Mbed マイクロマウスの解説

2013.7　葉山清輝（熊本高専）

１．概略

ニューテクノロジー振興財団の主催するマイクロマウス大会・マイクロマウスクラッシック競技に合わせて，プロトタイピングツールの mbed（http://mbed.org/）を利用して設計したマイクロマウスです．２相ユニポーラステッピングモータ２個を走行用のモータとして利用します．迷路の壁を検知するフォトセンサを搭載しており，壁の有無や距離を読み取ってモータを制御して迷路内を走行します．mbed の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 のサイトにアクセスすれば，ブラウザ上で開発環境を利用でき，mbed 基板と PC を USB で接続して bin ファイルをダウンロードし，リセットボタンを押すことで実できます．

図１は低価格で製作するために，秋月電子通商で販売されている安価なステッピングモータと取扱いが容易で安価な単３電池による構成例です（基板設計前にユニバーサル基板で試作したもの）．図２は大会参加を視野に入れて高トルクなステッピングモータと小型軽量なリチウムポリマーバッテリーを使って作ったもので高速走行が可能です．走行時の安定性を増すためにジャイロセンサも利用できます．

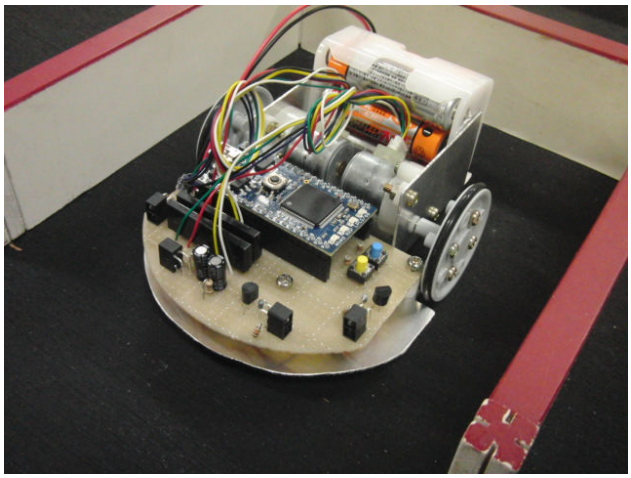

図１　安価な部品構成（ユニバーサル基板による試作）

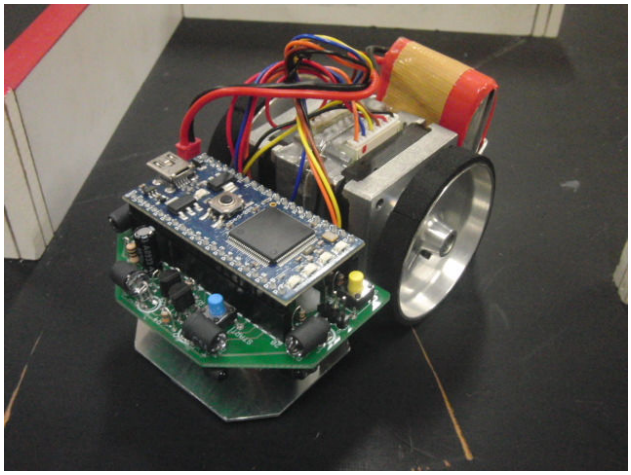

図２　高速走行可能な部品構成

２．基板と回路の説明

図３は回路の設計画面，図４は回路図です．迷路の壁を検知するために前に２個，左右に１個ずつで計４個のフォトトランジスタを搭載しています．前壁を検知する際には CN1-p9 に接続されたトランジスタを動作させて LED1 と LED２を点灯させて 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，左右の壁を検知する際には，CN1-p10 に接続されたトランジスタにより LED3 と LED4 を点灯させて，PT3 と PT4 の出力を読み取ります．

ステッピングモータの駆動回路は安価に構成するためにパワーFET モジュール（MP4401）にモータを直結して回すようにしました．モータの逆起電力に対する保護回路や，モータの電流制限回路は入っていませんので，場合によっては mbed ボードにダメージを与える可能性がありますが，試作品では今のところ問題なく動作しています．

そのほか，基板にはモード選択用のセットスイッチ（CN2-1），スタートスイッチ（CN2-2）を搭載しています．

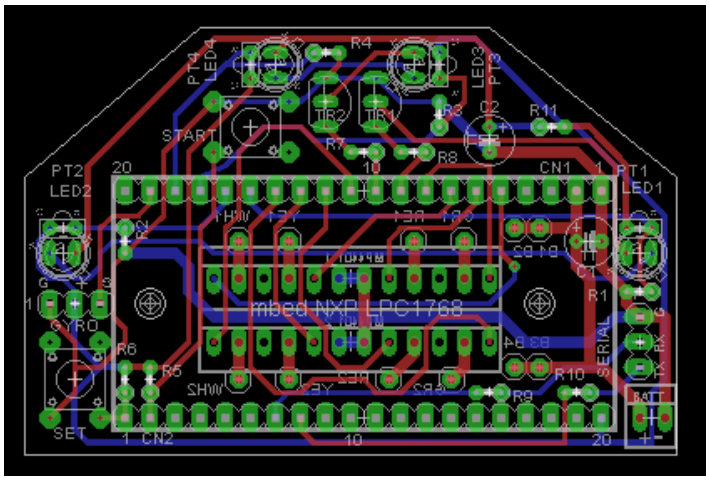

図３　回路基板の設計画面

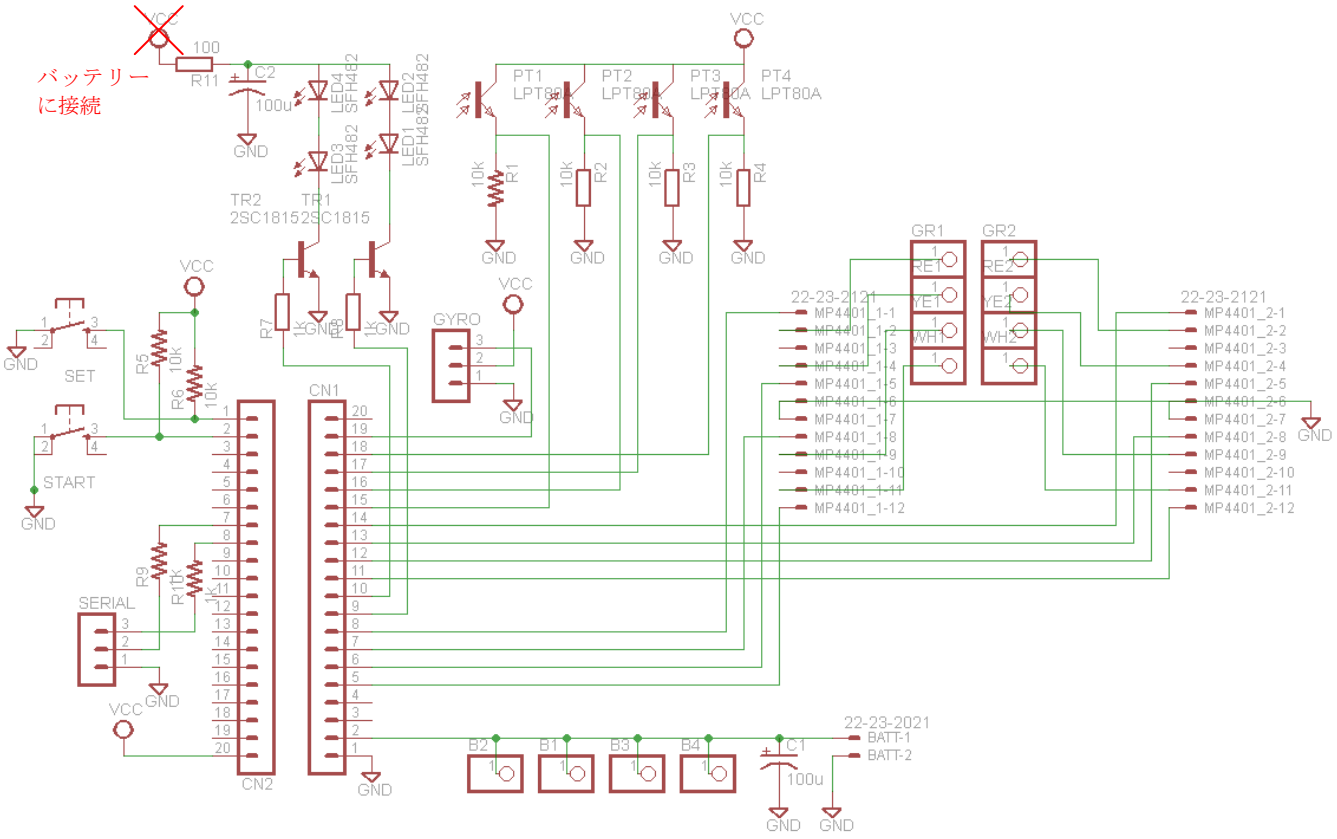

図４　回路図

３．基板の製作と組立

部品表は表１に示す通りです．マイクロマウス基板に部品をはんだ付けして組み立てます．大変申し訳ありませんが回路は１箇所ミスがあります．センサ用の LED 点灯回路で Vcc（3.3V）を接続するパターンとなっていますが電圧が足りないので，バッテリーの電圧を加えなければなりません．図６の写真の通り基板の裏面で 100Ω の R11 を接続してください．

斜め走行も可能なように基板寸法を小さく作りましたので，図７に示すようにパワーMOSFET モジュールは基板の裏面に配置するようにしています．必然的に基板を少し持ち上げないといけないので，基板の表面に LED，裏面にフォトトランジスタを接続すると，両者の干渉が少なくなります．その場合

でも，フォトトランジスタに横からの直接光が入らないように側面にカバーをしてください．LED とフォトトランジスタは LBR-127HLD 等のフォトリフレクタでも代用出来ますが，感度が弱くなってしまいますので高速走行には向きません．フォトリフレクタを取り付ける際には９０度足を曲げて，センサを全面と左右に向けることを忘れないようにしてください．

表１　部品表

	品名	基板上の記号，備考
	mbed マイコンボード　NXP　LPC1768	mbed　NXP　LPC1768
1	教育用マイクロマウス基板	
2	ピンソケット（20P×2個）	
3	ピンヘッダ（3P×2個）	（ジャイロとシリアル通信を使わなければ不要）
4	カーボン抵抗（炭素皮膜抵抗）1/6W　10kΩ	R1,R2,R3,R4,R5,R6
5	カーボン抵抗（炭素皮膜抵抗）1/6W　1kΩ	R7,R8（,R9,R10 はシリアル通信しなければ不要）
6	カーボン抵抗（炭素皮膜抵抗）1/4W 100Ω	R11
7	電解コンデンサー　100μF　25V	C1,C2
8	トランジスタ　2SC1815	TR1,TR2
9	超高輝度赤LED　5mm　OSHR5111A－TU	LED1,LED2,LED3,LED4
10	照度センサ（フォトトランジスタ） （フォトリフレクタ（LBR-127HLD 等）を使用可）	PT1,PT2,PT3,PT4
11	パワーMOS－FETモジュール　MP4401	MP4401-1,MP4401-2
12	タクトスイッチ	SET，START
13	24対1ユニポーラステッピングモータ	MP4401-1,MP4401-2
14	自在タイヤφ40　2個入り	SET，START
15	電池ボックス（単3X6・白・スナップ）	G_VCC_TX_RX_DTR
16	バッテリースナップ（電池スナップ）プラスチック製	
17	スペーサ	長さ 20mm 程度
18	アルミ板材　A5 版　A5052	適宜，機体を自作してください
19	スベリ材用スポンジ	高さ調整用
20	スベリ材	カグスベールなど
21	（ジャイロセンサ，必要な場合）	GYRO

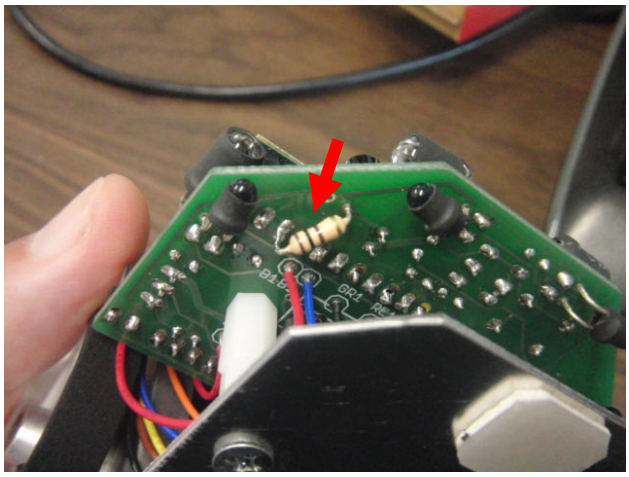

図６　基板の修正点

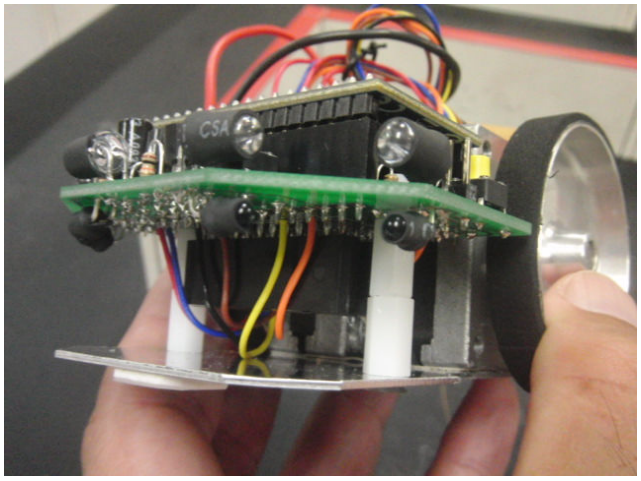

図７　センサと MP4401 の取付け

トランジスタ，パワーMOSFET モジュール，電解コンデンサなど極性のある部品については回路図と基板のパターンの対応を考えながら方向を間違えないように取り付けてください．

mbed 基板の下に収まる電解コンデンサは，足を折り曲げて横に倒してください．バッテリースナップ

は動かしていると断線しやすいので配線が終わった後に接着剤等で固定するとよいでしょう．

ステッピングモータの配線は，秋月電子通商から入手できる SPG20-1362 に付属の配線の色を元に表示しています．基板上の GR1 が右モータの緑色の配線です．そのほか，RE1 が赤色，YE1 が黄色，WH1 が白色の配線となります．GR2，RE2,YE2,WH2 には左モータの配線を接続してください．左右モータの黒と青の配線は電源に接続しますのでバッテリースナップ横の B1,B2,B3,B4 にまとめてハンダ付けしてください．

他社のモータを利用する場合は，GR，RE,YE,WH の順に励磁するとモータが回転するように取り付けてください（回転方向については，ソフトウェアで変えられます）．

低速で走行する場合にはジャイロセンサは不要ですが，高速走行を目指すなら適当なものを探して取り付けると良いでしょう．基板の外側から GND,Vcc，信号（S）となっています．

シリアル通信用の端子を設けましたが，次項で述べるように USB を経由したシリアル通信ができますので，この端子と R9 と R10 の接続は不要です．

使用するモーターとタイヤ・ホイール，バッテリーに合わせて機体を制作し，基板を取付ければ完成です．

４．サンプルプログラムと操作方法について

mbed の使用方法は，書籍や Web サイトを参照してください．例えば，

mbed のホームページ　http://mbed.org/

じぇーけーそふとのこーなー　http://jksoft.cocolog-nifty.com/blog/mbed/

超お手軽マイコン mbed 入門，http://shop.cqpub.co.jp/book_guide/detail/17521/

などが参考になります．

サンプルプログラムは，　mbed のサイト内に公開している以下のリンクから入手出来ます．

http://mbed.org/users/hayama/code/mbedMicromouse/

zip 形式でダウンロードして，自分の mbed の workspace にインポートすると良いです．コンパイルして mbed に書き込んでください．サンプルプログラムはリスト１に示す通りです．

操作方法について説明します．

１）リセットボタンを押してプログラムを動作させます．

２）セットボタンで動作モードを選択します．動作モードは mbed ボード上の４個の LED に２進数で表示されます．

３）スタートボタンで選択したモードで動作させます．

モード０：　センサチェック，シリアル通信（USB 経由）でセンサの読みを PC に表示させます．

モード１：　１区画前進

モード２：　右９０度ターン

モード３：　左９０度ターン

モード４：　右１８０度ターン

モード５：　U ターン走行（直進－前壁を見つけたら U ターンを繰り返す）

モード６：　左手法による迷路探索走行

モード７：　最短経路走行

となっています．モード０のセンサチェックでは，USB 接続を利用したシリアル通信を行います．以下の URL を参考にドライバーをインストールしてください．

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

サンプルプログラムではセンサの基準値は以下の通りとなっていますので，迷路内にマイクロマウスを置いて読み取ったセンサの値をもとに，迷路走行に必要なしきい値を決めてプログラム中の基準値の修正を行わなければなりません．

```
#define DISFR     16      // マウスを迷路中央においた時の前壁右側のセンサの読み
#define DISFL     16      // マウスを迷路中央においた時の前壁左側のセンサの読み
#define DISR      8       // マウスを迷路中央においた時の右壁センサの読み
#define DISL      8       // マウスを迷路中央においた時の左壁センサの読み
#define DISFMAX 2       // 2/3区画進んだ時に前面のセンサにより前壁の有無を判断するしきい値
#define DISRMAX 2       // 右壁の有無を判断するしきい値（区画左いっぱいにおいた時の右センサの読み）
#define DISLMAX 2       // 左壁の有無を判断するしきい値（区画右いっぱいにおいた時の左センサの読み）
```

マウスは低速と高速の２通りのスピードを切り替えながら走行するように作ってあります．ステッピングモータの回転を進める際の待ち時間を与えていますので，値が大きいほど低速になります．高速でもモータが脱調しない値を選んでください．

```
#define LSPD      5       // 低速の待ち時間
#define HSPD      3       // 高速の待ち時間
```

モード１の１区画前進では，１区画の歩数を調整します．実際に走らせてみて，プログラム中の STEP1 の基準値を調整してください．

```
#define STEP1     665     // 迷路一区画を進ませるモータのステップ数
```

モード２，３，４でそれぞれの角度でターンを行うのに必要なステップ数を以下の基準値を書き換えて調整してください．

```
#define R90       265     // 右９０度回転を行うステップ数
#define L90       265     // 左９０度回転を行うステップ数
#define R180      530     // 右１８０度回転を行うステップ数
```

基準値の修正したらマイクロマウスにプログラムを書き込むと，モード６で探索走行ができる．ただし，サンプルプログラムでは走行誤差に対する補正を入れていませんので，このままでは数区画しか迷路を走行できません．難しい迷路でも完走できるようにプログラムを改良してみてください．

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．小さな迷路で走らせる場合は，run_saitan 関数の中に書かれているゴール座標を書き換えてテストしてください．

ゴールに到達後にスタート位置まで戻ってきたら，マウスの向きをスタート方向に向けなおし，モード７に切替えて再度スタートボタンを押します．最短走行を行いゴール位置で停止するはずです．

大会に出場する場合は，走行時の誤差に対する様々な補正を行い，確実な探索走行ができるようにして，さらに探索走行を賢くして探索時間を短くし，２回目以降は加減速しながら最短・最速の高速走行をさせます．どのようにコースを記録し，高速走行させるかが大会の醍醐味ですのでご自分で作ってみてください．

リスト１　サンプルプログラム

```
//*****************************************************************************
//
//      KNCT-MMEdu for mbed
//      (c) Kiyoteru Hayama(Kumamoto National College of Technology)
//
//*****************************************************************************
#include "mbed.h"

// run parameters
#define LSPD       5                          // timer count for low speed
#define HSPD       3                          // timer count for high speed
#define STEP1      665                        // number of step for 1 maze area
#define R90        265                        // number of step for 90 degree right turn
#define L90        265                        //    number of step for 90 degree left turn
#define R180       530                        //    number of step for 180 degree u-turn
#define DISFR      16                         // front right sensor value in normal micromouse position
#define DISFL      16                         // front left sensor value in normal micromouse position
#define DISR       8                          // right sensor value in normal micromouse position
#define DISL       8                          // left sensor value in normal micromouse position
#define DISFMAX 2                             // threshold of sensor value for front wall detection
#define DISRMAX 2                             // threshold of sensor value for right    wall detection
#define DISLMAX 2                             // threshold of sensor value for left wall detection

// pattern table for stepping motor
const unsigned char RMOTOR[]={0x03, 0x06, 0x0C, 0x09, 0x00};       // magnetization pattern for right motor
const unsigned char LMOTOR[]={0x09, 0x0C, 0x06, 0x03, 0x00};       // magnetization pattern for left 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 unsigned char 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 ・・:no wall, 1:exist wall・・
                 unsigned char e:1;                         // 1 bit for east wall ・・:no wall, 1:exist wall・・
                 unsigned char s:1;                         // 1 bit for south wall ・・:no wall, 1:exist wall・・
                 unsigned char w:1;                         // 1 bit for west wall ・・:no wall, 1:exist wall・・
                 unsigned char d:4;                         // 4bit for history
               };
      } mmap[16][16];

Ticker timer;                       // defince interval timer

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;
}

//------------------------------------------------------------------------
// interrupt by timer
//------------------------------------------------------------------------
void SensAndMotor() {

// motor rotation,   mode = 0: free・・: forward・・: reverse・・: break
// right motor rotation
  if (timR>0) timR--;                           //count down timR・詣hen 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
  }
// left motor rotation
  if (timL>0) timL--;                           //count down timL・詣hen 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; patL=4; }    // motor free when mode=0
    motorR= RMOTOR[patR];           // pattern output to right motor
    motorL= LMOTOR[patL];           // pattern output to left motor

    // 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(100);                           // delay
        sensFR=(ptFR-ptFRB)*20;
        sensFL=(ptFL-ptFLB)*20;
        ledFout=0;                              // LED-OFF
    }
    if (timS==10){
```

```
            ptRB=ptR;
            ptLB=ptL;
            ledRLout=1;
            wait_us(100);
            sensR=(ptR-ptRB)*20;
            sensL=(ptL-ptLB)*20;
            ledRLout=0;
        }

// 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)>4) fL=0;                       // set low speed for left moter, when close to the right wall
            if ((sensR-DISR)<-4) 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)>4) fR=0;                       // similar to the control by right wall.
            if ((sensL-DISL)<-4) fL=0;
        }
        } else { fR=fL=0; }                                 // when fS=0, set low speed for both motor
}

//---------------------------------------------------------------------------
// check sensor value using serial port
//---------------------------------------------------------------------------
void check_sens(){
    while (1){
      // Serial.print(0x0c,BYTE);
      // Serial.print("Sensor FR:"); Serial.println(sensFR);
      // Serial.print("Sensor FL:"); Serial.println(sensFL);
      // Serial.print("Sensor    R:"); Serial.println(sensR);
      // Serial.print("Sensor    L:"); Serial.println(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))>4){    // do adjustment when difference of sensor value larger than
threshold(20)
    if ((sensFR-DISFR)>(sensFL-DISFL)) {
          modeR=2; modeL=1;                         // turn right
       } else {
          modeR=1; modeL=2;                             // turn left
       }
    }
    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=20;                              // 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=STEP1-20;
   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 method・・sing direction history・・
//    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 mouse・North:1・East: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 ・・apF,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(); md=DtoR[md]; md=DtoR[md];}       //u-turn

// go forward and detect walls
      wS=0; wF=0; wR=0; wL=0;                               // reset of wall flags
      slow_start();                                         // slow start
      stepR=STEP1;
      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; break; }    // detection of front wall. exit from loop
when front wall detected.
          }
       }

          // 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(); break; }           // finish search run when mouse return start position
    }
}

//---------------------------------------------------------------------------
// 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;

        timer.attach(&SensAndMotor, 0.001);     // set timer to 1ms for timer interrupt
        // Serial.begin(9600);                          // open serial port for PC connection (read value of sensors)

        while (1) {

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

                while (startSw==1) {
                        if (setSw==0) {
                                wait(0.01);
```

```
                while (setSw==0);
                wait(0.01);
                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: run_step(); break;           // run 1 area step
            case  2: run_R90();   break;          // 90 deg. turn right
            case  3: run_L90();   break;          // 90 deg. turn left
            case  4: run_R180(); break;           // u-turn
            case  5: run_Turn(0);     break;      // go forward and u-turn
            case  6: run_Hidarite(); break;       // left hand rule
            case  7: run_saitan();    break;      // fast run for minimum route
        }
    }
}
```