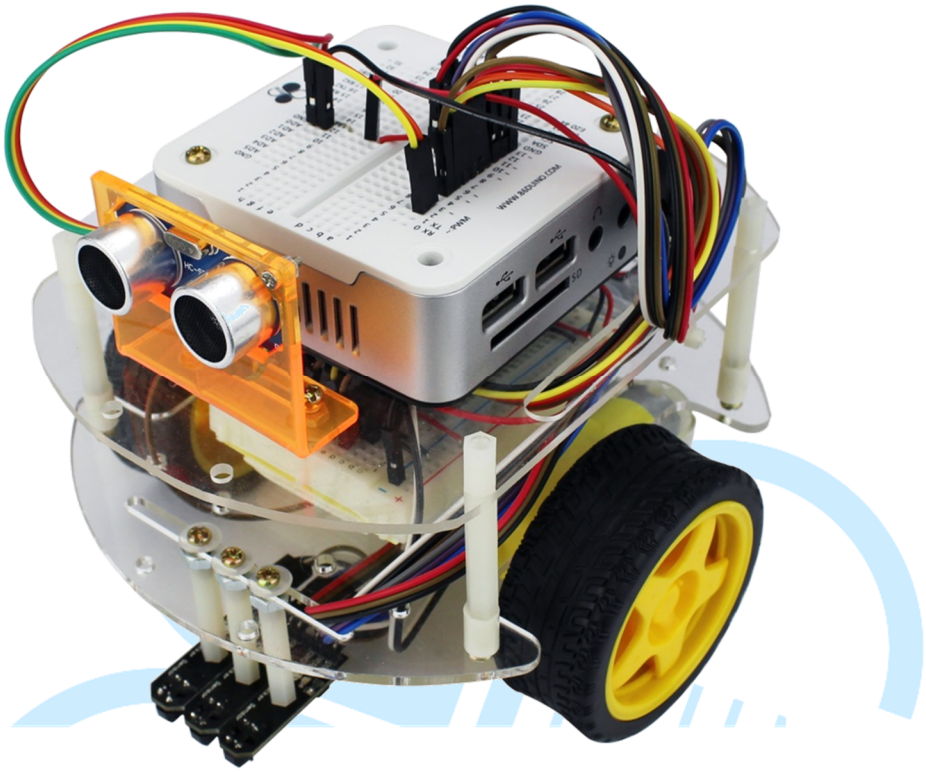
EduCake Robotic Rover



1. Robotic Rover (Automated Vehicle)

In the previous chapters, we talked about different sensors and control on the 86Duino hardware platform. In this chapter, we will implement a robotic rover using an 86Duino EduCake as the controller, with some of these sensors and control.

There are many area a robotic rover can be useful, such as the high profile Mars Rover we sent to explore the planet Mars, which roam around Mars' surface, capturing images and collecting valuable scientific data to send back to Earth.

http://en.wikipedia.org/wiki/Mars_rover

While the Mars Rover seems like from a science-fiction story that is not achievable by most of the average developer, there are other automated robotic vehicles designed with functionality to serve purposes which many of us can relate to, such as:

- Google's driverless car project, which already performed real-life testing on street in multiple cities around the world.

  http://en.wikipedia.org/wiki/Google_driverless_car

- The world's largest online retailer, Amazon, is using automated rovers that roam around their warehouse, automatically picking up merchandise from the shelf to fill customer order.

- The Roomba automated vacuum machine from iRobot, which can roam around your house by itself to vacuum and clean the floor.

While the Mars Rover and Google's driverless car projects are research project that involve 100 of millions dollars in funding, which is out of reach by the average developer. The Roomba automated vacuum robot from iRobot and other automated vacuuming robots are consumer product that cost just a few hundred dollars, which most of the household in developed country can easily afford and purchase. For just a few hundred dollars, consumer is able to purchase one of these intelligent vacuum robot with the following:

- Automatically vacuum and clean the floor in a designated area.

- Automatically perform designed vacuum tasks based on pre-programmed schedule.

- Avoid obstacle as it roams around to perform its duty.

- When battery is low, it's able to automatically return to its charging station to recharge and continue to perform its task after charging.

Imagine this: Perhaps, within 5 to 10 years or sometime in the near future, we can sent driverless car to pick up our friends and love one from the airport.

2. Motion and Control

In order for a robotic rover (Robot) to move around, it needs wheels and motor. For the example in this chapter, we are putting together a simple Robot using geared DC motor and 65mm diameter wheels, as shown in figure-1. The working voltage range for the DC motor is from 0 ~ 6V. At 0 volt, the Robot does not move. As the voltage to the motor gradually increases, and reaching the 1 V range, the motor starts to turn slowly. As the input voltage increases, the motor turn faster and faster and draw approximately 100 ~ 500mA of current per motor, which can draw up to 1A of current between the two motors combine. The L293D dual H-bridge motor driver IC is able to provide sufficient current to support the motors for the project.

Figure-1: Small geared motor assembly with wheel

As shown in figure-2, three signal wires from the 86Duino EduCake are needed to control each motor. One for PWM signal to control speed and the other two to control direction of movement. Together, six signal wires are needed to control the two motors for the Robot. As shown in figure-2, signal wires 1, 2 and 3 are used to control the first motor and wires 4, 5 and 6 are used to control the second motor, using available signal output from the EduCake. To control motor speed, we can use the analogWrite () function. Using analogWrite() function to control motor with PWM is covered in the earlier chapter and will not be covered here.

Please note, the max voltage to the motor is 6V. When using DC to DC converter or voltage source different from the four 1.5V battery connected in series, as shown in figure-2, make sure the voltage is within the 6V range, to avoid damage. While the motor can tolerate little higher voltage, such as 7V, applying high voltage to the motor continuously for prolong period of time will damage and shorten the motor's useful life.

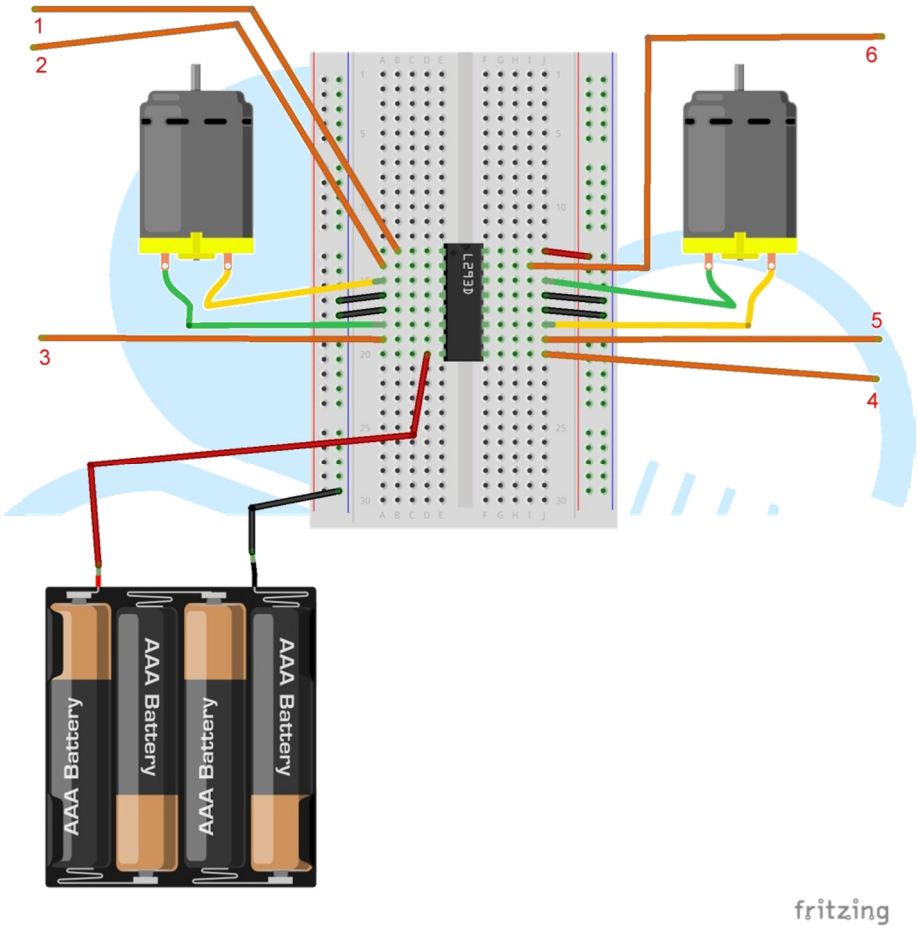


Figure-2: Motor control circuit

Sample codes to control the motors:

```
int M1a=6,M1b=7; // Control 1st motor's direction
int ENa=5; // control 1st motor's speed
int M2a=9,M2b=8; // control 2nd motor's direction
int ENb=10; // control 2nd motor's speed
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); // gradually increase 1st motor's speed
    delay(20);
  }
  digitalWrite(ENa, LOW); // Stop
}
```

To accurately detect the robot's movement, we will use encoder with encoder-disk, as shown in figure-3.

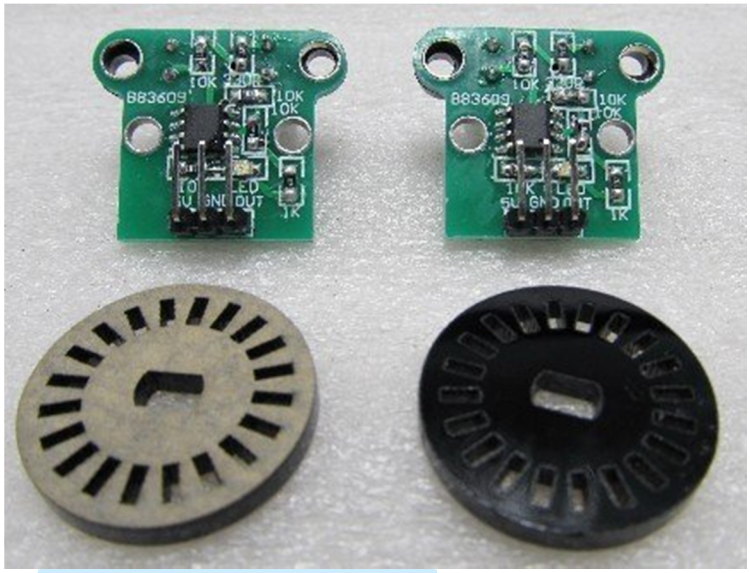


Figure-3: Encoder

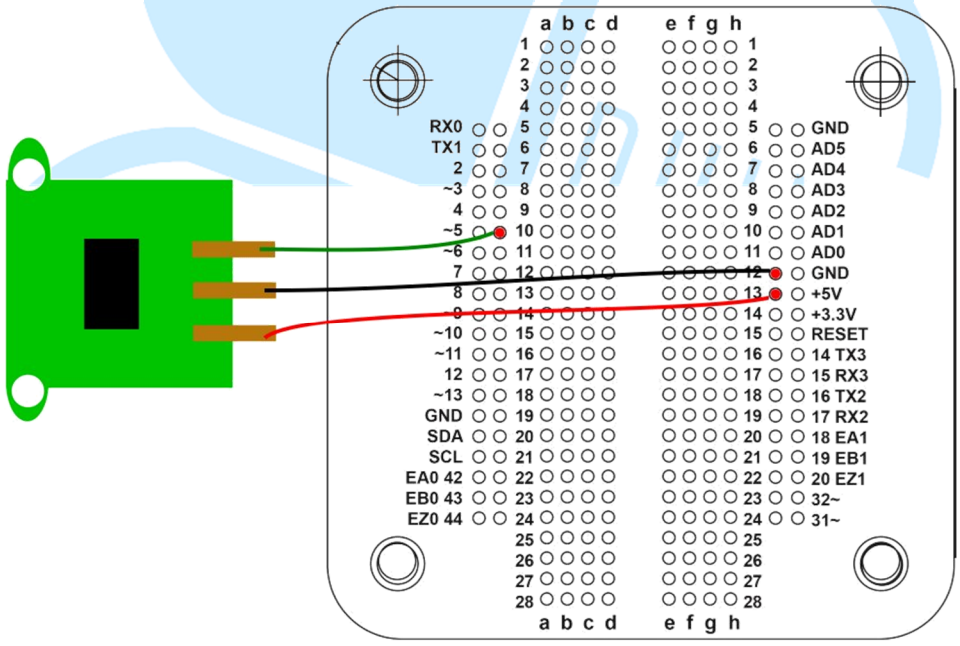


Figure-4: Encoder circuit

When working with robotics vehicle, the encoder disk is typically attached to the wheels, centering on the wheel's axle. The encoder, using optical sensor, is position in a location where the light passes through the slots on encoder disk. As the encoder disk turn, light source reaches the encoder optical sensor in series of pulses, which can be used to determine wheel movement. By detecting the number of light pulse and time delay between each pulse, it's possible to calculate speed and distance.

As the Robot moves around, the attached encoder disks will move in relatively high speed and is not practical to read sensor input from the encoder using the digitalRead() function in a loop. In addition to poor performance, the digitalRead() function may not be able to yield accurate result. For this example, the pulseIn() function is a far better option.

As the encoder disk turns, light passes through the open slot on the disk and hit the optical sensor, which trigger the light detected condition. As the encoder disk turn and the solid portion of the wheel passes over the sensor, blocking light source from the sensor, a no-light-detected condition is created. Using the pulseIn() function, we can detect the length of time needed for the encoder disk to move from one open slot to the next. There are 20 open slot and 20 blocked position on the encoder disk, by multiplying the transition time between light-detected and no-light condition, by 40, we can calculate the time needed for the encoder disk to complete on full turn. Based on the wheels' diameter, 6.5cm, we can calculate the travel distance by multiplying the Robot's wheel radius and the time it takes for the wheel to complete one full turn, using the following formula to calculate the wheel radius:

Travel distance of one full turn = 6.5cm * 3.14 = 20.41 cm

By dividing the total travel distance when the wheel complete a full turn by the number of open slots on the encoder disk, we get the travel distance per open slot. Since there are 20 open slot on the encoder disk, we can calculate the travel distance relative to the encoder disk movement from one open slot to the next by dividing the travel distance for one full turn by 20, as follow:

Travel distance between each open slot = 20.41cm / 20 = 1.02cm

In addition to the above calculation, we need to take into account other factors that can affect the accuracy of the calculated outcome, such as the wheels may skid on slippery surface, weight from the robot may alter the wheel diameter which affect the actual travel distance, and etc.

We can improve the accuracy by using variety of methods. For example, the encoder mechanism can be placed in a different location, such as to detect and measure motor movement at the point prior to the gear reduction mechanism.

Assuming that we are working with a geared motor with 150:1 ratio of gear reduction, by attaching the encoder to detect motor movement prior to the gear reduction mechanism, the accuracy is increased by 150 times. At the same time, this increased accuracy also generate 150 times additional interrupt events. Instead of 20 interrupt events per revolution, the system now trigger 3000 interrupt events per revolution, which the processor must respond to. It takes CPU time and processing resources to handle each of these interrupt events, which may cause the system not able to process other required computing tasks in a timely manner.

It's best to select a balanced approach that delivers acceptable level of accuracy with minimal impact to the system's performance.

Let's take a look at the interrupt handling code:

```
volatile int counter ;
unsigned long oldtime;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(2, RPM, RISING);// Interrupt triggered when
                                  // signal change: LOW to HIGH
 Serial.begin(9600);
 oldtime= millis(); //current time
 counter =0;
}

void loop()
{
  long rpm;
  if (counter  >= 20) // counter to detect completion of 1 turn
  {
    rpm = millis() - oldtime; // elapsed time
    rpm = 60 * 1000 / rpm; // calculate speed
    oldtime = millis(); // reset new time
    counter  = 0; // reset counter
    Serial.println(rpm, DEC);
  }
}

void RPM ()
{
 // when an interrupt event occur, increment counter by 1
 // Keep thing simple to minimize interrupt handling loading
 // and avoid affecting other part of the program
 // As shown in figure-5, interrupt event is triggered when
 // encoder signal transition from LOW to HIGH
 // when interrupt event is triggered, increase counter by 1.
 //
 counter ++;
}
```

Based on the time needed for the encoder disk to complete one full revolution and total traveling time, we can put together mathematic formula to calculate the speed and distance. In the above code, the calculation is based on millisecond, where there are 60 seconds to one minute and 1 second is equivalent to 1000ms (millisecond). When the wheel is moving in high speed, calculation based on the millisecond resolution may not provide sufficient accuracy for certain application. If this is the case, you can consider using microsecond as the calculation parameter, which increase accuracy by factor of 1000.

The Robot we are working with is equipped with a 70 RPM geared motor, which can rotate 70 revolution within one minute, which rounded to approximately 1 revolution per second. With 20 open slot on the encoder disk to trigger 20 interrupt event per revolution, the motor is capable to generate an interrupt event every 50ms. For our purpose, the millisecond calculation parameter provides sufficient accuracy.

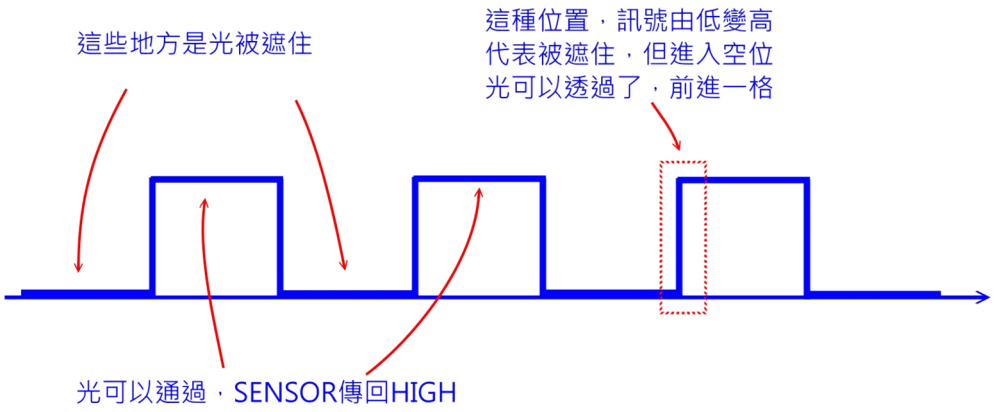


Figure-5: Encoder signal transition

With the wheel's diameter (65mm) and rotation speed, we can calculate how far the Robot can travel in one minute, and how long it takes for the Robot to travel one meter or whatever distance needed to plan the path for the Robot to travel. The hardware used in this example is able to deliver limited level of accuracy. To improve accuracy, we can use a higher resolution rotary encoder and a faster CPU, which also increase the cost.

3. Cargo Box Control

To use the Robot to transport things from one location to another, a cargo box is needed. To avoid things from falling off or removed by others during transport, an enclosed cargo box is needed.

There are different options to keep the enclosed cargo box close and secure during transport, such as using an electromagnetic mechanism, servo and others. For this example, a servo is used to control the cargo box's opening and closing, as shown in figure-6.

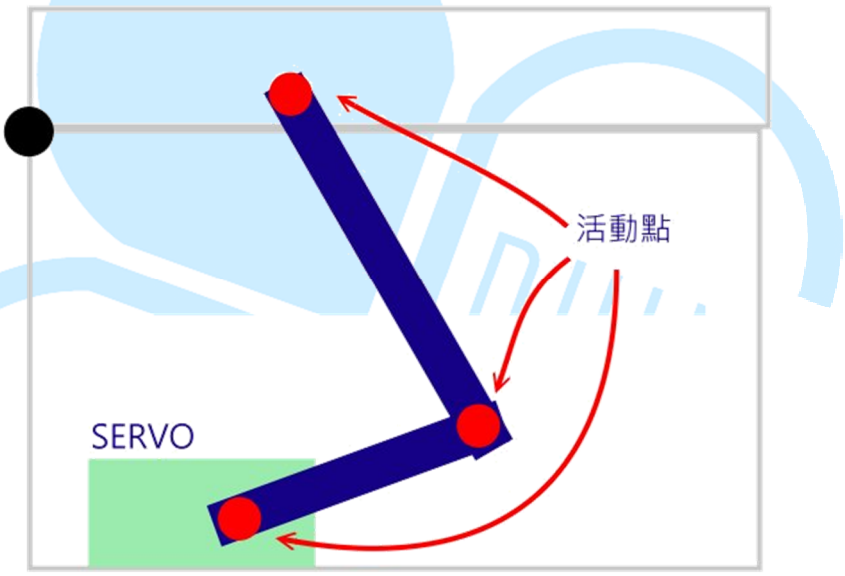


Figure-6: Servo controlled cargo box

Starting in the close position as in figure-6, turn the servo arm pushes and open the cover, as shown in figure-7. Let's assume when PWM 1000us is applied to the Servo, the cargo box is in the close position, and when PWM 1750us is applied to the Servo the cargo box cover is fully open, we can use a simple program to control the opening and closing of the cargo box based on these parameters. In addition, it's possible to use RFID, remote control, IR, Bluetooth and other resources to trigger the opening and closing of the cargo box's door. For example, with the cargo box, it's possible to put together a Robot to deliver lunch box to co-workers in the office, as follow:

- An RFID tag is attached to each co-worker's desktop, which function as a check point for the Robot to stop for approximately 5 to 10 seconds, for the person at the desk to respond.

- Each co-worker is wearing an RFID tag on a wristband, as the person at the desk reach their hands toward the Robot, the RFID tag triggers the cargo box to open and push out a lunch box to the person at the desk.

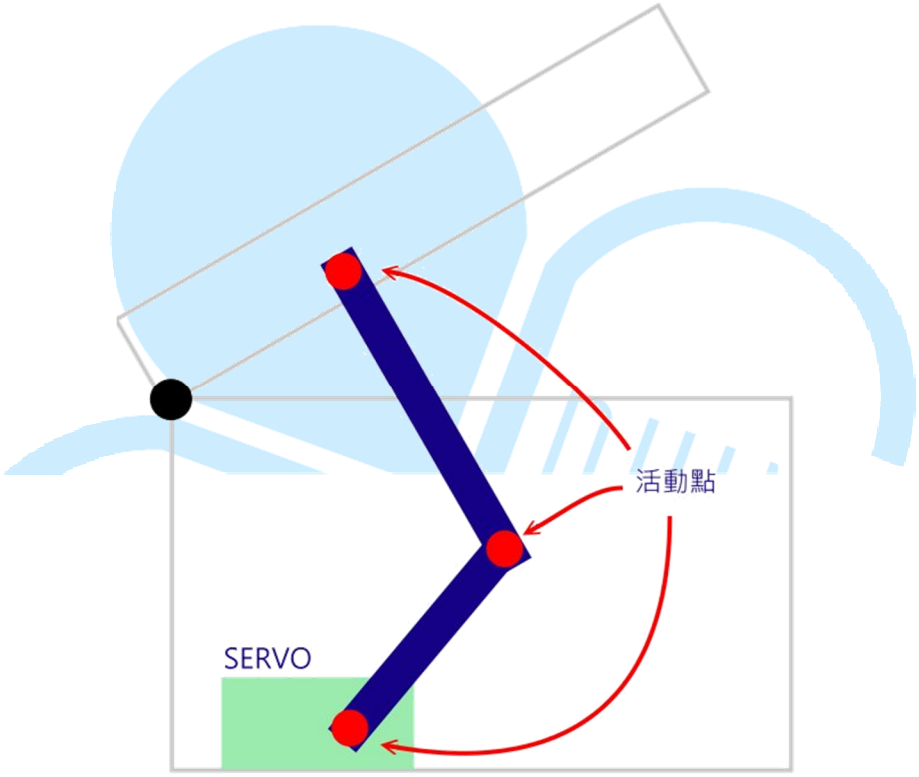


Figure-7: Cargo box with servo control

Here is a listing of code to control the servo:

```
#include <Servo.h>
Servo myservo; // Creating the servo object
void setup()
{
  pinMode(2,INPUT); // setup digital pin 2 as input from a
                    // button
  myservo.attach(3);  // attach servo object to digital pin 3
}
void loop()
{
  int b=digitalRead(2);
  if (b==HIGH) // 代表按鈕被按下
    myservo. writeMicroseconds(1750); // open cargo box
 else
    myservo. writeMicroseconds(1000); // close cargo box
}
```

The above codes can easily be tested using a button interface. When the button is pushed, the cargo box open. When the button is released, the cargo box close.

As an alternative to using servo to control the cargo box's opening and closing, sliding track with stepper motor can be used to yield better result. However, it will increase the cost and complexity. For our example, the servo option is sufficient.

Before continuing, let's talk briefly about some good practices for programming. Instead of writing a long program within a single module, which may result in 100s even 1000s lines of codes that are difficult to trace and debug, it's best to write code in small chunk, separating the codes in multiple modules and call each of these modules from the main program. By doing this, the code is easier to trace and debug. In addition, each of these module can easily be reuse in other projects. Since these codes has been validated and tested as part of an existing program, you can save lots of time and efforts, when developing new program that can take advantage of these existing modules.

4. Infrared

In order for the Robot to travel along a track, marked by tape, we need to use sensor to detect the surrounding. While image recognition provide good result, it requires lots of processing resources and high speed imagine capture camera, which is complex and costly. An alternative low cost and simple approach is to use infrared sensor, as shown in figure-8.

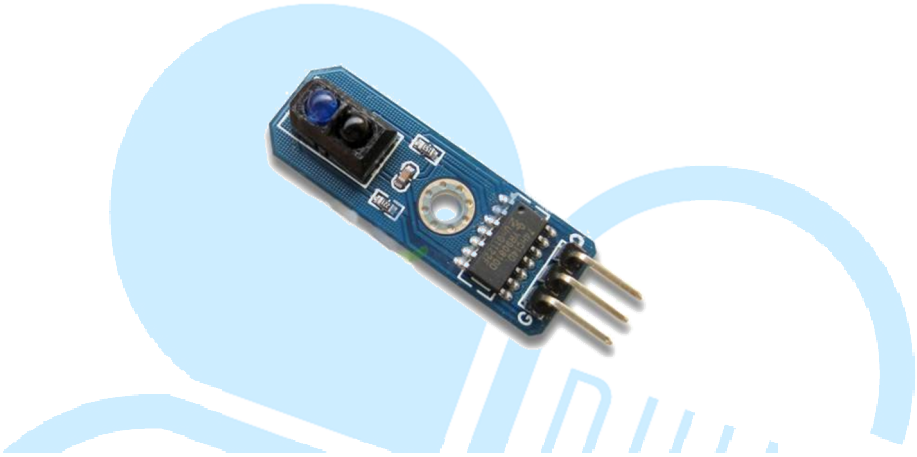


Figure-8: Infrared sensor module

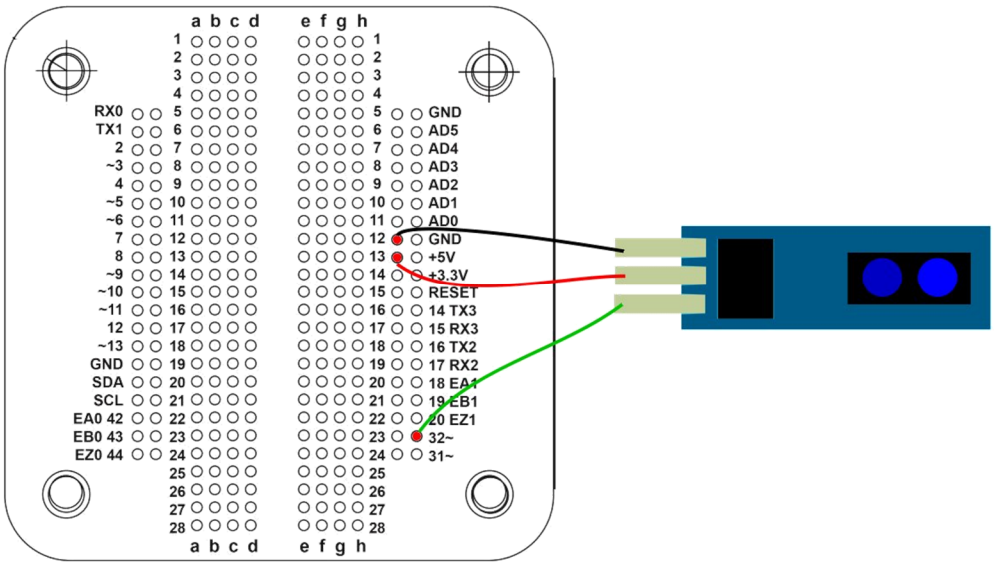


Figure-9: Infrared sensor module circuit

As shown in figure-9 above, the infrared sensor module has 3 wires, Vcc, Gnd and Sng.

This type of infrared sensor module has an emitter that emits a low intensity infrared

signal and bounced back after hitting a surface. Then, the receiver that is part of the infrared module, detect and sense the infrared signals that are bouncing back. In general, surface with different color can reflect and bounce high level of infrared back to the sensor module, except for black color, which absorb majority of the infrared, which can be used to detect whether the infrared beam is reflecting back from a black or non-black surface.

Today, in many Robot competition that involves having the robot following along a marked path in black color, it's common to see infrared sensors being used in these competitions. To avoid interference, the infrared sensor module is typically placed in a position that is close to the traveling surface, approximately 1~2 mm away. Infrared sensor module's usage is also simple, which involve 3 wires: VCC, GND and SNG. SNG is the signal output from the sensor, when black color surface is encountered the output is 0. When the sensor encounters surface with color other than black, the output is 1. Using the digitalRead() function to read the output from an infrared sensor, we can easily determine whether the surface it encounters is black or non-black.

In the situation where paths with different color marking are used to make the competition more interesting, it requires color sensor module and is far more complicated, which is not within the scope for us to cover in this chapter.

It's common to see infrared sensor module with group of 3, 5 or 7 sensors, positioned side by side in a row with even space between them. This type of implementation is best for detecting and following a marked path, as demonstrated in figure-10.

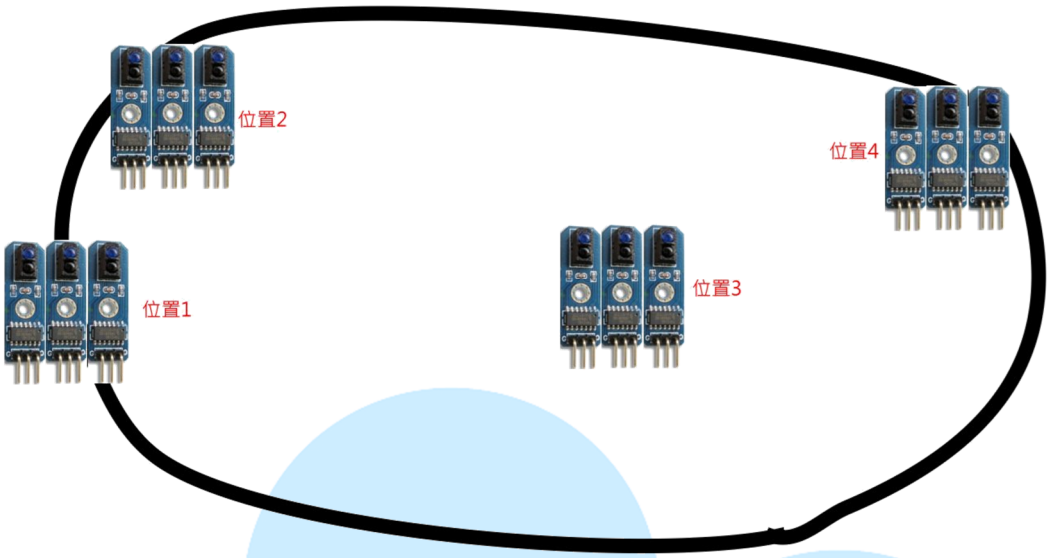


Figure-10: Different infrared sensor module condition in line-following application

In figure-10, the simple terrain along with the placement of sensor modules, each with group of 3 infrared sensors, in different position along the path to demonstrate how to use these sensors to detect and follow the marked path.

For a simple path, a group of 3 infrared sensors is sufficient to deliver good result. For complex path with lots of un-uniform turn and curve, more than 3 sensors are needed to deliver acceptable result.

The path in figure-10 is marked by a 1.5cm wide black tape. Within the group of 3 infrared sensors, the space between each sensors is 1.2cm. As shown by the sensors at the 1st position in figure-10, when the sensor in the middle is position at the center of the black tape and send out a digital LOW signal, both of the sensors to the left and right are detecting the non-black surface and send out digital HIGH signal. The combined signals from all 3 sensors indicate the Robot is centered on the marked path and should maintain the same traveling direction.

As indicate by the group of sensors at the 2nd location in figure-10, the left and middle sensors are above the black tape and both send out a LOW signal. The sensor to the right are above the non-black surface and send out a HIGH signal. This condition indicates the Robot is traveling too far to the right. The formula need to issue command for the Robot to adjust its traveling direction slightly to the left, to move the Robot toward the center of the marked path.

When both of the left and right motors are moving at the same speed, the robotics rover travel in a straight line. To adjust the robotics rover's traveling direction, to the left or right, you can use the analogWrite() function to control the left and right motors to move at different speed. When the left motor is moving faster than the right one, the rover turns to the right. When the right motor is moving faster than the left one, the rover turns to the left. To make a sharp turn quickly, where it cannot be accomplish by controlling the left and right motor to move at different speed, such as a curve that is less than 90 degree, you can achieve this by controlling the left and right motors to move at opposite direction. When attempting to control the rover to make a sharp turn quickly, it may be necessary to reduce the robot's traveling speed just before the turn, to prevent the robot to roll-over as the result of traveling too fast while making a sharp turn.

The terrain in figure-10 is a simple one, without any sharp turn. We can use the simple method to adjust the robotics rover's traveling direction, by controlling the left and right motors to move at different speed.

In figure-10, the sensors at the 3rd location is completely separated from the marked path. When all three sensors is detecting the black tape on the marked path and send out digital LOW signal, which indicates the robot is traveling within the marked path, you can let the robot continue to travel at whichever direction it's moving toward.

In the case as shown for the sensors at the 3rd location, all three sensors is not detecting the marked path where the robotics rover does not have any point of reference that can be used to figure out its current location relative to where the marked path. Allowing the robot to travel toward a random direction may move the robot further away from where it needs to be. In scenario like this, we can implement additional resource such as Bluetooth, by placing a Bluetooth transmitter close to the center of the marked path and Bluetooth receiver on the robotics rover. As the robotics rover move further away from the marked path, the Bluetooth signal become weaker, which can be used to indicate whether the robot is moving in the correct direction, back toward the marked path. Adding Bluetooth involves additional cost and increased complexity which is not within this chapter's objective. There will be more coverage about Bluetooth in the later chapter.

The sensors at the 4th location is the opposite from the sensors at the 2nd location,

where the middle and right sensors are above the marked path and both send out digital LOW signal. The left sensor is above the white surface and send out digital HIGH signal, which indicates the robotics rover is traveling too far to the left and need to adjust its heading to the right to continue and follow the marked path.

Based on the signal from the 3 infrared sensors, where each sensor may send digital LOW or HIGH condition, there are 8 different possible combination of signals from the 3 sensors.

Here is a listing of codes to handle each of these conditions:

```
int Left_IR=5,Middle_IR=7,Right_IR=6;
int spd_a=255, spd_b=255; // variable to control L & R motor speed
                          //  L = Left,  R = Right
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  all sensors on marked path, turn L (ok to turn R)
    turn_left();
    break;
  case 1://001 traveling slightly to the right, turn L
    turn_left();
    break;
  case 2:// 010
  // Since the black tape used to mark the path is 1.5cm
  // and the space between each infrared sensors is 1.2 cm
  // It's not possible for this condition to happen.
  //
  // Under other situation, this condition can happen
  //  and need to act accordingly
    break;
  case 2:// 010
  // Since the black tape used to mark the path is 1.5cm
  // and the space between each infrared sensors is 1.2 cm
  // It's not possible for this condition to happen.
  //
  // Under other situation, this condition can happen
  //  and need to act accordingly
    break;
case 2:// 010
  // Since the black tape used to mark the path is 1.5cm
```

```
     // and the space between each infrared sensors is 1.2 cm
    // It's not possible for this condition to happen.
    //
    // Under other situation, this condition can happen
    //  and need to act accordingly
      break;
    case 3: //011 · similar to case1, with both middle and right
            // sensors off the marked path, the robot further
            // to the right in comparison to case1
            // and need to call the turn_left() function twice
            // to adjust the heading.
      turn_left();
      turn_left();
      // Instead of calling the turn_left() function twice,
      // we can use a different function or include a variable to
      // increase the length of time for the robot to move toward the
      // left.
      //
      break;
    case 4://100 · traveling slightly to the L, turn R
      turn_right();
      break;
    case 5://101 · Robot is following the marked path
      move_forward()
      break;
  case 6://110 · traveling far to the L, adjust heading to the R
      turn_right();
      turn_right();
      break;
    case 7://  111 marked path not detected
           //  continue in the same heading
      move_forward();
      break;
    }
    delay(100);
  }

  void move_forward()
  {
     digitalWrite(M1a,HIGH);
     digitalWrite(M1b, LOW);
     analogWrite(ENa, spd_a);  //control L motor speed
     digitalWrite(M2a,HIGH);
     digitalWrite(M2b, LOW);
     analogWrite(ENb, spd_b);  //control R motor speed
  }

  void turn_right()
  {
     digitalWrite(M1a,HIGH);
     digitalWrite(M1b, LOW);
     analogWrite(ENa, spd_a);  //control L motor speed
     digitalWrite(M2a,HIGH);
     digitalWrite(M2b, LOW);
     analogWrite(ENb, spd_b-100); //decrease R motor speed to turn R
  }
```

```
void turn_left()
{
    digitalWrite(M1a,HIGH);
    digitalWrite(M1b, LOW);
    analogWrite(ENa, spd_a-100); //decrease L motor speed to turn L
    digitalWrite(M2a,HIGH);
    digitalWrite(M2b, LOW);
    analogWrite(ENb, spd_b);  //control R motor speed
}

void turn_left_rotate() // Turn left without moving forward
{
    digitalWrite(M1a, LOW); // These two lines of code configure
    digitalWrite(M1b, HIGH); // the L & R motors to move in
                             // different direction

    analogWrite(ENa, spd_a);  // control left motor speed
    digitalWrite(M2a,HIGH);
    digitalWrite(M2b, LOW);
    analogWrite(ENb, spd_b);  // control right motor speed
}

void move_back() // move backward
{
    digitalWrite(M1a,LOW);
    digitalWrite(M1b, HIGH);
    analogWrite(ENa, spd_a);  //control L motor speed
    digitalWrite(M2a, LOW);
    digitalWrite(M2b, HIGH);
    analogWrite(ENb, spd_b);  //control R motor speed
}
```

There are many other situation where infrared sensor is useful, such as detecting the surface edge to avoid falling off the platform, such as in the scenario where the robotics rover move around near a stair without a fence to protect it from falling off the stairway, infrared sensors can be used to detect when the robot is moving near the edge of the stair and change the traveling direction to avoid falling.

There are infrared sensor designed to sensor the distance from the reflecting surface, such as the Sharp infrared sensor shown in figure-11, which can be used to measure distance from 10cm to 80cm.

By placing an infrared sensor in the forward facing position pointing downward to detect the distance to the surface in front of the robot as it travel, when the robot is

traveling close to the edge of a stair, as the infrared beam passed the edge of the stair, the distance to the surface will dramatically increase, which can be used as a signal to indicate the robot is moving toward the edge of the traveling surface and programmatically adjust the heading to avoid falling off the edge.

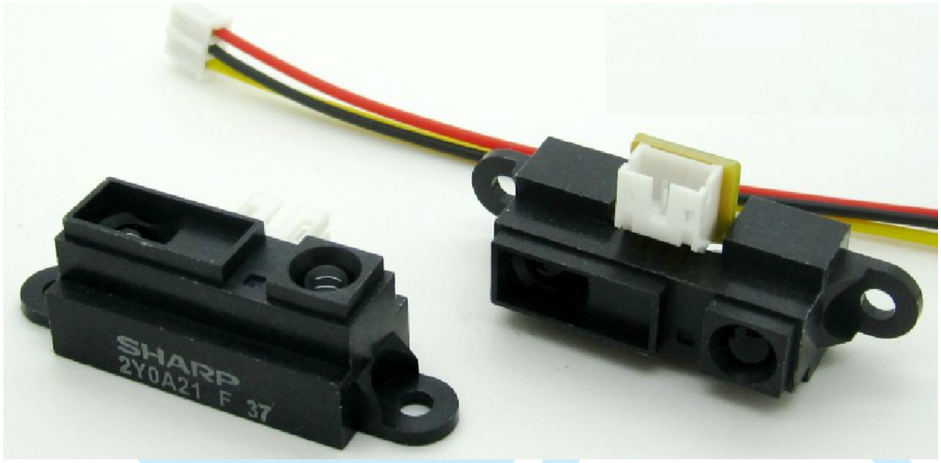


Figure-11: Infrared sensor to measure distance

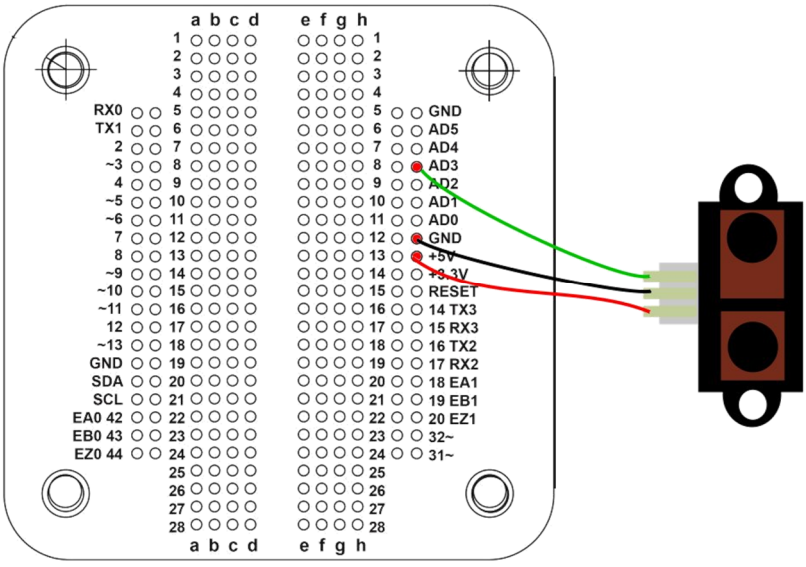


Figure-12: Circuit showing infrared sensor attaching to EduCake

Codes to read data from infrared sensors to measure distance:

```
// GP2D12 distance measuring infrared sensor
void setup()
{
 Serial.begin(9600);
}
}
void loop()
{
 int IR;
 IR = read_IR (3); // Attach to analog P3
 Serial.print("IR distance = ");
 Serial.print(IR);
 Serial.println(" cm");
 delay(500); //delay 500ms before the next reading
}

float  read_IR(byte pin)
{
 int tmp;
 tmp = analogRead(pin);
 if (tmp < 3) return -1; // when less than 3, value is improper
   return (6762.0 /((float)tmp - 9.0)) - 4.0;
}
```

Sensor value, in voltage, from the distance measuring infrared sensor relative to the actual distance is not linear where the actual result is calculated using the following formula:

Sensor data = (6762.0/((float)temp – 9.0)) – 4.0;

```
void move_back() // move backward
{
    digitalWrite(M1a,LOW);
    digitalWrite(M1b, HIGH);
    analogWrite(ENa, spd_a);  //control L motor speed
    digitalWrite(M2a, LOW);
    digitalWrite(M2b, HIGH);
    analogWrite(ENb, spd_b);  //control R motor speed
}
```

5. Ultrasound

In the previous section, we talked about using infrared sensor to measure distance. While infrared sensor is low cost and deliver acceptable accuracy in measuring, it can only measure short distance. In the case of the Sharp sensor used in the previous section, the effective measurement range is from 10cm to 80cm. Ultrasound sensor is used to measure longer distance.

To measure distance, an ultrasound sensor emit high frequency sound waves in the 40 KHz range, which is above the range human can hear, where these sound waves will reflect back to the sensor from the surface of object they hit. Distance measurement can be calculate based on the time it takes for the sound waves to reflect back to the sensor, using the speed of sound as an additional parameter.

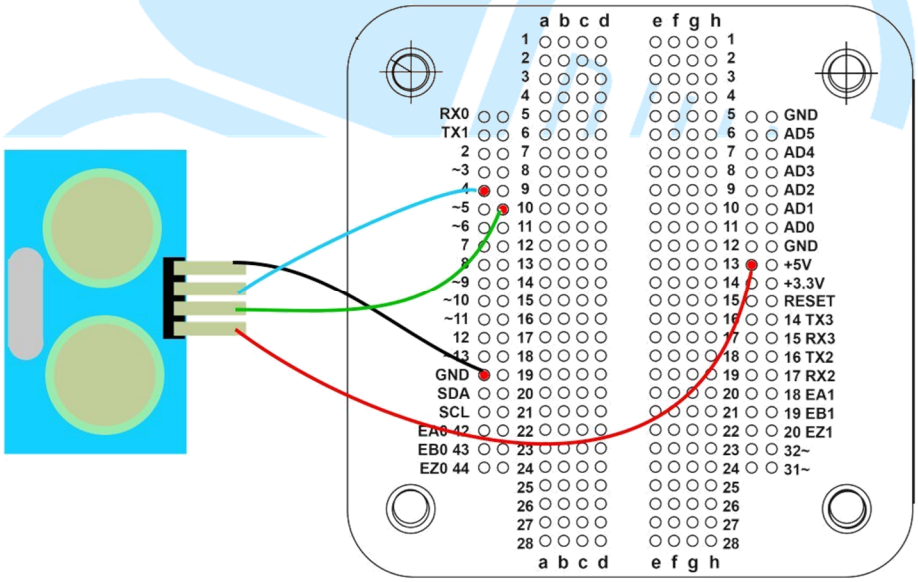


Figure-13: Circuit to attach ultrasound sensor module to EduCake

Here is a listing of codes to measure distance using ultrasound sensor:

```
int sonic_echo =4; // attach to digital pin #4
int sonic_trig=5; // attach to digital pin #5

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);  // set digital pin HIGH
                                   // for 10µs
  delayMicroseconds(10); // delay 10us for the module to
                         //  prepare & emit ultrasound

  digitalWrite(sonic_trig, LOW); // after delay
                                 // set digital pin to LOW
  // wait for reflected sound wave and calculate the time
  int distance = pulseIn(sonic_echo, HIGH);
  distance= distance/58;   // convert distance to CM
  Serial.println(distance);
  delay(300);
}
```

The simple codes in the listing above can be used to measure distance ranging from 1cm to 500cm. Comparing to infrared, ultrasound sensor is more susceptible to interference. There are different conditions which can cause the ultrasound sensor to lose its accurate measurement ability, such as:

- When the sound wave is reaching a soft surface such as sponge, the sound wave may be absorbed by the surface and does not reflect back to the sensor and cause a time-out condition for the pulseIn() function.
- When the sound wave fails to reach a 90 degree angle surface, the sound wave may bounce to another surface before reflecting back to the sensor.
- When the sound wave is traveling too far and become too weak to reflect back to the sensor.

While there are ultrasound sensor capable to deliver accurate measure for longer distance, most of the sensor for the DIY market are limited to the 1cm to 500cm range.

For the ultrasound sensor we are using, the effective range is approximately 1cm to 220cm.

6. Introduction to Path Planning for Robotic Vacuum

This section talks about the challenges associate with path planning for robotic vacuum robots. Based on our observation, many robotic vacuum robots are designed in such a way that the robot attempt to cover all of the accessible floor space in a room and move on to the next room until all of the rooms in the house are covered. There are broad range of approaches to accomplish these objectives, ranging from simple to complex system.

Many low-cost robotics vacuum machines uses a simple approach where multiple mechanical sensors are attached to the robot's bumper. The robot would travel along a reference path, which may be randomly selected, until it encounter obstacle along the path then change its travel direction randomly. This approach generally is not able to cover all of the floor space and rarely able to do a good job automatically without some human intervention.

To maximize the robot's ability to cover all floor space within a room, it's necessary to implement computing algorithm to move the robot around in a Z pattern or circulating a room's parameter while gradually reducing the circulating diameter and move toward the center of the room, as indicated by the drawing in figure-16. In addition to the selected travel pattern, there are more complex computation & algorithm involve to create a function robotic vacuum with the ability to cover a living quarter with multiple rooms, such as the apartment floor plan in figure-14.

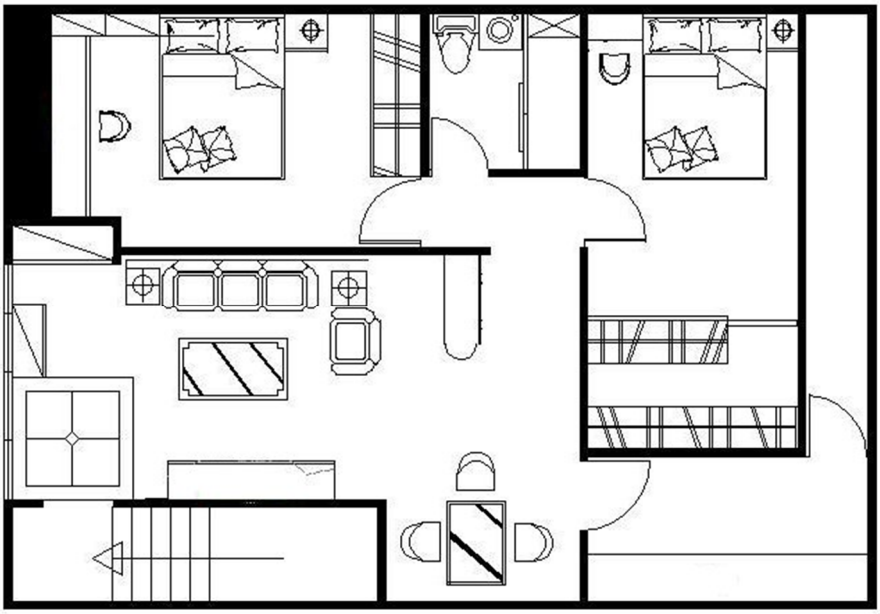


Figure-14: Floor plan for a multi rooms apartment

In order for a robotic vacuum robot to efficiently clean all of the rooms in the floor plan, as shown in figure-14, the robot's wheels need to be equipped with encoder and other sensors to detect and record the terrain it travels through, to create a virtual map in its memory. Using the SLAM (Simultaneous Localization and Mapping) approach, along with the associated sensor and algorithm, the robot establish multiple coordinates on the virtual map and use these coordinates to identify obstacle, path that link one room to another and separate the floor plan into different zones where it covers each of these zones individually and move on to the next zone after completing the current one until the whole apartment or house is covered, as shown in figure-15.

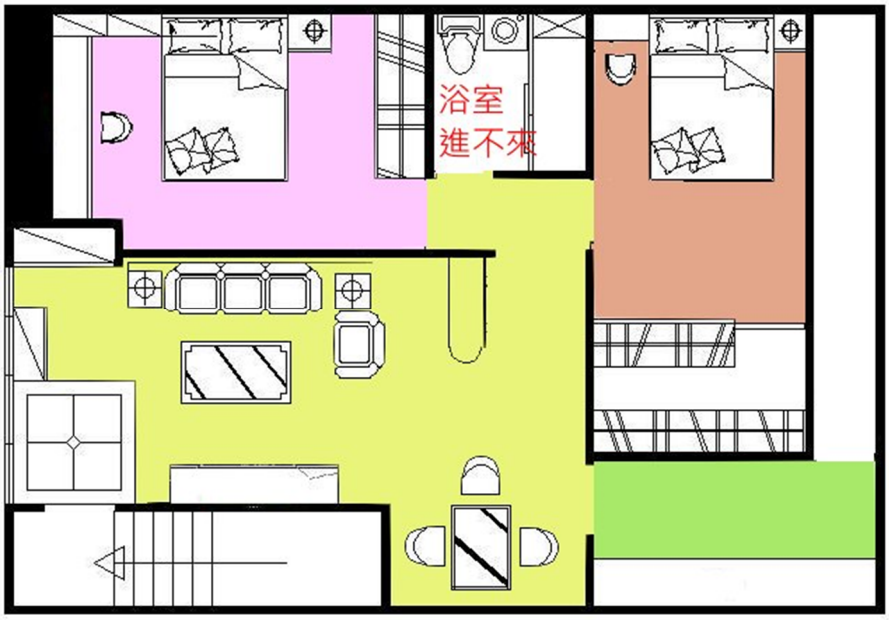


Figure-15: Floor plan separated into zones

As shown in figure-15, the floor plan is separated into multiple zones where the door-frame or entry point to each room are the coordinates used to separate each of these zones. Area that is not to be clean (or not accessible) by the robot is also marked, such as the bath room.

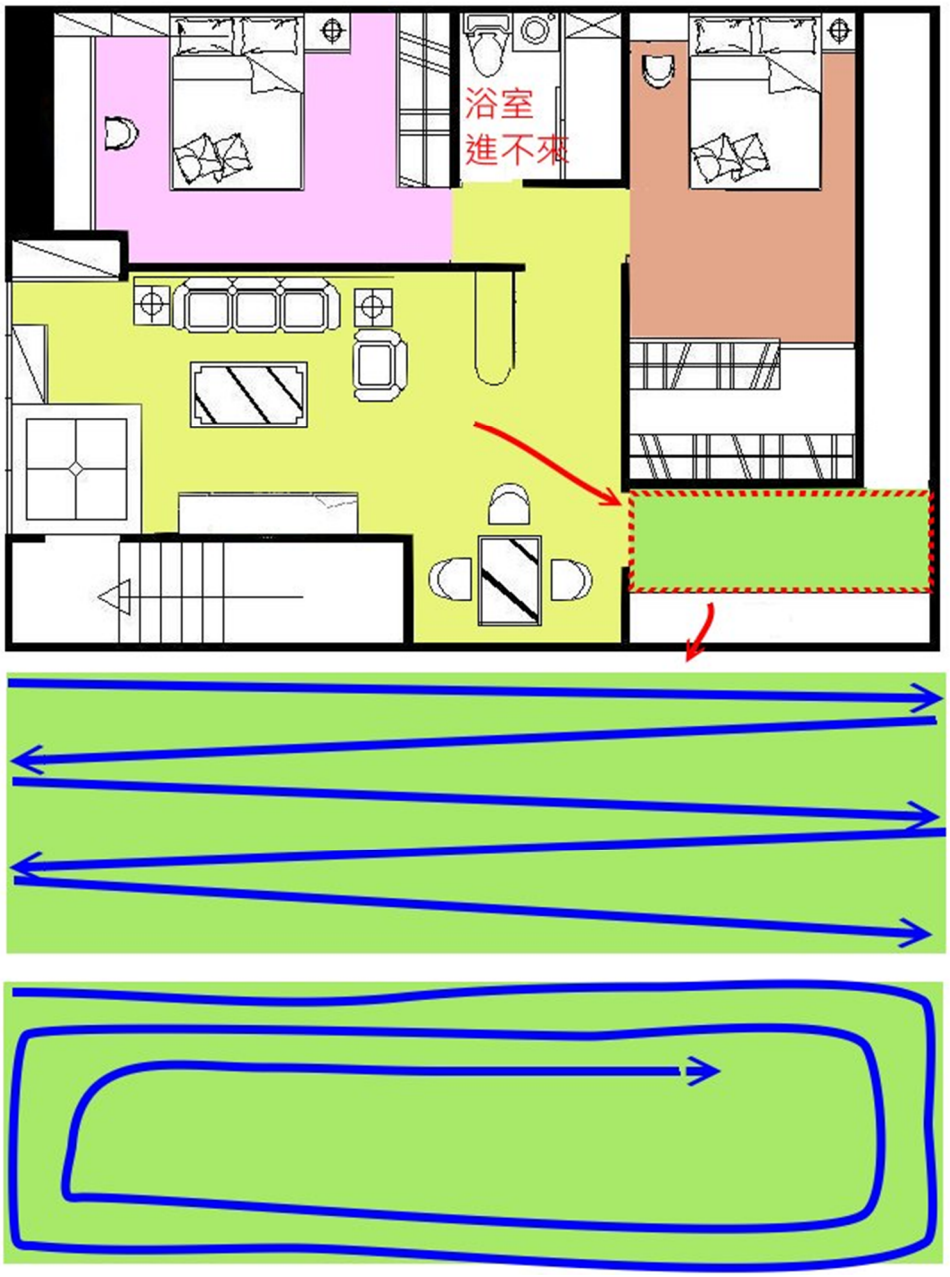


Figure-16: Two movement algorithm targeting open area in the kitchen

With the virtual map and zones in its memory, the robots can efficiently move around and clean all of the floor, one zone at a time. To insure maximum efficiency and covering all of the floor space thoroughly, some robotic vacuum robot uses an algorithm to control the robot to move in a Z pattern or circulating around the covered area and gradually reducing the circulating diameter, moving toward the center.

When using the Z pattern algorithm, the robot initially move along one of the zone's outer edge until it encounters an obstacle, or reaching a coordinate that

identify the outer limit for the current zone, the robot change the travel direction by 175 degree to create the Z travel pattern (changing travel direction by 180 degree would control the robot to travel in the exact opposite direction it just came from). With the Z movement pattern, some of the floor space is not covered.

To improve floor space coverage, the robot can be control to change it travel direction by 90 degree, move a short distance that is equal to the robot's width and then change the travel direction by 90 degree and travel back to the exact direction it came from in parallel to the path already cleaned. While this may seem like a better approach, it require highly accurate encoder & sensor, with increased cost and complexity, to insure the robot's movement is just right.

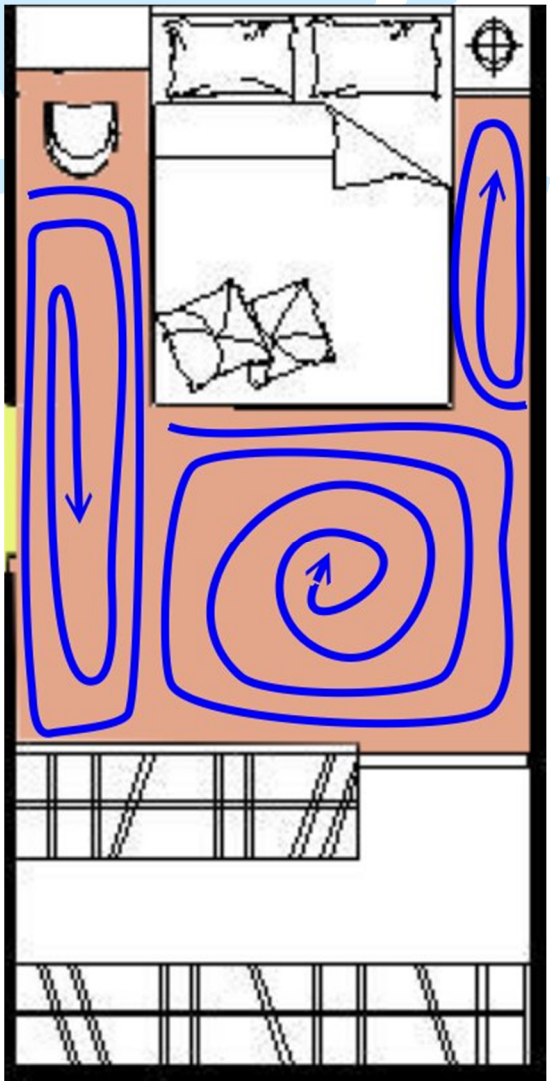


Figure-17: More complex coverage

In situation where there are large obstacle in a room, such as the bed in a bedroom, as shown in figure-17, it's a bit more complex to fully cover the room. The robot needs to adopt an algorithm to separate the room into different zones and cover each of these zones separately.

7. Summary

By combining different type of sensors and control mechanism, a robotics rover can be adopt to serve many useful purpose. The only limit is our imagination. In the later chapters, we will talk about other implementation tips to help accomplish other useful tasks.