
LEGO MindStorm EV3 の RobotC による制御プログラミング

2017年10月26日版

西井 淳

目次

1	はじめに	2
1.1	LEGO MindStorm と RobotC について	2
1.2	演習課題について	2
2	RobotC の準備	2
2.1	RobotC のインストール	2
2.2	RobotC のアクティベーション	2
3	MindStorm を動かしてみよう	3
3.1	LEGO MindStorm EV3 の準備	3
3.2	RobotC の設定	3
3.2.1	モータポートの設定	4
3.2.2	センサポートの設定	4
3.3	プログラムを書いて MindStorm を動かす	4
3.4	プログラムにエラーがおきた時	5
3.5	ファイルの消し方	5
4	RobotC の基本文法	6
4.1	モータを動かす	6
4.2	LCD ディスプレイ表示	6
4.3	変数の型と基本演算	7
4.4	関数	8
4.4.1	関数の定義	8
4.4.2	引数のアドレス渡し	8
4.4.3	引数のアドレス渡し (C++ の書式)	9
4.4.4	配列・構造体	9
4.5	制御構文	11
4.5.1	条件式	11
4.5.2	if 文, while 文	11
4.5.3	for 文	11
4.5.4	do~while 文	12
5	センサーを使う	13
5.1	タッチセンサを使う	13
5.2	カラーセンサを使う: 明るさを取得する	14
5.3	カラーセンサを使う: 色情報を取得する	15
5.4	旧型光センサを使う	15
5.5	超音波センサを使う	16
5.6	ジャイロセンサを使う	16
6	並列タスク	18
A	付録 A 用意されている数学関数の例	21

1 はじめに

1.1 LEGO MindStorm と RobotC について

LEGO MindStorm は LEGO が MIT と共同開発したロボット制御の学習用キットです。本テキストでは、LEGO MindStorm EV3 をプログラミング言語 RobotC を使って制御する方法を説明します。RobotC では、3種類のプログラムスタイルを利用できますが、演習では Text-Based ROBOTC Commands と呼ばれる C 言語と似た言語を使います。

1.2 演習課題について

このテキストで「**課題**」とある箇所は、関連する質問や指示が elearning のページにあります。**必ず回答してください。**「**問**」とある箇所は、指示に従って設問の内容を各自考えましょう。結果を報告する必要はありませんが、その後の課題や演習のレースプログラム作成のために必要になることがあるので各自必ず動作確認をしましょう。

2 RobotC の準備

2.1 RobotC のインストール

演習の elearning のページに、RobotC のインストーラのダウンロードサイトのリンク情報があります。インストーラをダウンロード・実行して、Yes か Next をひたすら選択すればインストールできます。インストール後にデスクトップに表示されるアイコン “ROBOTC for LEGO MINDSTORMS” をクリックすると図 1 のような画面がでてきます。

2.2 RobotC のアクティベーション

以下の手順でアクティベーションをしてください(図 2)。ただし、大学での情報コンセントではアクティベーションできないので、各自自宅で実行してください下さい。(自宅でできない人は担当教員に相談)

1. RobotC のメニューで”Help/Manage License”を選択
2. 表示されるウィンドウで”Add License”を選択
3. Product は”ROBOTC for LEGO MINDSTORMS 4.x”を選択
4. アカウント情報を入力 (elearning のページ参照)

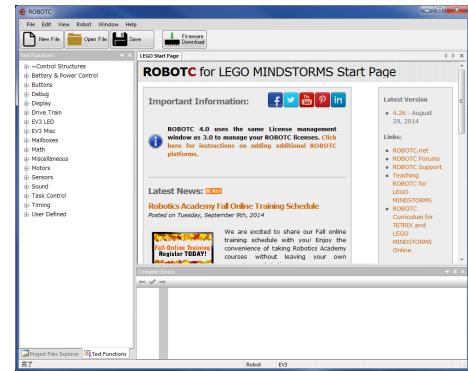
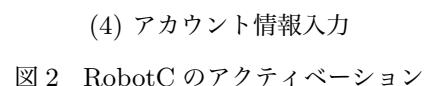
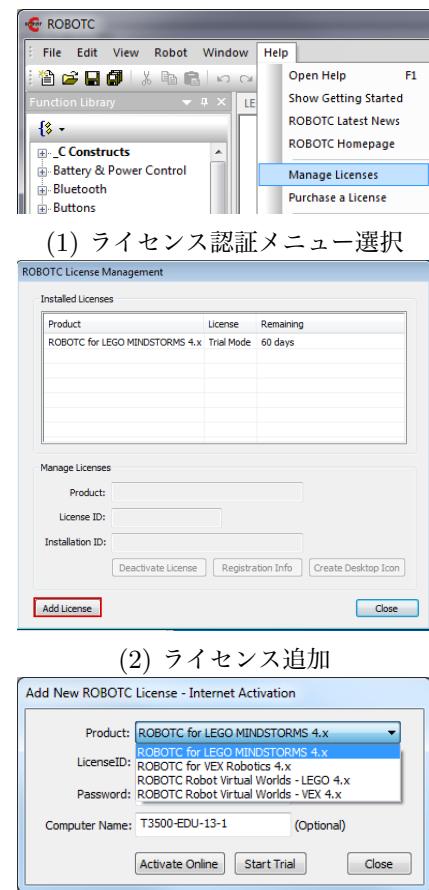


図 1 RobotC の起動画面



(4) アカウント情報入力

図 2 RobotC のアクティベーション

3 MindStorm を動かしてみよう

3.1 LEGO MindStorm EV3 の準備

MindStorm EV3 の上部にある **B ポート**に左モータが、**C ポート**に右モータが接続されている事を確認しましょう。次に、センサがEV3 下部の各ポートに、以下のように接続されていることを確認してください。

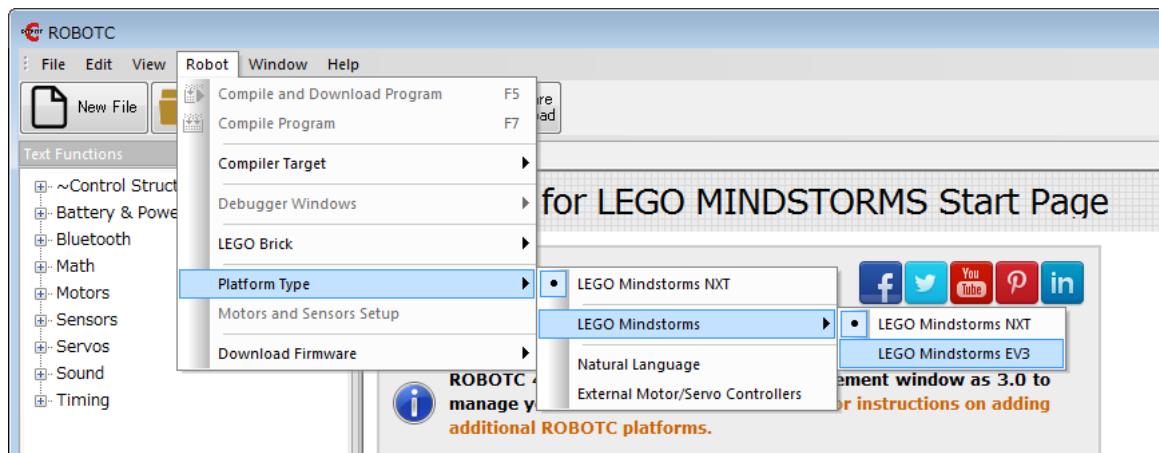
1. 入力ポート 1 にタッチセンサ
2. 入力ポート 2 にジャイロセンサ
3. 入力ポート 3 にカラーセンサ
4. 入力ポート 4 に超音波センサ

確認したら、USB ケーブルでパソコンと MindStorm EV3 を接続し、MindStorm EV3 を起動しましょう。LEGO MindStorm EV3 のボタンの説明は右図を参照してください。



3.2 RobotC の設定

RobotC で利用する MindStorm の機種設定を行います。RobotC の上部メニューの **Robot → Platform Type → LEGO Mindsotrms** から「**LEGO Mindsotrms EV3**」を選択します。



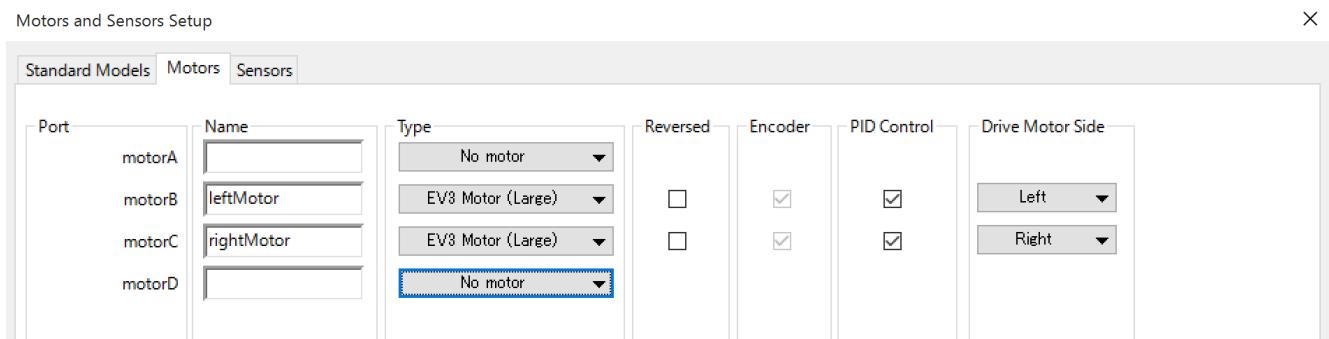
次に、プログラムを書く準備のためにメニューの **File → New** を選択しましょう。その後、モータやセンサについての設定を行います。上部の”**Motors and Sensors Setup**” のアイコンを選択してください。



かわりに、メニューの **Robot → Motors and Sensors Setup** を選択しても同じ設定をできます。

3.2.1 モータポートの設定

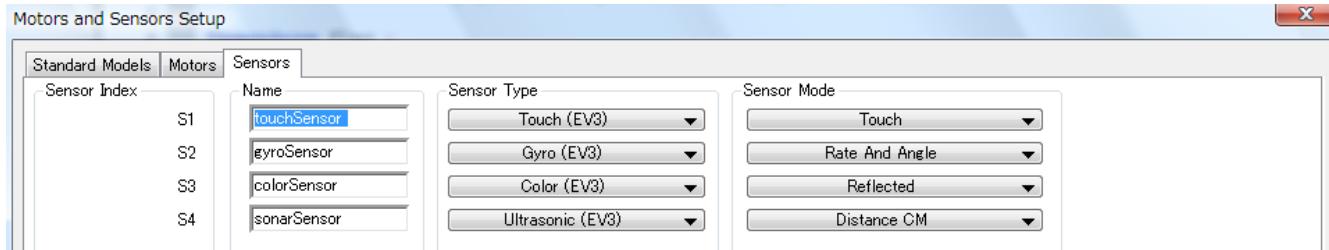
“Motors”と書かれたタブを開いて、motorAからmotorDまでの各欄を下図のように設定してください。Name欄には、各モータの名前をleftMotor, rightMotorと登録しましょう。以下のプログラムでは、各モータにこの名前を使ってアクセスします。



[課題 3.1] モータポートの設定をよく確認したら、elearning の課題をしましょう。

3.2.2 センサポートの設定

“Sensors”と書かれたタブを開いて、S1からS4の各欄を以下のように設定してください。



Name欄には、各センサの名前をtouchSensor, gyroSensor, colorSensor, sonarSensorと登録しましょう。以下のプログラムでは、各センサにこの名前を使ってアクセスします。ポートS4に接続している超音波センサのSensor Mode欄で選択しているDistance CMは、対象までの距離を単位cmで計測するようにするための設定です。

[課題 3.2] センサポートの設定をよく確認したら、elearning の課題をしましょう。

3.3 プログラムを書いて MindStorm を動かす

以下のようなプログラムを書いてみましょう。

```

1 task main()
2 {
3     setMotorSpeed(rightMotor,50);
4     setMotorSpeed(leftMotor,50);
5     sleep(2000);
6 }
```

以下の手順でプログラムを実行しましょう。

1. [保存] プログラムを書いたら「Save」をクリックしてファイルを保存します。
 - (a) ファイル名は日本語不可です。必ず英数文字のみの名前にします。日本語を使うと、コンパイルは出来ても実行が出来なくなります。
 - (b) ”()”などの記号も使えません。
2. [コンパイル] 「Compile Program」(下図)をクリックします。このとき、プログラム名をすでに指定しているときには、プログラムは自動で上書き保存されます。
3. [ロボットにダウンロード] 「Download to Robot」(下図)をクリックします。プログラムを変更したときには、この転送をお忘れ無く。

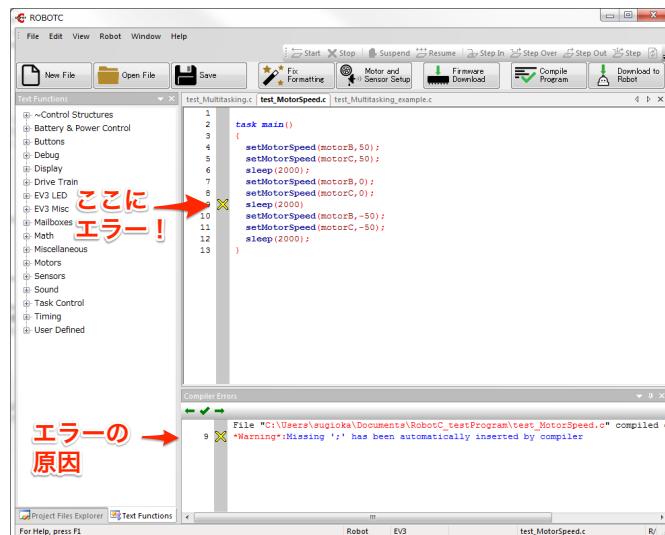


注意) EV3 を初めてノートパソコンに接続したときには、ドライバの自動インストールのため、プログラムのダウンロードができるようになるまで少し時間がかかることがあります。

4. [実行] ダウンロード後表示される「Start」ボタンをクリックすれば実行できます。プログラムを MindStorm にダウンロードした後は、USB ケーブルをはずして MindStorm のボタンを操作してもプログラムの実行はできます。

3.4 プログラムにエラーがおきた時

コンパイル時にエラー行に ×印が表示されます。エラーの原因は下のウィンドウに表示されます。



プログラムが暴走したときには、「電源ボタン」と「戻る」ボタンを同時に押すと強制終了できます。

3.5 ファイルの消し方

演習終了時には、MindStorm 上にある自分のプログラムを消去しましょう。RobotC の上部メニュー”Robot/LEGO Brick/File Management Utility”で、MindStorm 上のファイル操作ができます。

4 RobotC の基本文法

4.1 モータを動かす

C 言語のプログラムでは必ず main 関数を一つだけ書きますが、ROBOTC では main() という名前の task(タスク) を必ず一つだけ書きます。タスクと関数は役割が少し違いますが、それについて後述します。

```

1 task main()
2 {
3     setMotorSpeed(rightMotor,50);
4     setMotorSpeed(leftMotor,50);
5     sleep(2000); //wait 2000 milliseconds
6     setMotorSpeed(rightMotor,-30);
7     setMotorSpeed(leftMotor,-30);
8     sleep(2000);
9     setMotorSpeed(rightMotor,0);
10    setMotorSpeed(leftMotor,0);
11 }
```

プログラム中の関数や記号の意味は以下の通りです。

- 各行において”//”より後はコメントとみなされます。C 言語と同様に/* */ではさむこともできます。
ただし、**コメント文には日本語は使えません。**
- `setMotorSpeed(port,speed)` は、指定したポート (port) に接続されているモータを、指定した出力 (speed [%]) で動かす命令です。speed は int 型の -100 から 100 の数値で指定します。引数 port にはモータを接続しているポート番号か、3.2.1 節で設定したモータ名を指定します。モータの回転方向を変えるには、出力の符号を変えます。
- `sleep(time)` は、指定した時間 (time [ms]) だけ次の行の命令の実行を待ち、それまでの命令を継続する命令です。

4.2 LCD ディスプレイ表示

以下は MindStorm の LCD ディスプレイに文字や数値を出力するプログラム例です。

```

1 task main()
2 {
3     int i=1234;
4     displayTextLine(0,"i=%d",i);
5     sleep(2000);
6     displayCenteredTextLine(2,"HelloWorld");
7     sleep(2000);
8     eraseDisplay();
9     sleep(2000);
10    displayBigTextLine(3,"HelloWorld");
11    sleep(2000);
12    displayCenteredBigTextLine(5,"i=%d",i-1000);
13    sleep(2000);
```

14 }

上記のプログラムで使っている関数の詳細は以下の通りです。

関数名	機能
eraseDisplay()	画面を消す
displayTextLine(n, string, ...)	n 行目に文字 string を表示
displayCenteredTextLine(n, string, ...)	n 行目に文字 string を中央揃えで表示
displayBigTextLine(n, string, ...)	n 行目に文字 string を大文字で表示
displayCenteredBigTextLine(n, string, ...)	n 行目に文字 string を大文字かつ中央揃えで表示

なお、残念ながら日本語の表示はできません。

4.3 変数の型と基本演算

RobotC では、C 言語のように変数を使うこともできます。以下は利用できる変数の型の例です。

bool	... true か false の二値をとる変数型
byte	... 整数型 (8bit, -128~127)
char	... 整数型 (8bit, -128~127)
float	... 浮動小数点型 (32bit)
long	... 整数型 (32bit, -2,147,483,648~2,147,483,647)
int	... 整数型 (16bit, -32,768~32,767)
string	... 文字型

数値演算のときにも C 言語と同様に、+，-，++，--，/，&，+=，-=などの演算子を使うことができます。

[課題 4.1] 以下のプログラムを作成して各変数の値を MindStorm の LCD ディスプレイに表示し、予想通りの値が表示されているか確認しなさい。

```

1 int a, b, c, d;
2
3 task main()
4 {
5     a=5;
6     b=2*a+1;
7     b/=a;
8     a%=3;
9     c=++a;
10    d=c++;
11    displayTextLine(0, "a=%d", a);
12    displayTextLine(1, "b=%d", b);
13    displayTextLine(2, "c=%d", c);
14    displayTextLine(3, "d=%d", d);
15    sleep(2000); /* wait 2s */
16 }
```

4.4 関数

4.4.1 関数の定義

RobotC にも C 言語の関数と同様の関数 (function) があります。以下の例では関数の返り値は `void` ですが、C 言語と同様に `int` や `float` 等を返り値にすることもできます。

```

1 void straight(int pwr)
2 {
3     setMotorSpeed(rightMotor, pwr);
4     setMotorSpeed(leftMotor, pwr);
5 }
6
7 task main()
8 {
9     straight(50);
10    sleep(1000);
11    straight(-30);
12    sleep(1000);
13    straight(0);
14 }
```

4.4.2 引数のアドレス渡し

C 言語と同様にポインタを使って引数のアドレス渡しを実現できます以下はその例です。

```

1 int foo(int x)
2 {
3     static int y=2;
4     x++;
5     y++;
6     return(y);
7 }
8
9 void bar(int *x)
10 {
11     (*x)++;
12 }
13
14 task main()
15 {
16     int x=1,y,z;
17     y=x++;
18     z=foo(y);
19     displayCenteredBigTextLine(2,"x=%d",x);
20     displayCenteredBigTextLine(3,"y=%d",y);
21     displayCenteredBigTextLine(4,"z=%d",z);
22     z=++x;
23     bar(&z);
24     displayCenteredBigTextLine(5,"z=%d",z);
```

```

25     z=foo(y);
26     displayCenteredBigTextLine(6,"z=%d",z);
27     sleep(2000);
28 }
```

この例では、関数 foo において**静的変数**も使われています。

[課題 4.2] 上のプログラムを作成し、変数 y の値が予想通りの値に表示されるか確認しなさい。

4.4.3 引数のアドレス渡し (C++ の書式)

引数のアドレス渡しは、C++ 言語と同様に関数の定義部の引数に`&`をつけることで、アドレス渡しの指定をすることもできます。以下の 2 つのプログラムは、同じプログラムを C の記法と C++ の記法で書いたものです。C++ の記法の場合、関数 bar の引数定義 (1 行目) には`&`をつけ、それ以外 (関数 bar 内で引数の値を修正するとき (3 行目) や、関数 main 内で関数 bar を呼び出すとき (9 行目)) の変数名は、通常の変数表記にすれば良いので、表記が簡潔になります。

```

1 void bar(int *x) //アドレス渡しなのでポインタでアドレスを受け取る
2 {
3     (*x)++;
4 }
5
6 task main()
7 {
8     int x=1;
9     bar(&x); // 変数のアドレスを関数 bar に渡す
10    displayCenteredBigTextLine(5,"x=%d",x);
11    sleep(2000);
12 }
```

```

1 void bar(int &x) //アドレス渡し
2 {
3     x++;
4 }
5
6 task main()
7 {
8     int x=1;
9     bar(x); // アドレス渡しでも単に変数名を書くだけで良い
10    displayCenteredBigTextLine(5,"x=%d",x);
11    sleep(2000);
12 }
```

4.4.4 配列・構造体

配列や構造体も C 言語と同様に利用できます。配列の各要素は 0 に初期化されます。以下の二つのプログラムは、同じアルゴリズムを配列と構造体で作った例です。

```
1 #define RIGHT 0
```

```

2 #define LEFT 1
3
4 void initSpeed(int *v) //値を書き換えるのでアドレス渡し
5 {
6     v[RIGHT]=100;
7     v[LEFT]=50;
8 }
9
10 void setSpeed(int *v) //配列を受け取るのでアドレス渡し
11 {
12     setMotorSpeed(rightMotor,v[RIGHT]);
13     setMotorSpeed(leftMotor,v[LEFT]);
14 }
15
16 task main()
17 {
18     int v[2];
19
20     initSpeed(v);
21     setSpeed(v);
22     while(true);
23 }
```

```

1 typedef struct{
2     int right;
3     int left;
4 } motorSpeed;
5
6 void initSpeed(motorSpeed &v) //値を書き換えるのでアドレス渡し
7 {
8     v.right=100;
9     v.left=50;
10 }
11
12 void setSpeed(motorSpeed v) //構造体(値は書き換えない)だと参照渡しでOK
13 {
14     setMotorSpeed(rightMotor,v.right);
15     setMotorSpeed(leftMotor,v.left);
16 }
17
18 task main()
19 {
20     motorSpeed v;
21
22     initSpeed(v);
23     setSpeed(v);
24     while(true);
25 }
```

関数 `initSpeed()` では引数 `v` の値を書き換えるので、引数をアドレス渡しで受取る必要があります。

4.5 制御構文

4.5.1 条件式

if 文等の条件式において値が非0ならば真, 0ならば偽と判定されます。条件式では C 言語と同様に ==, >, <= 等の多くの演算子を使うことが出来ます。「真を表す記号」として”true”, 「偽を表す記号」として”false”を使うことも出来ます。

4.5.2 if 文, while 文

if 文や while 文も C 言語と同様に使えます。以下のプログラム例で, while(true) の true は「真」を意味するので, while 文のループ内は永久に実行され続けることになります。

```

1 #define MOVE_TIME 1000
2 #define TURN_TIME 500
3
4 void forward(int pwr)
5 {
6     setMotorSpeed(rightMotor,pwr);
7     setMotorSpeed(leftMotor,pwr);
8 }
9
10 task main()
11 {
12     srand(nSysTime);
13     while(true){
14         forward(50);
15         sleep(MOVE_TIME);
16         if(random(1) == 0){
17             forward(-50);
18         } else {
19             setMotorSpeed(rightMotor,-50);
20         }
21         sleep(TURN_TIME);
22     }
23 }
```

プログラム中の関数 random(n) は 0 から n までの整数乱数を返します。関数 srand() は nSysTime を引数として乱数の初期値を設定しています。nSysTime はロボットの起動時間 (msec) の下 16 ビットの値を示す変数です。

[課題 4.3] 上のプログラムを組んでロボットの動作をよく理解したら, elearning の設問に答えなさい。

4.5.3 for 文

for 文も C 言語と同様に使うことが出来ます。

```

1 #define MOVE_TIME 1000
2 #define TURN_TIME 500
3
```

```
4 task main()
5 {
6     for(int i=1;i<=3;i++){
7         setMotorSpeed(rightMotor,50);
8         setMotorSpeed(leftMotor,50);
9         sleep(MOVE_TIME);
10        setMotorSpeed(leftMotor,-50);
11        sleep(TURN_TIME);
12    }
13 }
```

4.5.4 do~while 文

以下のように do~while 文も使うことができます。

```
1 do {
2     ...
3 }
4 while (condition);
```

do 文ではまず do に続く {}内の命令を一度実行した後に、while() で指定した条件が評価されることに気をつけましょう。

[課題 4.4] 4.5.3 節のプログラムを do~while 文を使って書き換えなさい。

5 センサーを使う

5.1 タッチセンサを使う

以下は何かにぶつかるまで直進するプログラム例です。

```

1 void forward(int pwr)
2 {
3     setMotorSpeed(rightMotor, pwr);
4     setMotorSpeed(leftMotor, pwr);
5 }
6
7 task main()
8 {
9     forward(50);
10    while (getTouchValue(touchSensor)==false){}
11    forward(0);
12 }
```

関数 `getTouchValue(port)` は、指定したポートに接続しているタッチセンサがなにかに接触した時には 1 (true) を、接触していない時には 0 (false) を返します。引数 port にはタッチセンサを接続しているポート番号か、3.2.2 節で設定したタッチセンサ名を指定します。

以下は障害物に当たったら少し後ろに下がってから向きを変えて動き続けるプログラム例です。

```

1 void forward(int pwr)
2 {
3     setMotorSpeed(rightMotor, pwr);
4     setMotorSpeed(leftMotor, pwr);
5 }
6
7 task main()
8 {
9     while(true){
10        forward(50);
11        if(getTouchValue(touchSensor)==true){
12            forward(-50);
13            sleep(500);
14            setMotorSpeed(rightMotor, 50);
15            sleep(500);
16        }
17    }
18 }
```

[課題 5.1] 上記のプログラムに、タッチセンサが衝突を検出した回数をカウント・表示する関数を追加し、main 関数から呼び出すよう修正しなさい。回数のカウントには静的変数 (static) を新しく作る関数内で定義して用いること。

5.2 カラーセンサを使う: 明るさを取得する

カラーセンサの横には LED ライトがついています。LED ライトを消灯すると周囲の明るさや色情報を、LED ライトを点灯するとその反射光によって、カラーセンサのすぐ前にある対象の明度や色の情報を得ることができます。

LED ライトの反射光から対象の明度を取得する関数には以下の 2 つがあります。

- int getColorReflected(port): 赤色 LED を点灯して反射光の明度を表す 0 から 100 までの数値を返します。返り値が大きいほど明度が高いことを示します。
- int getColorAmbient(port): 赤色以外の LED を点灯して、反射光の明度を返します。

以下は地面が明るいと時計回り、暗い時には反時計回りに回転するプログラムです。光センサの値は Mind-Storm の LCD ディスプレイに表示されます。

```

1 #define THRESHOLD 40
2
3 void clockwise(long pwr)
4 {
5     setMotorSpeed(rightMotor,-pwr);
6     setMotorSpeed(leftMotor,pwr);
7 }
8
9 task main()
10 {
11     int s;
12     while(true){
13         s=getColorReflected(colorSensor);
14         displayCenteredBigTextLine(5,"Light:%d",s);
15         if(s>THRESHOLD){
16             clockwise(50);
17         } else {
18             clockwise(-50);
19         }
20     }
21 }
```

[問 5.1] 上記のプログラムを改造して、カラーセンサを使って床面に描かれている黒線の右縁に沿ってロボットを動かすライントレースのプログラムを作成してみましょう。カラーセンサで取得した床面の明度は LCD に表示させるようにしてください。

[課題 5.2] まっすぐ直進しながら、床面にある黒線を横切った回数を数え、4 本目の黒線を通過したところで停止するプログラムを作成してください。黒線を横切った回数はディスプレイに表示するようにしなさい。

5.3 カラーセンサを使う：色情報を取得する

関数 `getColorRGB(port,&red,&green,&blue)` ^{*1}を使うと、カラーセンサを使って対象の赤、青、緑の各色の強さを 0 から 255 の数値で知ることができます。以下は地面の色の情報を MindStorm の LCD ディスプレイに表示するプログラムです。

```

1 task main()
2 {
3     long redValue;
4     long greenValue;
5     long blueValue;
6
7     while (true){
8         getColorRGB(colorSensor, redValue, greenValue, blueValue);
9         displayCenteredBigTextLine(5,"RGB:%d,%d,%d", redValue, greenValue, blueValue);
10        sleep(100);
11    }
12 }
```

[課題 5.3] 右図のように赤く塗りつぶされた 2 地点がある。その一方から出発し、他方に到達したら後進してもとの場所に戻る動作を繰り返すプログラムを作り、その動作を確認しなさい。



5.4 旧型光センサを使う

明るさのみを検出できる（色情報は取得できない）旧型（MindStorm NXT 用）の光センサを使う場合、**3.2.2 節のセンサ設定**でセンサ名（Name）を `lightSensor` とし、Sensor Type は Light-Reflected(NXT) を指定して下さい。プログラム中で LED を消灯したい時には以下を実行します。

```
1 SensorType[lightSensor]=sensorLightInactive;
```

点灯する時には右辺を `sensorLightActive` とします。

旧型光センサ

以下は、周囲の明るさを取得して表示するプログラムです。



```

1 task main()
2 {
3     while(true){
4         displayCenteredBigTextLine(5,"Light:%3d",SensorValue(lightSensor));
5     }
6 }
```

関数 `SensorValue(port)` は、指定ポートのセンサ値を 0 から 100 までの値で返します。

^{*1} &についている引数は、ポインタを使わずにアドレス渡しを実現できることを示します。C++ 言語の書式と同じです。

5.5 超音波センサを使う

超音波センサは、出した音が返ってくるまでの時間を計ることで前方の物体までの距離を測ることができるセンサです。距離は関数 `getUSDistance(port)` を使って単位 cm または inch で取得できます。どちらの単位を使うかは 3.2.2 節で指定できます。

[問 5.2] 以下のプログラムの動作を説明し、実際にロボットを使って動作確認をしなさい。

```

1 #define NEAR 15 //cm
2
3 task main()
4 {
5     while(true){
6         setMotorSpeed(rightMotor,50);
7         setMotorSpeed(leftMotor,50);
8         while(getUSDistance(sonarSensor) > NEAR){}
9         setMotorSpeed(rightMotor,0);
10        setMotorSpeed(leftMotor,-50);
11        sleep(400);
12    }
13 }
```

[課題 5.4] ロボットがヒトの後ろをついてくるようなプログラムを作りたい。ヒトの代わりに、下敷き等をロボットの前にかざして、ロボットの前後運動を操作できるプログラムをここでは作成する。超音波センサでロボットの前方にある物体までの距離を測り、前方の物体が動いても常に 30cm 程度の距離を保つように動くことができるプログラムを作りなさい。

5.6 ジャイロセンサを使う

ジャイロセンサは、回転の角速度を計測することができるセンサーです。その積分により回転角度も得ることが出来ますが、積分により求められる**角度は時間とともに誤差が蓄積するので注意しましょう**。以下がジャイロセンサを用いるための関数です。

- `long getGyroDegrees(port):` ジャイロセンサを MindStorm に接続したとき、もしくは MindStorm を起動してからの回転角度を度数法で返す。時計回りに回転する時にはこの値は増え、反時計回りのときは減る。
- `void resetGyro(port):` ジャイロセンサの初期化 (現在の角度を 0 度と定義し直す) をする。上述のように、一般にジャイロセンサを用いて求めた角度には誤差が蓄積しやすいので、MindStorm がある方向を向くとわかる適当なタイミングが時々あるなら、そのタイミングでジャイロセンサを初期化すると良い。
- `void getGyroHeading(port):` 最後にジャイロセンサをリセットしてからの回転角度を度数法で返す。時計回りに回転する時にはこの値は増え、反時計回りのときは減る。
- `long getGyroRate(port):` ジャイロセンサの角速度 [degree/s] を返す。返り値は-440 degree/s から 440 degree/s の値を取る。

以下はジャイロセンサを使って 90 度回転するプログラム例です。

```

1 void rotate(int pwr)
2 {
3     setMotorSpeed(rightMotor, -pwr);
4     setMotorSpeed(leftMotor, pwr);
5 }
6
7 task main()
8 {
9     resetGyro(gyroSensor);
10
11    while(getGyroDegrees(gyroSensor) < 90){
12        rotate(50);
13    }
14
15    rotate(0);
16 }
```

以下は、ジャイロセンサで計測した角速度に応じて進行方向を調整するプログラムです。

```

1 void turn_right(int power,int turn)
2 {
3     setMotorSpeed(rightMotor, power-turn);
4     setMotorSpeed(leftMotor, power+turn);
5 }
6
7 task main()
8 {
9     while(true){
10         long omega=getGyroRate(gyroSensor);
11
12         if(abs(omega) > 3){
13             turn_right(50,5*sgn(omega));
14         } else {
15             turn_right(50,0);
16         }
17     }
18 }
```

`sgn(float)` は、指定した値が正なら 1 を、負なら-1 を、0 なら 0 を返す関数、`abs(float)` は絶対値を返す関数です。

[問 5.3] MindStorm に接続させている全てのセンサから得られる様々な値 (接触センサ、超音波センサ、光センサ (R,G,B)、ジャイロセンサ (角度、角速度)) の一覧を画面に表示するプログラムを作りなさい。センサ値は 1 秒ごとに更新し、光センサで得られる明度と (R,G,B) 値は 1 秒ごとに交互に表示するようにしなさい。

[課題 5.5] 課題 5.3 と同様に赤く塗りつぶされた 2 地点がある。その一方から出発し、他方に到達したところで 180 度回転して、もとの場所に戻る動作を繰り返すプログラムを以下のように作り、その動作を確認しなさい。回転角度の検出にはジャイロセンサを用いること。

6 並列タスク

RobotC には**並列計算**の機能があり、複数の作業を同時に使うことができます。**並列して実行する作業単位**を **task** と呼びます。task main() も task を定義する関数のひとつで、プログラム実行時にはまずこれが呼び出されます。他の task は task main() から関数 startTask() で呼び出します。以下は、ロボットが障害物に当たったら方向転換しながら動き回るプログラム例です。ロボットを直進させるタスク move_forward() と、タッチセンサの入力を監視しながら入力があれば方向転換をするタスク check_touch() を並列処理しています。

```

1  TSemaphore flag;
2
3  void forward(int pwr)
4  {
5      setMotorSpeed(rightMotor,pwr);
6      setMotorSpeed(leftMotor,pwr);
7  }
8
9  task move_forward()
10 {
11     while(true){
12         semaphoreLock(flag);
13         forward(50);
14         semaphoreUnlock(flag);
15     }
16 }
17
18 task check_touch()
19 {
20     while(true){
21         semaphoreLock(flag);
22         if(getTouchValue(touchSensor) == true){
23             forward(-50);
24             sleep(500);
25             setMotorSpeed(leftMotor,50);
26             sleep(500);
27         }
28         semaphoreUnlock(flag);
29     }
30 }
31
32 task main()
33 {
34     semaphoreInitialize(flag); //Initialize the semaphore
35     startTask(move_forward);
36     startTask(check_touch);
37
38     while(true); // タスク実行をずっと続ける
39 }
```

複数のタスクを同時に実行していると、複数のタスクが1つのモータに異なる方向に動くように命令を出してしまいます。このような問題を防ぐため、**TSemaphore** (セマフォ、手旗信号) という型の変数を使います。上の例では、以下のようにセマフォを使っています。

1. 3行目で **TSemaphore** 型の変数 **flag** を宣言します。
2. タスク **main** で関数 **semaphoreInitialize()** により **flag** を初期化します。**このようにセマフォの初期化は main タスクで忘れずに実行しましょう。**
3. タスク **main** では次に、**StartTask()** で2つのtaskを順に起動しています。この2つのtaskがセマフォ **flag** を利用してモータの実行権の取り合いをします。先に関数 **semaphoreLock()** を実行した方が変数 **flag** にある値を書き込むことで、セマフォのロック状態であることを示し、他方が **flag** をロックできなくなります。
4. モータの実行が終わったときには、**semaphoreUnLock()** により **flag** の値を変更することで、セマフォのロック解除を示す状態にします。 **flag** がロック解除されるまで、もう一方のtaskは **semaphoreLock()** できず、その後の命令も実行されません。

セマフォを用いた実行権の奪い合いは **mutex** (相互排除) とも呼ばれ、並列計算で一般によく使われます。

[課題 6.1] 以下は、通常時は前進、障害物に当たったら後退して方向転換をするプログラムを並列タスクを利用して作ろうとしたものです。悪い点を指摘し、修正案を書きなさい。

```

1 task check_touch()
2 {
3     while(true){
4         if(getTouchValue(touchSensor) == true){
5             setMotorSpeed(rightMotor,-50);
6             setMotorSpeed(leftMotor,-50);
7             sleep(700);
8             setMotorSpeed(rightMotor,50);
9             sleep(700);
10        }
11    }
12 }
13 task forward()
14 {
15     while(true){
16         setMotorSpeed(rightMotor,50);
17         setMotorSpeed(leftMotor,50);
18         sleep(100);
19     }
20 }
21 task main()
22 {
23     startTask(check_touch);
24     startTask(forward);
25
26     while(true);

```

[課題 6.2] ライントレースをしながらもし何かにぶつかりそうになったら、もしくはぶつかったら少し後ろに下がってからランダムに方向転換してラインを探し、ラインを見つけたらまたライントレースをするプログラムを並列タスクを利用して作りましょう。障害物の検出は超音波センサとタッチセンサを使い、どちらのセンサが反応しても障害物を回避できるようにしましょう。

付録 A 用意されている数学関数の例

- `float abs(float x)` x の絶対値を返す
- `short sgn(float x)` 指定した値が正なら 1 を, 負なら-1 を, 0 なら 0 を返す.
- `float sqrt(float x)` x の平方根を返す
- `float sin(float x)` 指定した角度 x [rad] に対する sin 関数の値を返す
- `float cos(float x)` 指定した角度 x [rad] に対する cos 関数の値を返す
- `float exp(float x)` $\exp(x)$ の値を返す
- `float log(float x)` $\log(x)$ の値を返す
- `int random(int n)` 0 から n までの整数乱数を返す。
- `void srand(const long x)` 亂数の初期値を設定する。例えば, `srand(nSysTime)` とすると, `nSysTime` の値により乱数の初期化をする。`nSysTime` はロボットの起動時間 (msec) の下 16 ビットの値を示す変数。

その他

RobotC のメニュー”View/Function Library (Text)”を有効にすると, RobotC で使うことができる関数や変数の一覧が表示されます。

索引

Symbols

#define 11

A

abs() 17, 21

B

bool 7

byte 7

C

char 7

cos() 21

D

displayBigTextLine() 7

displayCenteredBigTextLine() 7

displayCenteredTextLine() 7

displayTextLine() 7

do~while 文 12

E

eraseDisplay() 7

exp() 21

F

float 7

for 文 11

function 8

G

getColorAmbient() 14

getColorReflected() 14

getColorRGB 15

getGyroDegrees() 16

getGyroHeading() 16

getGyroRatio() 16

getTouchValue() 13

getUSDistance() 16

I

if 文 11

int 7

L

LEGO MindStorm 2

log() 21

long 7

M

MindStorm 2

mutex 19

N

nSysTime 11

R

random() 11, 21

resetGyro() 16

RobotC 2

S

semaphoreInitialize() 19

semaphoreLock() 19

semaphoreUnlock() 19
 SensorValue 15
 setMotorSpeed() 6
 sgn() 17, 21
 short 7
 sin() 21
 sleep() 6
 sqrt() 21
 srand() 11, 21
 startTask() 18
 string 7

T

task 18
 TSemaphore 19
 TSemaphore() 19

U

ubyte 7

V

void 7

W

while 文 11
 word 7

あ

アドレス渡し 8, 9
 演算子 7

か

カラーセンサ 3
 構造体 9
 コメント 6

さ

ジャイロセンサ 3
 条件式 11
 静的変数 9

た

タッチセンサ 3
 超音波センサ 3

は

配列 9
 並列計算 18
 変数 7

ら

乱数 11, 21