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

 ${\ensuremath{\mathbb C}}$ 2018 The University of Aizu

目次

1	課題		1
	1.1 課	題説明	1
	1.2 ⊐	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1	.0
	6.1 R'	TCBuilder の起動1	0
	6.1.1	ようこそ画面1	0
	6.1.2	パースペクティブを開く1	.0
	6.1.3	新規プロジェクトの作成1	.2
	6.1.4	プロファイル情報入力とコードの生成1	.4
	6.1.5	コード生成1	.8
7	EV3	の前進とコンポーネントの作成1	.9
	7.1 EV	√3 を前進させるサンプルソース(Python)1	.9
	7.2 前	進コンポーネントの作成2	20
	7.2.1	コード編集2	20
	7.2.2	動作確認	21
	7.2.3	vx の値を変えて動かす2	26
8	旋回	をするコンポーネントの作成2	28
	8.1 EV	√3 を旋回させるサンプルソース(Python)2	28
	8.2 旋	回コンポーネントの作成2	28
	8.2.1	コード編集2	28
	8.2.2	動作確認	29
	8.2.3	値を変えて動かす。2	29
9	数秒	前進後数秒旋回するコンポーネントの作成3	0
	9.1 2 7	砂間前進、2 秒旋回の手順	0

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	害物を検知し旋回後前進するコンポーネントの作成	
10.1	①超音波センサーの値を InPort で取得する手順	
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	1 サンプルソース (Python)	36
10.3.2	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</double>	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 \circ 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列に並べたもの)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/idl/1.1/idlreference_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、

「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。
- 「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。左上に割り当てらえた 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	$3 \text{cm} \sim 250 \text{cm}$
計測角度	約 20°
計測制度	+/-1cm

3.3 超音波センサーの値の取り方



- (1) 電源が入っていなければ、「<u>2 EV3の起動</u>」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, va = 0.0, vy = 0.0

後退させたい場合は、

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。

EV3を旋回させたい場合は

vx = 0.0, va = 0.1, vy = 0.0

となり、vx が 0 かつ va 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、vx が 0 かつ va 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、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旋回します。

4.3 値の制限

EV3に与える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	最小值	最大値
VX	-0.5m/s	0.5m/s
va	-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のようになります。



(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。

ファイル(E) 編集(E) 音 ③ ようご・ ※ 載	ナビゲート(<u>N</u>) 検索(<u>A</u>) プロジェクト(<u>E</u>) 実行(<u>B</u>)	(出)でいく (出)でいく (出)です。 (出)でいく (出)でいく (出)です。 (出)でいく (出)です。 (出) (出) (出) (出) (出) (出) (出) (出)] == # ¶• ຍ<<>>#
	Eclipse へようこそ		
Q	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	
*	サンプル サンプルの試行	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		

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。

Java - 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(<u>P</u>) 実行(<u>R</u>) ウィンドウ(<u>W</u>) ヘルプ(<u>H</u>)					
🖻 • 🗏 🕼 🗁 🕸 • O • 💁 • 🕅 🖉 •	@ ∦ • [2] • [2] • (+ + + + + +				クイック・アクセス 📄 🐉 Jav	a
は パッケージ・エクスプローラー ☆ □				- 0	アウトライン Sパースペクティブを開	<
E 😫 🔻					表示するアウトラインはありません。	
	開題 X @ lavadoc @ 真言				▼ □ E	5
	0項目					
	記述/説明		リソース	パス	ロケーショ	=>
		m				۲

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。

● パースペクティブを開く	- • •
 CVS リポジトリー・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 レソース 	
ОК ‡	ヤンセル

「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 RTC Builder 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。

۲	RTC Builder - Eclipse SDK												
77	・イル(F) 編集(E) ソース(S)	ノファクタリング(T)	+12	⊊ L(N) 10/00 (A)	プロジェクト	(P) 実行(R)	ーウィンドウ(W)) ヘレプ(H)					
	新規(N)	Alt+シフト+N I		プロジェクト(R)						クイック・ア	クセス	😫 🐉 Java	RTC Builder
	ファイルを聞く(.)			サンプル(X)							- 0		
	閉じる(C)	Ctrl+W	-	その他(の)…	Ctrl+N								
	すべて閉じる(L)	Ctrl+シフト+W	-	00010000									
	保管(S)	Ctrl+S											
	別名保存(A)		ι.										
	すべて保管(E)	Ctrl+シフト+S	ι.										
	前回保管した状態に戻す(T)		L .										
	移動(V)		ι.										
	名前変更(M)	F2	ι.										
8	更新(F)	F5	ι.										
	行区切り文子の変換(D)	•	ι.										
۵	印刷(P)	Ctrl+P	ι.										
	ワークスペースの切り替え(W)	•	L .										
	再開		ι.										
2	インポート(I)												
<u>Na</u>	エクスポート(0)		ι.										
RT	Open New Builder Editor												
	プロパティ(R)	Alt+Enter		M									
	終了/出口(X)		ľ	2.5									
_													

「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。

● 新規プロジェクト	- • •
ウィザードを選択	
ウィザード(<u>W</u>):	
フィルタ入力	
 ※ 既存 Ant ピルド・ファイルからの Java プロジェクト > → 一般 > → CVS > → Eclipse Modeling Framework > → Java > → Xcore > → ブラグイン開発 > → その他 > ▲ RTC Bulder > → TS = 	E
(?) < 戻る(B) 次へ(N) > 終了(E) キャ	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」ボタンを左クリックします。

RT-Component Builder Project				
プロジェクト名(<u>P</u>): Subject1				
☑ デフォルト・ロケーションの使用	(<u>D</u>)			
ロケーション(L): C:¥rtcws¥Subjec	:1			参照(<u>R</u>)
0	< = Z (D)	17 A (NI) >	\$\$Z(5)	التحفي حصيا ل
	< 床る(世)	////(<u>N</u>) >	▶ 終「(上)	キャンセル

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

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。ラベルが赤字の項目は必須項目です。その他はデフォルトの状態で変更は 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.

C RTC Builder - Subject1/RTC.xml - Eclipse S	ЮK						
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(B) ウィンドウ(W) ヘルプ(H)							
🖻 • 🗵 🕼 🗠 🏦 💁 • 🛷 • 🛓 •	$[2] \star \Leftrightarrow \Leftrightarrow \star \Rightarrow \star$			クイック・アクセス 😰 🐉 Java 💦 RTC Builder			
🔋 パッケージ・エクスプロー 🙁 😐 🗆	≯*Subject1 ⋈						
⊨ 🔄 🌣	基本			, in the second s			
		sic Profile					
	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		この名称はコンハーネントのヘ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。			
	*ハージョン: *ベンダ名:	Alzu	パージョン:	RTコンポーネントのバージョンを指定します。 x.v.z(x.v.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コンポーネントを作成した作者名、ベンダ名を指定しま 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カデ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 ・UNIOUF: 動的に生成/削除されるっニークなRTC			
				E E			
	基本 アクティビティ テー	-タホート サービスホート コンフィギュレーション トギュメント生成 言語・環境	RTC.xml				
	🙀 BuildView 🔀						
		Subject1					
		Subject					

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。

RTC Builder - Subject1/RTC.xml - Eclipse SE	DK				
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>₩)</u> ヘルプ(<u>H</u>)			
🖻 🕶 🔛 🕼 🚔 💁 🕶 🛷 🕶 🛃 🕶	$a \star \leftarrow \leftarrow \star \Rightarrow \star$				クイック・アクセス 😰 🖏 Java 😭 RTC Builder
🛤 パッケージ・エクスプロー 🛛 🖳 🗆	A Subject1 W				
📄 😫 🤝	onActivated	Deactivated	onAborting	onError onReset onExecute	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。
		Dataflow型コンポーネントのフ	アクション	onStatel Indate	onExecuteの後毎回呼ばれます。
	onExecute	onStateUpdate	onRateChanged	onRateChanged	ExecutionContextのrateが変更されたとき呼ばれます。
		FSM型コンポーネントのアク	7ション	onAction	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			onModeChange	d モードが変更された時に呼ばれます。
		Mode型コンポーネントのアク	ウション	21. /- ar 22.	
	onModeChanged			動作成要: 高前条件:	アクティビティの奴要式明を記述します。 アクティビティを実行する前に成立すべき事前条件を記述
	- Documentation			車治条件・	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
	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 アクティビティ名: onActi	ンヨンの根要を説明するドキュメントを ると、それぞれのドキュメントを記述で vated	記述します。 きます。 ● <u>2013</u> (OF	F	
	基 アクティドティ データル	ペート サービスボート コンフィギュ!	 ノーション ドキュメント生成 言語・環境	RTC.xml	
	BuildView 🖂				
			Subject1		
				1	

最終的に下図のようになります。

onInitialize	onFinalize	
	実行コンテキストの起動と停止に関するア	クション
onStartup	onShutdown	
	alive状態でのコンポーネントアクショ	シ
onActivated	onDeactivated	onAborting
onError	onReset	
	Dataflow型コンポーネントのアクショ	1>
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	2

onModeChanged

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: ultraso データ型: RTC::R	nic angeData	・OutPo ポート名 データ型	rt : vel : RTC::TimedVelocity2D		
发数名:ultrasoni	С	发纵名·V	发数名: vel		
表示位置: left		表示位置	: right		
○ RTC Builder - Subject1/RTC.xml - Eclipse : ファイル(E) 編集(E) ナビゲート(N) 検索(A)	SDK) プロジェクト(E) 実行(E) ウィンドウ(W) ヘルプ(出)				
	*Subject 12				
E Subjecti	プータボート DataPortプロフィル "のナクションアは8TコンボーネントのDataBodt"= - 57 "パート名(InPort) Jdtrasonic Detail このセクションではデータボート旬の概要を説明するドキュニムのデータボートを追引すると、それぞれのドキュメントカ ボート名: ultrasonic (InPort) "データ名 RTC::RangeData 変影名: ultrasonic 変影名: ultrasonic 第二 アクティビア アージス: ロビカションマントカ アービスボート コンフ BuildView 13	ボートンの道観を設定します。 ボートを(OutPort) el ロメントを記述します。 が記述できます。 ・ ・ ・ ・ ・ ・			
		Subject1			

ここで設定した情報は Pyhton 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	selfd_変数名
OutPort ポート名	selfポート名 Out
OutPort 変数名	selfd_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	selfultrasonicIn
InPort ultrasonic	selfd_ultrasonic
OutPort vel	<pre>selfvelOut</pre>
OutPort vel	<pre>selfd_vel</pre>

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<=x<=2.5 Widget: slider

○ RTC Builder - Subject1/RTC.xml - Eclipse SDK コーズル(の) 時間(の) キジモート(の) 時間(の) プロミークト(の) 市田(の) の(つ(の)) の((つ(の)))				
		クイック・ア	クセス 🔹 😰 Java 🛃 RTC Builder	
は バッケージ・エクスプロー ※ □ □ → *Subject1 ※				
B Subject1 コンフィギュレーション・パラメータ ◆ RT-Component Configuration Parameter Definitions		+ ヒント	î	
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・バラメータを指定します。 *名称 TurnDistance	Add	Config. Param. :	RTコンポ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・パラメータは実(再利用性を向上させるために、RTコンポー 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 名前にはアルファベット、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、ペクトル型、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		変数名:	コンフィギュレーション・パラメータに関) 実際の名称は言語により異なります。	
*データ型 double		単位: 制約条件:	コンフィギュレーション・パラメータの単 コンフィギュレーション・パラメータの制 ******	
g数名: TurnDistance			・10にない。王曰 ・100(即値):100 ・範囲指定・< > <= >= 下	
基本 アクティビティ データボート サービスボード コンフィギュレーション ドキュメント生成	言語・環境 RTC.xml			
🙀 BuildView 😒				
Z ultrasonie		vel	Û,	
. Kim	HTT		•	

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。

C RTC Builder - Subject1/RTC.xml - Eclipse S	5DK			
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>)		
🖻 • 🗄 🕲 🛆 🌺 🎴 • 🔗 • 🖄 •	$\{1, \dots, 2^{n}\} \rightarrow \{2^{n}\} $			クイック・アクセス 😰 😂 Java 🛃 RTC Builder
パッケージ・エクスプロー ☆ □	≯ *Subject1 ⋈			
🕒 😓 🗢 Subject1	言語・環境			
	▼ 三語			* E21
	このセクションでは使用する ● C+++ ● Python ● Java ● Ruby	3豊語を指定します	Use old build environment.	言語: RTコンボーネントを作成する言語を選択します。リスト中の宣言からう 環境: 言語:このライブランの伝知解除: 使用するのとどの環境を選択し 詳細環範で設定したが容 (OSI領紙、ライブラリ情報など) は、プロフ □
	▼環境			
	このヤクションでは依存する	5ライブラリや使用するOSなどを指定	ा/ ह व	
	Version	os	Add Delete	
	→詳細情報			
	基本 アクティビティ データオ	ペート 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	TC.xml
	🙀 BuildView 🖂			
		2 ultrasonic		
	<	·	Subject1	

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。

🕒 RTC Builder - Subject//RTC.xml - Eclipse SDK							
ファイル(E) 編集(E) ナビゲート(<u>N</u>) 検索(<u>A</u>) 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>)							
□ * □ ○ △ ▲ ▲ ▲ * ダ * シ * ジ * ジ * ジ * ジ * ジ * ジ * ジ * ジ * ジ							
はパッケージ・エクスプロー 🛛 🗖 🗆	は バッケージ・エクスプロー ※ ⁻ □ ▶*Subject1 ※ - □						
🕒 🔄 🌣	コンボーネント種類:	V 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 ・UNIQUE: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コンポ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		 PERIODIC: 実行周期が固定の周期実行型RTC SPORADIC: 実行周期が非固定の周期実行型RTC EVENTDRIVEN: イベント駆動型RTC 			
	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: 有限状態運移型RTC MultiMode: マルチモード型RTC 			
	▼ コード生成とパッケー	- 汐化	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			
		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]単位で指定し 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1	(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	ンボートおよびエクスボートを行います。	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 エクスパ	K- F					
		107	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 +			
	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 言語・環境	RTC.xml	,			
iii BuildView 2							
		4uitrasonic Subject1		vel			

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()

[write0]は OutPort から値を出力する関数 変数に値を設定し、write0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 on Execute(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。

 $C{:} \$rtcws \$Subject 1 \$Subject 1. py$

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。

ユーザ名 : robot

パスフレーズ:makerを入力します。

■ 192.168.0. ファイル(F)	SSH認証		
	ログイン中: 192.1680.8 認証が必要です. ユーザ名(N): robot パスフレーズ(P): ●●●●● ダパスワードをメ	ユーザ名(N):robot パスフレーズ(P):maker	この
	 ユージェンドキム ・ ブレインテキストを使う(L) RSA/DSA鍵を使う 	和密鍵(K):	
	 rhosts(SSH1)を使う 本スト額 	ユーカルのユーザ名(U): 【F: 】	
	 ● チャレンジレスボンス認証を使う(- ● Presentを使う 	キーボードインタラクティブ)(C)	
	OK	接続斯(D)	

	セキュリティ警告	x
	knows bostel 17 b由のtt = 15" 100 168 11 100"の1	-2,
20	の画面が表示されたら、「続行」	
ボク	タンを押下してください。	
	サーバ側のホスト鍵指紋	
	c3:3e:0b:2a:2e:26:19:1f:6e:d9:6f:f2:9d:0a:0d:4d	
	+[ECDSA 256]+	*
	E 0. S o = +o.o +=+.+ ++	÷
	 既存の鍵を、新しい鍵で上書きする(B) ・続行(C) 接続断(D) 	

この画面が表示されたらログイン完了です。

🧧 192.168.0.8:22 - robot@ev3dev: ~ VT
ファイル(F) 編集(E) 設定(S) コントロール(O) 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: Tue Sep 13 01:22:52 2016 from 192.168.0.9 robot@ev3dev: \$
*

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出来ます。

\$ 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。



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

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。



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。



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。

n 💀 🖉 🖉 🔐			
🛿 Name Ser 👘 Repositor 🖓	🗇 🗑 *System Diagram 🙁 🗖 🗖	□ プロパティー	
🟠 🗇 🗢 📑 🌺 🤣 🌶		プロパティー	値
RT 192.168.3.55		ब तर System Diagram	
<pre>ev3dev host_cxt</pre>		System ID	
EV3Control0 rtc		Kind	ONLIN
त्र localhost		Create Date	
NB1503013 host_cxt		Update Date	
Subject10 rtc	ultrasonic vel vel ultrasonic	Composite	None
	EV3Control0		
	Configur ZT Manager ZT Composi ZT Executio ZT RT Log 🙄 🗖		
	ComponentName: ConfigurationSet:		
	active config name value 通用		

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。

vx の値の範囲は以下になります。

	最小値	最大値
VX	-0.5	0.5

vx>0の場合前進、**vx<0**の場合後退します。**vx=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、Vxの値を0にして、Va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 on Execute(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。

```
def onExecute(self, ec_id):

# 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()

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 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
<pre>selfd_vel.data.vx = 0.04</pre>
selfd_vel.data.va = 0.0
<pre>selfd_vel.data.vy = 0.0</pre>
<pre>selfvelOut.write()</pre>
②2 秒間コンポーネントを停止
<pre>time.sleep(2)</pre>
③旋回指令
<pre>selfd_vel.data.vx = 0.0</pre>
<pre>selfd_vel.data.va = 0.5</pre>
<pre>selfd_vel.data.vy = 0.0</pre>
<pre>selfvelOut.write()</pre>
④2 秒間コンポーネントを停止
<pre>time.sleep(2)</pre>
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 度旋回を行う

現状 va の値が 0.5 の場合 2 秒では 90 度旋回は出来ません。 va の値を色々変更して 90 度旋回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旋回の値を探索すると、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の旋回速度をみます。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見ます。仮に目的の速度が

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。

10.1①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(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]

[isNew0]は新しい値を受信しているか確認する関数

[read0]新しい値を読み込む関数

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。

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]:</pre>
                    #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。

RT System Editor RCP	_		_		
File Window Help					
al aft 🖉 🖉 🖉 🔐					
💉 Name Ser 🕅 Repositor 🖓 🗖	ត្រា *System Diagram	X	- 0	□ プロパティー	~
🌣 🗢 🖓 🗱 🐌 🗢				プロパティー	値
RT 192.168.3.32				ਸ System Diagram	
<pre>ev3dev host_cxt</pre>				System ID	
EV3Control0 rtc				Kind	ONLINE
⊼⊤ localhost				Create Date	
NB1503013 host_cxt	Γ Γ		T	Update Date	
Subject10 rtc	ultrasonic	Vel		Composite	None
			ultrasonic		
	S	Subject10	color		
			touch		
		EV/2Control/	81		
		Evacontrolo			
	Configur KT Ma	anager 🚮 Composi 🚮 Execut	tio 🕅 RT Log 🖓 🗖		
	ComponentName:	ConfigurationSet:	編集		
	active config	name value			
			週用		
			キャンセル		
		28 bn Ballio			
	15.2	· 追加		< III	F.
	L	1			

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。

RT System Editor RCP	
File Window Help	
0 🖳 0 🚛 🔎 👹 🤐 🔐	
💓 Name Ser 🔞 Repositor 🖓 🗖 🗑 🖓 *System Diagram 🔀	
	プロパティー 値 ^
	A 52 Subject10
	Path URI localhc
	Instance Name Subjec
default	Type Name Subjec
ConfigurationSet : default	Description Subjec
TurpDictance 0.1	Version 1.0.0
(III)	Vendor Vender
	Category Catego
	State INACT
	- a owned
(3)OK を押下	Apply Apply SecutionCo
OK	ID 0
	State RUNNI
	Kind PERIO
	Rate 1000.0
	sync_tra YES
	① 問わな切下 transitio 0.5
	U用如211 「 type Periodi
Configur KI Manager KI Composi KI Executio	KI RI Log L rate 1000
ComponentName: ConfigurationSet: default	編集 Propertie
active config name value	Name Subject
G default TumDistance 0.1	適用 Data Type IDL:PI
	キャンセル Interface Ty corba
	Dataflow Ty pull.pu
複製 追加 削除 🕅	
Pos: (88,134) Size: (111,39)	