中級者向け講習会課題 2-2

会津大学　ＲＴミドルウェア講習会

カラーセンサーで障害物を検知し、それを避けて進むプログラムを作成する

目次

※ 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)

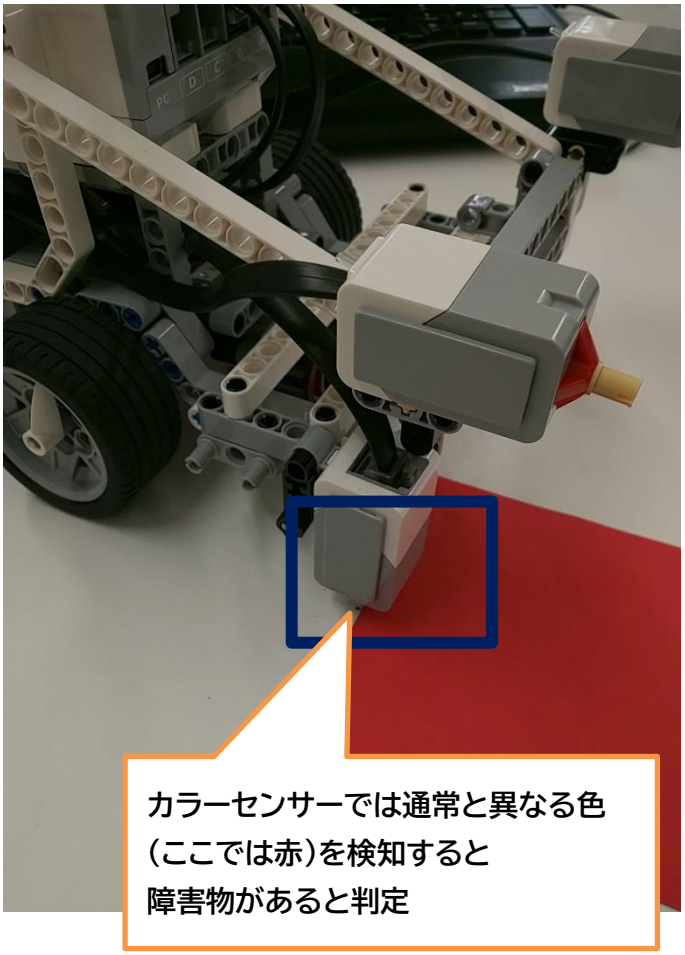
・LEGO Mindstorms EV3 活用事例

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

1 システム概要

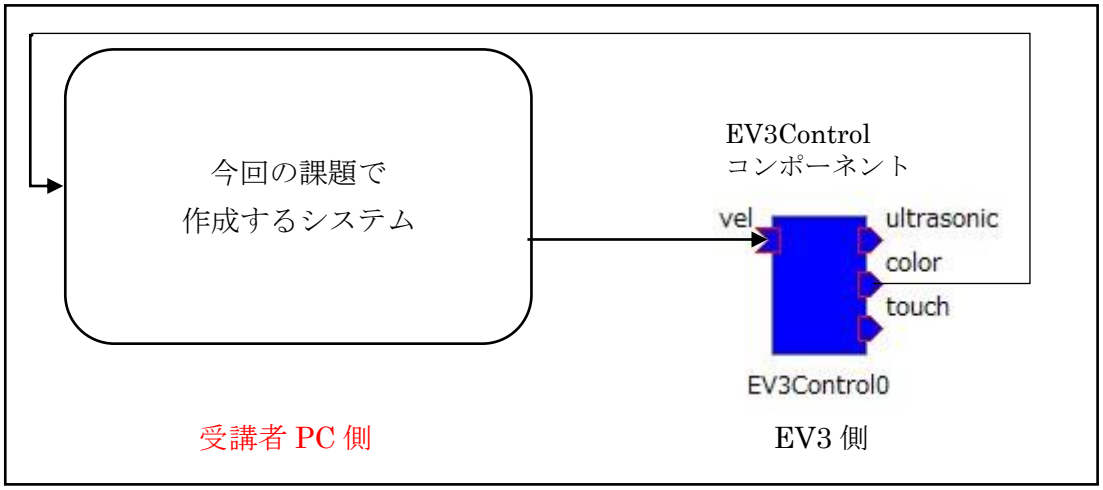
1.1 システム概要

EV3 に接続されたカラーセンサーで障害物を検知し、後退した後に旋回して障害物を避けて進むプログラ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の値を各 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

1.2.2 カラーセンサーのデータ型

EV3Control のカラーセンサーのデータ型は以下のようになります。

【RTC.TimedString】

型	変数名	意味
string	data	文字データ
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 data にはカラー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

1.2.3 サンプルソース

カラーセンサーの値取得のサンプルソース（Python）

```
        # カラーセンサーの値が更新されている場合
        if self._colorIn.isNew():
                # カラーセンサーの値を読み込み
                self._d_color = self._colorIn.read()
```

カラーセンサーの値比較のサンプルソース（Python）

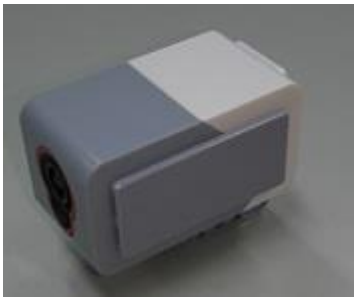
```
        # カラーセンサーの値が青の場合
        if self._d_color.data == "2":
                #ここに処理を追加する#

        # カラーセンサーの値が赤の場合
        if self._d_color.data == "5":
                #ここに処理を追加する#

        # カラーセンサーの値が青か赤の場合
        if self._d_color.data == "2" OR self._d_color.data == "5":
                #ここに処理を追加する#
```

「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))
```

2 カラーセンサーの仕様

2.1 カラーセンサーについて



カラーセンサーとは、表面の色を検知することが出来るセンサーです。EV3 で使用しているカラーセンサーは左図となります。このカラーセンサーを EV3 に接続することによりカラーセンサーを使用することが出来るようになります。

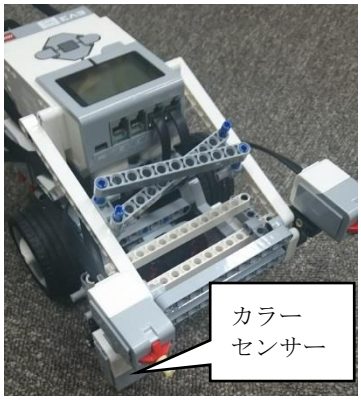
2.2 カラーセンサーの性能

色	値
なし	0
黒	1
青	2
緑	3
黄色	4
赤	5
白	6
茶	7

2.3 カラー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してください。

EV3 に接続されたカラー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-color at in1」と選択します。
※「lego-ev3-color at in1」at in1 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、「Set mode」で「COL-COLOR」を選択します。
(4) 選択後、「Watch values」を選択します。
(5) カラー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 に接続されたカラーセンサーの前にある物体の色によって値が変化します。

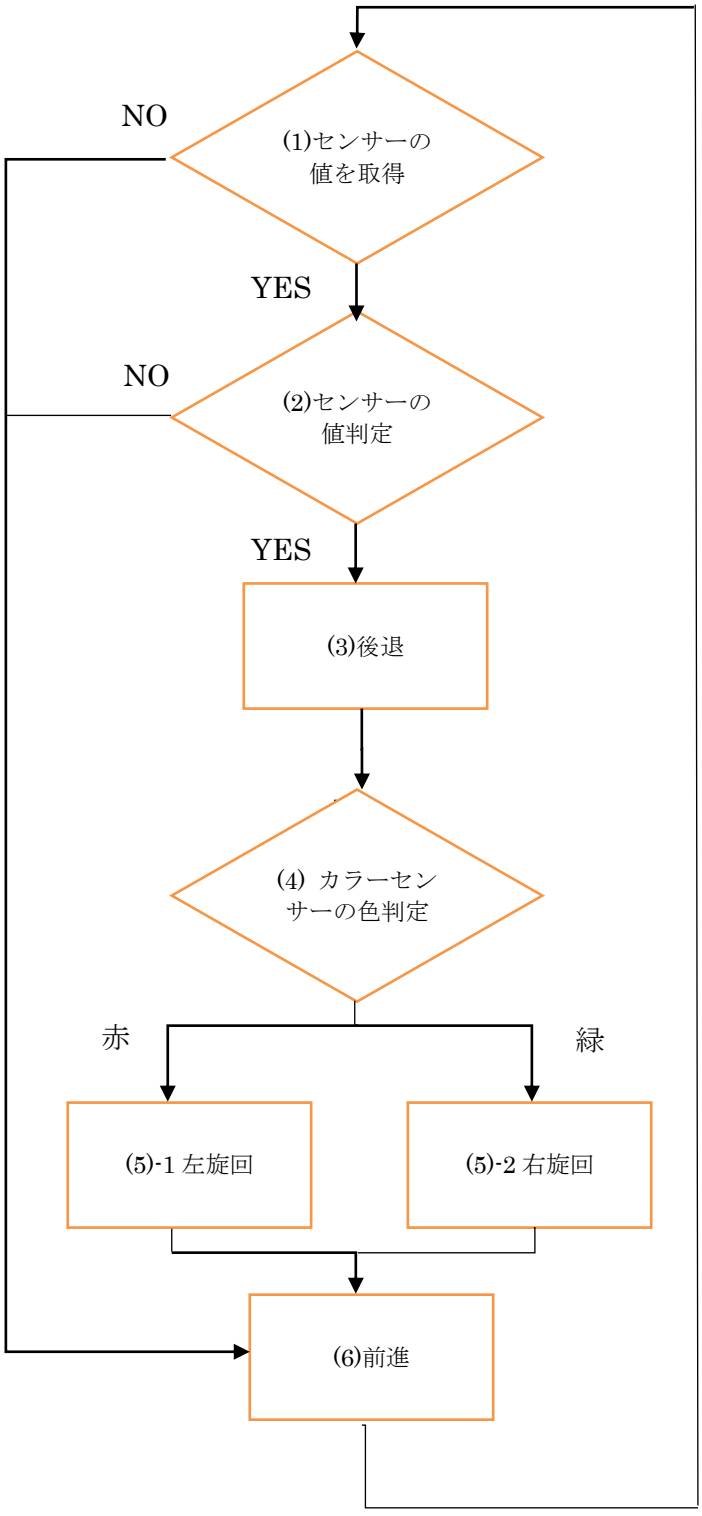
3 速度の与え方

速度の与え方については課題 1 で説明した通りです。

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ようになります。



(1) センサーの値を取得できたか判定。NO なら(6)前進の処理に移行。YES なら(2)の判定に移行。
(2) センサーの値が赤か緑か判定。赤か緑なら YES、赤か緑以外なら NO。NO なら(6)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。後退処理終了後に(4)カラーセンサーの色判定に移行。
(4) センサーの値が赤か緑かを判定。赤の場合(5)-1 左旋回の処理に移行。緑の場合(5)-2 右旋回の処理に移行。
(5) -1 左旋回の処理実行。旋回処理終了後に(6)前進の処理に移行。
-2 右旋回の処理実行。旋回処理終了後に(6)前進の処理に移行。
(6) 前進の処理実行。前進処理終了後に(1)の判定に戻る。

といったようになります。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。
※カラーセンサーの値は光の影響により正しく取得できない場合があります。
その影響も踏まえてプログラムしてください。

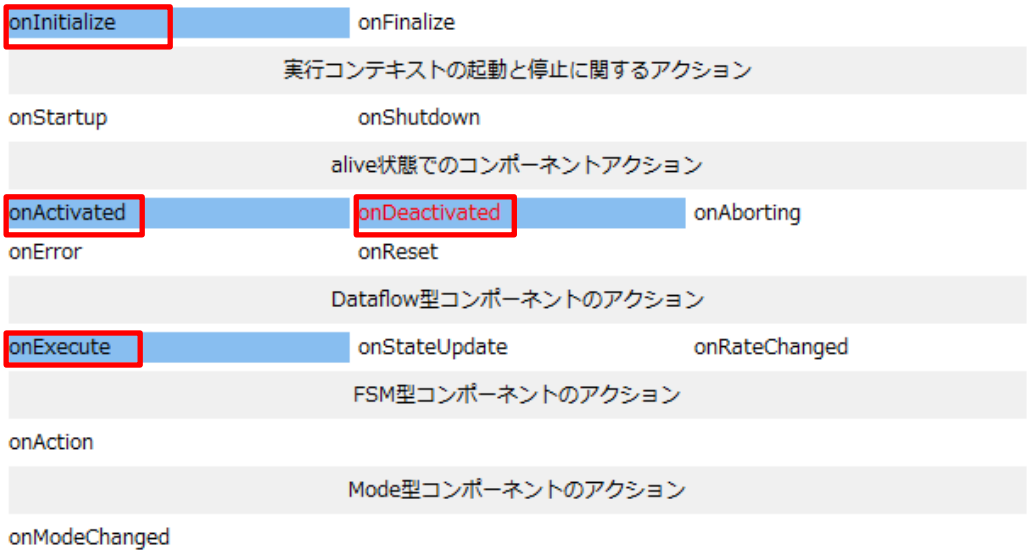
4.2 雛型の値

RTCBuilder で作成する雛形の例を以下に記載します。

4.2.1.1 基本

モジュール名：Subject3
モジュール概要：Subject3 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.

4.2.1.2 アクティビティ



4.2.1.3 データポート

・InPort
ポート名: color
データ型: RTC::TimedString
変数名: color
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

4.2.1.4　コンフィギュレーション

なし

4.2.1.5　言語・環境

“Python”を選択

4.3 障害物を検知して旋回するサンプルソース

カラーセンサーを使用したサンプルソースを以下に載せます。穴埋め式になっていますので赤文字の所を自分で埋めてみてください。

4.3.1 サンプルソース（Python）

```
def onExecute(self, ec_id):

        # カラーセンサーの値が更新されている場合
        if self._ #InPort のポート名#In.isNew():
                # カラーセンサーの値取得
                self._d_#InPort の変数名# = self._ #InPort のポート名#In.read()
                #カラーセンサーの値が 2（青）または 5（赤）の場合
                if #カラーセンサーの値が 2（青）または 5（赤）の場合:
                        # 後退指定
                        self._d_vel.data.vx = -0.04
                        self._d_vel.data.va = 0.0
                        self._d_vel.data.vy = 0.0
                        self._velOut.write()
                        # Sleep
                        time.sleep(2);

                        #カラーセンサーの値が 2 の場合
                        if #カラーセンサーの値が 2 の場合#:
                                # 左旋回指定
                                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)
                        elif #カラーセンサーの値が 5 の場合#:
                                # 右旋回指定
                                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)
        # 前進指定
        self._d_vel.data.vx = 0.04
        self._d_vel.data.va = 0.0
        self._d_vel.data.vy = 0.0
        self._velOut.write()

        return RTC.RTC_OK
```

「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))
```