リアルタイム OS を用いた mbed マイクロマウス

山田　理園

1.概略

ロボットを制御するマイコンに mbed を利用したマイクロマウスです．マイクロマウスとは，迷路を自力で探索しゴールへ走り抜ける自律型移動ロボットである．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．迷路内の壁を検知するためにフォトセンサを 4 個搭載しています．壁の有無をフォトセンサで検知し，ステッピング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す．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 が公式に公開しているので，それを利用しました．

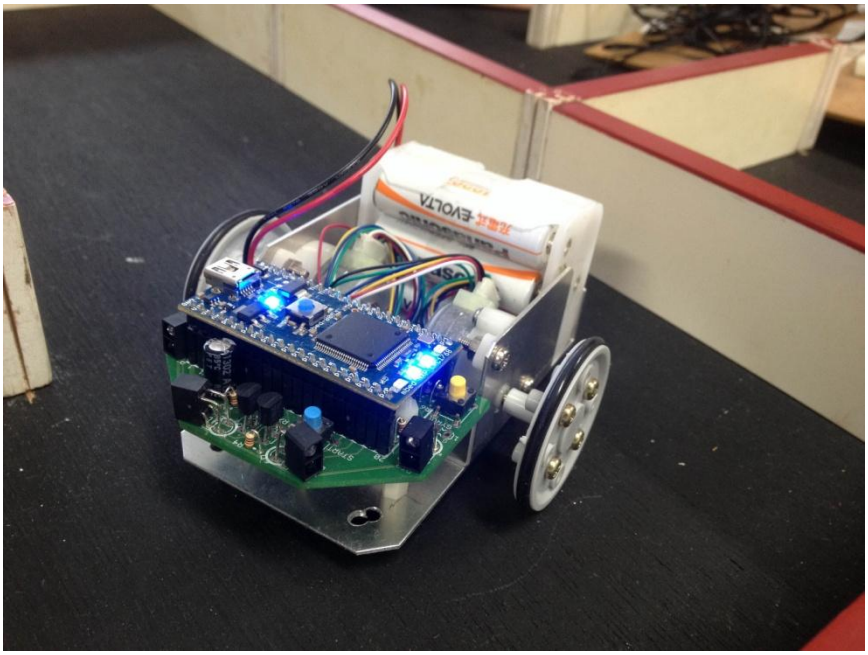

図 1　マイクロマウスの全体図

2. 回路について

図 2 はマイクロマウスに搭載されている基板の回路図です．マイクロマウスには迷路内の壁を検知するためにフォトセンサが前に 2 個，左右に 1 個ずつ搭載されています．前壁を検知する際には LED1 と LED2 を点灯させ，PT1 と PT2 のフォトトランジスタの出力をアナログ値で読み取ります．左右の壁を検知する際には LED3 と LED4 を点灯させて，PT3 と PT4 のフォトトランジスタの出力をアナログ値で読み取ります．

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．

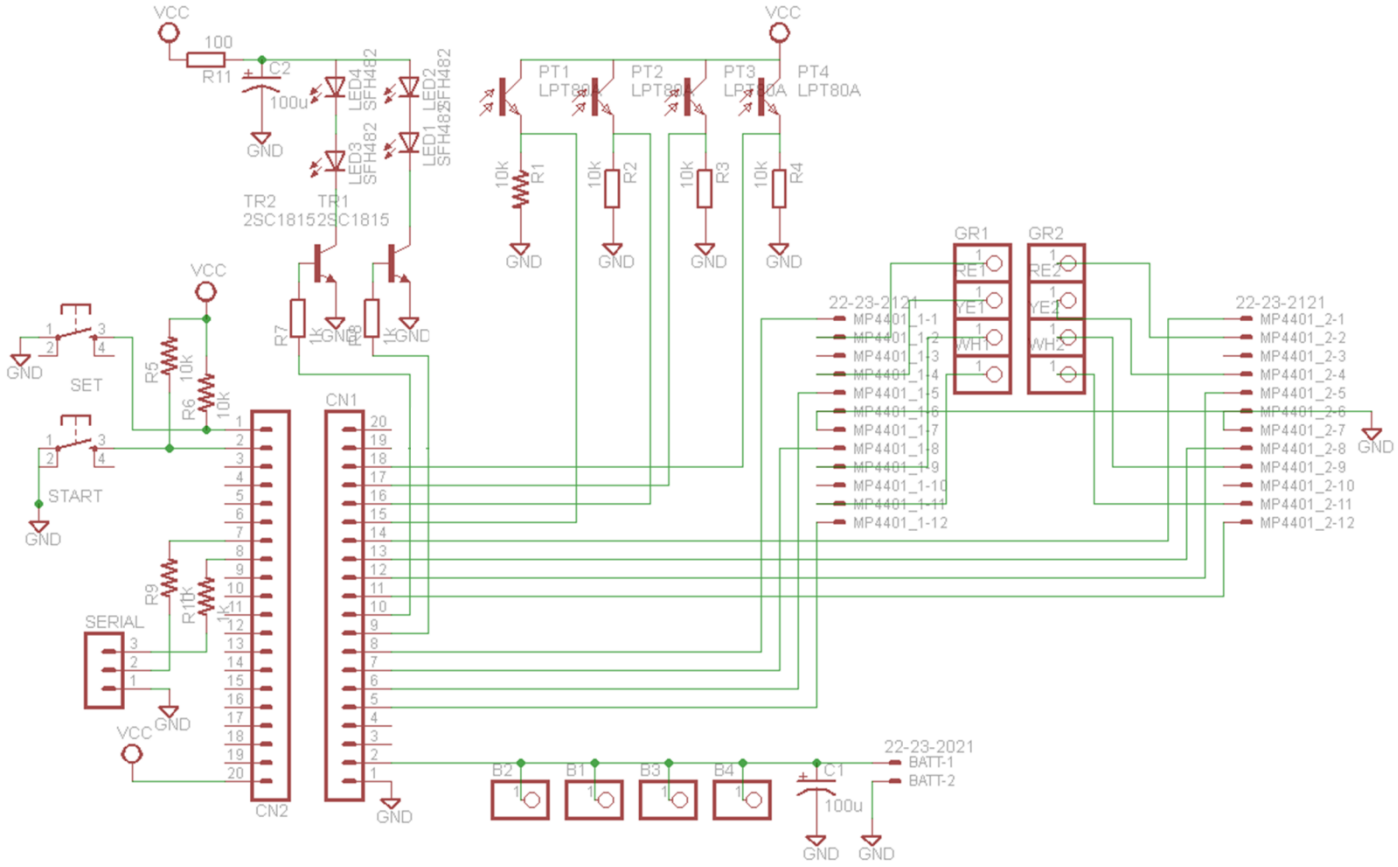

図 2　回路図

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協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/mbedMicromouse_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： 1 区画前進
モード 2： 90 度右回転
モード 3： 90 度左回転
モード 4：180 度右回転
モード 5：U ターン走行
モード 6：拡張左手法による迷路探索
モード 7：最短経路走行

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 関数も 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　タスクと 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

左右モータの加減速を行い左右壁との距離の調整や，超信地旋回させるために左右モータを逆回転させなければならないため，左右のモータは別々のタスクとして制御している．

2）センサの読取り

マイクロマウスは前壁の検出に前方に 2 個，左右壁の検出にセンサを 1 個ずつ左右に搭載しています．走行時は壁との距離や有無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，迷路探索プログラム，最短経路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

```
//--------------------------------------------------------------------------
// 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) {
        if (timR>0) timR--;                          //  timR の値をカウントダウン
        if (timR==0) {
            if (fR==0) timR=LSPD;                   // 速度を遅くする
            else timR=HSPD;                         // 速度を速くする
            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 || modeL==0) {                 // フリー
            patR=4;
            patL=4;
        }
        motorR= RMOTOR[patR];                        // ステッピングモータの励磁出力
```

```
        Thread::wait(5);                    // スレッドを停止
    }
}
```

スレッドは main 関数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 関数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};
```

マイクロマウス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．

モータの速度は以下の高速度と低速度が定義されています．

```
#define LSPD     3          // 低速度
#define HSPD     2          // 高速度
```

速度はフラグ fR の値によって変化します．fR が 0 のときは timR に低速度 LSPD が代入されます．モータの制御は timR をカウントダウンしていき timR が 0 になったタイミングで行われます．timR に LSPD が代入されると，高速度の場合に比べてモータ制御をおこなう頻度が減るためモータの速度は遅くなります．

5.3 RTOS タイマーによるセンサ読取り

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 関数の中で前壁の検出は右前と左前に搭載されたセンサで行います．右前センサの読取りを以下のリスト 3 に示します．

リスト 3　右前センサの読取り

```
ptFRB=ptFR;                    // バックグラウンドの値を取得
ledFout=1;                     // LED-ON
wait_us(20);                   // 20μs 待つ
sensFR=(ptFR-ptFRB)*20;        // 前壁センサの値
ledFout=0;                     // LED-OFF
```

左前センサや左右壁を検出する 2 個のセンサも同様の方法で読取りを行います．

その後，取得したセンサの値を使ってモータ加減速のためのフラグのセットをします．

リスト 4　モータ加減速フラグのセット

```
if (fS==1) {
        fR=fL=1;                                    // 左右モータの速度を高速度に設定する
        if(sensR>DISRMAX) {                         // 右壁を検出している
            if ((sensR-DISR)>0.5) fL=0;             // 右壁に近寄っている場合左モータの速度を低速度に設定
            if ((sensR-DISR)<-0.5) fR=0;            // 左壁に近寄っている場合右モータの速度を低速度に設定
        } else if(sensL>DISLMAX) {                  // 左壁だけ検出している
            if ((sensL-DISL)>0.5) fR=0;             // 左壁に近寄っている場合右モータの速度を低速度に設定
            if ((sensL-DISL)<-0.5) fL=0;            // 右壁に近寄っている場合左モータの速度を低速度に設定
        }
    } else {
        fR=fL=0;                                    // fS=0 のとき左右モータの速度を低速度に設定する
    }
```

センサの値に応じて速度制御を行う際に，左右壁を検知している，または右壁を検出している場合は図 3 のように，右壁との距離に応じてモータの速度を変化させます．左壁のみ検出している場合は左壁との距離に応じてモータの速度を変化させます．

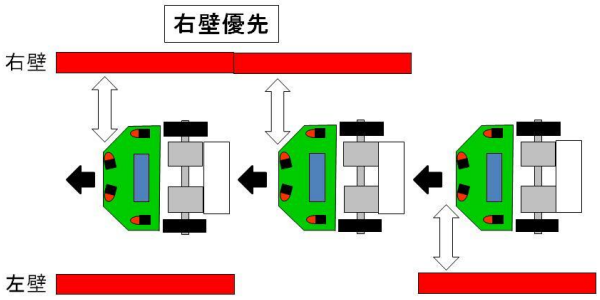

図 3　速度制御

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 タイマーの定義の他に，迷路内のマップの初期化や 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);
        while (setSw==0);
        wait(0.1);
        pmode++;
        if (pmode>7) 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();
             break;                          // センサチェック
    case  1: thread1.signal_set(0x1);        // 1 区画前進
             thread2.signal_set(0x1);
             run_step();
              break;
              ・・・・・中略・・・・・
    case  6: thread1.signal_set(0x1);       // 拡張左手法による迷路探索
             thread2.signal_set(0x1);
             run_Hidarite();
             break;
    case  7: run_saitan();
             break;                         // 最短経路走行
        }
```

テストプログラムにはシリアル通信を利用したセンサのチェックや 1 区画前進，90 度右回転，90 度左回転，180 度右回転，U ターン走行がある.

case 6 では拡張左手法による迷路探索を行う．迷路探索の方法には他に左手法がある．左手法とは，左側の壁に沿って迷路を探索する手法です[4]．拡張左手法とは，逆戻りする場合を除いて探索途中に仮想的な壁を立て，すでに通ったことのある区画への再入を禁止し，かつ左手法と同じ優先順位に従って探索する方法です.

ゴール位置は（７，７），（７，８），（８，７），（８，８）の 4 箇所のいずれかに設定されています．ゴール位置を変える場合は，run_saitan 関数の中に書かれているゴール座標を書き換えてください.

case 7 では最短経路走行を行います．迷路探索で得た壁の有無の情報をもとに最短経路を求めてスタートからゴールまで走行します.

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　高速マイクロマウスの作り方”，東京電機大学出版局，(2002)

[4] 左手法 - マイクロマウス委員会関西支部，http://mmk.rulez.jp/?page_id=408

サンプルプログラム

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

// run parameters
#define LSPD       3                          // timer count for low speed
#define HSPD       2                          // timer count for high speed
#define STEP0      20                         // munber of step for slow start
#define STEP1      500                        // number of step for 1 maze area
#define R90        207                        // number of step for 90 degree right turn
#define L90        207                        // number of step for 90 degree left turn
#define R180       406                        // number of step for 180 degree u-turn
#define DISFR      4.8                        // front right sensor value in normal micromouse position
#define DISFL      4.8                        // front left sensor value in normal micromouse position
#define DISR       1.8                        // right sensor value in normal micromouse position
#define DISL       1.8                        // left sensor value in normal micromouse position
#define DISFMAX 0.3                           // threshold of sensor value for front wall detection
#define DISRMAX 0.6                           // threshold of sensor value for right    wall detection
#define DISLMAX 0.5                           // threshold of sensor value for left wall detection

// 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 unsigned char DtoR[]= {0,2,4,0,8,0,0,0,1};       // table indicating to the right direction
const unsigned char DtoL[]= {0,8,1,0,2,0,0,0,4};       // table indicating to the left direction

unsigned char pmode=0;                                        //    program mode

// Variables. It is necessary to define as a Volatile    when the variable used in interrupt.
volatile float ptFRB, ptFLB, ptRB, ptLB;                // sensor values during turn-off the LED
volatile float sensFR, sensFL, sensR, sensL;          // sensor values

volatile unsigned char modeR=0, modeL=0;                // run forward both motor
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 float timR=0, timL=0;                          // waiting timer for motors
volatile unsigned char timS;                            // waiting timer for sensors
volatile unsigned char fS=0;                            // flag for control of distanse from R,L walls
volatile unsigned char fR=0, fL=0;                      // flag of R, L motors, 0: low speed, 1:hight speed

union {                                                 // struct and union define for access map
    unsigned char all;                                  // map access by 1 byte
    struct {
        unsigned char n:1;                              // 1 bit for north wall i0:no wall, 1:exist wallj
        unsigned char e:1;                              // 1 bit for east wall i0:no wall, 1:exist wallj
        unsigned char s:1;                              // 1 bit for south wall i0:no wall, 1:exist wallj
        unsigned char w:1;                              // 1 bit for west wall i0:no wall, 1:exist wallj
        unsigned char d:4;                              // 4bit for history
    };
} mmap[16][16];

Serial pc(USBTX, USBRX);

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 ptFR(p15);                             // front right sensor, analog input
AnalogIn ptFL(p16);                             // front left sensor, analog input
AnalogIn ptR(p17);                              // right sensor, analog input
AnalogIn ptL(p18);                              // left sensor, analog input
AnalogIn gyro(p19);                             // for Gyro, analog input, reserved
DigitalIn setSw(p21);                           // set-switch, digital input
DigitalIn startSw(p22);                         // start-switch, digital input
DigitalOut ledFout(p9);                         // LED output signal for front wall detection
DigitalOut ledRLout(p10);                       // LED output signal for side wall detection

//------------------------------------------------------------------------
//   LED display
//------------------------------------------------------------------------
void dispLED(unsigned char n)
{
    leds=n;
```

```
}

//-----------------------------------------------------------------------
// RTOS Timer
//-----------------------------------------------------------------------
void Sensor(void const *argument)
{
    // read sensors
    // 1st-step:measure background during LED-off, 2nd-step: measure reflecting light during LED-on. sensor value is differnce of both.
    if (timS<20) timS++; else timS=0;              // set counter timS
    if (timS==0){
    ptFRB=ptFR;                                    // measure all background values
    ledFout=1;                                     // LED-ON
    wait_us(20);                                   // delay
    sensFR=(ptFR-ptFRB)*20;                        // front right sensor values
    ledFout=0;                                     // LED-OFF
    }
    if (timS==5){
    ptFLB=ptFL;                                    // measure all background values
    ledFout=1;                                     // LED-ON
    wait_us(20);                                   // delay
    sensFL=(ptFL-ptFLB)*20;                        // front left sensor values
    ledFout=0;                                     // LED-OFF
    }
    if (timS==10){
    ptRB=ptR;                                      // measure all background values
    ledRLout=1;                                    // LED-ON
    wait_us(20);                                   // delay
    sensR=(ptR-ptRB)*20;                           // right sensor values
    ledRLout=0;                                    // LED-OFF
    }
    if (timS==15){
    ptLB=ptL;                                      // measure all background values
    ledRLout=1;                                    // LED-ON
    wait_us(20);                                   // delay
    sensL=(ptL-ptLB)*20;                           // left sensor values
    ledRLout=0;                                    // LED-OFF
    }
    // set motor control flag by distance of both side walls
    // use only right wall when right wall detected,    use left wall when detected left wall only.
    if (fS==1) {                                   // do the following process, when flag fS=1
```

```
        fR=fL=1;                                    // set high speed for both motor
        if(sensR>DISRMAX) {                         // when right wall exists,
            if ((sensR-DISR)>0.5) fL=0;      // set low speed for left moter, when close to the right wall
            if ((sensR-DISR)<-0.5) fR=0;     // set low speed for right moter, when close to the left wall
        } else if(sensL>DISLMAX) {                  // when existing left wall only,
            if ((sensL-DISL)>0.5) fR=0;      // set low speed for right moter, when close to the left wall
            if ((sensL-DISL)<-0.5) fL=0;     // set low speed for left moter, when close to the right wall
        }
    } else {
        fR=fL=0;       // when fS=0, set low speed for both motor
    }
}

//-------------------------------------------------------------------------
// 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) {
        if (timR>0) timR--;                                 //count down timR when timR=0 do next process
        if (timR==0) {
            if (fR==0) timR=LSPD;
            else timR=HSPD;
            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 || modeL==0) {
            patR=4;       // motor free when mode=0
            patL=4;
        }
        motorR= RMOTOR[patR];                               // pattern output to right motor
        Thread::wait(5);
```

```
    }
}

//-----------------------------------------------------------------------
// 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) {
        if (timL>0) timL--;                          //count down timLCwhen timL=0 do next process
        if (timL==0) {
            if (fL==0) timL=LSPD;
            else timL=HSPD;
            if (modeL==1) {
                if (patL < 3) patL++;
                else patL = 0;
            }
            if (modeL==2) {
                if (patL > 0) patL--;
                else patL = 3;
            }
            cntL++;                                  // count up left moter step
        }
        if (modeR==0 || modeL==0) {
            patR=4;      // motor free when mode=0
            patL=4;
        }
        motorL= LMOTOR[patL];                        // pattern output to left motor
        Thread::wait(5);
    }
}

//-----------------------------------------------------------------------
// check sensor value using serial port
//-----------------------------------------------------------------------
void check_sens()
{
    while (1) {
        pc.printf("¥f");
```

```
            pc.printf("Sensor FR:%f ¥n",sensFR);
            pc.printf("Sensor FL:%f ¥n",sensFL);
            pc.printf("Sensor  R:%f ¥n",sensR);
            pc.printf("Sensor  L:%f ¥n",sensL);
            wait (0.5);
        }
}

//---------------------------------------------------------------------
// break motors
//---------------------------------------------------------------------
void run_break()
{
    modeR=0;
    modeL=0;                        // mode 0 means break the motor
}

//---------------------------------------------------------------------
// adjustment by front wall
//---------------------------------------------------------------------
void adjust()
{
    fS=0;                                       // set low speed
    while(abs((sensFR-DISFR)-(sensFL-DISFL))>0.4) {     // do adjustment when difference of sensor value larger than threshold
        if ((sensFR-DISFR)>(sensFL-DISFL)) {
            modeR=2;    // turn right
            modeL=1;
        } else {
            modeR=1;    // turn left
            modeL=2;
        }
    }
    run_break();
}

//---------------------------------------------------------------------
// slow start of the motors
//---------------------------------------------------------------------
void slow_start()
{
    fS=0;                           // set low speed
```

```
    modeR=modeL=1;                  // set mode for run forward
    cntR=0;
    stepR=STEP0;          // run 20 step at low speed
    while (cntR<stepR);
}

//---------------------------------------------------------------------
//      run forwad of 1 maze area
//---------------------------------------------------------------------
void run_step()
{
    slow_start();
    fS=1;                                   // change to high speed
    cntR=0;
    stepR=STEP1-STEP0;
    while (cntR<stepR);
    run_break();
}

//---------------------------------------------------------------------
// 90 degree turn right
//---------------------------------------------------------------------
void run_R90()
{
    fS=0;                           // set low speed
    cntR=0;
    stepR=R90;        // set motor step for turn 90 degree
    modeR=2;
    modeL=1;          // right motor: reverse, left motor: forward
    while (cntR<stepR);
    run_break();
}

//---------------------------------------------------------------------
// 90 degree turn left
//---------------------------------------------------------------------
void run_L90()
{
    fS=0;                           // set low speed
    cntL=0;
    stepL=L90;          // set motor step for turn 90 degree
    modeR=1;
```

```
	modeL=2;		// right motor: forward, left motor: reverse
	while (cntL<stepL);
	modeR=0;
	modeL=0;
	run_break();
}

//---------------------------------------------------------------------
// u-turn
//---------------------------------------------------------------------
void run_R180()
{
	fS=0;				// set low speed
	cntR=0;
	stepR=R180;		// set motor step for turn 180 degree
	modeR=2;
	modeL=1;		// right motor: reverse, left motor: forward
	while (cntR<stepR);
	run_break();
}

//---------------------------------------------------------------------
// run forward and u-turn when front wall detected
//---------------------------------------------------------------------
void run_Turn(unsigned char n)
{
	while (1) {
		slow_start();
		fS=1;					// set high speed
		while (sensFR<DISFR);		// run forward to normal distanse from front wall
		adjust();				// adjestment by front wall
		if (n==0) run_R180();
		else run_R90();	// u-turn or turn right by the value of n
	}
}

//---------------------------------------------------------------------
// left hand methodiusing direction historyj
//	priority is left, front and right. if all the wall exists, then u-turn.
//---------------------------------------------------------------------
void run_Hidarite()
{
```

```
unsigned char wF, wR, wL;                    // flag for front right left walls
unsigned char wS;                            // flag for sensing the walls
unsigned char mapF, mapR, mapL;              // variable to read the history
unsigned char mx,my;                         // x and y axis of mouse, start posisiton is 0,0
unsigned char md;                            // direction of the mouseCnorth:1Ceast:2, south:4, west:8

mapF=0;
mapR=0;
mapL=0;                      // initiallize
mx=0;
my=0;
md=1;                            // initiallize
wF=0;
wR=1;
wL=1;                            // initiallize
mmap[0][0].d=1;                          // inital direction is north (1)

while (startSw==1) {                     // repeat durning no sw input detection (push start-sw to exit )

    // reade history    imapF,mapR,mapL are the history of front, right, left area
    // no access to the out of range
    switch (md) {
        case 1:
            if (my<15) mapF=mmap[my+1][mx].d; // when mouse direction is north,
            if (mx<15) mapR=mmap[my][mx+1].d;
            if (mx>0)    mapL=mmap[my][mx-1].d;
            break;
        case 2:
            if (mx<15) mapF=mmap[my][mx+1].d; // when mouse direction is east,
            if (my>0)    mapR=mmap[my-1][mx].d;
            if (my<15) mapL=mmap[my+1][mx].d;
            break;
        case 4:
            if (my>0)    mapF=mmap[my-1][mx].d; // when mouse direction is south,
            if (mx>0)    mapR=mmap[my][mx-1].d;
            if (mx<15) mapL=mmap[my][mx+1].d;
            break;
        case 8:
            if (mx>0)    mapF=mmap[my][mx-1].d; // when mouse direction is west,
            if (my<15) mapR=mmap[my+1][mx].d;
            if (my>0)    mapL=mmap[my-1][mx].d;
            break;
```

```
        }


        // decision by left hand rule
        if (wL ==0 && (mapL==0 || mapL==DtoL[md])) {      // left turn when no left wall and no history or available history
            run_L90();
            md=DtoL[md];
        } else if (wF==0 && (mapF==0 || mapF==md)) {}     // go forward when no front wall and no history or available history (pass the turn process)
        else if (wR==0 && (mapR==0 || mapR==DtoR[md])) { // right turn when no left wall and no history or available history
            run_R90();
            md=DtoR[md];
        } else {
            run_R180();      //u-turn
            md=DtoR[md];
            md=DtoR[md];
        }

        // go forward and detect walls
        wS=0;
        wF=0;
        wR=0;
        wL=0;                                   // reset of wall flags
        slow_start();                                           // slow start
        stepR=STEP1-STEP0;
        fS=1;                                                   // change high speed
        while (cntR<stepR) {                                    // go forward
            if (cntR > (STEP1*2/3) && wS==0) {                  // wall detection when the mouse run 2/3 step of area
                wS=1;                                           // set flag to detect wall at once.
                if (sensR > DISRMAX) wR=1;
                else wR=0;          // detection of right wall
                if (sensL > DISLMAX) wL=1;
                else wL=0;          // detection of left wall
                if ((sensFR>DISFMAX || sensFL>DISFMAX)) {
                    wF=1;       // detection of front wall. exit from loop when front wall detected.
                    break;
                }
            }
        }
```

```
// go forwrd to adjust distanse of front wall, when exit the above loop by front wall detection.
if (wF==1) {
    while (sensFR<DISFR);                    // go forward to have normal distance to front wall
    adjust();                                // adjustment by front wall
}

// write map and history ( opposit direction of out of the area )
// record map after update mouse axis
switch (md) {
    case 1:
        mmap[my][mx].d=4;
        my++;
        mmap[my][mx].n=wF;
        mmap[my][mx].e=wR;
        mmap[my][mx].w=wL;
        break;
    case 2:
        mmap[my][mx].d=8;
        mx++;
        mmap[my][mx].e=wF;
        mmap[my][mx].s=wR;
        mmap[my][mx].n=wL;
        break;
    case 4:
        mmap[my][mx].d=1;
        my--;
        mmap[my][mx].s=wF;
        mmap[my][mx].w=wR;
        mmap[my][mx].e=wL;
        break;
    case 8:
        mmap[my][mx].d=2;
        mx--;
        mmap[my][mx].w=wF;
        mmap[my][mx].n=wR;
        mmap[my][mx].s=wL;
        break;
}
if (mx==0 && my==0) {
    run_break();      // finish search run when mouse return start position
    break;
```

```
        }
    }
}

//---------------------------------------------------------------------
// fast run, find minimum route to goal and run fast
//---------------------------------------------------------------------
void run_saitan()
{
    unsigned char i,j,k,m;
    unsigned char smap[16][16];              // map for calculate minimum route
    unsigned char run[256];                  // array for run pattern
    unsigned char md;                         // direction of mouse 1:north, 2:east, 4:south, 8:west

    // clear map and set walls for no histry area.
    for(i=0; i<16; i++) {
        for(j=0; j<16; j++) {
            smap[i][j]=0;
            if (mmap[i][j].d==0) {
                mmap[i][j].n=1;
                if (i<15) mmap[i+1][j].s=1;
                mmap[i][j].e=1;
                if (j<15) mmap[i][j+1].w=1;
                mmap[i][j].s=1;
                if (i>0)    mmap[i-1][j].n=1;
                mmap[i][j].w=1;
                if (j>0)    mmap[i][j-1].e=1;
            }
        }
    }

    // write steps to smap from goal position
    // goal position set to m=1, find same value of m in smap and put m+1 to no wall direction, increment of
m,
    // go out roop when reach to stat position.
    smap[7][7]=1;
    smap[7][8]=1;
    smap[8][7]=1;
    smap[8][8]=1; // goal position set to 1

    m=1;                                                    // set m=1
    for(k=0; k<255; k++) {                                  // repeat maximun 255 times
```

```
        for(i=0; i<16; i++) {
            for(j=0; j<16; j++) {                                   // scan all areas
                if (smap[i][j]==m) {
                    if (mmap[i][j].n==0 && i<15 && smap[i+1][j]==0) smap[i+1][j]=m+1;
                    if (mmap[i][j].e==0 && j<15 && smap[i][j+1]==0) smap[i][j+1]=m+1;
                    if (mmap[i][j].s==0 && i>0   && smap[i-1][j]==0) smap[i-1][j]=m+1;
                    if (mmap[i][j].w==0 && j>0   && smap[i][j-1]==0) smap[i][j-1]=m+1;
                }
            }
        }
        m++;                                            // increment of m
        if (smap[0][0]!=0) break;                   // go out of loop
    }

    // make run pattern to run[k]   array
    // k:number of run pattern, 1:go forward, 2:turn right, 3:turn left
    m=smap[0][0]-1;                             // set m to start position
    i=0;
    j=0;
    k=0;
    md=1;
    while (m>0) {                               // loop while reach to goal position
        switch(md) {
            case 1:
                if (mmap[i][j].n==0 && smap[i+1][j]==m && i<15) {
                    run[k]=1;
                    i++;
                    m--;
                    break;
                }
                if (mmap[i][j].e==0 && smap[i][j+1]==m && j<15) {
                    run[k]=2;
                    md=DtoR[md];
                    break;
                }
                if (mmap[i][j].w==0 && smap[i][j-1]==m && j>0 ) {
                    run[k]=3;
                    md=DtoL[md];
                    break;
                }
            case 2:
                if (mmap[i][j].e==0 && smap[i][j+1]==m && j<15) {
```

```
                run[k]=1;
                j++;
                m--;
                break;
            }
            if (mmap[i][j].s==0 && smap[i-1][j]==m && i>0 ) {
                run[k]=2;
                md=DtoR[md];
                break;
            }
            if (mmap[i][j].n==0 && smap[i+1][j]==m && i<15) {
                run[k]=3;
                md=DtoL[md];
                break;
            }
        case 4:
            if (mmap[i][j].s==0 && smap[i-1][j]==m && i>0 ) {
                run[k]=1;
                i--;
                m--;
                break;
            }
            if (mmap[i][j].w==0 && smap[i][j-1]==m && j>0 ) {
                run[k]=2;
                md=DtoR[md];
                break;
            }
            if (mmap[i][j].e==0 && smap[i][j+1]==m && j<15) {
                run[k]=3;
                md=DtoL[md];
                break;
            }
        case 8:
            if (mmap[i][j].w==0 && smap[i][j-1]==m && j>0 ) {
                run[k]=1;
                j--;
                m--;
                break;
            }
            if (mmap[i][j].n==0 && smap[i+1][j]==m && i<15) {
                run[k]=2;
                md=DtoR[md];
```

```
                    break;
                }
                if (mmap[i][j].s==0 && smap[i-1][j]==m && i>0 ) {
                    run[k]=3;
                    md=DtoL[md];
                    break;
                }
            }
            k++;
        }

        // run minimun route
        i=0;
        while (i<k) {
            if (run[i]==1) {
                run_step();
                i++;
            }
            if (run[i]==2) {
                run_R90();
                i++;
            }
            if (run[i]==3) {
                run_L90();
                i++;
            }
        }
}

//---------------------------------------------------------------------
//      main
//---------------------------------------------------------------------
int main()
{
        int i,j;

        // initialize map
        for (i=0; i<16; i++) for (j=0; j<16; j++) mmap[i][j].all=0; // clear map
        for (i=0; i<16; i++) {
            mmap[i][0].w=1;
            mmap[i][15].e=1;            // set east and west wall
            mmap[0][i].s=1;
```

```
        mmap[15][i].n=1;            // set north and south wall
    }
    mmap[0][0].e=1;

    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

    // initialize parameters
    ledFout=ledRLout=0;
    motorR=motorL=0;

    while (1) {
        while (startSw==1) {
            if (setSw==0) {
                wait(0.1);
                while (setSw==0);
                wait(0.1);
                pmode++;
                if (pmode>7) pmode=0;
            }
            leds=pmode;
        }
        leds=0;
        wait(0.5);

        // go selected functions
        switch(pmode) {
            case  0:
                check_sens();
                break;      // check sensors
            case  1:
                thread1.signal_set(0x1);    // run 1 area step
                thread2.signal_set(0x1);
                run_step();
                break;
            case  2:
                thread1.signal_set(0x1);    // 90 deg. turn right
                thread2.signal_set(0x1);
                run_R90();
```

```
                break;
            case  3:
                thread1.signal_set(0x1);    // 90 deg. turn left
                thread2.signal_set(0x1);
                run_L90();
                break;
            case  4:
                thread1.signal_set(0x1);    // u-turn
                thread2.signal_set(0x1);
                run_R180();
                break;
            case  5:
                thread1.signal_set(0x1);    // go forward and u-turn, 0: u-turn, else: 90 deg.turn
                thread2.signal_set(0x1);
                run_Turn(0);
                break;
            case  6:
                thread1.signal_set(0x1);    // extended left hand rule
                thread2.signal_set(0x1);
                run_Hidarite();
                break;
            case  7:
                run_saitan();
                break;       // fast run for minimum route
        }
    }
}
```