限りがあります。その値が以下になります。

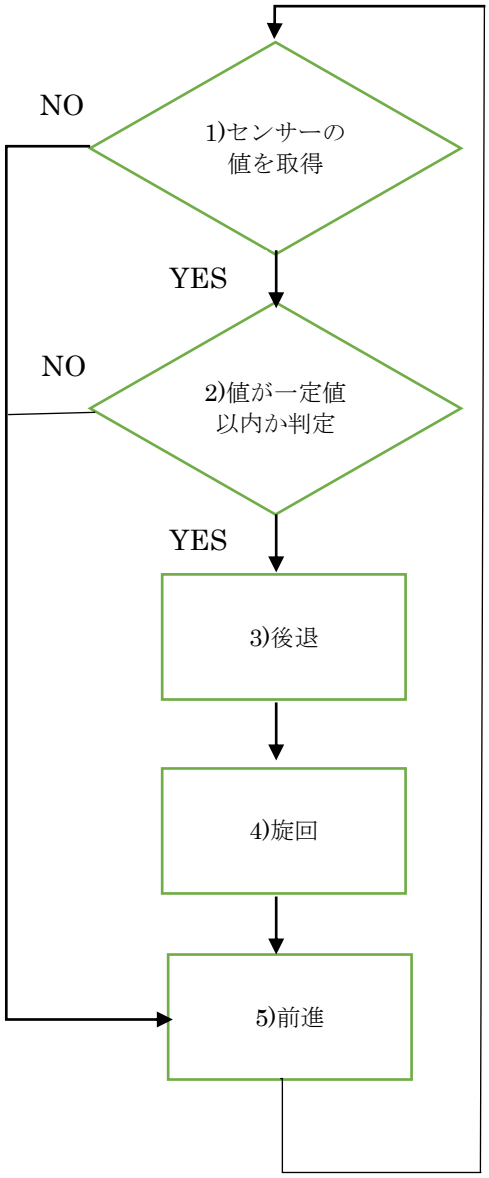
	最小値	最大値
vx	-0.5	0.5
va	-8.5	8.5

この範囲以外の値を入れ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。

4. 作成のヒント

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の様になります。



1)センサーの値の取得の判定。NO なら 5)前進の処理に移行。YES なら 2)の処理に移行。
2)値が一定値以内か判定。NO なら 5)前進の処理に移行。YES なら 3)後退の処理に移行。
3)後退の処理実行。終了後 4)旋回の処理に移行。
4)90 度旋回の処理実行。終了後 5)前進の処理に移行。
5)前進の処理実行。終了後 1)の判定に戻る。
といった様になります。後退や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。

5. 雛型の作成

超音波センサーを一緒に作成します。以下の手順に従って作成します。

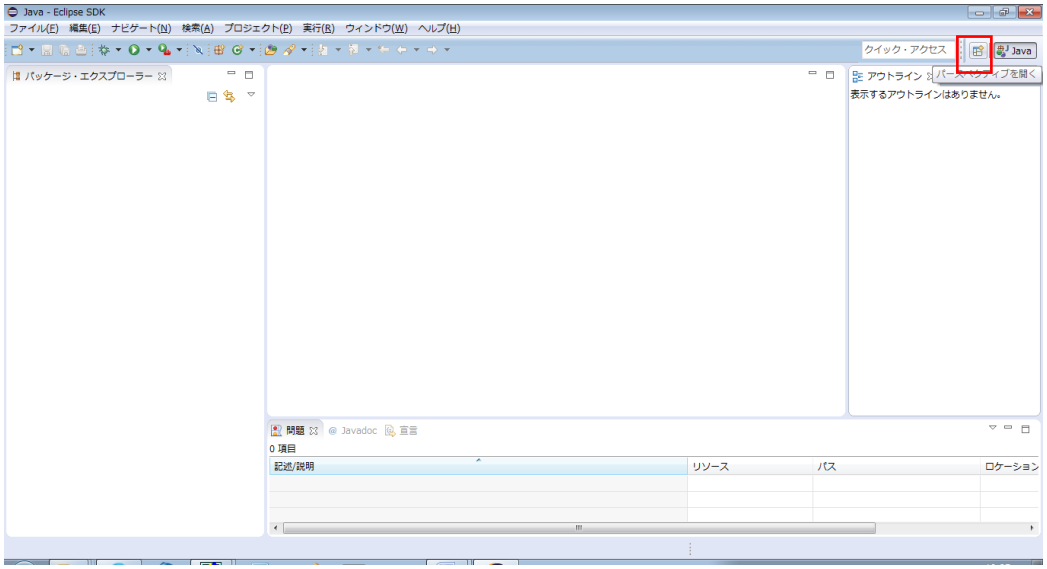
1．RTCBuilder の起動

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 C 直下の下記ディレクトリに保存します。

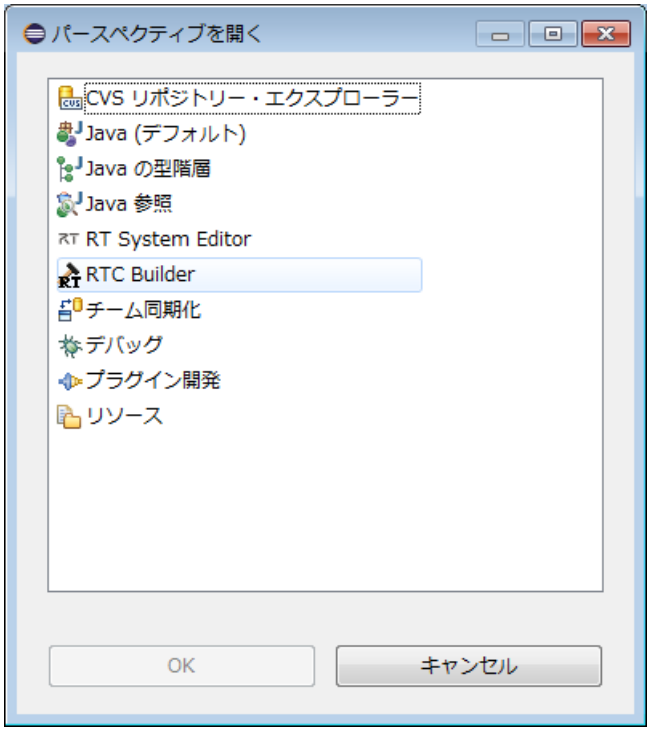
[C:¥rtcws]

最初に起動したとき下記画面がでます。この画面は使用しないので左上の×ボタンを押します。

×ボタンを押す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]をクリックしてください。



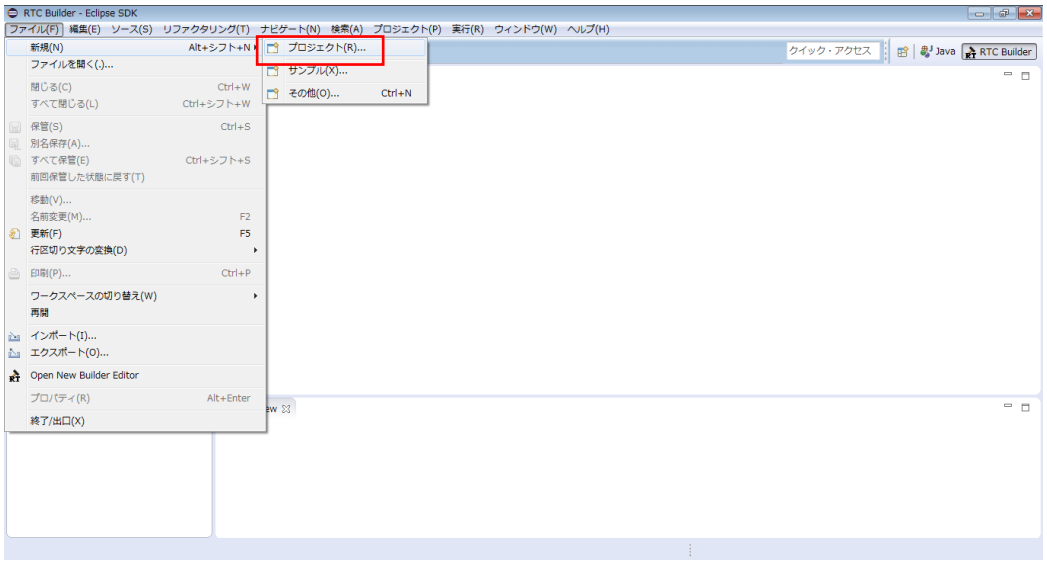
下記ウィンドウが出ますので[RTC Builder]を選択し、「OK」をクリックします。



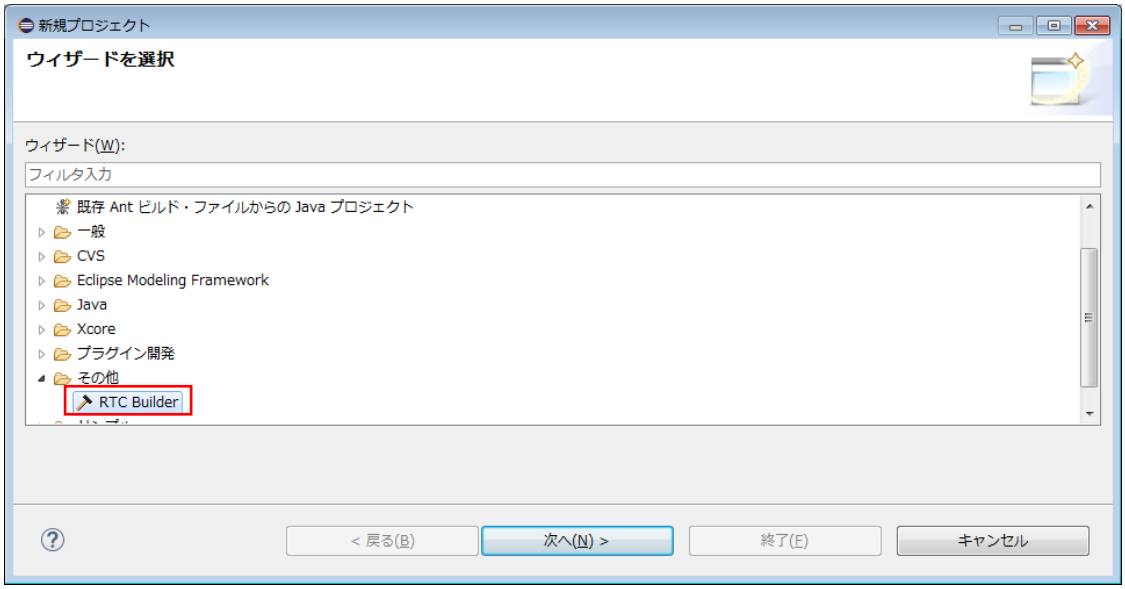
「RTC Builder」を選択することで、RTCBuilder が起動します。メニューバーに「カナヅチと RT」の RTCBuilder のアイコンが現れれば完了です。

2. 新規プロジェクトの作成

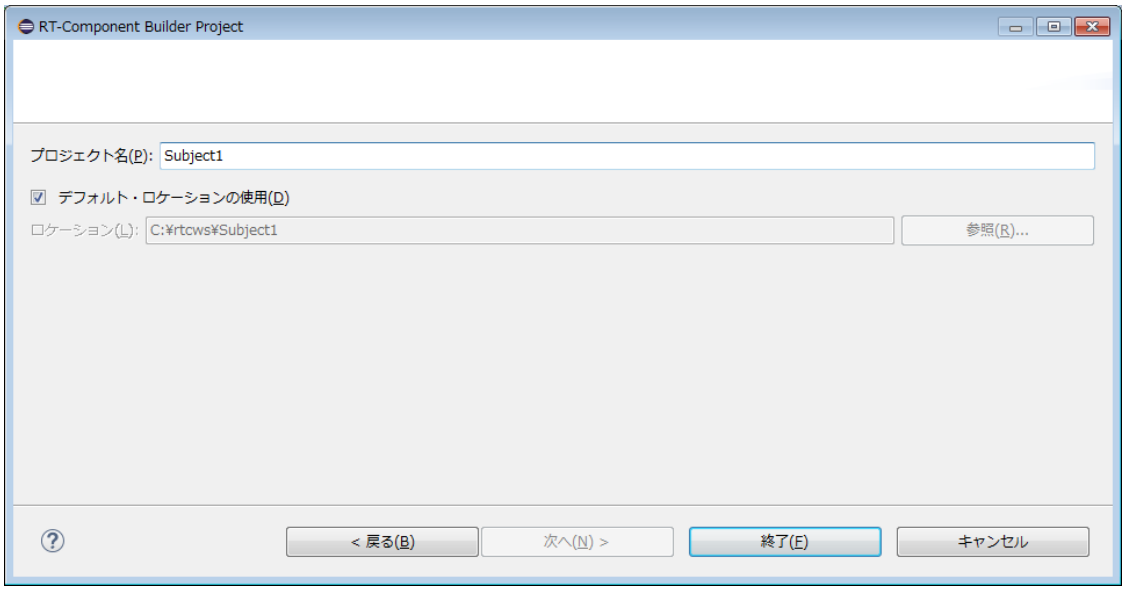
画面上部のメニューから[ファイル]－[新規]－[プロジェクト]を選択



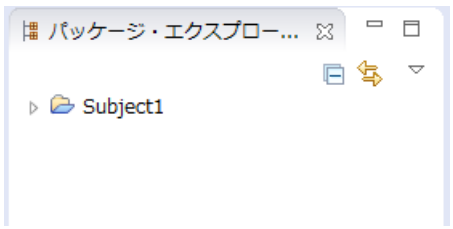
「新規プロジェクト」画面において，「その他」－「RTC Builder」を選択し、「次へ」 をクリック



「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」をクリックします。



下記画面の様にパッケージエクスプローラ内にプロジェクトが追加されれば完了です。

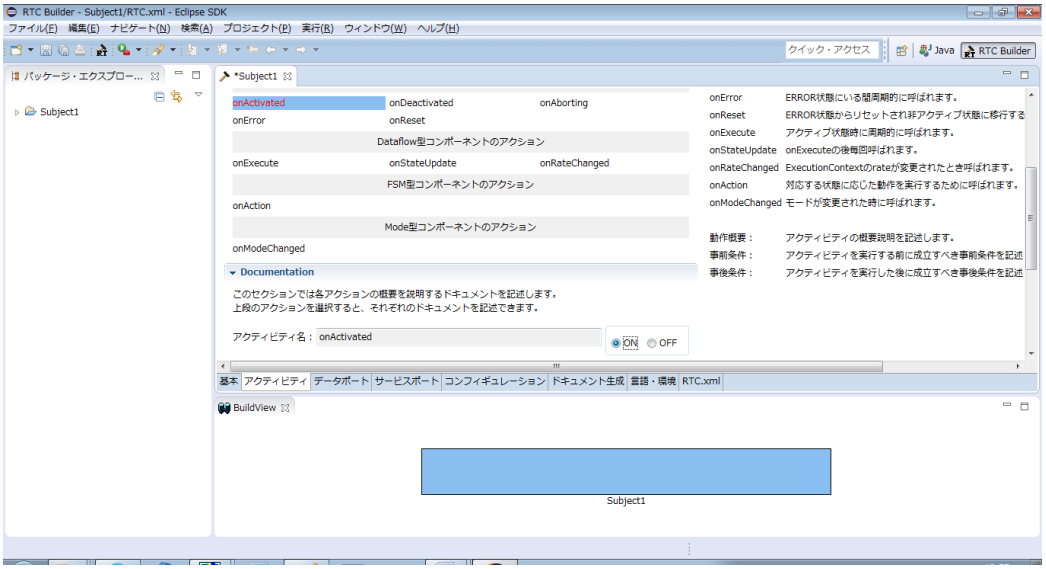


3. プロファイル情報入力とコードの生成

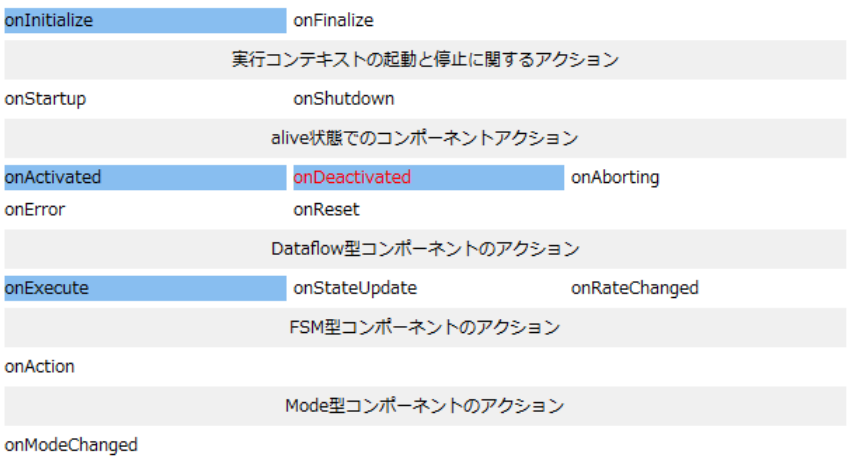
一番左の「基本」タブを選択し、基本情報を設定します。コンポーネントの名前や概要などを記入します。ラベルが赤文字の項目は必須項目です。その他はデフォルトのままで大丈夫です。

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:1000.0

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 onAtivated をクリック後に赤枠のラジオボタンにて"on"にチェックを入れます。onDeactivated,onExecute についても同様の手順を行います。



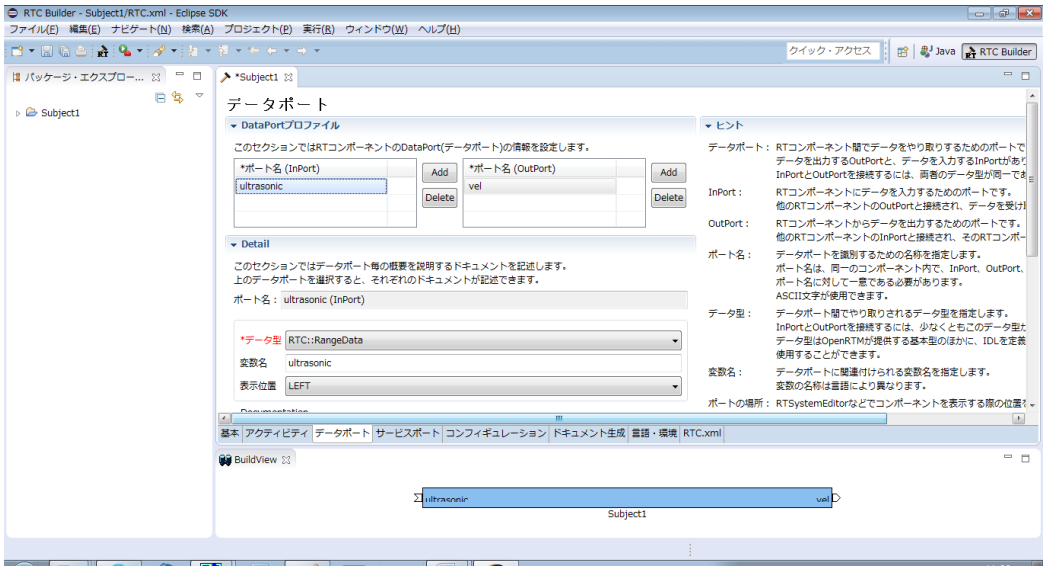
最終的に下図の様になります。



「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。

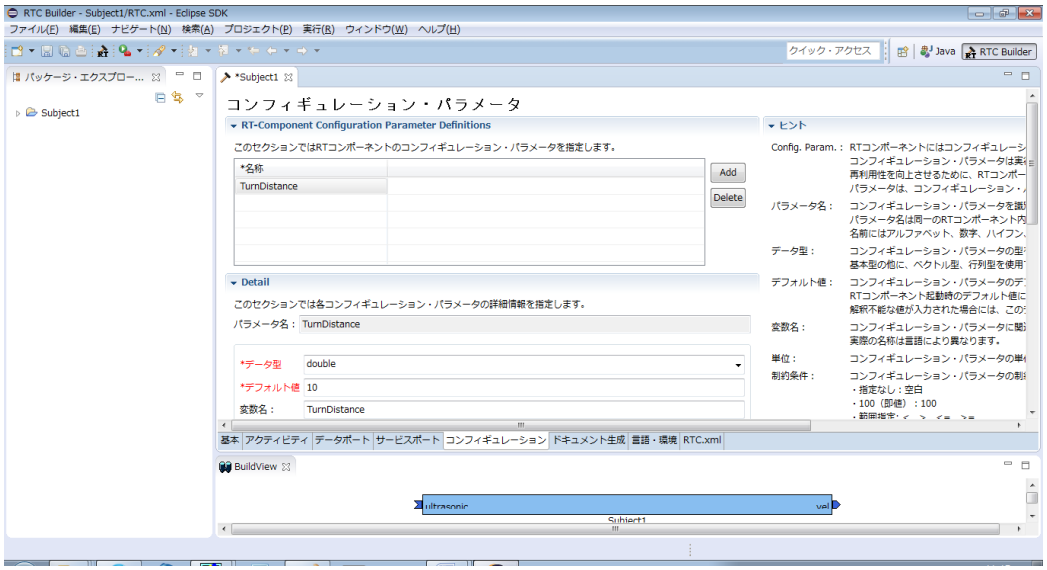
・InPort
ポート名: ultrasonic
データ型: RTC::RangeData
変数名: ultrasonic
表示位置: left

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

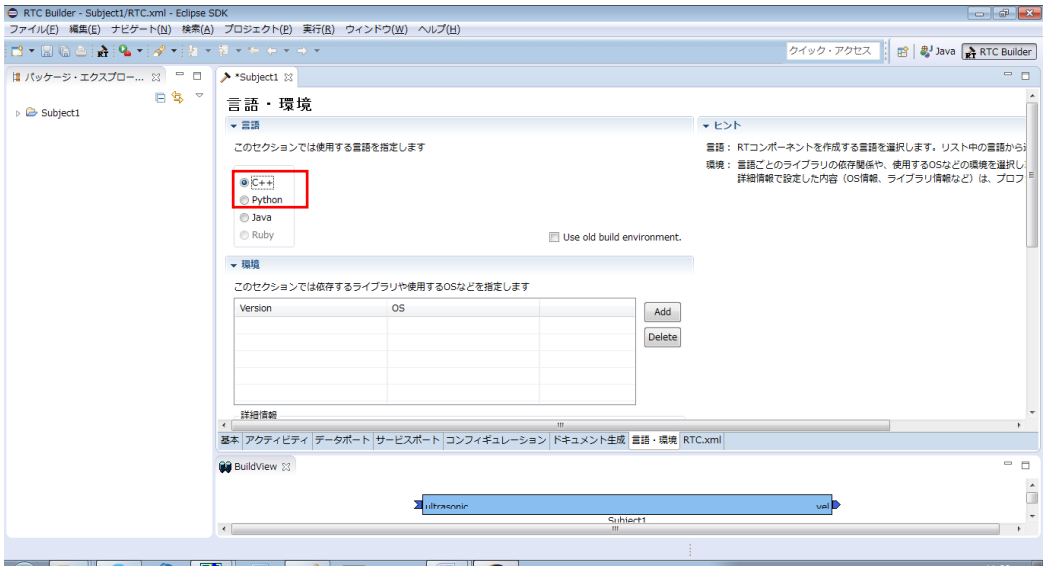


「コンフィギュレーション」タブを選択し、Configuration の情報を入力します。制約条件および Widget とは、RTSystemEditor でコンポーネントのコンフィギュレーションパラメータを表示する際に GUI で値の変更を行うための形式を表すものです。
[Add]ボタンを押して新しいコンフィギュレーションを追加します。
ここでは例として超音波センサーの距離が何 cm で回避動作を行うかの値を追加します。
後で調整したい値を適宜、追加しておくとよいでしょう。

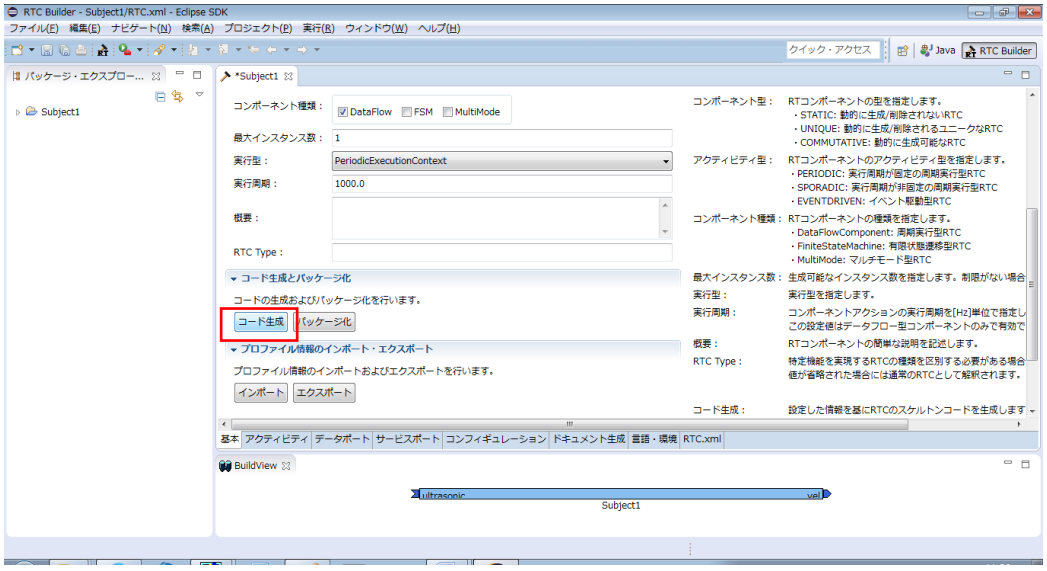
名称: TurnDistance
データ型: double
デフォルト値: 10
変数名: TurnDistance
制約条件: 3<=x<=250
Widget: slider



「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、C++、または 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になりますので、 必ず言語の指定を行うようにしてください。



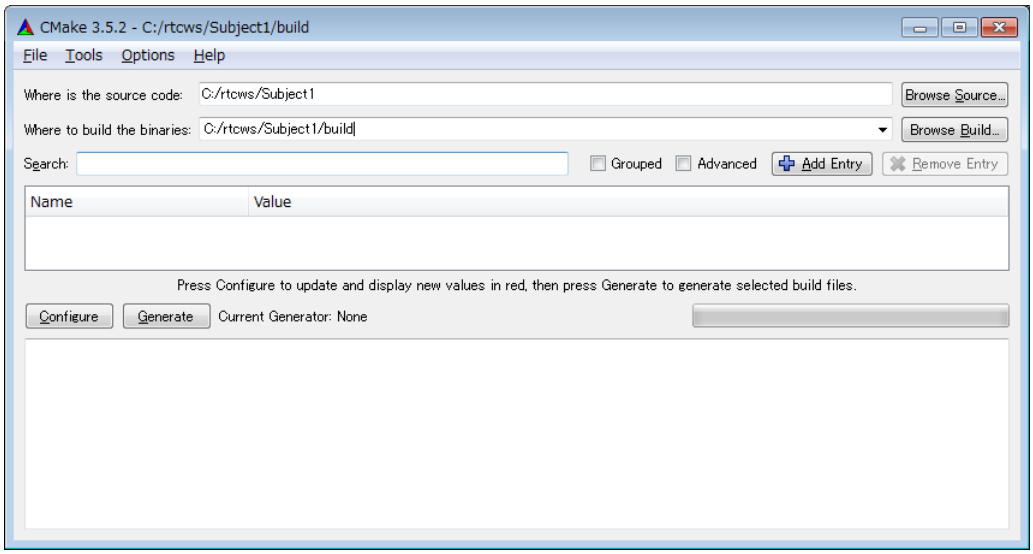
全ての設定が完了しましたら、「基本」タブに戻りコード生成ボタンをクリックします。問題がなければコンポーネントの雛型が生成されます。



4. ソリューションファイルの作成

※ 「言語・環境」タブのプログラミング言語で C++を選択した場合行います。

ここまでの作業で Subject1 コンポーネントの雛型が生成されました。
次の作業として CMake を利用してビルド環境の Configure を行います。
スタートメニューなどから CMake (cmake-gui)を起動します。



画面上部に以下のようなテキストボックスがあります。

・Where is the soruce code

・Where to build the binaries

「Where is the soruce code」に CMakeList.txt が有る場所、「Where to build the binaries」にビルドディレクトリを指定します。

CMakeList.txt はデフォルトでは<ワークス ペースディレクトリ>/ Subject1 になります。

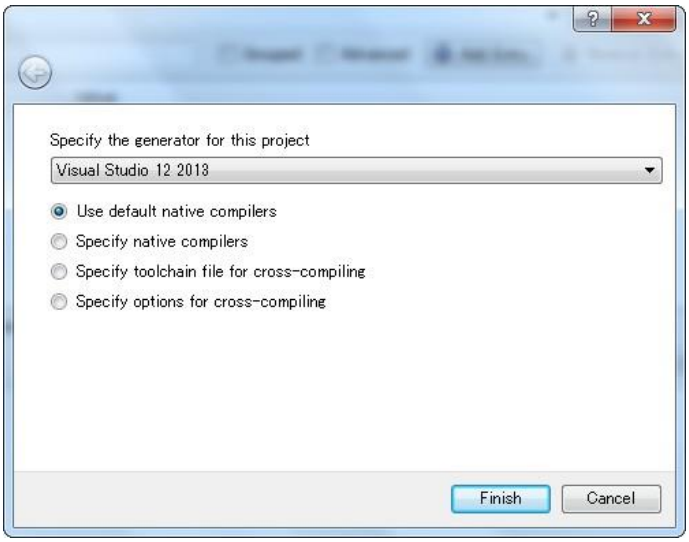
ビルドディレクトリとは、ビルドするためのプロジェクトファイル やオブジェクトファイル、バイナリを格納する場所のことです。場所は任意ですが、この場合 <ワークスペースディレクトリ>/Subject1/build のように分かりやすい名前をつけた Subject1 のサブディレクトリを指定することをお勧めします。

ディレクトリは自動で作成されるので指定前に作成する必要はありません。

今回は以下の様になるはずです。

Where is the soruce code	C:¥rtcws¥Subject1
Where to build the binaries	C:¥rtcws¥Subject1¥build

指定したら、下の Configure ボタンを押します。すると下図のようなダイアログが表示されますので、生成したいプロジェクトの種類を指定します。



Visual Studio バージョン	32/64 bit	生成したいプロジェクトの種類
Visual Studio 2013	32 bit	Visual Studio　12 2013
	64 bit	Visual Studio　12 2013 Win 64
Visual Studio 2015	32 bit	Visual Studio　14 2015
	64 bit	Visual Studio　14 2015 Win 64

ダイアログで Finish を押すと Configure が始まります。問題がなければ下部のログウインドウに Configuring done と出力されますので、続けて Generate ボタンを押します。Generating done と出ればプロジェクトファイル・ソリューションファイル等の出力が完了します。
次に先ほど指定した build ディレクトリの中の Subject1.sln をダブルクリックして Visual Studio を起動します。

6. コンポーネント作成

作成した雛型をもとにコンポーネントを作成します。

1. EV3 が前進

   EV3 を直進させるコンポーネントを作成します。

   [3. 速度の与え方]の所で説明をしていますが、EV3 を直進させるには

   ```
   vx = 0.04 ,      va = 0,     vy= 0
   ```

   というように vx のみに値を入れ、va,vy には 0 を入れます。

   これをプログラムに直すと以下の様になります。

   ・EV3 を前進させるサンプルソース（C++）

   ```
   // 変数に前進の値を代入
   m_vel.data.va = 0;
   m_vel.data.vx = 0.04;
   m_vel.data.vy = 0;

   // アウトポート（vel）に書込み
   m_velOut.write();
   ```

   ・EV3 を前進させるサンプルソース（Python）

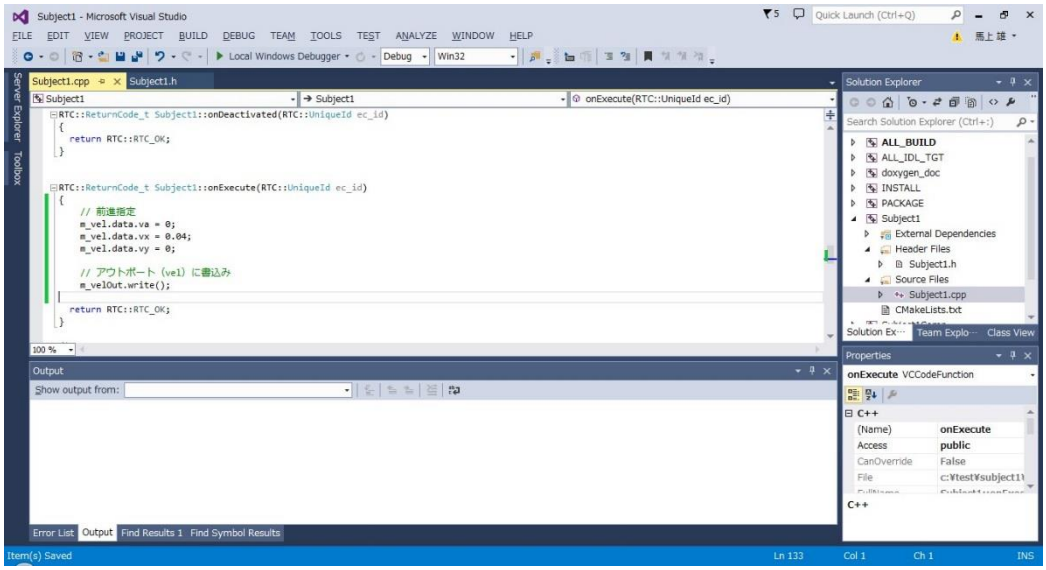
   ```
   # 変数に前進の値を代入
   self._d_vel.data = RTC.Velocity2D(0.04, 0.0, 0.0)

   # アウトポート（vel）に書込み
   self._velOut.write()
   ```

   変数に値を指定して write()でデータを OutPort から送信します。

   上記の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。

   ・C++

   先ほど指定した build ディレクトリの中の Subject1.sln をダブルクリックして Visual Studio を起動します。ソースコードの[onExecute]内にサンプルソースを貼り付けてください。



ソリューションエクスプローラーの ALL_BUILD を右クリックしビルドを選択してビルドします。作成されたコンポーネント(C:¥rtcws¥ Subject1¥build¥src¥Debug¥Subject1Comp.exe)をダブルクリックして起動してください。

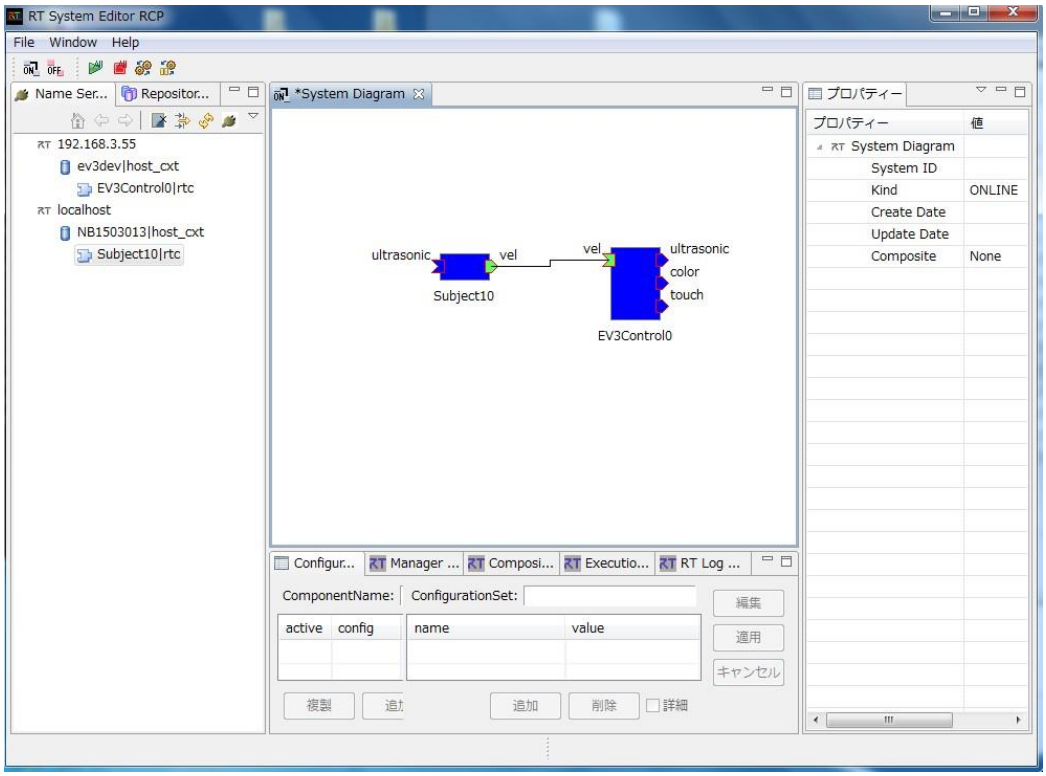
・Python
Subject1.py を開き[def onExecute(self, ec_id):]内にサンプルソースを貼り付けてください。
そして[def __init__(self, manager):]内を以下の様に修正します。

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))

その後 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てください。

EV3 制御用コンポーネントとネームサーバを起動します。起動方法は EV3 起動手順を参照してください。

コンポーネントを起動したら[Start Naming Service]と[RTSystemEditor]を起動しコンポーネントを接続します。



接続後、All Active(緑のアイコン)を押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。

2. EV3 が前進中、目の前に障害物があったら数秒間後退

作成した前進のコンポーネントにコードを追加して、目の前の障害物を感知し 2 秒間後退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。超音波センサーの値を InPort で取得し、値が一定以内になったら、後退の値を OutPort から送信する流れになります。

InPort からデータの値を参照するには以下の手順で行います。

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、その後データを読込

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のようになります。

・InPort のデータを読み込むサンプルソース（C++）

```
double ranges;

// ①超音波センサーの値が更新された場合
if (m_ultrasonicIn.isNew())
{
        // ②超音波センサーの値 読み込み
        m_ultrasonicIn.read();
        // ③超音波センサーの値 取得
        ranges = m_ultrasonic.ranges[0];
}
```

・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 読み込み
        self._d_ultrasonic = self._ultrasonicIn.read()
        # ③超音波センサーの値 取得
        distance = self._d_ultrasonic.ranges[0]
```

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。

2 秒間後退とは後退のデータを送信後 2 秒後に前進のデータを送信すると表すことが出来ます。ここでは sleep 関数を使用して 2 秒後。手順としては以下になります。

① 後退のデータを送信

② 2 秒間コンポーネントを停止(Sleep)

③ 前進のデータを送信

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なります。

・2 秒間後退するサンプルソース（C++）

```
// ①後退指定
m_vel.data.va = 0;
m_vel.data.vx = -0.04;
m_vel.data.vy = 0;
m_velOut.write();
// ②2 秒間コンポーネントを停止
sleep(2000)
// ③前進指定
m_vel.data.va = 0;
m_vel.data.vx = 0.04;
m_vel.data.vy = 0;
m_velOut.write();
```

・2 秒間後退するサンプルソース（Python）

```
# ①後退指定
self._d_vel.data = RTC.Velocity2D(-0.04, 0.0, 0.0)
self._velOut.write()
# ②2 秒間コンポーネントを停止
time.sleep(2);
# ③前進指定
self._d_vel.data = RTC.Velocity2D(0.04, 0.0, 0.0)
self._velOut.write()
```

以上のことを踏まえて、EV3 が前進中目の前に障害物があったら 2 秒間後退するコンポーネントは以下のプログラムで作成します。

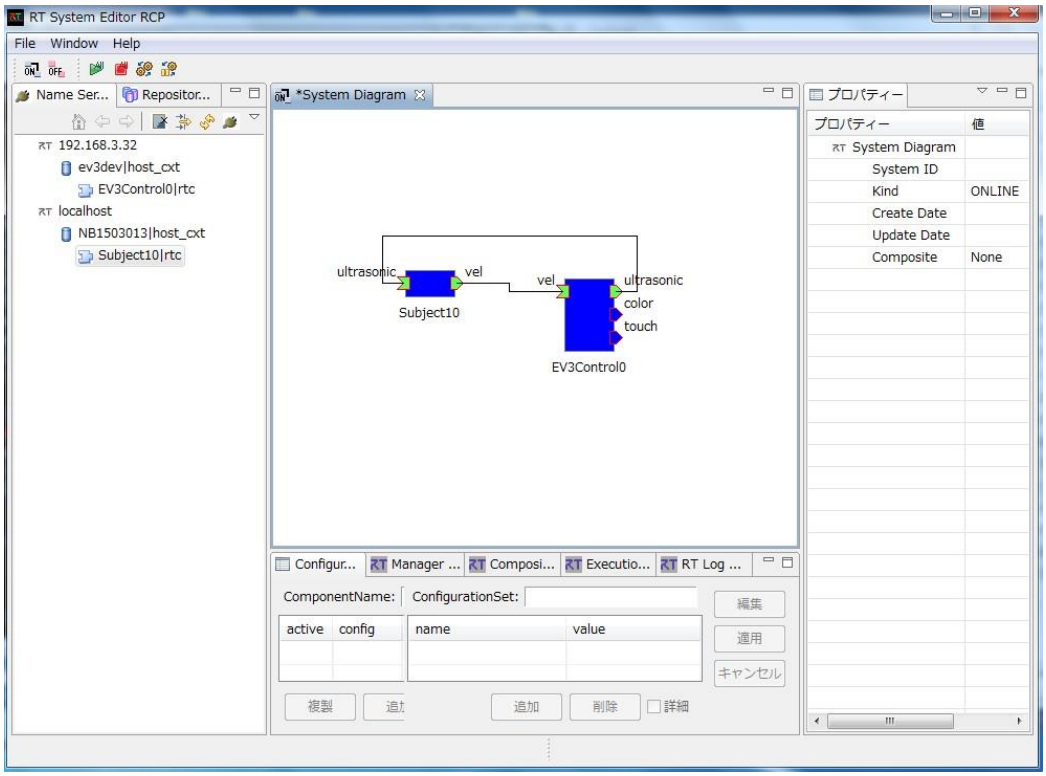
・サンプルソース（C++）

```
m_vel.data.va = 0;
m_vel.data.vx = 0.04;
m_vel.data.vy = 0;

double ranges;
if (m_ultrasonicIn.isNew()){
        m_ultrasonicIn.read();
        ranges = m_ultrasonic.ranges[0];
        if(ranges< m_TurnDistance){
                m_vel.data.va = 0;
                m_vel.data.vx = -0.04;
                m_vel.data.vy = 0;
                m_velOut.write();

                Sleep(2000);

                m_vel.data.va = 0;
                m_vel.data.vx = 0.04;
                m_vel.data.vy = 0;
        }
}
m_velOut.write();
```

・サンプルソース（Python）

```
self._d_vel.data = RTC.Velocity2D(0.04, 0.0, 0.0)

if self._ultrasonicIn.isNew():
        self._d_ultrasonic = self._ultrasonicIn.read()
        distance = self._d_ultrasonic.ranges[0]
        if distance <= self._TurnDistance[0]:
                self._d_vel.data = RTC.Velocity2D(-0.04, 0.0, 0.0)
                self._velOut.write()
                time.sleep(2);
                self._d_vel.data = RTC.Velocity2D(0.04, 0.0, 0.0)

        self._velOut.write()
```

上記のソースコードを使用して実際に試してください。
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)を接続します。



3. EV3 が前進中、目の前に障害物があったら 2 秒間後退後左旋回をする
前に作成した前進のコンポーネントに追加して、EV3 が前進中目の前に障害物があったら 2 秒間後退後左旋回をして再び前進をするコンポーネントを作成します。

[3. 速度の与え方]の所で説明をしていますが、EV3 を旋回させるには

vx = 0 ,　　va = 0.1,　　vy= 0

というように va のみに値を入れ、vx,vy には 0 を入れます。

このことを踏まえて、コンポーネントを作成してみましょう。

7. EV3 のモータの動かし方

モータを動かすにはまず与えられた速度からモータの角速度を求めます。そして角速度からモータが一秒に何度動くか、度毎秒[deg/s]を求めます。求めた度毎秒[deg/s]からモータを一秒間に何カウント動くかの値、カウント毎秒を求め、カウント毎秒を指定するとモータが動きます。以下の内容は EV3 制御用コンポーネント内でのモータを動かすための計算方法です。速度の与え方を考える際に参考にしてください。

1) 角速度の求め方

EV3 制御用コンポーネントは二次元速度ベクトルを受け取り、それぞれのモータを動かします。モータを動かすには角速度を求める必要があり、EV3 制御用コンポーネントの内部ではそれぞれのモータの角速度を計算しています。右車輪の角速度を Wr、左車輪の角速度を Wl とし、車輪の半径を r、中心から車輪までの距離(トレッドの 1/2)を d とすると角速度は以下の計算式になります。

Wr=(vx+va×d)/r Wl=(vx -va×d)/r

なぜこのような計算式になるのかはスライドと下記 URL を参考にしてください。

車輪移動ロボット

http://www.mech.tohoku-gakuin.ac.jp/rde/contents/course/robotics/wheelrobot.html

EV3 の車輪の半径とトレッド幅は以下の値になります。

車輪の半径 r	28mm (0.028m)
トレッド幅 d	59.25mm (0.05925m)

この式に実際に下記表の実際の値を当てはめると次の様になります。

Wr=(vx+va×0.05925)/ 0.028 Wl=(vx-va×0.05925)/ 0.028

これで左右のモータの角速度が求められます。

2) モータの動かし方

ここまででそれぞれのモータの角速度を算出する方法が分かりました。

しかし、EV3 ではモータを動かすには角速度でなく 1 秒間にモータを何カウント動かすかの値を指定して求めます。従って角速度から 1 秒間にモータを何カウント動かすかの値を求める必要があります。

計算方法は角速度から角度、度毎秒[deg/s]を求め、その値に一度ごとのカウント数、毎度カウント[count/deg]をかけます。そうすると一秒間にモータを何カウント動かすか毎秒カウント[count/s]が求められます。

角度を求める式は「角度 ＝ ラジアン * 180 / π」になります。毎度カウントを求める式は[毎度カウント = 1 回転するのに必要なカウント/360 度]で求められます。1 回転するのに必要なカウントは ev3dev の設定により決まっていて値は 360 です。右車輪の度毎秒を Cr、左車輪の度毎秒を Cl とし、1 回転するのに必要なカウントを count としたとき計算式は以下になります。

```
Cr =Wr*(180/π)/( count/360)
Cl =Wl*(180/π) /( count/360)
```

これで各モータの度毎秒が求められます。この値で実際にモータを動かし、EV3 を動かすことが出来ます。

上記の計算式をまとめると

```
Cr =((vx+va×0.05925)/ 0.028)*(180/π)
Cl =((vx-va×0.05925)/ 0.028)*(180/π)
```

となります。

よって 1 秒に 1 回転とは va=0 のとき vx=0.176 前後のときになります。

モータの値ですが-2000～2000 まで設定することが出来ます。しかし、実際に回転速度として反映されるのは 1000 までででそれ以降の値は回転速度に反映されません。