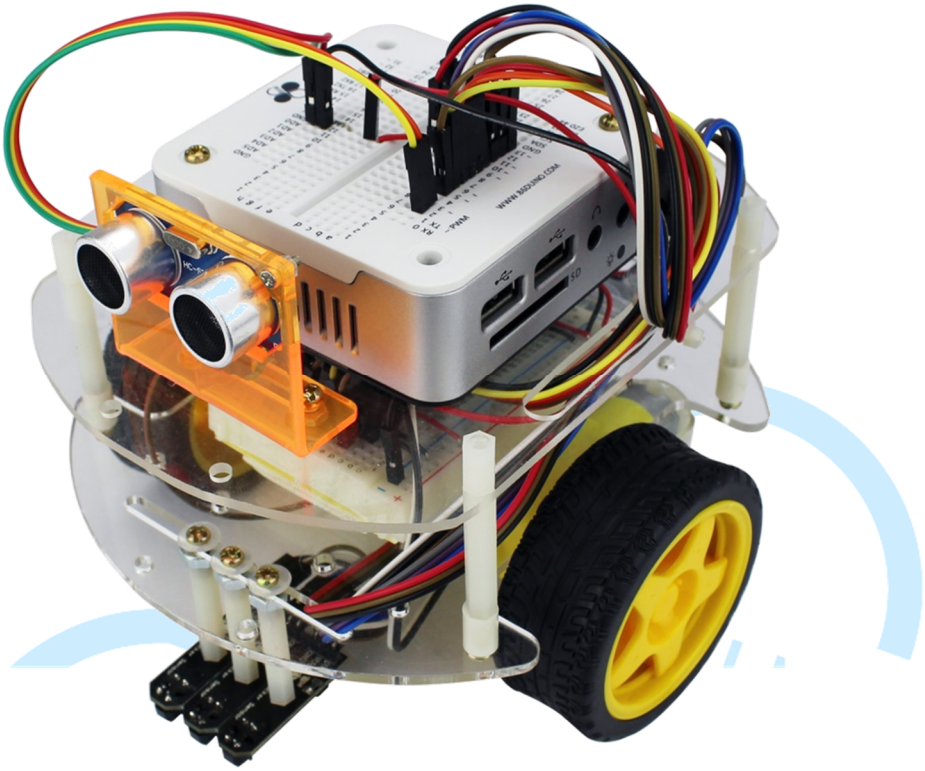
EduCake による自走車



1. 自走車はどのような機能を備えているべきか

86Duino EduCake の各種シグナルコントロール、相互方法を概ね紹介した後、この章では実際に応用された自走車の範例を実作したいと思う。自走車は非常に多くの場所で用いられているが、最も有名なのは火星上陸においてであり、普通の小型車のような大きさの火星探査機キュリオシティ号は、今尚火星で人類のために、火星の各大地と地表のデータを探検し、調査を続けており、絶えず火星の地表の高解像度の写真を送信し続けている。興味を感じた読者は、これらを下記に挙げるウィキペディアのホームページ上で閲覧することが可能である。

http://zh.wikipedia.org/wiki/%E5%A5%BD%E5%A5%87%E8%99%9F

火星上陸は、全く不可能なことだと思われるかもしれないが、地球に目をやれば数え切れないほど多くの自走車が生活の中で応用されている。例えば、GOOGLE は既にオートマチック・カーの実用段階に入っており、100 万キロを走行済みで、市街地での走行テストを開始している。（http://www.ithome.com.tw/news/87223）

５，６年以内の未来において、この機能は普通自動車において実用可能の予定である。会社の門で来客の応対に当たり、お茶出しをする様なロボットも、また多くの場所で実作され、広く使用されている。各種の工場内では、大型の部品を運搬するときに自動搬運車が迅速かつ正確に貨物を所定の場所に運び、生産効率を上げている。Amazon の様な大型量販店のステーションでは、購入者の注文表に基づいて数多くの自動運送自走車が倉庫内で忙しく走り回り、商品を集めてくる。生活において、最もよく目にする自走車は清掃車である。もし要求がそれほど高くなければ、1000 数百万台湾元ほどで、家庭用の清掃機を購入することが可能である。最も高額なものであっても３万から５万円程の元手で、清潔に清掃を行い、規律正しいルート通りに動き、自動で充電を行い、定時に清掃を行い、障害を回避しながら動く等の機能を揃えた物を購入することが可能である。

この様な各種の機械のそれぞれ異なる動力、センサー機器は主な機能の仕組みは、下記に挙げる 86Duino EduCake にそれらの機能が如何に用いられ、使用されているか説明するのに役立つことであろう。

2. 動力と運動コントロール

車は当然のことながらタイヤが動くことによって、前進することが可能である。動力部分は車の性能を表し、優れた耐荷能力と特殊機能を持ち合わせていなくても、簡単なギアボックスと普通の DC モーター及び 65mm タイヤさえあれば走行は可能である。参考のために下記に図 1 を掲載する。この種のモーターは使用する電圧制限が 0~6V，0V の時に完

全に停止し、電圧を徐々に上げ、1V となった時にゆっくりと回転を開始する。電圧を上げ続ければ、モーターの動く速度は更に向上する。これはなぜなら、必要とされる電流が 100~500mA ほどであり、二つのタイヤを合わせても 1A 程度だからである。この時使用される普通の L293D モーターが IC を駆動することによって容易に制御することが可能である。

図 1 小型車用の原則モーターとタイヤ

モーターの動きを制御する電気回路は図 2 で表される。回転をコントロールするために各モーターには 2 本のシグナル回線と 1 本の PWM 速度制御シグナル回線が備わっており、２つのモーターは 6 本の回線によって正確に駆動することが可能となるのである。図の中の 1.2.3 番の回線は第 1 番目のモータを制御し、4.5.6 番の回線は 2 番目のモーターを制御する。これら 6 本の回線は EduCake 上部の未だその他の装置が digital ピンに取り付けられ、使用されていない状態時に確認することが可能である。各モーターの速度をコントロールする際には **analogWrite(EN,速度);**が使用される。この指令は、先に PWM の章で既に説明を行っているので、ここでは重複を避けたい。モータの電圧は最高で 6 でなくてはならないので、電気回路図には 4 つの 1.5V の電池が記載されている。実

際には、変圧器と DC converter を用いて 6V に落として供給する際に使用される。その他、これらの電圧は必ずしも 6V する必要はない。わずかに 7V に上げても、筆者は測定使用することができた。しかしながらそれがモーターの寿命に影響を与えることは、言うまでもないだろう。

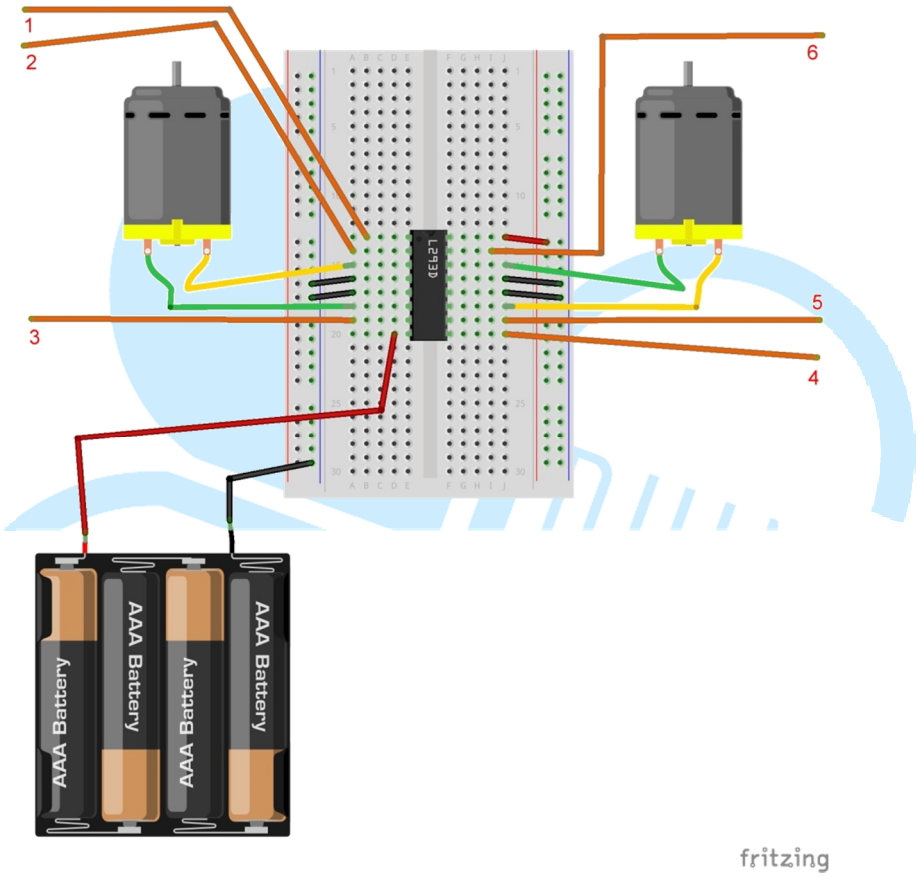


図 2　モーター制御電気回路図

制御コードは以下の通りである。

```
int M1a=6,M1b=7; //    これにより第一のモーターが反転する
int ENa=5; //第一のモーターの速度
int M2a=9,M2b=8; //    これにより第二のモーターを制御
int ENb=10; //第二のモーターの速度
void setup()
{
  pinMode(M1a,OUTPUT);
```

```
  pinMode(M1b,OUTPUT);
  pinMode(ENa,OUTPUT);
  pinMode(M2a,OUTPUT);
  pinMode(M2b,OUTPUT);
  pinMode(ENb,OUTPUT);
}
void loop()
{
  int a;
  for(a = 0; a <= 255; a+=5)
  {
    digitalWrite(M1a,HIGH);
    digitalWrite(M1b, LOW);
    analogWrite(ENa, a); //  第一のモーターの速度を弱める
    delay(20);
  }
  digitalWrite(ENa, LOW); //  停止
}
```

続いて機器の動作状況について知りたいと思われていることと思う。必要となるエンコーダーは図 3 となる。

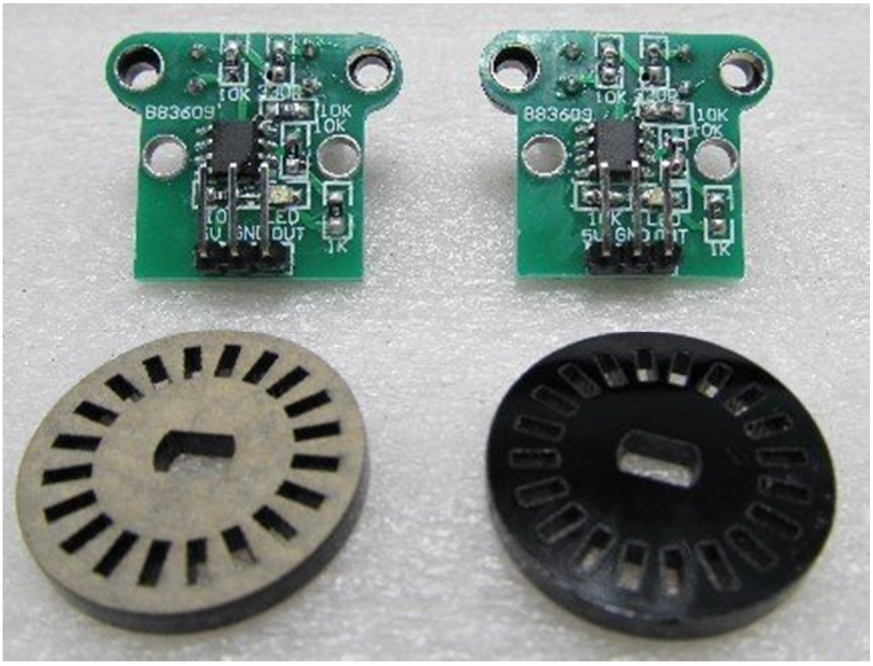


図 3　エンコーダー

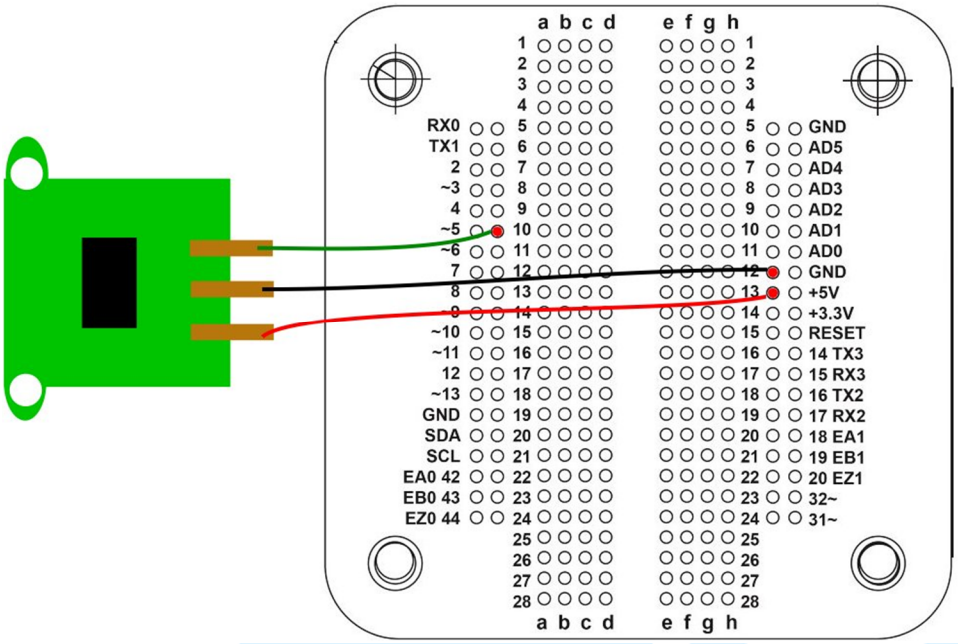


図 4　エンコーダー配線図

通常、エンコーダーは車輪上部に配置され、車輪が振動すると同時に、配線図内で示されているエンコーダー一つ一つが連動する。エンコーダーは光線を遮るものの、丈夫に空いた穴から光が差し込むことが可能となるため、光度感応器を利用し、車輪が動いた距離を算出することが可能である。

実際に使用してみると、車輪の動きは速く、digitalRead 等の感測器で読み取ることは不可能である。効率が劣り、また、基準ではない物を除き、Read では変化した位置を読み取ることは不可能であり、大きな誤差を生みだし、また、プログラムカードの設置された場所を損なう恐れも生じてしまう。

簡単な方法は pulseIn();関数を使用することである。なぜなら、エンコーダーの穴は光度感応器を一回転させることが可能だからである。(光が通過した場合)穴の無い所は回転しない。この時使用される pulseIn(pin, HIGH);はかかった時間を計ることが可能であり、エンコーダー上には合計 20 の穴が空いている。穴の無いものも同様に 20 存在する。合計 40 個で一回転にかかる時間を知ることが可能である。車輪の長さは事前に分かってい

るので、このように簡単に算出することができるのだ。例を挙げてみよう。車輪の直径が6.5cm であるとするなら、車輪が一回転することが移動した距離は以下の通りとなる。

6.5*3.14=20.41cm(当然ながら、ここでの円周率を更に正確なものとし、より正確な数値を出すことも可能である)

エンコーダーには 20 の穴が空き、先に挙げた計算方法により算出された数値を 20cm とした場合、一つにつき前進するのは 20cm/20 個＝1cm となる。然しながらこのような結果は実際、大きな誤差を生む恐れがあり、円周率を除いて、機器がどの方向に滑りゆくかは、車輪が耐荷物によって変形してしまった、などと言う種々の不測の事態などにより異なってくるため、測定の場合は注意する必要が有る。

先に紹介した方法では大きな問題が存在する。それは、機器が万一作動しなかった場合、pulseIn 関数を使用することが問題となり、同様にプログラムカードから関数の timeout 時間にまで影響を及ぼすが、timeout 時間の長短は容易に設定することは不可能である。なぜなら、私たちは機器の作動する速度が速いのか、それとも遅いのか、知る由もないからだ。(当然、私たちは機器が作動した場合、それがある一定の速度で動くということのみ知ることができる)

このようなことは理想的なことではないが、よりよい方法は+millis()関数の使用を中断し、正確な経過時間を計ることである。このことは前章でお話ししているので、もう一度ご覧になっていただきたい。

その他のやり方としては、即ち、20 個の穴を 100 個とし、より正確に計測することである。その他の方法としては、設置位置を変更することだ。例えば、DC モーターを後方へとずらすといったような。減速ギアの速度比を 150:1 とすると、車軸が一回転するたびに DC モーターは 150 回転することになる。これは即ち、制度が 150 上がったということである。しか

しながらこのようなやり方はあまりいい方法とは言えない。なぜなら、処理速度が上がれば、

それだけ、途中でストップしてしまう可能性も高くなってしまうということだからだ。

大量の CPU の時間を消耗してしまう。中断された際のコードは以下の通りとなる。

```
volatile int counter ;
unsigned long oldtime;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(2, RPM, RISING);//シグナルにより LOW から
HIGH に変更がなされたときに中断される
 Serial.begin(9600);
 oldtime= millis(); //現状の時間の記録を行う
 counter =0;
}

void loop()
{
   long rpm;
   if (counter   >= 20) //  先の 20 個のうち，count=20 は 1 回転を
表す
{

   rpm = millis() - oldtime; //  経過時間の取得，単位は千分の一秒
      rpm = 60 * 1000 / rpm; //  回転速度の算出，一分 60 秒，一
秒 1000ms
      oldtime = millis(); //時間を再度計測
      counter   = 0; //  計数機稼働
      Serial.println(rpm, DEC);
   }
}

void RPM ()
```

```
{
 // 中断がなされる度にここへ来る，一つの事柄を行えば，変数は一加わる
 // 内部でシャットダウンを行うのにあたり、最も良い方法を用いる。このようにして、シャットダウンによる負担を軽減する。
 // また、外部のコードが重要な影響を受けることはない
 // シグナルの変化は図 5 の通り
 // シャットダウンが引き起こされた際，シグナルは低から高へと影響を受ける
 // それぞれの位置は全て以下の関数 counter に 1 を加えること.
 counter ++;
}
```

一分は60秒、一秒に就き1000ms。これにより毎分の速度を算出することができ、速度から距離を求めることが可能である。しかしながら注意しなくてはならないことは、関数**millis()**は厳密にいえば極めて正確であるとは言えず、そのことを考慮に入れたうえで用いなくてはならない。

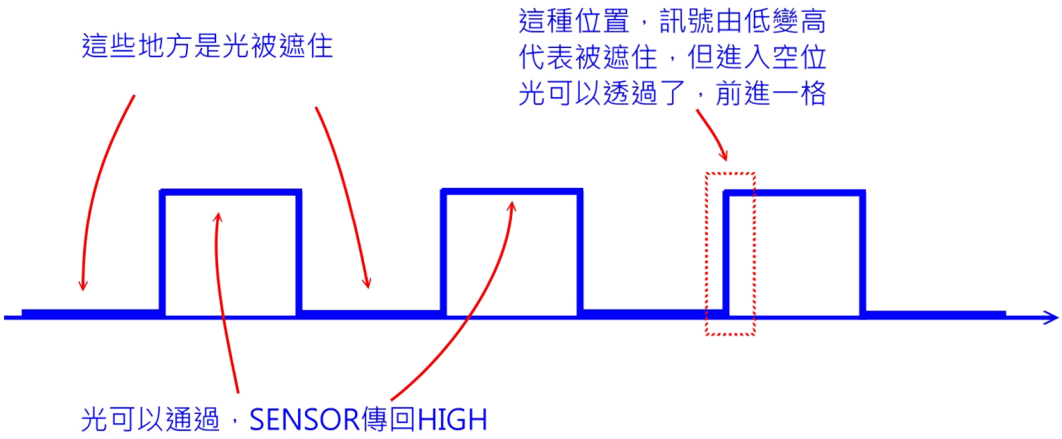


図 5　エンコーダーのシグナル変化

回転速度が分かれば、タイヤの円周が 65mm であることも分る。機器が 1 分間にどれだけ進むのかも、求めることは可能である。然しながらこのような方法の精度はあまり高いと言えないので、より、高度な解析を求めたいのであれば、運算速度が比較的高い CPU を解析し、求める必要が有る。コストは当然高くなるが、この方法については、後述したいと思う。

3. 専用箱の開け閉め

自動搬運自走車は、密閉された箱状のものでなくてはならないが、考えてみても欲しい、もし、有るものが人の手にわたる際、自走車が関係のない者の手にわたってしまったとしたら？

弁当配送車を作るのも同じことである。各人の人の手に確実に渡されるようにしなくてはならない。もし、最初の一人がすべてを持ち去ってしまったら、後の人たちの口に弁当は入らない。

このような現実味を帯びた考えのもと、開閉可能な密閉容器が必要となってくるのだ。このような方法で、モーターなど、様々なものを作成することができる。ここでは SERVO を用いて、簡単に説明したいと思う。例えば図 6 であるが、これは蓋付きの開閉可能な密閉容器である。SERVO とアームは連結し、これにより開閉がコントロールされる。

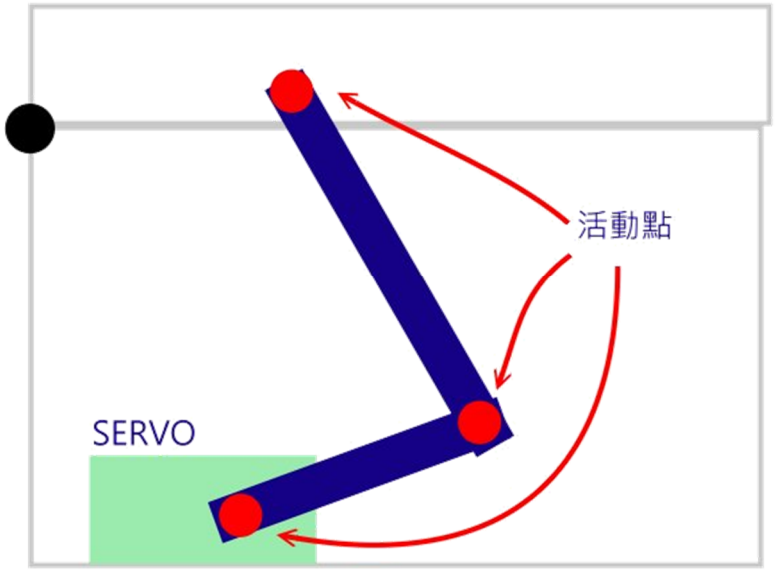


図 6　SERVO による蓋の制御

蓋は元来開閉可能なものであるが、SERVO がアームと連動することにより、蓋を開けることが可能となる。例えば図 7 の様に、SERVO が上に押されれば、それだけ、蓋もより高い場所で開く。このように、事前に蓋が開いた状態の SERVO の位置をあらかじめ予測しておけば (仮に PWM 1000us とする)、蓋が開いた時の理想的な SERVO の位置(仮に PWM 1750us とする)を計測することが可能である。この方程式を用い、容易に蓋の開閉を制御することが可能となる。

ひいては、どのような状況下で SERVO がこの様な動きをするのかが分かれば、RFID、コントローラー、赤外線、bluetooth 等に応用利用することが可能となろう。筆者の以前のオフィスの弁当児童配送車には、各職員の机に RFID が有り、機器は RFID の側に来ると 5 秒間停止し、職員が機器の中から弁当を取り出すと、また走って行ったものだ。

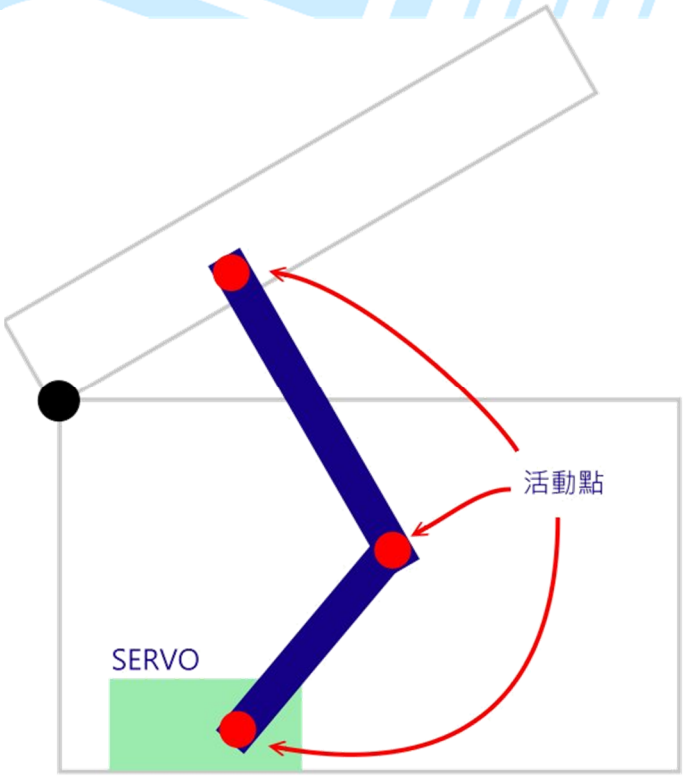


図 7 SERVO が制御する蓋の開閉

コードは以下の通りとなる。

```
#include <Servo.h>
Servo myservo; // Servo が起動し，モーターを制御する
void setup()
{
  pinMode(2,INPUT); // digital を設定し続いてボタンを一つ押す
  myservo.attach(3);   // digital ピン 3 を連結する
}
void loop()
{
  int b=digitalRead(2);
  if (b==HIGH) // クリックするように指示されたボタンを押す
    myservo. writeMicroseconds(1750); // 蓋の開閉を制御
 else
    myservo. writeMicroseconds(1000); // 蓋の開閉
}
```

このコードを利用したもっとも簡単な測定方式とは、ボタンを用いることである。即ち、ボタンを利用することにより、開閉を容易に制御するということである。この概念により導き出される原則は、ゆっくりと、後述していきたい。この機能により、どのような軌道を描くのか、モーターはどのようになるのか、などである。

しかし、このような小型の機器には、恐らく、そのような複雑な説明は不要であると思われる。

4. 赤外線の応用作業について

自走車を正確に走らせるため、CPU が利用されているが、その他にも赤外線も用いられている。この時使用される観測機は下記のようなものである。

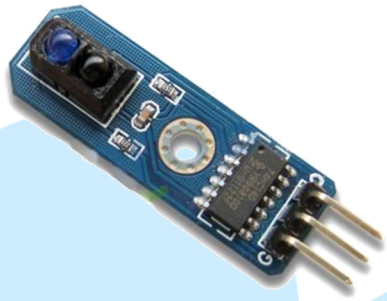


図 8　赤外線観測機

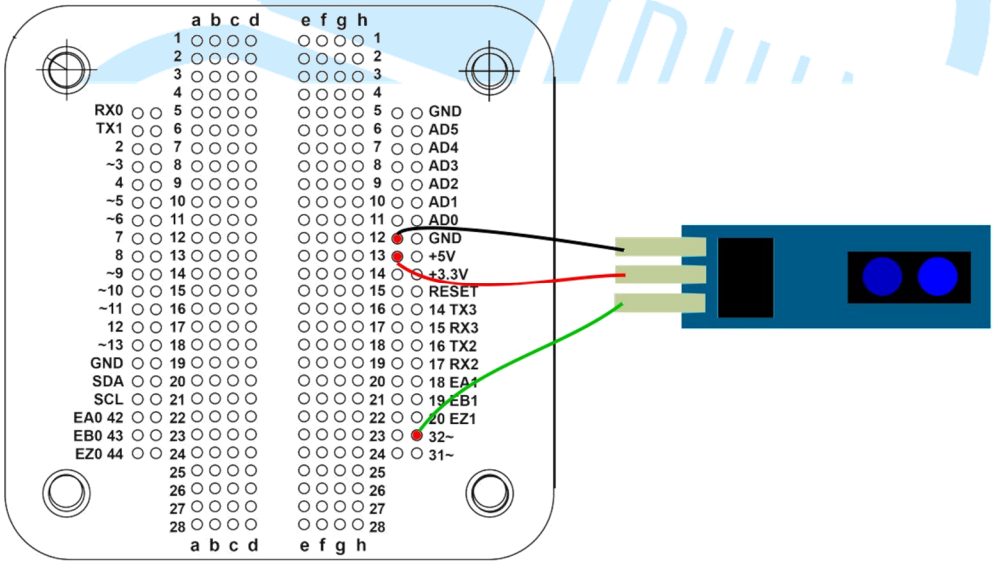


図 9　赤外線観測探知機配線図

使用される方法は一般に GND、VCC と digital ピンを接続する方法である。

3 つの線の区分はそれぞれ VCC、GND、SNG。SNG の黒いシグナル回線が 0 を出力し、その他の色の回線が 1 を出力する。このようにして digitalRead を利用することにより観測機に指令を送るのである。その他の詳しい解説、説明については煩雑を極めるため、こ

こでは説明を避けたい。

赤外線は通常三つを一組にして、或いは 5～7 個を一組とする。

なぜこのような方法がとられるのか？

それにはまず、下図をご覧いただきたい。

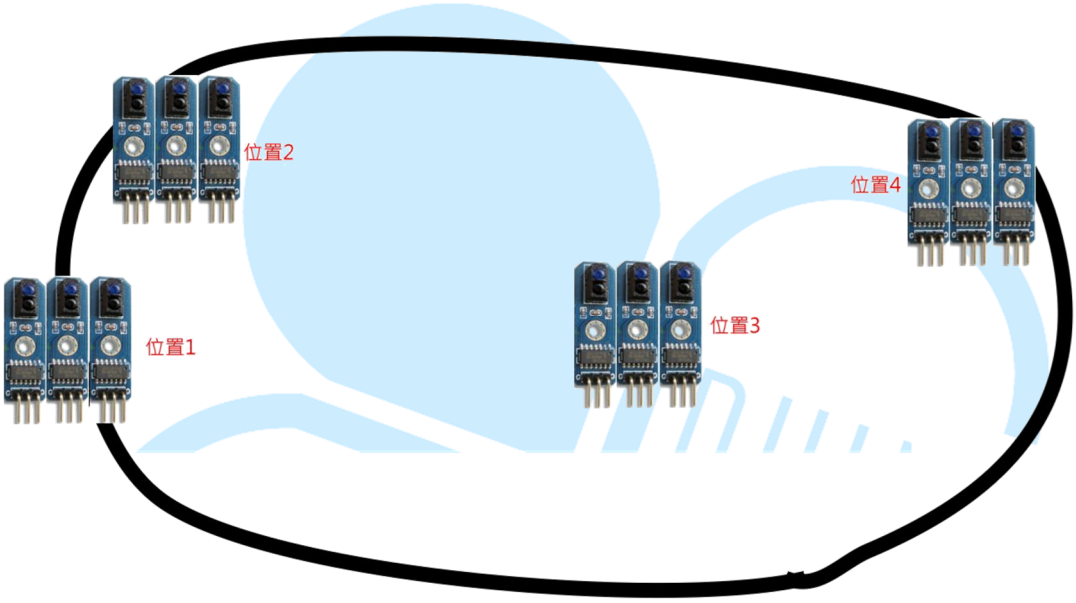


図 10　赤外線観測機にもたらされる各種状態

このような形を用い、図中で用いられているような 3 つの赤外線観測機を併用することで、処理速度を上げ、精度をより向上させる。

各種上記図のような方法があるが、最も的確なのは図 1 の位置である。

1.5 cmないし 1.2 cm幅で設置され、digital　LOW のシグナルは、左右両脇につけられたボードによって反射し、判断を行うのだ。

２の場合を見てみよう。この方法を用いた場合、速度はあまり速くない。なぜなら、赤外線探知機上のビニールが、全て LOW のシグナルを受信するが、右側の赤外線探知機が

HIGH のシグナルを受信してしまうため、左右の車輪の足並みがそろわず、結果として、速度が減速してしまうのである。

では、3 ではどうだろう？機器の位置がこの場所にあると、赤外線観測機に何の問題がなくとも、問題が生じてしまい、結果、自走車はあらぬ方向に走って行ってしまう。これを解決するには、回線の設置位置を修正せねばならないが、この方法は複雑なため、ここでは説明は避けたい。

位置 4 の場合には転倒の恐れが出てくる。右側と中間にある感応機が黒い回線の上で同時に感応し、自走車は大きく左に傾き、結果、バランスを崩した自走車は、大きく転倒するという惨事を招いてしまう。

上記のように考えると、3 つの観測機を設置するに当たり、次のような 8 つの状況が導き出せる。

コードは下記の通り

```
int Left_IR=5,Middle_IR=7,Right_IR=6;
int spd_a=255, spd_b=255; //  左右の速度を制御
void setup()
{
  pinMode(Left_IR,INPUT);
  pinMode(Middle_IR,INPUT);
  pinMode(Right_IR,INPUT);
}
void loop()
{
  int L,M,R;
  int pos;
  L=digitalRead(Left_IR);
  M=digitalRead(Middle_IR);
  R=digitalRead(Right_IR);
```

```
  pos=L*4+M*2+R;
  switch (pos)
  {
  case 0://000  ライン上に全て圧し，左回転する(当然右回転も可能である)
     turn_left();
     break;
  case 1://001，やや右に傾いているため，左に回転させる必要あり
     turn_left();
     break;
  case 2:// 010
  //左右がラインを感知するが，中間部は未感知である。このような状況が発生することはありえず，もし起こり得るとすれば先の処理上の問題である。
  //しかしコネクト部分においては、選ぶことが不可能である。
    break;
  case 3: //011， case1 に相対し ，右に傾きすぎており，より回転する必要がある
    turn_left();
     turn_left(); //  左回転を二回連続して行う
     //  或いは別の関数を用いたり，パラメータを用いることで開店時間を処理したり，左に自転することも可能である。
     break;
  case 4://100,   左に傾きすぎている、要右回転
     turn_right();
     break;
  case 5://101，ライン中央に位置し、直進する
     move_forward()
     break;
  case 6://110，左にかなり傾きすぎており，右に大きく回転させ、修正の必要あり
     turn_right();
     turn_right();
     break;
  case 7://  111  未感知状態であり、直進し，ラインに遭遇数かどうか見る
```

```
      move_forward();
      break;
    }
    delay(100);
}

void move_forward()
{
      digitalWrite(M1a,HIGH);
      digitalWrite(M1b, LOW);
      analogWrite(ENa, spd_a);   //左辺のモーター速度
      digitalWrite(M2a,HIGH);
      digitalWrite(M2b, LOW);
      analogWrite(ENb, spd_b);   //右辺のモーター速度
}

void turn_right()
    digitalWrite(M1a,HIGH);
      digitalWrite(M1b, LOW);
      analogWrite(ENa, spd_a);   //左辺のモーター速度
      digitalWrite(M2a,HIGH);
      digitalWrite(M2b, LOW);
      analogWrite(ENb, spd_b-100);   //右辺のモーター速度を減速
することにより、右回転を行う。

void turn_left()
{
      digitalWrite(M1a,HIGH);
      digitalWrite(M1b, LOW);
      analogWrite(ENa, spd_a-100);   //左辺モーター速度を減速す
る効果あり
      digitalWrite(M2a,HIGH);
      digitalWrite(M2b, LOW);
      analogWrite(ENb, spd_b);   //右辺モーター速度
}
```

```
void turn_left_rotate() //  左に向けて現地点から回転する
{
    digitalWrite(M1a, LOW); //  この二つの状態及び turn_left()が
転倒した場合、モーターは反転する
    digitalWrite(M1b, HIGH);
    analogWrite(ENa, spd_a);   //左辺のモーター速度

    digitalWrite(M2a,HIGH);
    digitalWrite(M2b, LOW);
    analogWrite(ENb, spd_b);   //右辺のモーター速度
}

void move_back() //  後退，実際は進行方向と反対の方向に作動す
る
digitalWrite(M1a,LOW);
    digitalWrite(M1b, HIGH);
    analogWrite(ENa, spd_a);   //左辺のモーター速度
    digitalWrite(M2a, LOW);
    digitalWrite(M2b, HIGH);
    analogWrite(ENb, spd_b);   //右辺のモーター速度
```

赤外線観測機の使われ方によって、自走車自身を破損してしまう危険性をはらんでいる。

一般的に、これを解決するのには、地面から計って 10 cm程の場所に設置し、図 11 の様なシャープのラインナップ製品のような赤外線観測機を使用し、その有効範囲を 10 cmから 80 cmの間に定めることである。

これが最も正しくかつ安全な方法である。

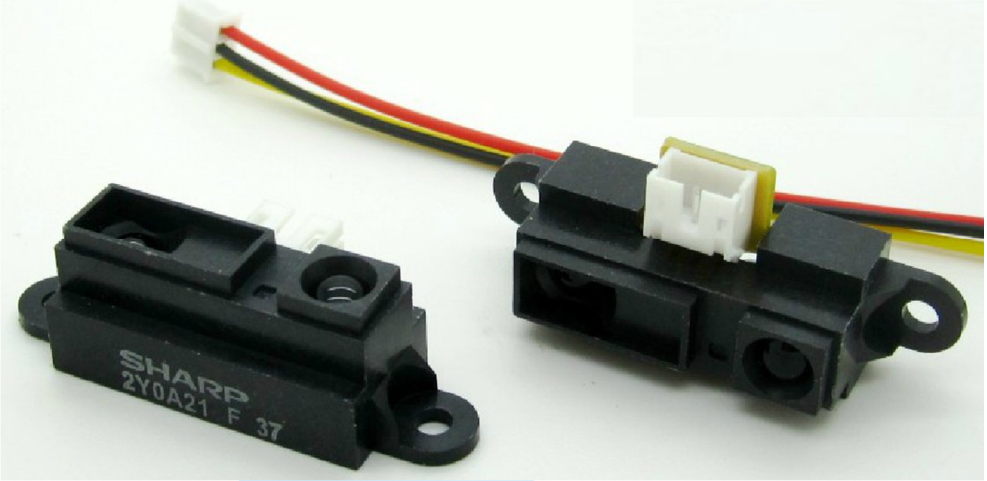


図 11　赤外線観測機

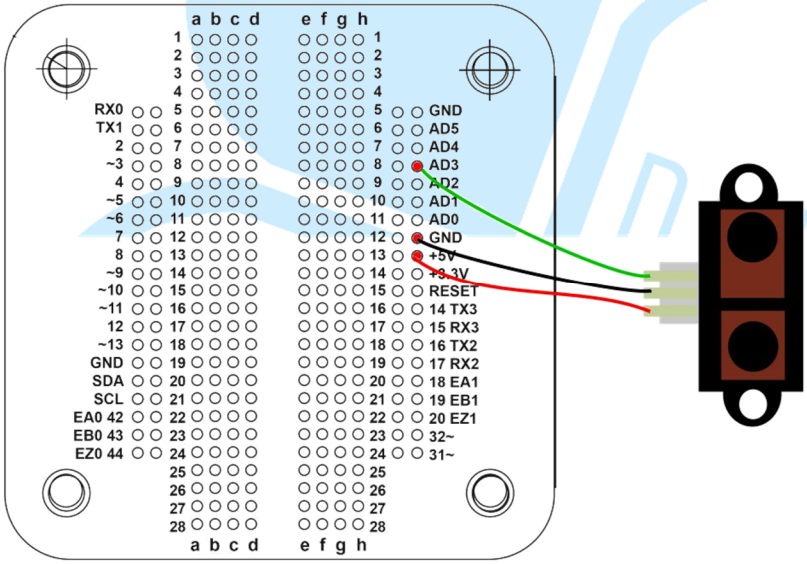


図 12　赤外線観測機回線図

赤外線観測機を制御するにあたり、必要なコードは下記の通りである。

```
// GP2D12 赤外線距離観測
void setup()
{
 Serial.begin(9600);
}
void loop()
{
 int IR;
```

```
 IR = read_IR (3); // analog P3 ピンを接続
 Serial.print("IR distance = ");
 Serial.print(IR);
 Serial.println(" cm");
 delay(500); //毎 0.5 秒間隔ごとに一度距離を示す
}

float   read_IR(byte pin)
{
int tmp;
tmp = analogRead(pin);
if (tmp < 3) return -1; //  少なくとも 3 以下が正確な値ではない
 return (6762.0 /((float)tmp - 9.0)) - 4.0;
}
```

このコードの中で特異なのは 6762.0 /((float)tmp - 9.0)) - 4.0;を用いていることであるが、これは電圧と距離がなす曲線の比例に関係するもので、直線によるものではない。その為、先に 10 ㎝から 80 ㎝の距離をあらかじめ確保しておき、EXCEL 内に必要な数値を入力しておく必要がある。

このようにして、初めて赤外線観測機の位置を正確に測ることが可能となるのだ。

5. 超音波の応用

上記のような方法により、赤外線観測機の距離は正確に測ることが可能となる。人の耳は、40KH z の高音波を聞き取ることが出来ないため、熱を発する機材の付近に何かぶつかったとに上記のような高音波が発せられても、人は箕聞き取ることが出来ないのである。

声は空気中で、摂氏 25 度の環境下で毎秒 331 ㎝進むことが可能である。観測機はおよそこの半分の時間を以て、音波を受信し、それにより距離を算出するのである。

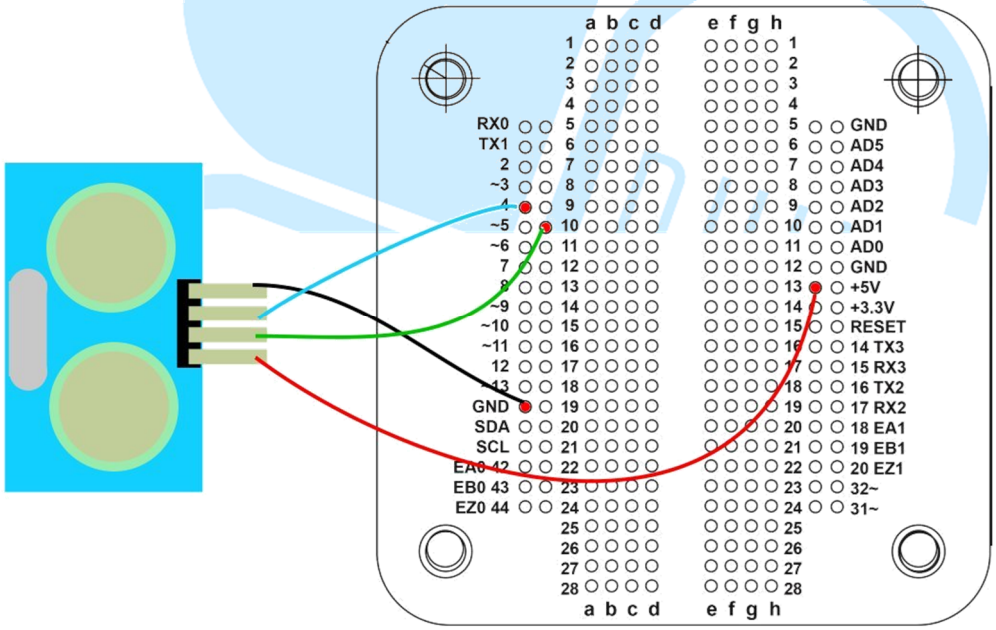


図 13 超音波距離観測機の回線図

制御するに当たり必要なコードは下記の通り

```
int sonic_echo =4; // digital 4 に接続
int sonic_trig=5; // digital 5 に
void setup()
{
 Serial.begin(9600);
 pinMode(sonic_echo, INPUT);
```

```
  pinMode(sonic_trig, OUTPUT);
}
void loop()
{
  digitalWrite(sonic_trig, LOW);
  delayMicroseconds(2);
  digitalWrite(sonic_trig, HIGH); // 高電位 10μs 送信
  delayMicroseconds(10); // 10us 時間後，動作準備が完了し、音波発生を開始する。
  digitalWrite(sonic_trig, LOW); // 時間を直ちに LOW に設定する
  // 超音波の反射を読み取り，経過時間の長短を計測する
  int distance = pulseIn(sonic_echo, HIGH);
  distance= distance/58;    // CM に転換する
  Serial.println(distance);
  delay(300);
}
```

このコードにより簡単に超音波の距離を測ることが可能である。しかしながら、その中の int distance = pulseIn(sonic_echo, HIGH);により取得できるのは、距離は観測できるものの、その範囲は 1~500cm となる。これはなぜか。

なぜなら、超音波と赤外線を比べると、その能力には大きな隔たりがある。音波が発せられた後、多様な阻害物により、その能力は制限され、pulseIn 関数は timeout を発し、算出する結果に大きな齟齬を生じてしまう。

筆者が実際に行ってみたが、1~220cm 範囲内であれば、その数値は信頼できるものであるが、これを超える範囲については、大きく疑問が生じる結果となった。

以上のことを踏まえると、赤外線利用の方に、やはり、多少の信がおけるようである。

とは言っても、赤外線の距離は 80 ㎝に限れるが。

6. 自動掃除機の経路規格につい簡単な紹介

続いて自動掃除機ついてご説明をしていきたい。筆者が実体験した各種多様なこのような危機の現状から顧みるに、一般的にこのような機器はその活躍の場所となる室内環境に大きく影響を受けるようである。

最も顕著なのはその移動経路である。障害にぶつかればその移動経路を変更せざるを得ないからだ。このような状況下に対処するため、必要なのは極めて高度な感応機の存在である。

しかしながら障害に遭遇するためにその進路を変えていては、その能力にも影響を生じてしまうのもやぶさか無いことである。1000 台湾元で買った自動掃除機であっても、その掃除は雑で、掃除結果は不完全であるとしか言いようがない。

このような状況を防ぐために、最も理想的な室内環境は、下記に挙げる図のようなものである。一つの区域が終われば次の区域に移ることができ、その動作はスムーズに行われ、自動掃除機が持つ本来の能力を遺憾なく発揮することが出来る。

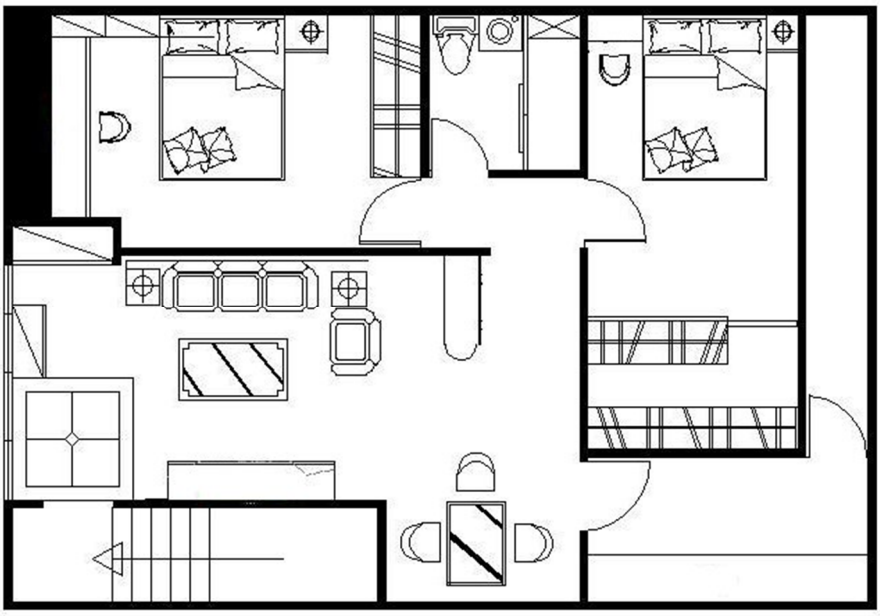


図 14　理想的な室内の間取り図

図 14 における室内はフラットであり、自動掃除機は内部を掃除するに当たり、自らにプログラミングされた情報に基づき、自らがたどった経路を記憶しながら、掃除をしてゆく。機器は全ての室内を掃除し終えると、ある種の地図を内部に保管する。これを SLAM(Simultaneous localization and mapping)と言う。

この塩酸方法については、この章で解説するにはあまりにも難解であり、また、紙面も足らないので、読者の皆様は、このような機能が内部に存在するのだ、とのみご理解いただければ幸いである。

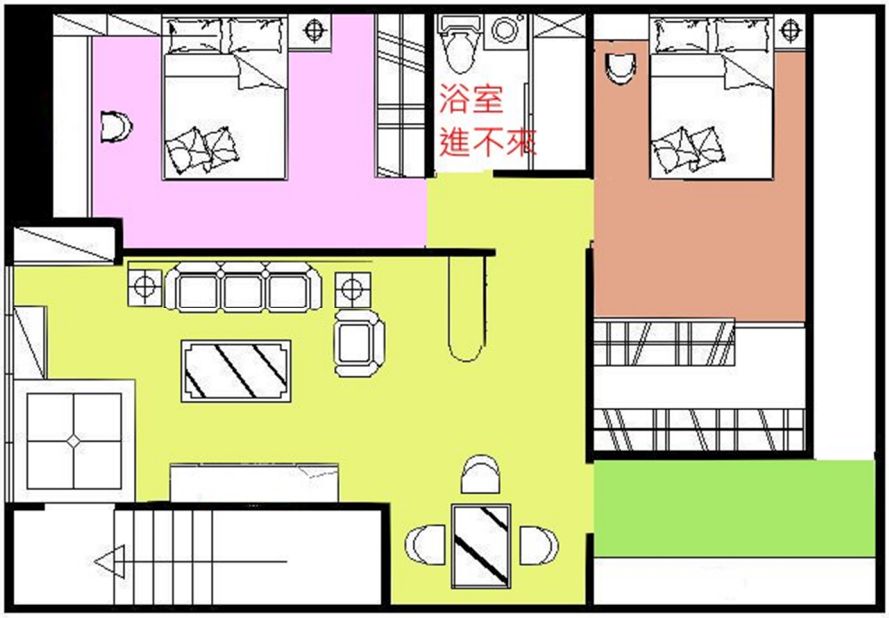


図 15　分割処理後の室内地図

図 15 は SLAM の処理を経て、それぞれの区域を割出し異なるカラーを以て、寝室やリビングを色分けしたものである。このような区分分けをするに当たり、重要視したのは、壁とドアの位置である。このような仮想地図を以て、自動掃除機は清掃作業を開始する。その時、自動掃除機が描く進路は漢字の「回」の字を描く。

なぜ、この様に回」の字を描くのかについては、下記の参考図 16 をご参照願いたい。

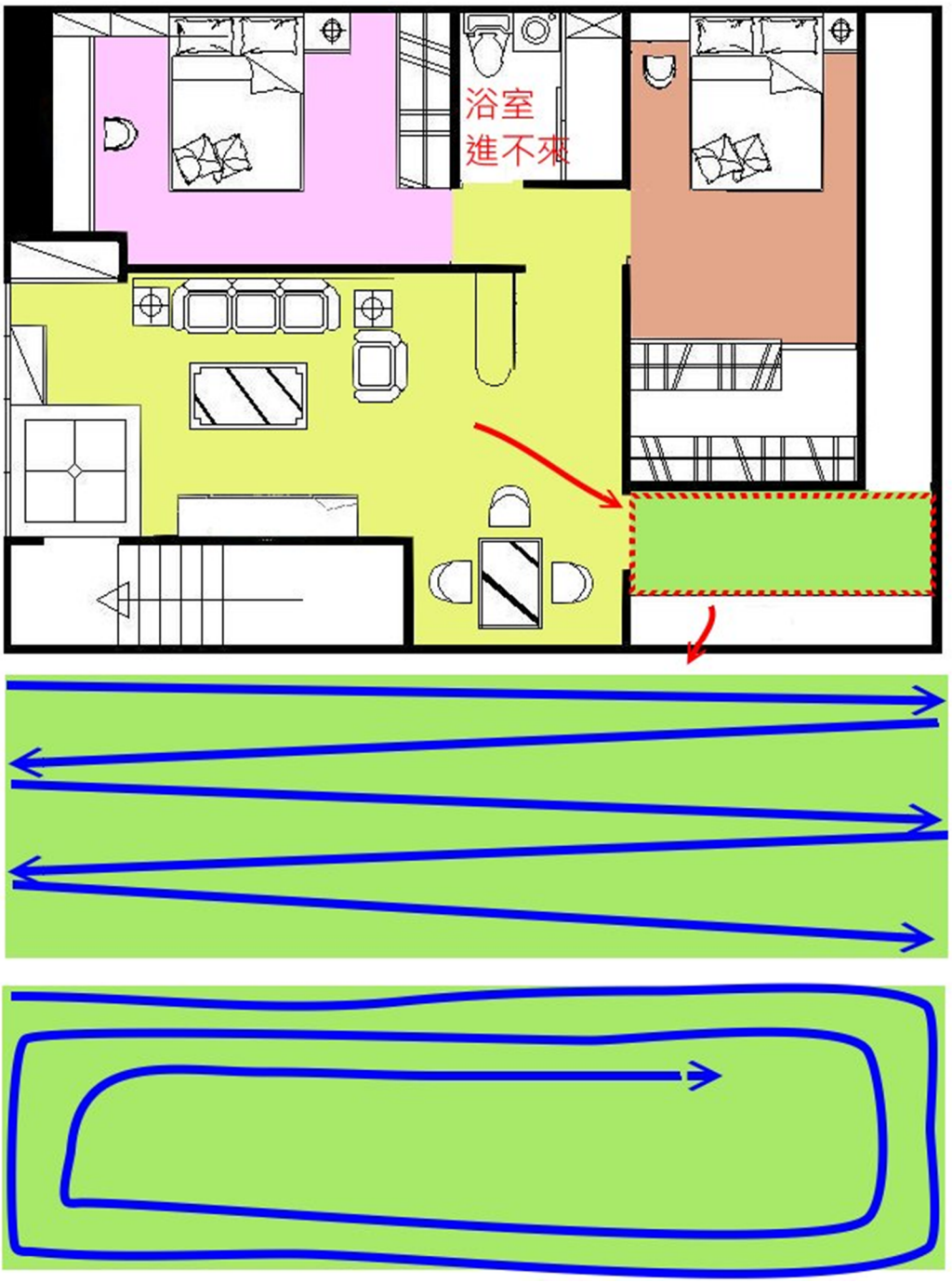


図 16　キッチン内部における 2 種類の異なる走行方法

図中に二つの走行方法を掲載したが、左右に行ったり来たりを繰り返す上段の走行図よりも、回の字型を描く走行方法の方が、効率がいいのがお分かり頂けると思う。

元々175 度（180 度に達すると逆戻りする）で回転しながら動く自動掃除機は、生涯に達するまでこの角度動作を守る。175 度での動作は 90 度で回転するよりも清掃するに

当たり、清掃をできない場所が生じてしまうが、これは仕方のないことである。

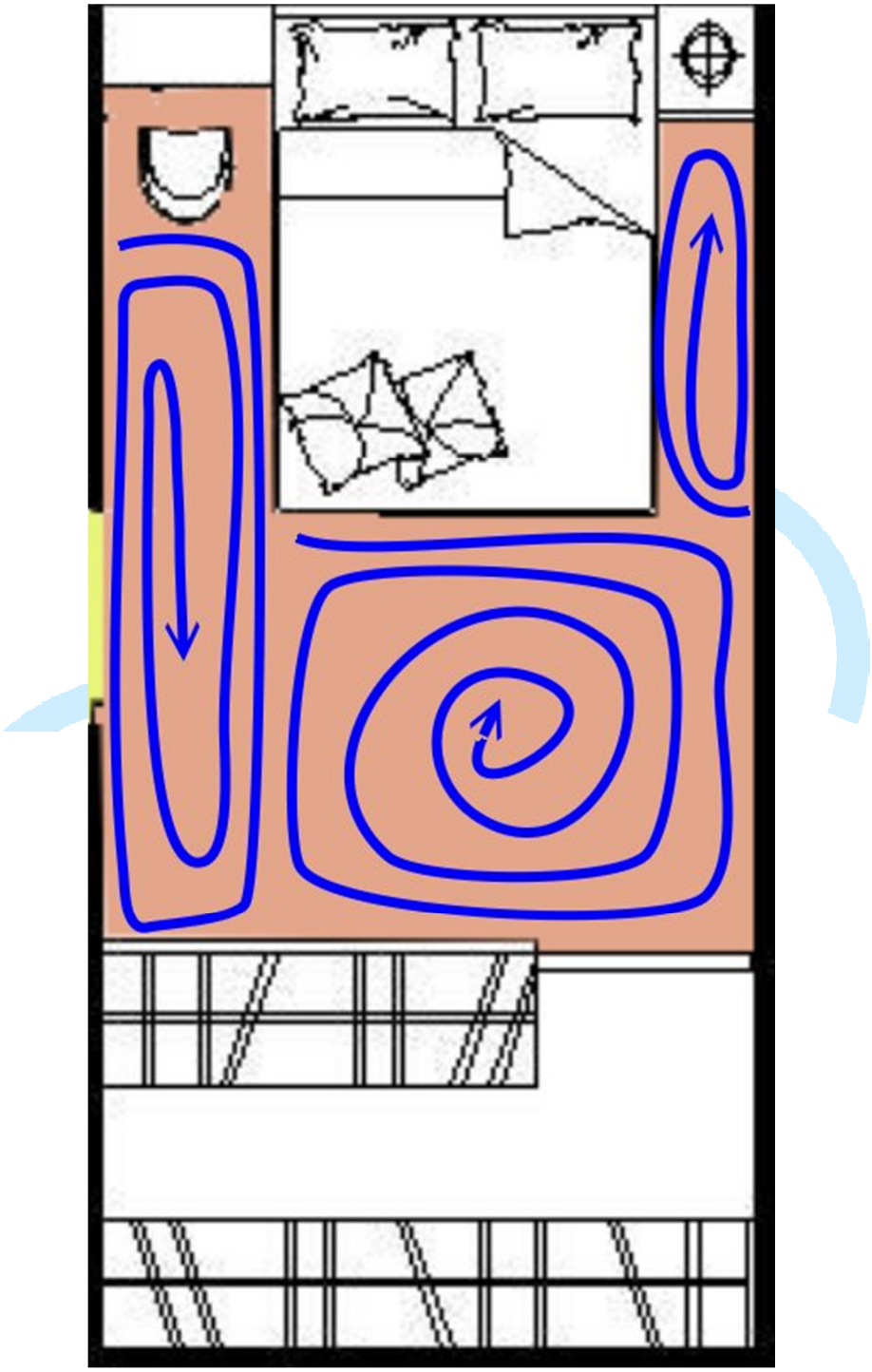


図 17　さらに複雑な環境下での動作状況

7. 結論

自走車様々な領域で応用可能であり、この様な応用需要は、様々な機能が一つに合わさった時に初めて実現可能となる。生活の中で自走車はその需要に基づき、今後益々生産量が増えていくと思われるが、実用的な自走車を

作るに当たり大いに想像力を発揮し、更なる可能性を生み出して欲しいものだ。

後の章で、自走車の更なる能力についてお話ししたいと思う。