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
1.1 システム概要	1
1.2 コンポーネント概要	2
1.2.1 入出力ポート	
1.2.2 超音波センサーのデータ型	
1.2.3 サンプルソース	
2 超音波センサーの仕様	
2.1 超音波センサーについて	
2.2 超音波センサーの性能	
2.3 超音波センサーの値の取り方	
 3 速度の与え方 	
3.1 データ型	
3.2 值	
3.3 値の制限	
4 作成のヒント	7
4.1 システムのアルゴリズム	7
5 雛型の作成	
5.1 RTCBuilder の起動	
5.1.1 ようこそ画面	
5.1.2 パースペクティブを開く	
5.1.3 新規プロジェクトの作成	
5.1.4 プロファイル情報入力とコードの生成	
5.1.5 コード生成	
5.2 ソリューションファイルの作成	
5.2.1 CMake	
6 EV3 の前進コンポーネントの作成	
6.1 EV3 を前進させるサンプルソース(C++)	
6.2 EV3 を前進させるサンプルソース(Python)	
6.3 前進コンポーネントの作成	
6.3.1 C++	
6.3.2 Python	
6.3.3 動作確認	
7 障害物を感知し数秒間後退するコンポーネントの作成	
7.1 InPort からデータの値を参照する手順	
7.2 2秒間後退の手順	
7.3 障害物を感知し2秒間後退するサンプルソース	
7.3.1 サンプルソース (C++)	
7.3.2 サンプルソース (Python)	

	7.3.3	動作確認	
8	EV3 が	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コンポージ	ネントの作
成	26		
8	5.1 EV	√3 のモータの動かし方	
	8.1.1	角速度の求め方	
	8.1.2	モータの動かし方	

※ 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 システム概要

1.1 システム概要

コースに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。



-1-

1.2 コンポーネント概要

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。



1.2.1 入出カポート

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

1.2.2 超音波センサーのデータ型

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。単位は m(メートル)です。

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

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(C++)



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

2.1 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。

2.2 超音波センサーの性能

計測可能範囲	$3 \text{cm} \sim 250 \text{cm}$
計測角度	約 20 度
計測制度	+/-1cm

2.3 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」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 の電源を入れます。しばらくすると初期画面が表示されます。
(2) +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。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。

3.1 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;
};
```

この型に値を与えて EV3 のモータを動かします。

3.2 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 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。

3.3 値の制限

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

4.1 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。

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。

5.1 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]

5.1.1 ようこそ画面

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。

ر چ	ava - Eclipse	SDK		
ファ	イル(E) 編集	(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(<u>W</u>) ヘルプ(<u>H</u>)	
	🚳 ようこそ	22		🏠 🗇 🔿 🖍 🛣 🖽 🖷 🖷
& J				『 や ワークベンチ
		Eclipse へようこそ		
	C C	概要 フィーチャーの概要	チュートリアル チュートリアルの実行	
	Ŷ	サンプル サンプルの航行	新機能 新機能について	
	()	マイグレーション 新規リリースへのマイグレーション		

5.1.2 パースペクティブを開く

×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。

Java - Eclipse SDK					
ファイル(E) 編集(E) ナピゲート(N) 検索(A) プロジェ	クト(<u>P</u>) 実行(<u>R</u>) ウィンドウ(<u>W</u>) ヘルプ(<u>H</u>)				_
📑 • 🗏 🕼 🗁 🕸 • O • 💁 • 🔀 🖶 🛛 •	@ A + 2 + 2 + 5 + + + + + +			クイック・アクセス	🐉 Java
😫 パッケージ・エクスプローラー 🛛 👘 🗖			(腟 アウトライン Σパースペク	ティブを開く
E 😫 🔻			-	表示するアウトラインはありまt	さん。
	■ 問題 ☆ @ Javadoc Q 車画				000
	0 4910 02:00 /04.88	111/-7	187		06-2-22
	86,822/87,473	99-X	~~~		07-232

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。

● パースペクティブを開く	- • •
 CVS リボジトリー・エクスプローラー Java (デフォルト) Java の型階層 Java 参照 AT RT System Editor RTC Builder デーム同期化 デバッグ プラグイン開発 リソース 	
ОК ≠1	マンセル

「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。

5.1.3 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。

「ァイル(F) 編集(E) ソース(S)	リファクタリング(T)	+ビゲート(≀	N) 検索(A)	プロジェクト(F	9) 実行(R)	ウィンドウ(W)	ヘルプ(H)				
新規(N)	Alt+シフト+N ▶	📑 วีอร	ジェクト(R)						クイック・アクセン	va 🗄 😰 🔝 🖉	a 🔉 RTC Bu
ファイルを聞く(.)			υμ(x)								(KI
閉じる(C)	Ctrl+W		100	Challen							
すべて閉じる(L)	Ctrl+シフト+W	-2010	3(0)	CUI+N							
保管(S)	Ctrl+S										
別名保存(A)											
すべて保管(E)	Ctrl+シフト+S										
前回保管した状態に戻す(T)											
移動(V)											
名前変更(M)	F2										
更新(F)	F5										
行区切り文字の変換(D)	+										
印刷(P)	Ctrl+P										
ワークスペースの切り替え(W) 車朝	÷										
()											
1 フルート(I) エクスポート(O)											
Open New Builder Editor											
open wew builder curtor											
フロパティ(R)	Alt+Enter	ew 23									-

「新規プロジェクト」画面で「その他」-「RTC Builder」を選択し、「次へ」をクリックします。

	- • •
ウィザードを選択	
ウィザード(<u>W</u>):	
ጋብሥቃλ <u>ታ</u>	
 ※ 既存 Ant ビルド・ファイルからの Java プロジェクト ▷ → 一般 ▷ ← CVS ▷ ← Eclipse Modeling Framework ▷ → Java ▷ → Xcore ▷ → プラグイン開発 △ ← その他 ▲ RTC Builder 	E
(2) (2) 次へ(N) > 終了(E) キャ	ッンセル

「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。

RT-Component Builder Project				
プロジェクト名(P): Subject1				
▼ デフォルト・ロケーションの使用	D)			
 □ケーション(L): C:¥rtcws¥Subject	1			参照(<u>R</u>)
?	< 戻る(<u>B</u>)	次へ(N) >	終了(E)	キャンセル

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。

パッケージ・エクスプロー	×		
	E	€ ₽	\bigtriangledown
👂 🗁 Subject1			

5.1.4 プロファイル情報入力とコードの生成

5.1.4.1 基本

ー番左の「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

C RTC Builder - Subject1/RTC.xml - Eclipse S	SDK					
ファイル(E) 編集(E) ナビゲート(N) 検索(A)) プロジェクト(<u>P</u>) 実行(<u>P</u>) ウィンドウ(<u>W</u>) ヘルプ(<u>H</u>)				
🖻 • 🗄 🕲 🏠 💁 • 🛷 • 🖄 •	$[1] \star \phi \leftrightarrow \star \phi \star$				クイック・アクセス 😰 🐉 Java 💦 RTC Builder	
🏼 パッケージ・エクスプロー 🛛 🖳 🗆	אי Subject1 מ					
E 🕏 ▽	基本				ń	
				* 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。			モジュール名:	RTコンポーネントを識別する名前を指定します。	
	*モジュール名: Subject1				この名称はコンポーネントのペースインスタンス名にも使 使用できる文字はアルファベット、数字、ハイフン、アン	
モジュール概要:		Subject1 component		モジュール概要:	RTコンポーネントが提供する機能の概要を入力します。	
	*バージョン:	1.0.0			ASCII文字が使用できます。	
	*ベンダ名:	Aizu		バージョン:	RTコンボーネントのバージョンを指定します。 x.y.z(x,y,zは数字)の形式で入力してください。	
	*モジュールカテゴリ:	Category	•	ベンダ名:	RTコンボーネントを作成した作者名、ペンダ名を指定しま ASCII文字が使用できます。	
	コンポーネント型:	STATIC	•	モジュールカテゴリ:	RTコンポーネントのカテゴリを入力します。	
	アクティビティ型:	PERIODIC	-		選択肢にない場合は任意の力テゴリ名を入力することがで 使用できる文字は、アルファベット、数字、ハイフン、ア	
	コンポーネント種類:	DataFlow FSM MultiMode		コンポーネント型:	RTコンボーネントの型を指定します。 ・STATIC:動的に生成/削除されないRTC ・UNIQUE:動的に生成/削除されるコニークなRTC	
	<			(F F	
	基本 アクティヒティ テー	-9ホート サービスホート コンフィチュレー	ション ドキュメント生成 言語・環境	RTC.xml		
	BuildView 🖂					
			C. Martine			
			Subject1			

5.1.4.2 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、onActivated0,onDeactivated0,onExecute0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。

RTC Builder - Subject1/RTC.xml - Eclipse S	DK				
ファイル(E) 編集(E) ナピゲート(N) 検索(A)	プロジェクト(<u>P</u>) 実行(<u>R</u>) '	ウィンドウ(型) ヘルプ(且)			
i 📬 🕶 🔝 🕒 🛃 💊 🕶 🛷 🕶 🖄 👻	$[2] \star \leftarrow \leftarrow \star \Rightarrow \star$				クイック・アクセス 😰 🖏 Java 😭 RTC Builder
😫 パッケージ・エクスプロー 😫 🖳 🗆	➤*Subject1 🔀				
⊨ 🤤 ⊽ ⊳ 🗁 Subject1	onActivated	onDeactivated onReset Dataflow型コンポーネントのアクショ	onAborting ■ン	onError onReset onExecute onStateUpdate	ERROR状態にいる間周期的に呼ばれます。 ^ ERROR状態からリセットされ非アクティブ状態に移行する アクティブ状態時に周期的に呼ばれます。 onExecuteの後毎回呼ばれます。
	onExecute	onStateUpdate FSM型コンポーネントのアクション	onkatechanged	onRateChanged onAction onModeChange	ExecutionContextのrateが変更されたとき呼ばれます。 対応する状態に応じた動作を実行するために呼ばれます。
	onAction	Mode型コンボーネントのアクション	>	動作概要: 事前条件·	E アクティビティの概要説明を記述します。 アクティビティの概要説明を記述します。
	▼ Documentation このセクションでは各アクラ 上段のアクションを選択する	ションの概要を説明するドキュメントを記述し ると、それぞれのドキュメントを記述できます	事後条件:	アクティビティを実行した後に成立すべき事後条件を記述	
	アクティビティ名: onActi	vated	● <u>ON</u> ○ OFF		
	< 基本 アクティビティ データオ	ペート サービスボート コンフィギュレーショ	… ヨン ドキュメント生成 言語・環境 RT	C.xml	•
	😝 BuildView 🖾				
			Subject1		

最終的に下図の様になります。

onInitialize	onFinalize	
実行二	コンテキストの起動と停止に関するアク	フション
onStartup	onShutdown	
ā	live状態でのコンポーネントアクショ	>
onActivated	onDeactivated	onAborting
onError	onReset	
C	Dataflow型コンポーネントのアクショ	>
onExecute	onStateUpdate	onRateChanged
	FSM型コンポーネントのアクション	
onAction		
	Mode型コンポーネントのアクション	,

onModeChanged

5.1.4.3 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]ボタンを押して新しいデータポートを追加します。

• InPort		• OutPo	rt	
ポート名: ultrasor	nic	ポート名	: vel	
データ型: RTC::RangeData データ型			: RTC::TimedVelocity2D	
変数名:ultrasonic 変数名:			el	
表示位置: left		表示位置	: right	
○ RTC Builder - Subject1/RTC.xml - Eclipse S ファイル(E) 編集(E) ナビゲート(N) 検案(A)	iDK ・ プロジェクト(E) 実行(E) ウィンドウ(W) ヘルプ(H)			
📑 🔹 🕼 🗁 🏄 💁 🔹 🖉 🔹 🖄	習ょすかりや		クイック・アクセス 😰 🐉 Java 💽 RTC Builder	
🛿 パッケージ・エクスプロー 😫 🕒 🗆	➤ *Subject1 🙁			
E 🕏 🗢	データポート		ń.	
v 🖉 Subject	▼ DataPortプロファイル		* E>P	
	このセクションではRTコンボーネントのDataPort(データボ *ボート名(inPort)	ート)の情報を設定します。	データポート: RTコンポーネント間でデータをやり取りするためのポートで データを出力するOutPortと、データを入力するInPortがあり	
	ultrasonic Vel	Delete	InPortとOutPortを接続するには、両者のテータ生が同一で3 EnPort: RTコンポーネントにデータを入力するためのポートです。 他のRTコンポーネントのDutPortと提続され、データを受け	
	- Datail		OutPort: RTコンポーネントからデータを出力するためのポートです。 他のRTコンポーネントのInPortと接続され、そのRTコンポー	
	・ 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 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 サームの たい クローン	メントを記述します。 記述できます。	ポート名: データボートを識別するための名称を指定します。 ポート名は、同一のコンボーネント内で、InPort、OutPort、 ポート名に対して一意である必要があります。 ASCI11空本が由まできます。	
	ホート名: ultrasonic (InPort)		データ型: データボート間でやり取りされるデータ型を指定します。 InPortとOutPortを接続するには、少なくともこのデータ型た	
	*デーク型 RTC::RangeData	•	データ型はOpenRTMが提供する基本型のほかに、IDLを定義 使用することができます。	
	変数名 ultrasonic 表示位置 LEFT	•	変数名: データボートに関連付けられる変数名を指定します。 変数の名称は言語により異なります。	
	Dosumontation		ポートの場所: RTSystemEditorなどでコンポーネントを表示する際の位置? -	
	▲ アクティビティ データポート サービスポート コンフィ	''' ギュレーション ドキュメント生成 言語・環境 R'	IC.xml	
	BuildView 😫			
			h	
	∑ ultrasonic	Subject1	val D	

5.1.4.4 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

C RTC Builder - Subject1/RTC.xml - Eclipse SD	Ж		
ファイル(E) 編集(E) ナビゲート(N) 検索(A)	プロジェクト(E) 実行(B) ウィンドウ(W) ヘルプ(E)		
📑 🔹 🗟 🔮 💁 🖓 🔹 🖉 🔹	マイクタイシャ	クイック・ア	7クセス 📄 🖏 Java 🔒 RTC Builder
🖺 パッケージ・エクスプロー 🛛 🔍 🗆	➤*Subject1 🙁		- 0
🕞 🚖 🤝	コンフィギュレーション・パラメータ	- KNK	
	• KI-component comparation Parameter bennitions	▼ E2F	
	このセクションではRTコンボーネントのコンフィギュレーション・パラメータを指定します。 *名称 *名称 TumDistance Add	Config. Param.	: RTコンボーネントにはコンフィギュレーシ コンフィギュレーション・/ラメータは実(= 再利用性を向上させるために、RTコンボー パラメータは コンフィギュレーション・/
	Delete	パラメータ名:	コンフィギュレーション・パラメータを識 パラメータ名は同一のRTコンポーネント内 名前にはアルファペット、数字、ハイフン、
		データ型:	コンフィギュレーション・パラメータの型 [;] 基本型の他に、ベクトル型、行列型を使用
	✓ Detail このセクションでは各コンフィギュレーション・パラメータの詳細情報を指定します。	デフォルト値:	コンフィギュレーション・パラメータのデ RTコンポーネント起動時のデフォルト値に 解釈不能な値が入力された場合には、この
	パラメータ名: TumDistance	查数名:	コンフィギュレーション・パラメータに関う 実際の名称は言語により異なります。
	*データ型 double -	単位:	コンフィギュレーション・パラメータの単1
	*デフォルト値 10	制料樂件:	コンフィギュレーション・バラメータの制 ・指定なし:空白 ・100 (PD#1) + 100
	変数名: TurnDistance		・100 (phe) 、100 ・範囲指定・< > <= >= *
	▼ III 「「「」」 「「」」 「「」」 「」」 「」」 「」」 「」」 「」」 「」		•
	24 7974 E74 7 -9/1-1 9-EX/1-1 1224 410-232 1141 X21 124 136 466 KICXIII		
	BuildView SS		
		vel	<u> </u>
	Subject		-
	· · · · · · · · · · · · · · · · · · ·		

5.1.4.5 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、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。

C RTC Builder - Subject1/RTC.xml - Eclipse S	SDK						
ファイル(E) 編集(E) ナビゲート(N) 検索(A)) プロジェクト(<u>P</u>) 実行(<u>R</u>) ウ·	ィンドウ(<u>W)</u> ヘルプ(<u>H</u>)					
📑 🔹 🕼 🗢 🌺 💁 🖌 🖉 🔹 🖄	$[0] \bullet f = (0, + 1) \bullet \bullet \bullet$			クイック・アクセス 🔡 🖏 Java 🔒 RTC Builder			
🎁 パッケージ・エクスプロー 🛛 🖳 🗆	≯*Subject1 ☆						
🕞 😫 🤝	言語・環境						
	* Ξ:#			* ESP			
	このセクションでは使用する言語を指定します ● C+++ ● Python ● Java ● Ruby Use old build environment.			言語: KTコンボーネントを作成する言語を選択します。リスト中の言語から3 環境: 言語にくのライブランの伝知解除: 使用するのとなの環境を選択し、 詳細環報で設定した内容(OSI領稿、ライブラリ情報など)は、プロフ ^Ⅱ			
	- 151A						
	このセクションでは広告する	ライブラリや使用するのSthドを指定し	≠ 7				
	Version	05	~,				
			Add				
			Delete				
	☆詳細情報						
	基本 アクティビティ データボート 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 富語・環境 RTC.xml						
	BuildView 🛛						
		Ultrasonic					
	•	<u> </u>	Subject1	·			

5.1.5 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。

RTC Builder - Subject1/RTC.xml - Eclipse S	SDK			
ファイル(E) 編集(E) ナビゲート(N) 検索(A)) プロジェクト(<u>P</u>) 実行(<u>F</u>	 ウィンドウ(W) ヘルプ(H) 		
📑 • 🗉 🕼 🗠 🚵 💁 • 🖋 • 🛃 •	$\mathbb{S} \star \leftarrow \leftarrow \star \Rightarrow \star$			クイック・アクセス 🔛 🐉 Java 🔒 RTC Builder
📱 パッケージ・エクスプロー 😫 🖳 🗆	≯ *Subject1 ☆			
🕒 😓 Subject1	コンボーネント種類:	☑ DataFlow ■ FSM ■ MultiMode	コンポーネント型:	RTコンポーネントの型を指定します。 ・STATIC:動的に生成/削除されないRTC ・INIDUE:動的に生成/削除されるコニークがRTC
	最大インスタンス数:	1		· COMMUTATIVE: 動的に生成可能なRTC
	実行型:	PeriodicExecutionContext	アクティビティ型:	RTコンボーネントのアクティビティ型を指定します。
	実行周期:	1000.0		
	极要:		コンポーネント種類:	RTコンポーネントの種類を指定します。 ・DataFlowComponent: 周期実行型RTC
	RTC Type :			 FiniteStateMachine: 有限状態遷移型RTC MultiMode: マルチモード型RTC
	▼ コード生成とパッケー	-ジ化	最大インスタンス数:	生成可能なインスタンス数を指定します。制限がない場合
	コードの生成およびパ	ッケージ化を行います。	実行型:	実行型を指定します。
	コード生成 パッケー	-SAF	実行周期:	コンポーネントアクションの実行周期を[Hz]単位で描定し この設定値はデータフロー型コンポーネントのみで有効で
	▼ プロファイル情報の-	(ンポート・エクスポート	概要:	RTコンポーネントの簡単な説明を記述します。
	プロファイル情報のイン	ッポートおよびエクスポートを行います。	RTC Type :	特定機能を実現するRTCの種類を区別する必要がある場合 値が省略された場合には通常のRTCとして解釈されます。
	インポート エクスノ	K- 1-		
			コード生成:	設定した情報を基にRTCのスケルトンコードを生成します 👻
		 	DTC und	۶.
	* POTAETA 7	-タバート リービスバート コンフィキュレーション トキュメント生成 言語・環境	KTC.XIIII	
	BuildView 🔀			
				vel
		Subject1		

5.2 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。

5.2.1 CMake

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。

A CMake 3.5.2 - C:/rtcws/	Subject1/build			
<u>File Tools Options He</u>	elp			
Where is the source code: 0	:/rtcws/Subject1			Browse Source
Where to build the binaries: C	C:/rtcws/Subject1/build			▼ Browse <u>B</u> uild
S <u>e</u> arch:			Grouped C Advanced	Add Entry
Name	Value			
Press	Configure to update and displa	ay new values in red, then pre	ss Generate to generate sele	cted build files.
<u>C</u> onfigure <u>G</u> enerate	Current Generator: None			

5.2.1.1 ディレクトリの指定

画面上部にテキストボックスがあります。

- $\boldsymbol{\cdot}$ Where is the soruce code
- $\boldsymbol{\cdot}$ 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

5.2.1.2 Configure とプロジェクトの種類の指定

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。

pecity the generator	for this project		
Visual Studio 12 201	3		
Use default native	e compilers		
Specify native co	mpilers		
🔵 Specify toolchain	file for cross-compil	ling	
Specify options for	or cross-compiling		

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。

5.2.1.3 Generate

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。

5.2.1.4 (参考) Visual Studio の起動

先ほど指定した build ディレクトリの中の Subject1.sln をダブルクリックすると Visual Studio が起動します。

6 EV3 の前進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。

6.1 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();

6.2 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。

6.3 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。

6.3.1 **C++**

CMake で指定した 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。

6.3.2 **Python**

Subject1.py を開き[def on Execute(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))
```

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。

6.3.3 動作確認

6.3.3.1 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。起動方法は EV3 起動手順を参照してください。

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

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。

7 障害物を感知し数秒間後退するコンポーネントの作成

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。

7.1 InPort からデータの値を参照する手順

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++)

・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[read0]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[isNew0]が必要になります。

7.2 2秒間後退の手順

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ことで表すことが出来 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 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を作成します。

7.3 障害物を感知し2秒間後退するサンプルソース

7.3.1 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){</pre>
              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();
```

```
7.3.2 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()</pre>
```

7.3.3 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。

RT System Editor RCP	_	_	_		
File Window Help					
6R 6FL 🖻 💣 🤐 🔐					
🏄 Name Ser 👘 Repositor 🖓 🗖	_{ดีพ} ิ *System Diagram	8	- 8	□ プロパティー	~
🖞 🗘 🗘 🔛 🛣 🔅 🥔 🎽 🎽				プロパティー	値
RT 192.168.3.32				ਸਾ System Diagram	
<pre>ev3dev host_cxt</pre>				System ID	
EV3Control0 rtc				Kind	ONLINE
RT localhost				Create Date	
NB1503013[host_cxt			7	Update Date	-
2 Subject10/rtc	ultrasomic_	vel		Composite	None
			ultrasonic		
	S	Subject10	touch		
			couch		
		EV3Control0)		
	Configur KT Ma	anager 🐹 Composi 🔭 Execu	tio 🕅 RT Log 🗖 🗖		
	Componentivame:	ConfigurationSet:	編集		
	active config	name value	適用		
			キャンセル		
	複製 追加	追加 削除	□詳細		
			-	· ·	F

8 EV3 が前進中、目の前に障害物があったら2 秒間後退

後左旋回をするコンポーネントの作成

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]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。

8.1 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。

8.1.1 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

Wr=(vx+va×d)∕r		
Wl=(vx-va×d)/r		

算式になります。

Г

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。

8.1.2 **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 = $Wr^*(180/\pi)/(count/360)$ Cl = $Wl^*(180/\pi)/(count/360)$

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まででそれ以降の値は回転速度に反映されません。