リアルタイム OS を用いた mbed ライントレーサ

山田　理園

1.概略

ロボットを制御するマイコンに mbed を利用したライントレーサです．ライントレーサとは床面上に引かれたライン上を走行する自律移動ロボットです．走行用モータにステッピングモータを利用しており，電源には単 3 電池を使用しています．車体の前方にはライン検知のためにセンサを 6 個，スタート・ゴールマーカーとコーナーマーカー検知のために左右にセンサを 1 個ずつ搭載しています．センサを用いてラインの位置を読み取り，モータを制御して走行します．

mbed のソフトウェア開発はインターネットに接続された PC から mbed の公式サイトにアクセスし，ブラウザ上でプログラムの編集や保存，コンパイルが可能である．開発したプログラムをコンパイルすると， PC にバイナリファイルがダウンロードされます．PC と mbed 基板を USB ケーブルで接続し，ダウンロードされたバイナリファイルを mbed のメモリに書き込むことで，ロボットを動作させることができます．

また，ソフトウェアはリアルタイム OS（RTOS）を使って作成しています．リアルタイム OS に関するライブラリは mbed が公式に公開されているので，それを利用しました．

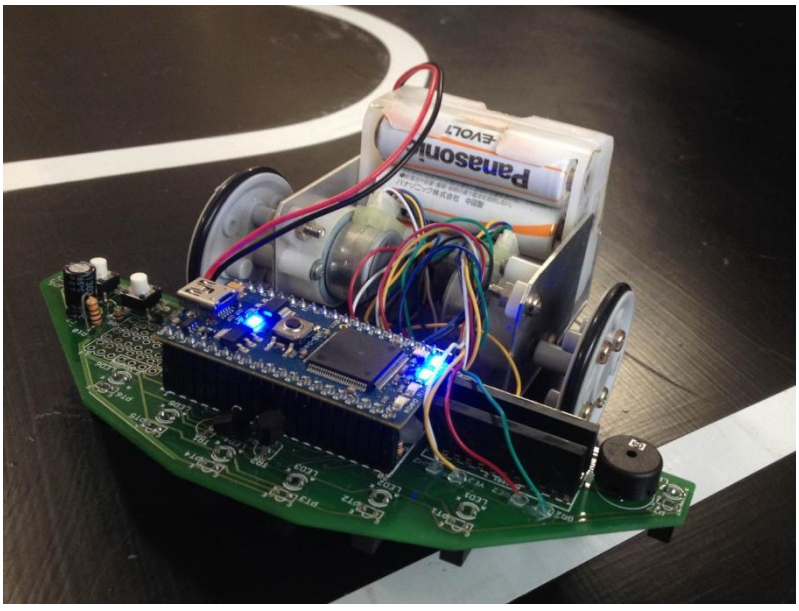

図 1　ライントレーサの全体図

2.回路について

図 2 はライントレーサに搭載されている基板の回路図です．mbed には 6 本のアナログ入力があり，1 本はユニバーサル領域，2 本はスタート・ゴールマーカーとコーナーマーカーセンサに接続されています．

そのため，残りの 3 本で 6 つのラインセンサに対応する必要があります．フォトトランジスタ 2 つを 1 組とし，3 本のうちの 1 本をアナログ入力に接続することで，ラインセンサの幅をできるだけ広くとることができるようになっています．ステッピングモータの駆動回路はパワーFET モジュール（MP4401）にモータを直結して回しています．その他に，基板にはモード選択用のスイッチ（CN2-1），スタートスイッチ（CN2-2）を搭載しています．

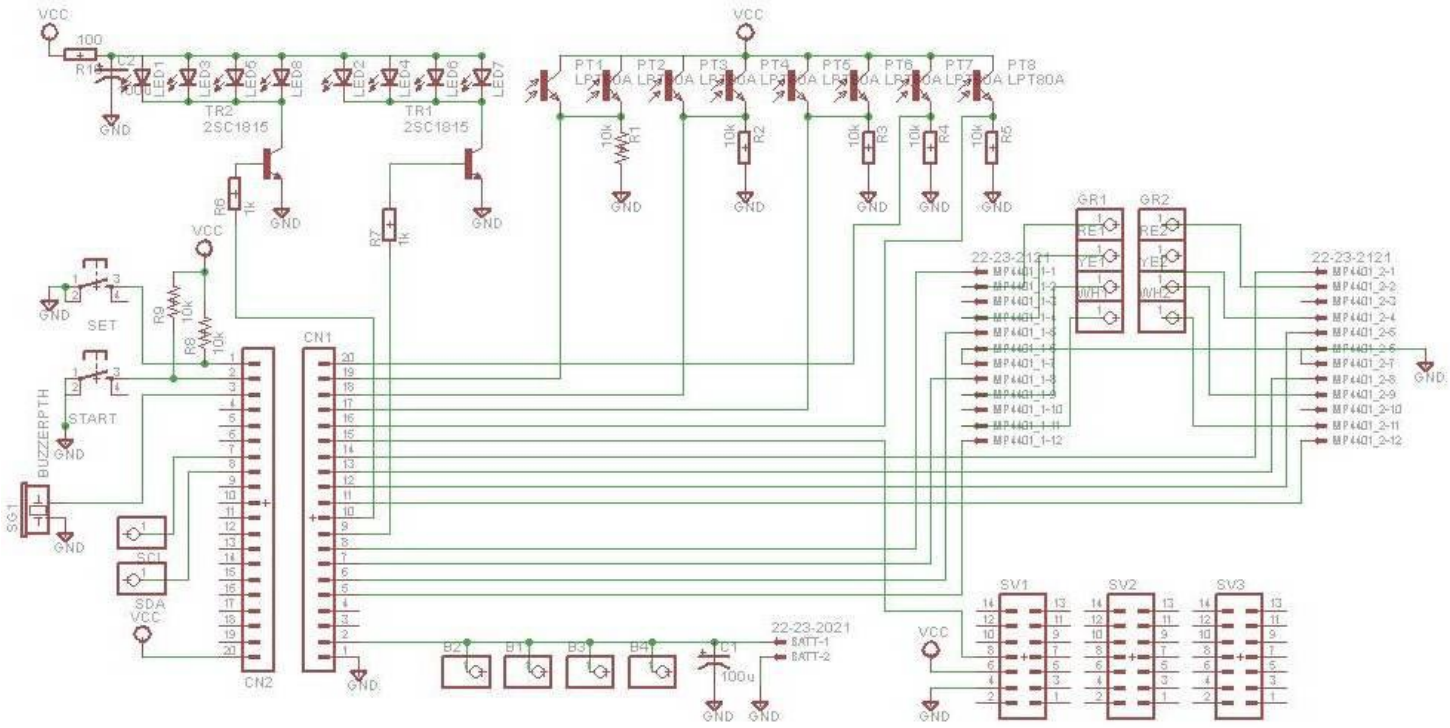

図 2　回路図

3.ソフトウェア開発環境

3.1　mbed について

mbed とは ARM コアを搭載したワンボードマイコンのことです．30 本のデジタル I/O ポートと共用の 6 本のアナログ入力ポート，シリアル，USB,CAN, Ethernet などの通信ポートが利用できます． ソフトウェアの開発環境は PC にインストールする必要がなく，インターネットに接続された PC から mbed の公式サイトにアクセスすることにより，ブラウザ上でプログラムの編集，保存，コンパイルが可能です．mbed の公式サイトでは，様々なライブラリが公開されているほか，個人の作成したプログラムやライブラリの公開もできる仕組みが用意されています．

3.2　RTOS について

リアルタイム OS（RTOS）とは，複数のタスクが並行動作するマルチタスクや一定時間内に処理を開始するリアルタイム性を重視して，その機能を実装する OS のことです．

携帯電話やテレビ，自動車，産業用ロボットなどマイコンを搭載した機器を組込みシステムと呼んでいます［1］．多くの場合，ある一つの組込みシステムの中には複数のマイコンが搭載されており，マイコンはそれぞれが担当する機器制御をしています．これらがばらばらの動作をするだけでは機器全体としての動作を制御できません．互いの処理は独立していながら，互いに通信を行い全体として強調動作をしなければなりません．組込みシステムにとって，複数のタスクが並行動作するマルチタスクや，外部からの事象によって必要なタスクへ素早く切り替わり一定時間内に処理を開始するリアルタイム性は重要なものになります．リアルタイム OS を使うことでこれらの要求を満たし，効率よくソフトウェアの開発を行うことができます．

mbed では RTOS のライブラリが公式に公開されているため，このライブラリをインポートすることで RTOS を使ったソフトウェアの開発が可能になります．サンプルプログラムやリファレンスに沿ってプログ小ラムを作成することで RTOS の標準的な機能を利用できます．文献[2]の mbed-rtos に関する Handbook に RTOS の持つ様々な機能の解説があるのでそちらも参照してみてください． mbed の RTOS ではタスクのこ

とをスレッドと呼びます.

4.サンプルプログラムと操作方法

サンプルプログラムは以下の URL のサイトに公開しています.

https://developer.mbed.org/users/hayama_lab/code/mbedLinetracer_RTOS/

ダウンロードして，mbed の program workspace にインポートするとプログラムの編集やコンパイルが可能です．コンパイルしてダウンロードしたバイナリファイルを mbed に書き込んでください.

操作方法について説明します.

① リセットボタンを押し，プログラムを動作させます.

② SET ボタンで動作モードを選択することができます．動作モードは mbed ボード上にある 4 個の LED に 2 進数で表示されます.

③ START ボタンを押すことで選択したモードが動作します．サンプルプログラムの動作モードは以下のものがあります.

モード 0： センサチェック，センサの値をシリアル通信で PC 画面に表示させます.

モード 1： 500 ステップ前進

モード 2： 500 ステップ右回転

モード 3： ライントレース走行

モード 0：センサチェックでは，USB 接続を利用してシリアル通信を行うため，以下の URL のサイトを参考にしてドライバーをインストールする必要があります.

https://mbed.org/handbook/Windows-serial-configuration

5.サンプルプログラムの説明

5.1 タスクと RTOS タイマーの定義

プログラムの main 関数ではスレッドと RTOS タイマーの定義を行っています．また，main も 1 つのスレッドであるため，スイッチ入力の監視や各種動作モードの選択・起動を行います.

スレッドと RTOS タイマーは以下のリスト 1 の様に定義されています.

リスト 1 スレッドと RTOS タイマーの定義

```
Thread thread1(motorR_thread , NULL ,  osPriorityHigh);        // 右モータのスレッド定義
Thread thread2(motorL_thread , NULL ,  osPriorityHigh);        // 左モータのスレッド定義

RtosTimer sensor_timer(Sensor, osTimerPeriodic, (void *)0);    // RTOS タイマー(センサ読取り)
sensor_timer.start(5);                                         // RTOS タイマーの開始
```

1）左右モータを別々のタスクで作成

床面上のラインに沿って左右モータの加減速を行いながら走行します.

2）センサの読取り

ライントレーサはラインセンサ 6 個，スタート・ゴールマーカ―センサとコーナーマーカーをそれぞれ 1 個搭載しています．6 個のラインセンサでラインの有無を検出します．走行時は床面上のラインの情報を常に更新しなければならないので，センサ読取りプログラムは 5ms 毎に起動します.

3）スイッチ監視・動作モード起動

メインタスクは，スイッチ入力の監視やテストプログラム，コース走行プログラムなど動作モードの選択・

起動を行います.

5.2　モータ回転スレッド

モータ回転のスレッドについて説明します．右モータ回転のスレッドをリスト 2 に示します.

リスト 2　右モータ回転スレッド

```
//-------------------------------------------------------------------------
// thread of right motor rotation
// motor rotation,   mode = 0: free,  1: forward,  2: reverse,  3: break
//-------------------------------------------------------------------------
void motorR_thread(void const *argsument)
{

    Thread::signal_wait(0x1);                   // シグナルが入ったらスレッドを動作

    while (true) {
        f=1;
        if(modeR==1) {                          // 前進
            if (patR < 3) patR++;
            else patR = 0;
        }
        if(modeR==2) {                          // 後退
            if (patR > 0) patR--;
            else patR = 3;
        }
        cntR++;
        if (modeR==0) patR=4;                   // フリー
        motorR= RMOTOR[patR];                   // ステッピングモータの励磁出力
        waitR=Motor_Control();                  // スレッドの停止時間
        Thread::wait(waitR);                    // スレッドを停止
    }
}
```

スレッドは main 部で定義すると同時に動作するので，動作モードの選択中にモータが回転を始めてしまうのを防ぐために Thread::signal_wait(0x1); を利用することにします．これは 0x1 というシグナルが通知されるまでスレッドの動作を停止させるものです．main 部で動作テストプログラムやコース走行プログラムなどが起動された場合，以下のように右モータスレッドにシグナル 0x1 が通知され，右モータが回転を始めます.

```
thread1.signal_set(0x1);
```

次にモータの回転方法について説明します．右モータの回転モード modeR は，フリー（0），前進（1），後退（2），ブレーキ（3）の 4 つの状態を取ります.

ステッピングモータはパルスによって制御を行うモータであり，一回のステップで決まった角度だけ回転するという特徴があります．ステッピングモータには数通りの励磁方式かあり，1 相励磁方式はコイルを一つずつ励磁していく方法です．2 相励磁方式はコイルを 2 つずつ励磁していくもので，1 相励磁に比べておよそ 2 倍のト

ルクを発生させます[3]．今回は 2 相励磁方式でステッピングモータを駆動させています．ステッピングモータの励磁出力は以下のようにバスで定義しています

ステッピングモータの励磁出力
BusOut motorR(p5,　p6,　p7,　p8);

モータの励磁パターンは以下のように定義しています．

ステッピングモータの励磁パターン
const unsigned char RMOTOR[]={0x09, 0x0C, 0x06, 0x03, 0x00};

ライントレーサを前進させる際は，この配列の引数を 0 から 3 まで変化させ，先ほどバスで定義した motorR へ出力することで前進できます．励磁パターンの配列の引数 patR はリスト 2 より，0 から 3 までの値を取ります．

次に右モータの回転速度の変化について説明します．回転速度の変化にはスレッドの停止を利用します．図 3 のようにライントレーサがライン上からずれて走行している場合を考えます．右モータは減速が必要であるため，右モータを制御するスレッドの実行回数を減らすことで右モータの減速を実現することができます．そこで今回は以下のように右モータ回転のスレッドを waitR ミリ秒間停止させています．

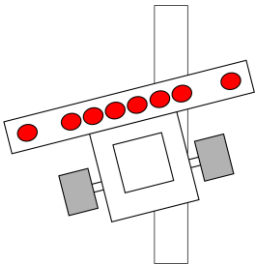

図 3　車体が左に傾いた場合

Thread::wait(waitR);

この waitR は Motor_Control 関数の中で，sensDout の値に応じて変化します．ライントレーサがライン上から大幅にずれて走行している場合 waitR の値も大きくなります．

左モータは，右モータと同じ制御方法でステッピングモータを駆動させています．

5.3　RTOS タイマーによるセンサ読み取り

RTOS タイマーによって起動されるセンサ読取りについて説明します．Sensor 関数は 5ms 毎にタイマー割り込みによって起動されます．Sensor 関数の中で以下のリスト 3 ようにしてアナログ入力に接続されたフォトトランジスタのアナログ値を取得します．

リスト 3　センサ値の取得

```
ledSGout=1;                    // LED-ON
sens1=pt12;                    // ラインセンサ値の取得
sens3=pt34;
sens5=pt56;
sensSG=ptSG;                   // スタート・ゴールマーカーセンサ値の取得
ledSGout=0;                    // LED-OFF

ledCout=1;                     // LED-ON
sens2=pt12;                    // ラインセンサ値の取得
sens4=pt34;
sens6=pt56;
sensC=ptC;                     // コーナーマーカーセンサ値の取得
ledCout=0;                     // LED-OFF
```

ラインセンサを半分に分け，発光した赤外線 LED の光をフォトトランジスタで受光し，交互に読み取りを行っています．同様にスタート・ゴールマーカーとコーナーマーカーも交互に読み取りを行っています．

床面上にひかれたラインの有無の検出にはフォトトランジスタで受光したセンサ値を，設定されたしきい値と比較することで 0 または 1 で判断します．図 4 のように，その値を十進化し，配列 sensArray を用いてラインの位置を±5 の数値で出力し，変数 sensDout に書き込まれます．そのプログラムはリスト 4 のようになっています．

リスト 4　ライン検出

```
sensD=0;
if (sens1>STH ) sensD  |= 0x20; else sensD &= ~(0x20);
if (sens2>STH ) sensD  |= 0x10; else sensD &= ~(0x10);
if (sens3>STH ) sensD  |= 0x08; else sensD &= ~(0x08);
if (sens4>STH ) sensD  |= 0x04; else sensD &= ~(0x04);
if (sens5>STH ) sensD  |= 0x02; else sensD &= ~(0x02);
if (sens6>STH ) sensD  |= 0x01; else sensD &= ~(0x01);
sensDout=sensArray[sensD];
```

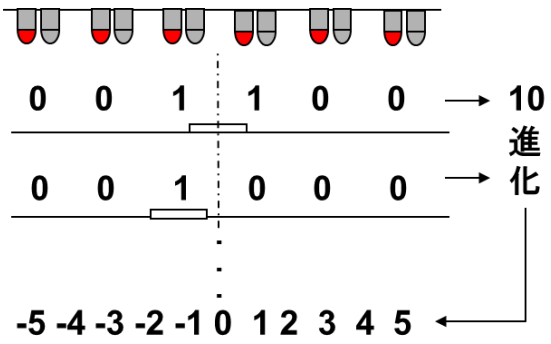

図 4　ライン検出

配列 sensArray は以下のように定義されています．

```
const int sensArray[64]=
{0,5,3,4,1,0,2,0,-1,0,0,0,0,0,0,0,-3,0,0,0,0,0,0,0,-2,0,0,0,0,0,0,0,-5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-4,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0};
```

また，スタート・ゴールマーカーやコーナーマーカーセンサの値もしきい値を設定し，マーカーの有無を判断しています．スタート・ゴールマーカを 2 回検出すると止まるようになっています．その他 Sensor 関数の中ではブザーの ON や OFF も行っており，スタート・ゴールマーカーおよびコーナーマーカーを検知するとブザーが鳴ります．

5.4　main 関数について

プログラムの main 関数ではスレッドと RTOS タイマーの定義を行った後に， while 文の中でスイッチ入力の監視や各種動作モードの選択・起動を行います．while 文の中のスイッチ入力監視のプログラムは以下のリスト 5 のようになっています．

リスト 5　スイッチ入力監視

```
while (startSw==1) {
    if (setSw==0) {
        wait(0.1);
        beep(50);
        while (setSw==0);
        wait(0.1);
        pmode++;
        if (pmode>3) pmode=0;
    }
    leds=pmode;                          //4 個の LED によって動作モードを二進数表示
}
```

モードを選択し，スタートボタンが押されたら SET スイッチの値に応じて分岐を行います．各種テストプロ

グラムと，コース走行プログラムが用意されており，以下のリスト 6 のようになっています.

リスト 6　動作モード選択

```
switch(pmode) {
        case  0: check_sens();                  // センサチェック
                 break;
        case  1: thread1.signal_set(0x1);    // 前進
                 thread2.signal_set(0x1);
                 runTurn(1,1,500);
                 break;
        case  2: thread1.signal_set(0x1);    // 右回転
                 thread2.signal_set(0x1);
                 runTurn(2,1,500);
                 break;
        case  3: thread1.signal_set(0x1);    // コース走行
                 thread2.signal_set(0x1);
                 run();
                 break;
    }
```

case 0 では USB 接続を利用してシリアル通信を行いセンサのチェックを行います．case 1 の runTurn(1,1,500);はライントレーサを 500 ステップ前進させることができ，case 2 の runTurn(2,1,500);では 500 ステップ右回転させることができます．床面上のコース走行は case 3 の run();によって行います.

参考文献

[1] 宇野俊夫，“C 言語によるリアルタイム組込み OS 自作講座”，翔泳社，(2012)

[2] RTOS - Handbook | mbed，https://developer.mbed.org/handbook/RTOS

[3] 浅野健一，“勝てるロボコン　ロボトレーサの作り方”，東京電機大学出版局，(2002)

サンプルプログラム

```
//*********************************
//
// mbed Linetracer using RTOS
//
// Rion Yamada(National Institute of Technology)
//
//*********************************
#include "mbed.h"
#include "rtos.h"

Serial pc(USBTX, USBRX);                    // serial port

// run parameters
#define STH       0.2                       // threshold value for digital photo sensor
#define SGTH      0.2                       // threshold value for start/goal marker
#define CTH       0.1                       // threshold value for corner marker

// pattern table for stepping motor
const unsigned char RMOTOR[]= {0x09, 0x0C, 0x06, 0x03, 0x00};     // magnetization pattern for left motor
const unsigned char LMOTOR[]= {0x03, 0x06, 0x0C, 0x09, 0x00};     // magnetization pattern for right motor

const int sensArray[64]=
{0,5,3,4,1,0,2,0,-1,0,0,0,0,0,0,0,-3,0,0,0,0,0,0,0,-2,0,0,0,0,0,0,0,-5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-4,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0};

unsigned char pmode=0;                      //   program mode

volatile float sens1, sens2, sens3, sens4, sens5, sens6, sensSG, sensC;          // sensor values
volatile int     sensD,sensDout=0;

unsigned char frun, fmarker;
volatile int markerSG, markerC, markerX;

int cntGoal=0;       // couter for run after goal



volatile unsigned char modeR=0, modeL=0;
volatile int stepR, stepL;                          // varilable for set step of motor
volatile unsigned char patR=0, patL=0;                // index of motor pattern
volatile int cntR, cntL;                            // count of motor steps
volatile int waitR,waitL;                                 //count of thread wait
```

```
volatile int bp=0;                                  // couter for beep
volatile int f;
volatile int r,l;



BusOut leds( LED4, LED3, LED2, LED1 );       // for LED display
BusOut motorR(p5,   p6,   p7,   p8 );          // output for right motor
BusOut motorL(p11, p12, p13, p14 );          // output for left motor

AnalogIn pt12(p19);                                 // front right sensor, analog input
AnalogIn pt34(p18);                                 // front left sensor, analog input
AnalogIn pt56(p17);                                 // right sensor, analog input
AnalogIn ptC(p20);                                  // left sensor, analog input
AnalogIn ptSG(p16);                                   // left sensor, analog input
AnalogIn gyro(p15);                                 // for Gyro, analog input, reserved

DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

DigitalIn setSw(p21);                                  // set-switch, digital input
DigitalIn startSw(p22);                             // start-switch, digital input
DigitalOut ledCout(p9);                             // LED output signal for corner marker
DigitalOut ledSGout(p10);                           // LED output signal for start/goal marker
DigitalOut buzzer(p23);                                // buzzer out

//------------------------------------------------------------------------
// RTOS Timer
//------------------------------------------------------------------------
void Sensor(void const *argument)
{

    ledSGout=1;                                     // LED-ON
    sens1=pt12;                                     // measure all sensor values
    sens3=pt34;
    sens5=pt56;
    sensSG=ptSG;
    ledSGout=0;                                     // LED-OFF

    ledCout=1;                                      // LED-ON
```

```
    sens2=pt12;                                 // measure all sensor values
    sens4=pt34;
    sens6=pt56;
    sensC=ptC;
    ledCout=0;                                  // LED-OFF

    sensD=0;
    if (sens1>STH ) sensD |= 0x20; else sensD &= ~(0x20);
    if (sens2>STH ) sensD |= 0x10; else sensD &= ~(0x10);
    if (sens3>STH ) sensD |= 0x08; else sensD &= ~(0x08);
    if (sens4>STH ) sensD |= 0x04; else sensD &= ~(0x04);
    if (sens5>STH ) sensD |= 0x02; else sensD &= ~(0x02);
    if (sens6>STH ) sensD |= 0x01; else sensD &= ~(0x01);
    sensDout=sensArray[sensD];

    if (sensSG>STH) markerSG++;
    else if (markerSG>0) markerSG--;
    if (sensC>CTH ) markerC++;
    else if (markerC>0)   markerC--;
    // cross line detection
    if (markerSG>1 && markerC>1) markerX=1;                 // both marker
    if (markerX==1 && markerSG==0 && markerC==0) markerX=0; // ignore cross line

    // buzzer
    if (bp>0) {
        bp--;
        if (buzzer==1) buzzer=0;
        else buzzer=1;      // alternate ON-OFF
    }
}

//---------------------------------------------------------------------
// Motor speed control
//---------------------------------------------------------------------
int Motor_Control()
{

    r=l=5;
    if (sensDout>0){
        r+=sensDout*4;
    } else {
        l-=sensDout*4;
```

```
    }
    if(f==1) return(r);
    else return(l);

}

//-------------------------------------------------------------------
// thread of right motor rotation
// motor rotation,   mode = 0: free,  1: forward,  2: reverse,  3: break
//-------------------------------------------------------------------
void motorR_thread(void const *argsument)
{

    Thread::signal_wait(0x1);

    while (true) {
        f=1;
        if(modeR==1) {
            if (patR < 3) patR++;
            else patR = 0;
        }
        if(modeR==2) {
            if (patR > 0) patR--;
            else patR = 3;
        }
        cntR++;                                          // count up right moter step
        if (modeR==0) patR=4;     // motor free when mode=0
        motorR= RMOTOR[patR];
        waitR=Motor_Control();
        Thread::wait(waitR);
    }
}

//-------------------------------------------------------------------
// thread of right motor rotation
// motor rotation,   mode = 0: free,  1: forward,  2: reverse,  3: break
//-------------------------------------------------------------------
void motorL_thread(void const *argument)
{

    Thread::signal_wait(0x1);
```

```
    while (true) {
        f=0;                //run の戻り値が左モータ専用
        if(modeL==1) {
            if (patL < 3) patL++;
            else patL = 0;
        }
        if(modeL==2) {
            if (patL > 0) patL--;
            else patL = 3;
        }
        if(modeL==3) patL=3;
        cntL++;                                         // count up left moter step
        if (modeL==0) patL=4;    // motor free when mode=0
        motorL= LMOTOR[patL];
        waitL=Motor_Control();
        Thread::wait(waitL);
    }
}

// ------------------------------------------------
// beep
// ------------------------------------------------
void beep(int n)
{
    bp=n;       // set beep couter
}

//--------------------------------------------------------------------------
// check sensor value using serial port
//--------------------------------------------------------------------------
void check_sens()
{
    while (1) {
        pc.printf("¥f");
        pc.printf("Sensor C:%f ¥n",sensC);
        pc.printf("Sensor SG:%f ¥n",sensSG);
        pc.printf("Sensor  1:%f ¥n",sens1);
        pc.printf("Sensor  2:%f ¥n",sens2);
        pc.printf("Sensor  3:%f ¥n",sens3);
        pc.printf("Sensor  4:%f ¥n",sens4);
        pc.printf("Sensor  5:%f ¥n",sens5);
        pc.printf("Sensor  6:%f ¥n",sens6);
```

```
            wait (1);
        }
}

//-----------------------------------------------------------------------
// break and release motors
//-----------------------------------------------------------------------
void run_release()
{
        modeR=0;
        modeL=0;                        // motor release
}
void run_break()
{
        modeR=3;
        modeL=3;                          // mode 0 means break the motor
        wait(0.5);
        run_release();
}

//-----------------------------------------------------------------------
// run and turn
// (mR,mL)=(1,1):forward, (2,1): turn right, (1,2): turn left, (2,2): Reverse
// nstep: number of step
//-----------------------------------------------------------------------
void runTurn(int mR,int mL , int nstep )
{
        modeR=mR;
        modeL=mL;
        cntR=0;
        stepR=nstep;
        while (cntR<stepR);
        run_break();
}

//----------------------
// run
//----------------------
void run()
{

        markerSG=0; markerC=0; markerX=0; fmarker=0;
```

```
    frun=0;
    modeR=modeL=1;

    while(startSw==1 ) {

        // corner marker check
        if (markerX==1) fmarker=0;
        if (markerX==0 && fmarker==0 && markerC>5) fmarker=1;
        if (markerX==0 && fmarker==1 && markerC==0) {
            fmarker=0;
            beep(100);
        }

        // start/goal marker check
        if (frun==0 && sensSG>SGTH) {
            frun=1;      // start marker detect
            beep(100);
        }
        if (frun==1 && markerSG==0) frun=2;                    // start marker fix
        if (frun==2 && sensSG>SGTH) frun=3;                    // goal marker detect
        if (frun==3 && markerX==1) {
            frun=2;      // ignor cross line
        }
        if (frun==3 && markerSG==0) {                          // goal marker fix
            frun=4;
            beep(50);
            break;
        }
    }
    run_break();
}

int main()
{

    Thread thread1(motorR_thread , NULL ,   osPriorityHigh);        // thread of right motor control
    Thread thread2(motorL_thread , NULL ,   osPriorityHigh);        // thread of left motor control

    RtosTimer sensor_timer(Sensor, osTimerPeriodic, (void *)0);    // set RTOS timer for sensor
    sensor_timer.start(5);                                          // start RTOS timer for sensor

    while (1) {
```

```
        while (startSw==1) {                // program mode selection
            if (setSw==0) {
                wait(0.1);
                beep(50);
                while (setSw==0);
                wait(0.1);
                pmode++;
                if (pmode>3) pmode=0;
            }
            leds=pmode;
        }
        leds=0;
        beep(50);
        wait(0.5);

        // run selected functions
        switch(pmode) {
            case  0: check_sens();                  // check sensors
                     break;
            case  1: thread1.signal_set(0x1);   // run forward
                     thread2.signal_set(0x1);
                     runTurn(1,1,500);
                     break;
            case  2: thread1.signal_set(0x1);   // turn right
                     thread2.signal_set(0x1);
                     runTurn(2,1,500);
                     break;
            case  3: thread1.signal_set(0x1);   // trace run
                     thread2.signal_set(0x1);
                     run();
                     break;
        }
    }
}
```