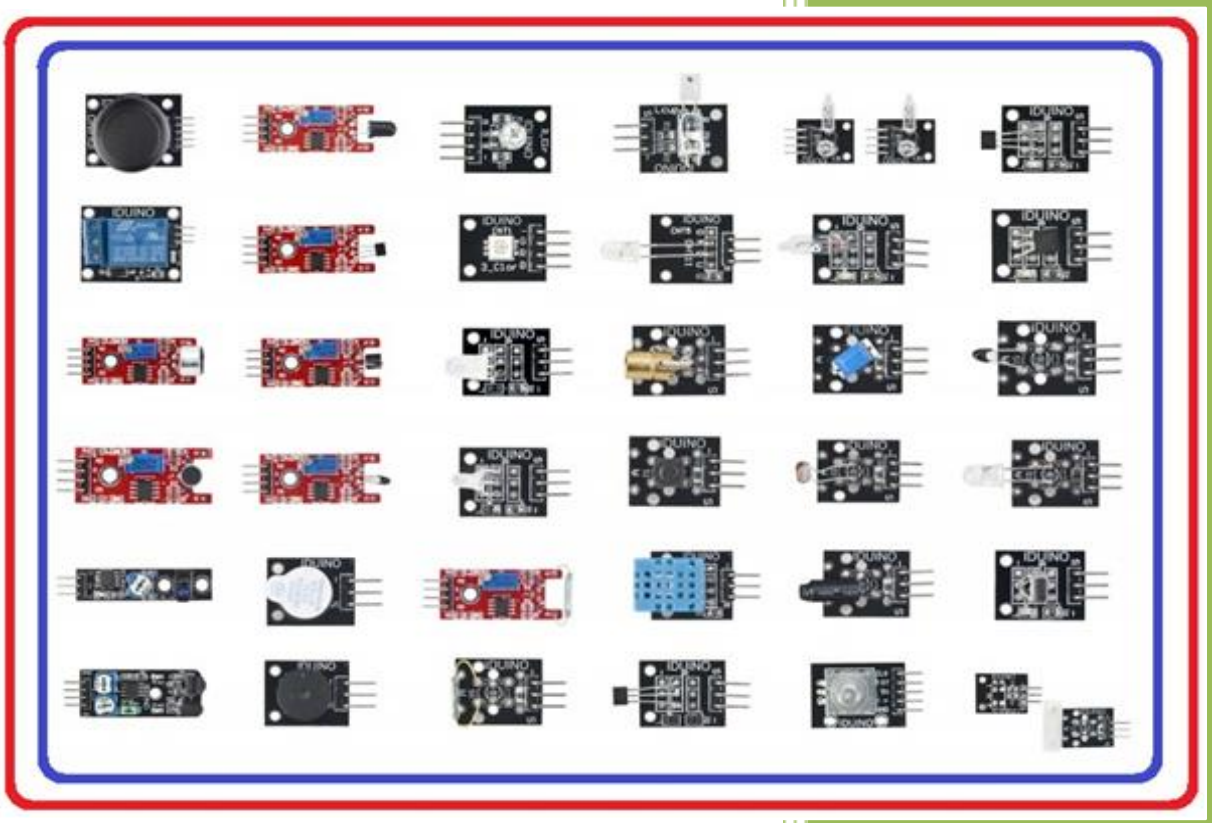
Introduction to 37 sensors



INDEX

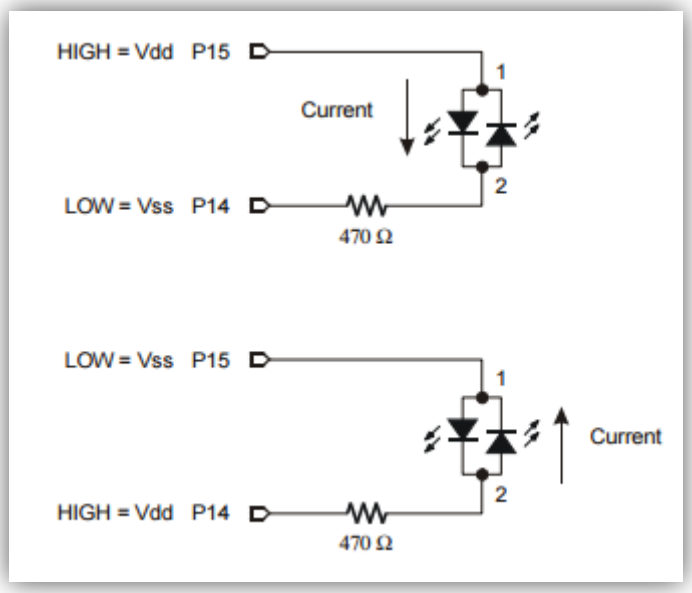
Two Color LED

- A **light-emitting diode** (**LED**) is a two-lead semiconductor light source. It is a pn-junction diode, which emits light when activated. When a suitable voltage is applied to the leads, electrons are able to recombine with electron holes within the device, releasing energy in the form of photons.
- A dual-color light emitting diode (LED) is capable of emitting two different colors of light, typically red and green, rather than only one color.

Working

Figure below shows how you can use P15 and P14 to control the current flow in the bicolor LED circuit. The upper schematic shows how current flows through the red LED when P15 is set to Vdd and P14 is set to Vss. This is because the red LED will let current flow through it when electrical pressure is applied as shown, but the green LED acts like a closed valve and does not let current through it.
The bi-color LED glows red. The lower schematic shows what happens when P15 is set to Vss and P14 is set to Vdd. The electrical pressure is now reversed. The red LED shuts off and does not allow current

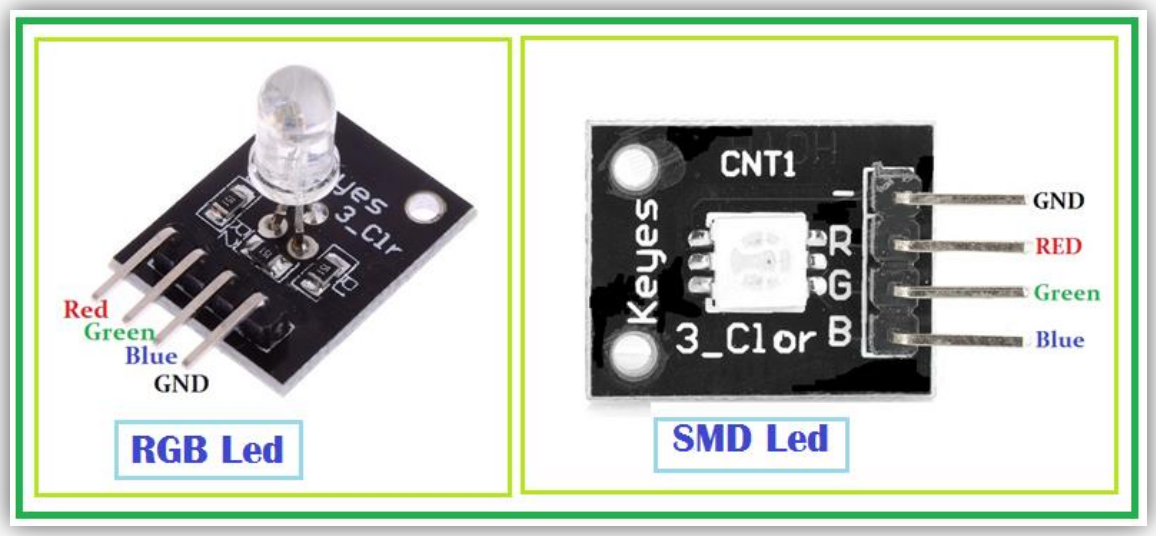
through. Meanwhile, the green LED turns on, and current passes through the circuit in the opposite direction.



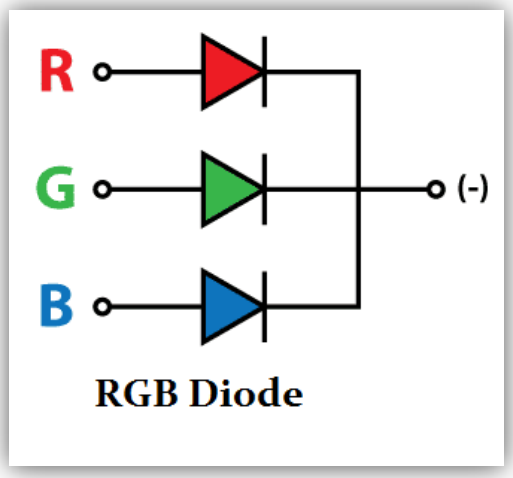
◆ PROGRAM

```
// Arduino test code for KY011
int redpin = 11; // select the pin for the red LED
int greenpin = 10; // select the pin for the green LED
int val;
void setup () {
  pinMode (redpin, OUTPUT);
  pinMode (greenpin, OUTPUT);
}
void loop () {
  for (val = 255; val> 0; val--)
    {
    analogWrite (greenpin, val);
    analogWrite (redpin, 255-val);
    delay (15);
  }
  for (val = 0; val <255; val++)
    {
    analogWrite (greenpin, val);
    analogWrite (redpin, 255-val);
    delay (15);
  }
}
```

RGB LED/SMD RGB



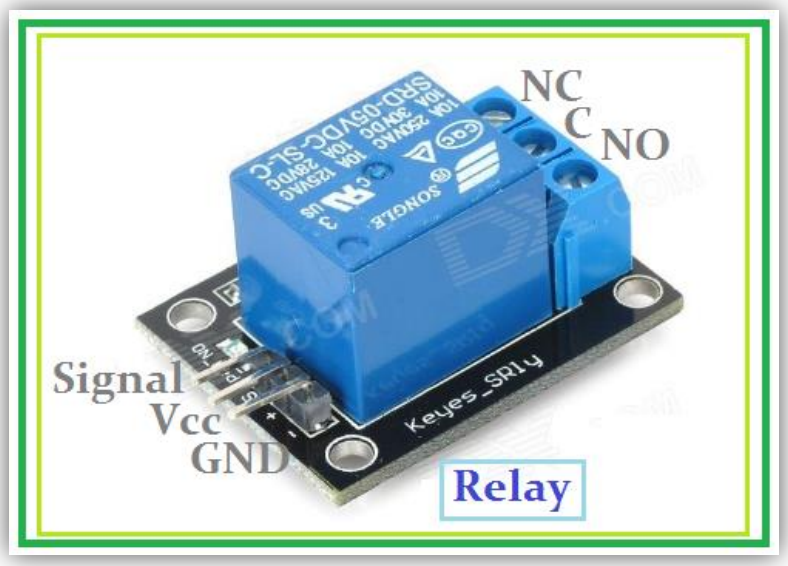
- The RGB LED can emit different colors by mixing the 3 basic colors red, green and blue.
- So it actually consists of 3 separate LEDs red, green and blue packed in a single case.
- That's why it has 4 leads, one lead for each of the 3 colors and one common cathode or anode depending of the RGB LED type.
- SMD RGB is used as a flash light in mobile phones.



- **PROGRAM**

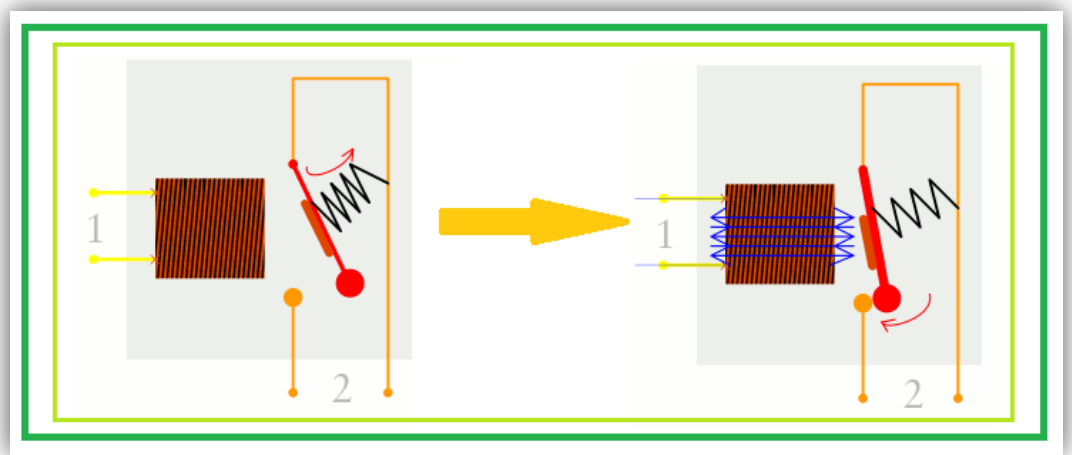
```
//KY016 3-color LED module
int redpin = 11; // select the pin for the red LED
int bluepin = 10; // select the pin for the blue LED
int greenpin = 9 ;// select the pin for the green LED
int val;
void setup () {
  pinMode (redpin, OUTPUT);
  pinMode (bluepin, OUTPUT);
  pinMode (greenpin, OUTPUT);
  Serial.begin (9600);
}
void loop ()
{
  for (val = 255; val> 0; val --)
  {
    analogWrite (11, val);
    analogWrite (10, 255-val);
    analogWrite (9, 128-val);
    delay (10);
    Serial.println (val, DEC);
  }
  for (val = 0; val <255; val ++)
  {
    analogWrite (11, val);
    analogWrite (10, 255-val);
    analogWrite (9, 128-val);
    delay (10);
    Serial.println (val, DEC);
  }
}
```

Relay

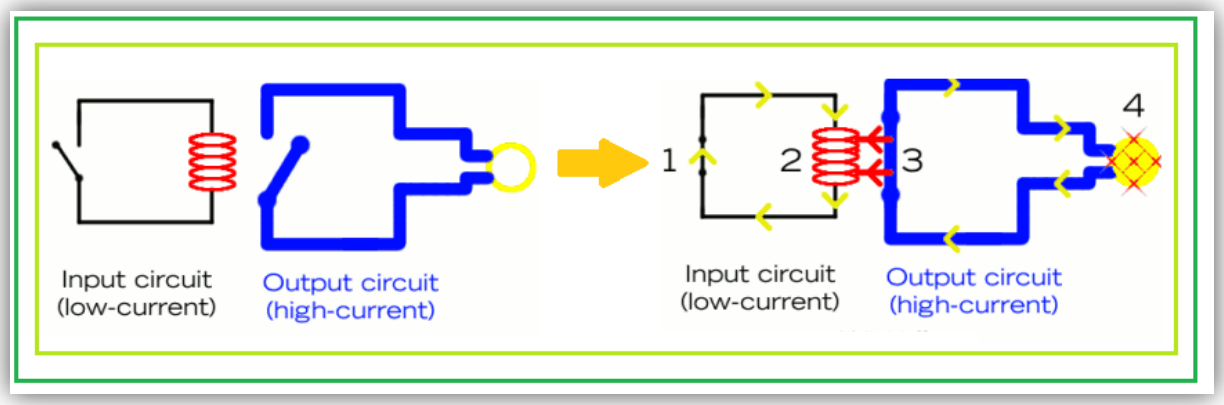


- A relay is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current. The heart of a relay is an electromagnet (a coil of wire that becomes a temporary magnet when electricity flows through it). You can think of a relay as a kind of electric lever: switch it on with a tiny current and it switches on ("leverages") another appliance using a much bigger current. Why is that useful? As the name suggests, many sensors are incredibly *sensitive* pieces of electronic equipment and produce only small electric currents. But often we need them to drive bigger pieces of apparatus that use bigger currents. Relays bridge the gap, making it possible for small currents to activate larger ones. That means relays can work either as switches (turning things on and off) or as amplifiers (converting small currents into larger ones).

Working



- In the above figure, When power flows through the first circuit (1), it activates the electromagnet (brown), generating a magnetic field (blue) that attracts a contact (red) and activates the second circuit (2). When the power is switched off, a spring pulls the contact back up to its original position, switching the second circuit off again.
- This is an example of a "normally open" (NO) relay: the contacts in the second circuit are not connected by default, and switch on only when a current flows through the magnet. Other relays are "normally closed" (NC; the contacts are connected so a current flows through them by default) and switch off only when the magnet is activated, pulling or pushing the contacts apart. Normally open relays are the most common.
- Here's another animation showing how a relay links two circuits together. It's essentially the same thing drawn in a slightly different way. On the left side, there's an input circuit powered by a switch or a sensor of some kind. When this circuit is activated, it feeds current to an electromagnet that pulls a metal switch closed and activates the second, output circuit (on the right side). The relatively small current in the input circuit thus activates the larger current in the output circuit:

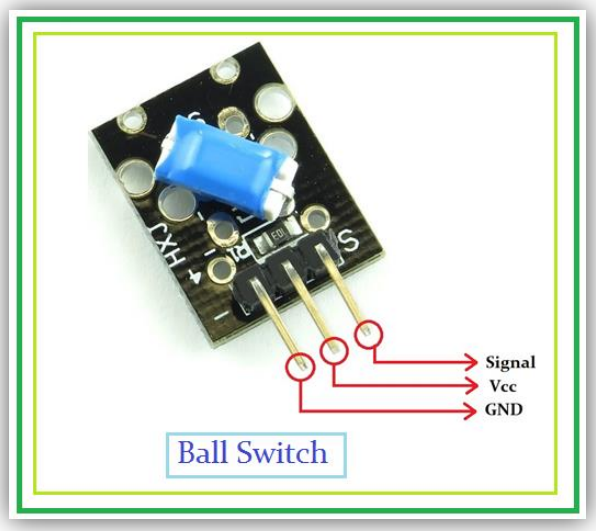


1. The input circuit (black loop) is switched off and no current flows through it until something (either a sensor or a switch closing) turns it on. The output circuit (blue loop) is also switched off.
2. When a small current flows in the input circuit, it activates the electromagnet (shown here as a red coil), which produces a magnetic field all around it.
3. The energized electromagnet pulls the metal bar in the output circuit toward it, closing the switch and allowing a much bigger current to flow through the output circuit.
4. The output circuit operates a high-current appliance such as a lamp or an electric motor.

◆ PROGRAM

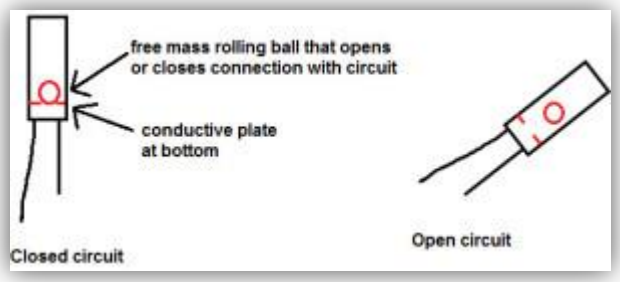
```
//KY019 5V relay module
int relay = 10; // relay turns trigger signal - active high;
void setup ()
{
  pinMode (relay, OUTPUT); // Define port attribute is output;
}
void loop ()
{
  digitalWrite (relay, HIGH); // relay conduction;
  delay (1000);
  digitalWrite (relay, LOW); // relay switch is turned off;
  delay (1000);
}
```

Ball Switch



- Tilt sensors are devices that produce an electrical signal that varies with an angular movement. These sensors are used to measure slope and tilt within a limited range of motion. Sometimes, the tilt sensors are referred to as inclinometers because the sensors just generate a signal but inclinometers generate both readout and a signal.

◆ **Working principle:**

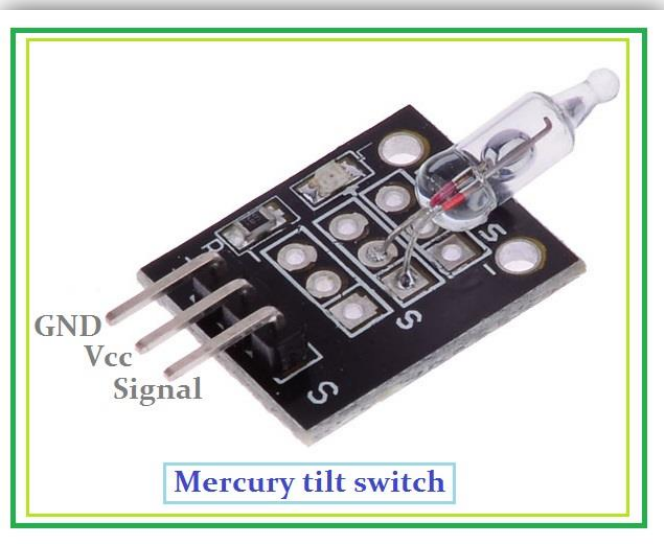


- These sensors consist of a rolling ball with a conductive plate beneath them. When the sensor gets power, the rolling ball falls to the bottom of the sensor to form an electrical connection. When the sensor is tilted, the rolling ball doesn't fall to the bottom so that the current cannot flow the two end terminals of the sensor.
- Because the sensor is very basic, it can only detect large changes when its tilt, and cannot measure the angle of its tilt.

- **PROGRAM**

```
int Led = 13 ;// define LED Interface
int buttonpin = 3; // define the tilt switch sensor interfaces
int val ;// define numeric variables val
void setup ()
{
  pinMode (Led, OUTPUT) ;// define LED as output interface
    pinMode (buttonpin, INPUT) ;//define the output interface tilt switch
sensor
}
void loop ()
{
  val = digitalRead (buttonpin) ;// digital interface will be assigned a value of 3
to read val
    if (val == HIGH) //When the tilt sensor detects a signal when the switch,
LED flashes
  {
    digitalWrite (Led, HIGH);
  }
  else
  {
    digitalWrite (Led, LOW);
  }
}
```

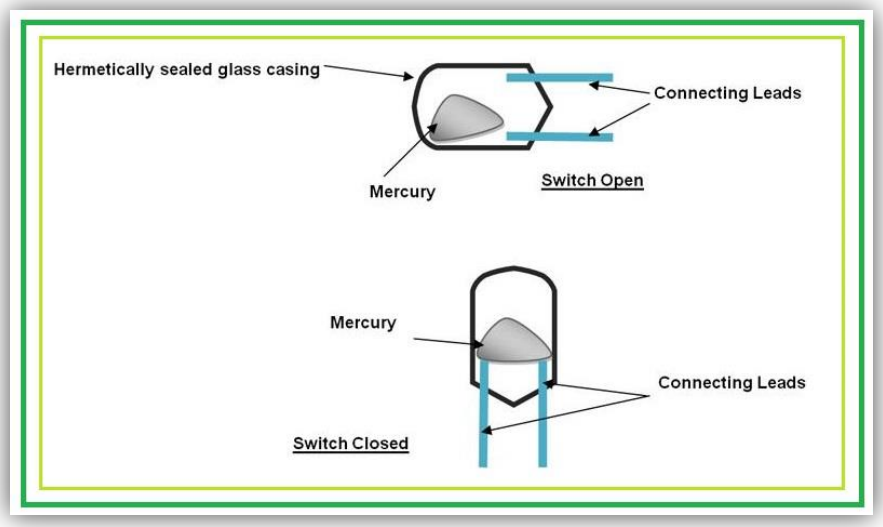
Mercury Tilt Switch



- A **Mercury switch** is an electrical switch that opens and closes a circuit when a small amount of the liquid metal mercury connects metal electrodes to close the circuit.

Working

- These switches employ a mercury bead which connects its terminals whenever it is tilted. One of the earliest types, the response of these tilt switches is not quick. Mercury tilt switches can be found in SPST and SPDT form depending upon the number of contacts used in them.



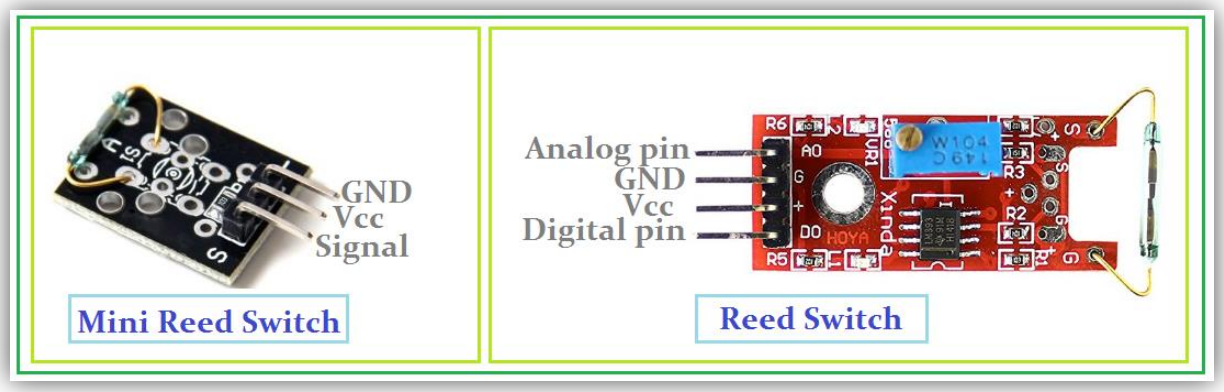
- As seen from the images above, mercury, being a liquid metal can flow down and establish contact between the leads of the switch.

The blob of mercury is able to provide resistance to vibrations as mercury is a dense liquid metal. Using mercury is discouraged as it is a toxic metal and poses a potential hazard to the user when the glass casing breaks and metal spillage take place.

- # PROGRAM

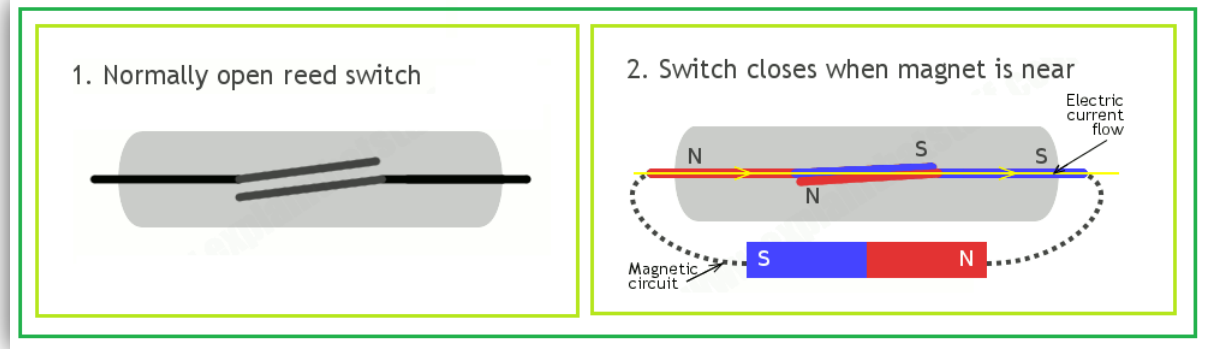
```
//KY017 Mercury open optical module
int Led = 13 ;// define LED Interface
int buttonpin = 3; // define the mercury tilt switch sensor interface
int val ;// define numeric variables val
void setup ()
{
  pinMode (Led, OUTPUT) ;// define LED as output interface
  pinMode (buttonpin, INPUT) ;// define the mercury tilt switch sensor output
interface
}
void loop ()
{
  val = digitalRead (buttonpin) ;// read the values assigned to the digital interface 3
val
  if (val == HIGH) // When the mercury tilt switch sensor detects a signal, LED
flashes
  {
    digitalWrite (Led, HIGH);
  }
  else
  {
    digitalWrite (Led, LOW);
  }
}
```

Reed Switch/Mini Reed Switch



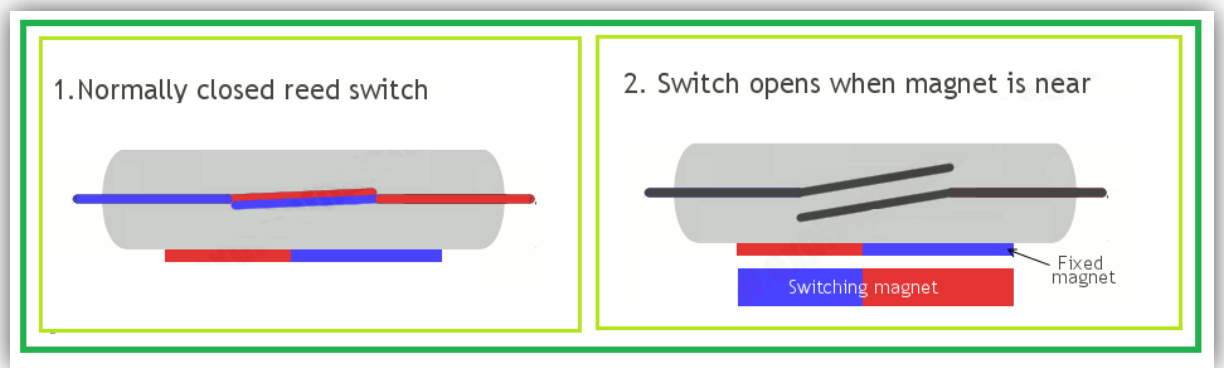
- Reed switches are magnetically-actuated electrical switches (not magically-actuated, though it seems that way sometimes). When the body of the switch is exposed to a magnetic field – like a magnet or even a strong electrical current – two ferrous materials inside pull together, the connection closes, and current can flow. In absence of a magnetic field, the switch opens and the flow of current in the circuit stops.
- Mini reed switch work is the smaller version of reed switch.
- Working
  - Reed switches come in two main varieties called normally open (normally switched off) and normally closed (normally switched on). The key to understanding how they work is to realize that they don't just work as an electrical bridge but as a *magnetic* one as well: magnetism flows through them as well as electricity.
  - **Normally open**
    1. As you bring a magnet up to the reed switch, the entire switch effectively becomes a part of a "magnetic circuit" that includes the magnet (the dotted line in the artwork shows part of the magnetic field). The two contacts of the reed switch become opposite magnetic poles, which is why they attract and snap together. It doesn't matter which end of the magnet approaches first: the contacts still polarize in opposite ways and attract one another. A reed switch like this is **normally open (NO)** (normally off), unless a magnet is positioned right next to it, when it switches on, allowing a current to flow through it.

2. Take the magnet away and the contacts—made from fairly stiff and springy metal—push apart again and return back to their original positions.



◆ Normally closed

1. You can also get reed switches that work the opposite way: the two contacts are normally snapped together and when you bring a magnet up to the switch, spring apart. Reed switches like this are called **normally closed (NC)** (normally switched on), so electricity flows through them most of the time. The easiest way of making one is to take a normally open switch and fix a magnet permanently to its glass case, flipping it over from its open to its closed state (as in the second frame in the normally open animation up above). This entire unit (normally open reed switch with magnet attached) becomes our normally closed reed switch. If you bring a second magnet up to it, with a magnetic field of opposite polarity to that of the first magnet, this new field cancels out the field of the first magnet so we have, in effect, exactly what we had in the first frame of the normally open animation: a reed switch with two contacts sprung apart.



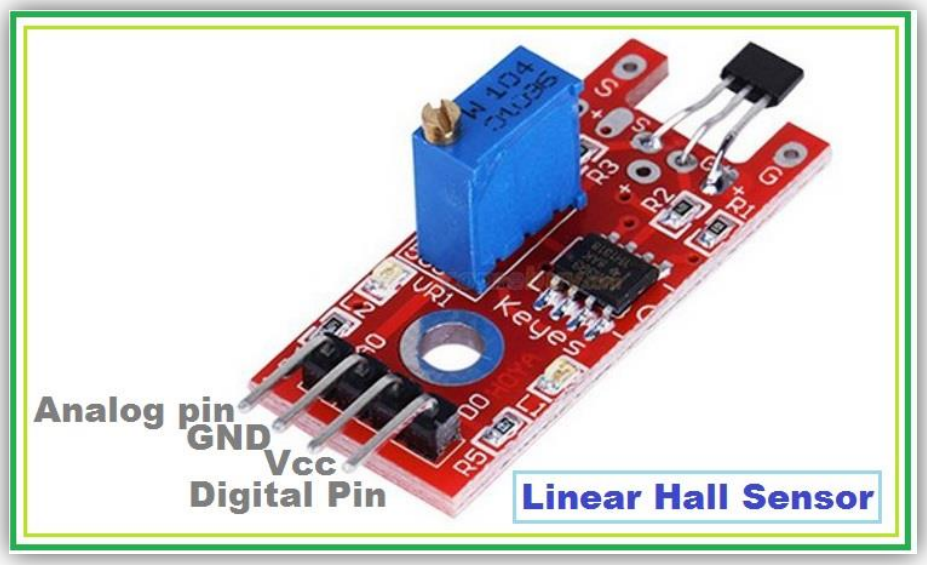
- Another crucial thing I need to point out is that reed switches don't simply switch on when a magnet moves up close and off when it moves away (in the case of a normally open/off switch): they will typically switch on and off several times as the magnet moves by, creating multiple on and off zones. They'll also respond differently according to the orientation of the magnet (whether it's parallel to the switch or perpendicular), what shape it is (because, as we all learned in school, different shaped magnets create different magnetic field patterns all around them), and how it moves past. This is really important when it comes to practical applications: you need to make sure you use the right magnet and that it moves in just the right way to actuate your reed switch. If you're using a reed switch as a counter, for example, it should actuate only once each time the magnet moves by (not three or four times, which would give a false reading). If you're using a reed switch in an alarm, you don't want your intruder switching the alarm on one second and then off again a second later because you put the magnet in the wrong place!

PROGRAM

```
// program for reed switch and mini reed switch
int Led = 13 ; // define LED Interface
int buttonpin = 3; // define the Reed sensor interfaces
int val ; // define numeric variables val
void setup ()
{
 pinMode (Led, OUTPUT) ; // define LED as output interface
```

```
  pinMode (buttonpin, INPUT) ; // output interface as defined Reed
sensor
}
void loop (){

  val = digitalRead (buttonpin) ; // digital interface will be assigned a
value of 3 to read val
  if (val == HIGH) // When the Reed sensor detects a signal, LED
flashes
  {
    digitalWrite (Led, HIGH);
  }
  else
  {
    digitalWrite (Led, LOW);
  }

}
```

Linear Hall Sensor

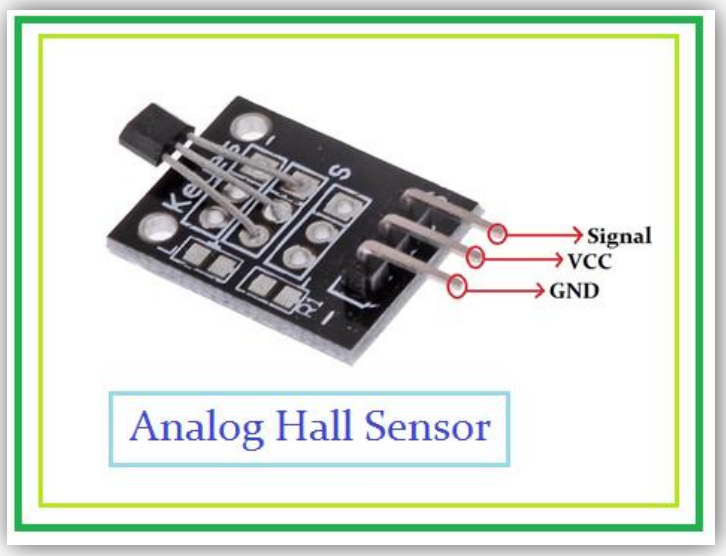


- A linear hall sensor is a digital/analog sensor used to return analog as well as digital reading by working on the same principle as the magnetic hall sensor.
- This sensor is used to detect magnetic fields. The current, and thus the voltage, running through the sensor is changed in relation to the presence of a magnetic field. This sensor is designed so it can tell you the amount the voltage has changed as an analog output, and if the voltage has changed as a digital output. Since the sensor can provide digital and analog outputs, there is a surprising amount you can do with it. You can easily make a working sensor, or make a much more complicated one. This sensor is built in such a way that the sensor itself has everything needed to operate, including an LED. The as such, you can get a basic magnetic switch with nothing more than a 5v battery by connecting the magnet ports to the + and G pins. When a magnet is brought close to the sensor, the voltage will change and send a signal to the sensors integrated LED, telling it to light up. This is as simple as this sensor comes. We would like to do something a bit more complicated though.

- # PROGRAM

```
int Led = 13 ; // define LED Interface
int buttonpin = 3; // define the linear Hall magnetic sensor interface
int val ; // define numeric variables val
void setup ()
{
pinMode (Led, OUTPUT) ; // define LED as output interface
pinMode (buttonpin, INPUT) ; // define linear Hall magnetic sensor output
interface
}
void loop ()
{
val = digitalRead (buttonpin) ; // digital interface will be assigned a value of 3 to
read val
if (val == HIGH) // When the linear Hall sensor detects a magnetic signal, LED
flashes
{
digitalWrite (Led, HIGH);
}
else
{
digitalWrite (Led, LOW);
}
}
```

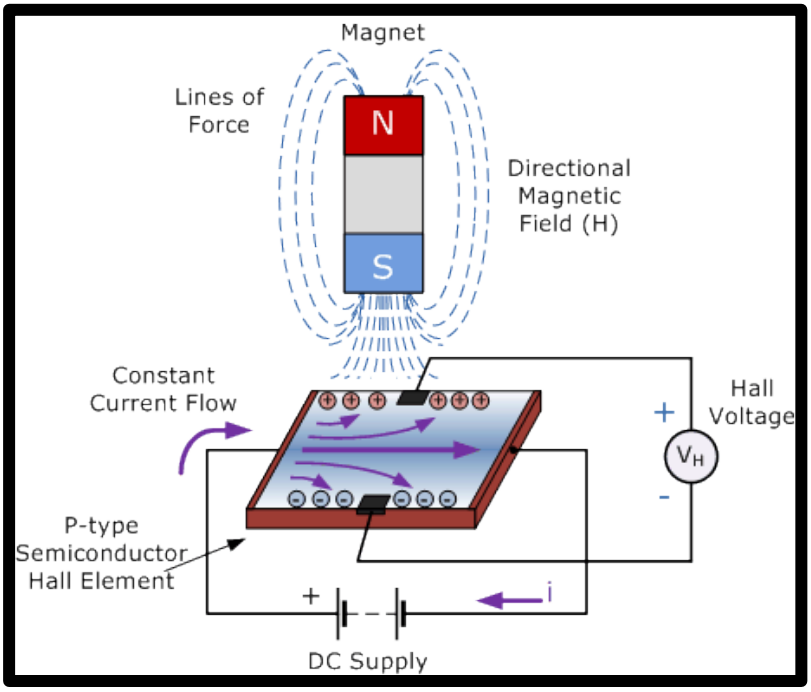
A314 Analog Hall Sensor



- A Magnetic hall sensor is an analog sensor which is used to detect magnetic field and return analog values only.
- Magnetic sensors convert magnetic or magnetically encoded information into electrical signals for processing by electronic circuits.
- Magnetic sensors are solid state devices that are becoming more and more popular because they can be used in many different types of application such as sensing position, velocity or directional movement. They are also a popular choice of sensor for the electronics designer due to their non-contact wear free operation, their low maintenance, robust design and as sealed Hall Effect devices are immune to vibration, dust and water.
- One of the main uses of magnetic sensors is in automotive systems for the sensing of position, distance and speed. For example, the angular position of the crank shaft for the firing angle of the spark plugs, the position of the car seats and seat belts for air-bag control or wheel speed detection for the anti-lock braking system, (ABS).
- Magnetic sensors are designed to respond to a wide range of positive and negative magnetic fields in a variety of different applications and one type of magnet sensor whose output signal is a

function of magnetic field density around it is called the **Hall Effect Sensor**.

- **Hall Effect Sensors** are devices which are activated by an external magnetic field. We know that a magnetic field has two important characteristics flux density, (B) and polarity (North and South Poles). The output signal from a Hall Effect sensor is the function of magnetic field density around the device. When the magnetic flux density around the sensor exceeds a certain pre-set threshold, the sensor detects it and generates an output voltage called the **Hall Voltage, V_H**.



- **Hall Effect Sensors** consist basically of a thin piece of rectangular p-type semiconductor material such as gallium arsenide (GaAs), indium antimonide (InSb) or indium arsenide (InAs) passing a continuous current through itself. When the device is placed within a magnetic field, the magnetic flux lines exert a force on the semiconductor material which deflects the charge carriers, electrons and holes, to either side of the semiconductor slab. This movement of charge carriers is a result of the magnetic force they experience passing through the semiconductor material.

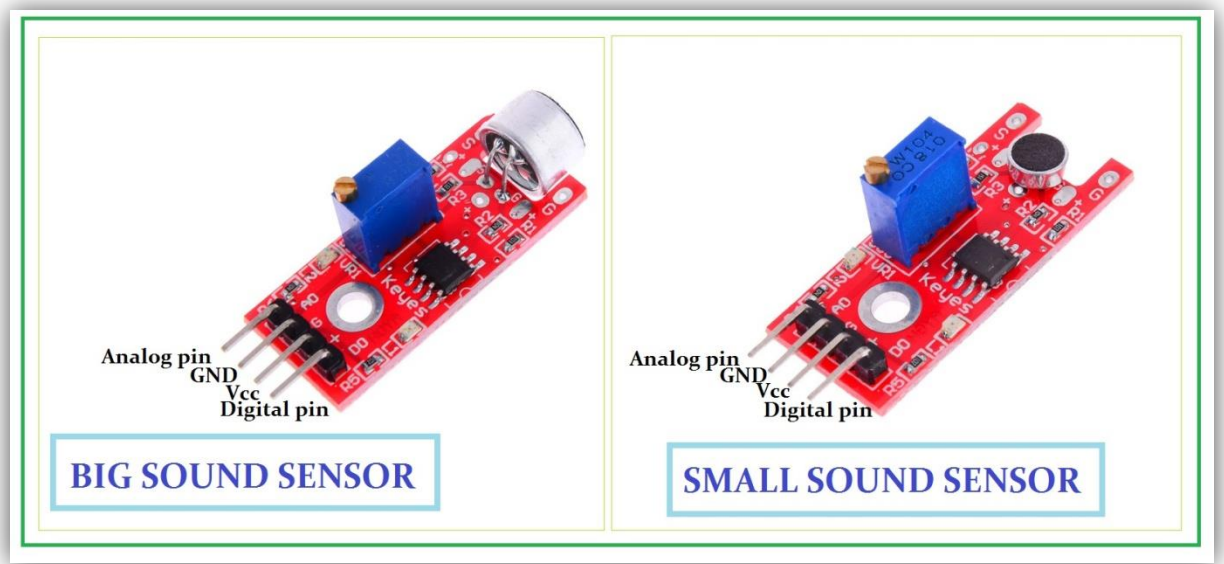
- As these electrons and holes move side wards a potential difference is produced between the two sides of the semiconductor material by the build-up of these charge carriers. Then the movement of electrons through the semiconductor material is affected by the

presence of an external magnetic field which is at right angles to it and this effect is greater in a flat rectangular shaped material.

- The effect of generating a measurable voltage by using a magnetic field is called the **Hall Effect** after Edwin Hall who discovered it back in the 1870's with the basic physical principle underlying the Hall Effect being Lorentz force. To generate a potential difference across the device the magnetic flux lines must be perpendicular, (90^0) to the flow of current and be of the correct polarity, generally a south pole.
- The Hall Effect provides information regarding the type of magnetic pole and magnitude of the magnetic field. For example, a south pole would cause the device to produce a voltage output while a north pole would have no effect. Generally, Hall Effect sensors and switches are designed to be in the "OFF", (open circuit condition) when there is no magnetic field present. They only turn "ON", (closed circuit condition) when subjected to a magnetic field of sufficient strength and polarity.
- **PROGRAM**

```
//KY-035 Hall analog sensor
int sensorPin = A5;    // select the input pin
int ledPin = 13;       // select the pin for the LED
int sensorValue = 0;   // variable to store the value coming from the sensor
void setup () {
  pinMode (ledPin, OUTPUT);
  Serial.begin (9600);
}
void loop () {
  sensorValue = analogRead (sensorPin);
  digitalWrite (ledPin, HIGH);
  delay (sensorValue);
  digitalWrite (ledPin, LOW);
  delay (sensorValue);
  Serial.println (sensorValue, DEC);
}
```

Big sound/Small sound Sensor



- Sound Sensor is a small board with a microphone which allows you to detect the sound in an environment.
- Sound sensors work by mimicking the human body process that involves the ears and signal transmission to the brain. Microphones are sound sensors that convert a sound signal into a voltage or current proportional to the detected signal. They typically have a small diaphragm made of magnets surrounded by coiled metal wire. Sound waves cause the diaphragm to vibrate, which vibrates the magnets and induces a current in the coil.
- **Uses**:
  - You could detect whether or not a motor is running.
  - You could set a threshold on pump sound so that you know whether or not there is cavitation.
  - In the presence of no sound, you might want to create an ambiance by turning on music.
  - In the presence of no sound and no motion, you may go into an energy savings mode and turn off the lights.
- Big sound sensor can be used where we need to detect sound and the area of detection is large. For e.g.: detecting sound in the environment, in classroom. Small sound sensor can be used where the area of detection is small. For e.g.: detecting sound inside the motor.

- **PROGRAM**

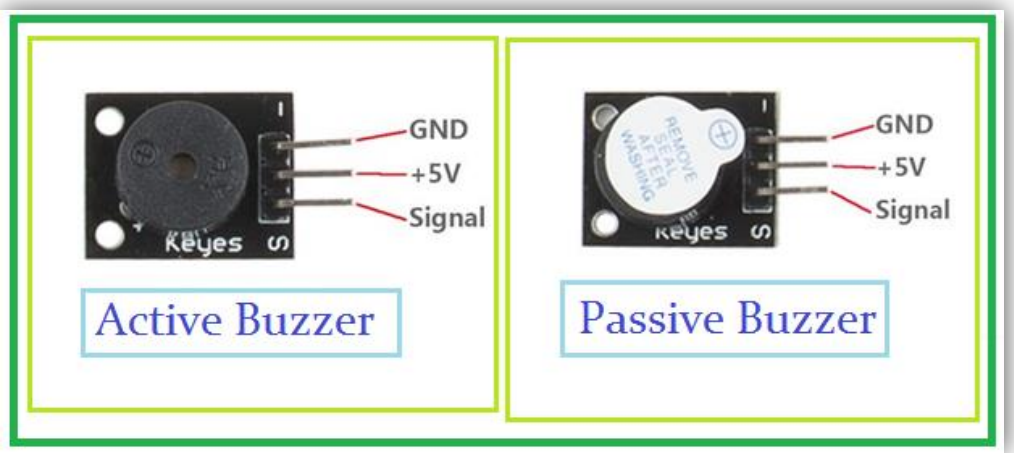
Sound-digital

```
int Led = 13 ;// define LED Interface
int buttonpin = 3; // define Do Sensor Interface
int val = 0;// define numeric variables val
void setup ()
{
  pinMode (Led, OUTPUT) ;// define LED as output interface
  pinMode (buttonpin, INPUT) ;// output interface Do is defined sensor
}
void loop ()
{
  val = digitalRead(buttonpin);// digital interface will be assigned a value of pin 3 to
read val
  if (val == HIGH) // When the sound detection module detects a signal, LED flashes
  {
    digitalWrite (Led, HIGH);
  }
  else
  {
    digitalWrite (Led, LOW);
  }
}
```

Sound-analog

```
int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor
void setup ()
{
  pinMode (ledPin, OUTPUT);
  Serial.begin (9600);
}

void loop ()
{
  sensorValue = analogRead (sensorPin);
  digitalWrite (ledPin, HIGH);
  delay (sensorValue);
  digitalWrite (ledPin, LOW);
  delay (sensorValue);
  Serial.println (sensorValue, DEC);
}
```

Buzzer



- A **buzzer** or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric.
- Typical uses of **buzzers** and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke.
- An **active buzzer** will generate a tone using an internal oscillator, so all that is needed is a DC voltage. The oscillator will turn the input voltage off and on very fast, producing a tone. The pitch can be varied with PWM.
- A **passive buzzer** requires an AC signal to make a sound. It is like an electromagnetic speaker, where a changing input signal produces the sound, rather than producing a tone automatically. This type of buzzer has no oscillator, and if you apply a DC current it will simply make a click sound as the diaphragm moves to its limit and stays there. If you apply a PWM signal it will make a sound, because PWM turns the supply on and off in a similar way to an oscillator.

- **PROGRAM**

Passive Buzzer

Example1.

```
//Example code KY012 active buzzer
#define TONE 20
int speakerPin = 8;
void setup () {
  pinMode (speakerPin, OUTPUT);
}
void loop () {
  //analogWrite(speakerPin, 255);
  digitalWrite (speakerPin, 1);
  delay (TONE);
  digitalWrite (speakerPin, 0);
  //analogWrite(speakerPin, 0);
  delay (TONE);
}
```

Example2. (Music)

```
int buzzer = 8 ;// setting controls the digital IO foot buzzer
void setup ()
{
  pinMode (buzzer, OUTPUT) ;// set the digital IO pin mode, OUTPUT out of Wen
}
void loop ()
{
  unsigned char i, j ;// define variables
  while (1)
  {
    for (i = 0; i <80; i++) // Wen a frequency sound
    {
      digitalWrite (buzzer, HIGH) ;// send voice
      delay (1) ;// Delay 1ms
      digitalWrite (buzzer, LOW) ;// do not send voice
      delay (1) ;// delay ms
    }
    for (i = 0; i <100; i++) // Wen Qie out another frequency sound
    {
      digitalWrite (buzzer, HIGH) ;// send voice
      delay (2) ;// delay 2ms
      digitalWrite (buzzer, LOW) ;// do not send voice
```

```
      delay (2) ;// delay 2ms
    }
  }
}
```

Example3.

```
//Specify digital pin on the Arduino that the positive lead of piezo buzzer is attached.
int piezoPin = 9;

void setup() {

}//close setup

void loop() {

  /*Tone needs 2 arguments, but can take three
    1) Pin#
    2) Frequency - this is in hertz (cycles per second) which determines the pitch of
the noise made
    3) Duration - how long teh tone plays
  */
 // tone(piezoPin, 1000, 500);
   tone(piezoPin, 2000, 500);

  delay(1000);

  //tone(piezoPin, 1000, 500);
  //delay(1000);

}
```

Example4.

```
/*
 Piezo

 This example shows how to run a Piezo Buzzer on pin 9
 using the analogWrite() function.

 It beeps 3 times fast at startup, waits a second then beeps continuously
 at a slower pace

 */

void setup()  {
  // declare pin 9 to be an output:
```

```
  pinMode(9, OUTPUT);
  beep(50);
  beep(50);
  beep(50);
  delay(1000);
}

void loop()  {
  beep(200);
}
void beep(unsigned char delayms){
  analogWrite(9, 20);      // Almost any value can be used except 0 and 255
                           // experiment to get the best tone

  delay(delayms);          // wait for a delayms ms
  analogWrite(9, 0);       // 0 turns it off
  delay(delayms);          // wait for a delayms ms
}
```

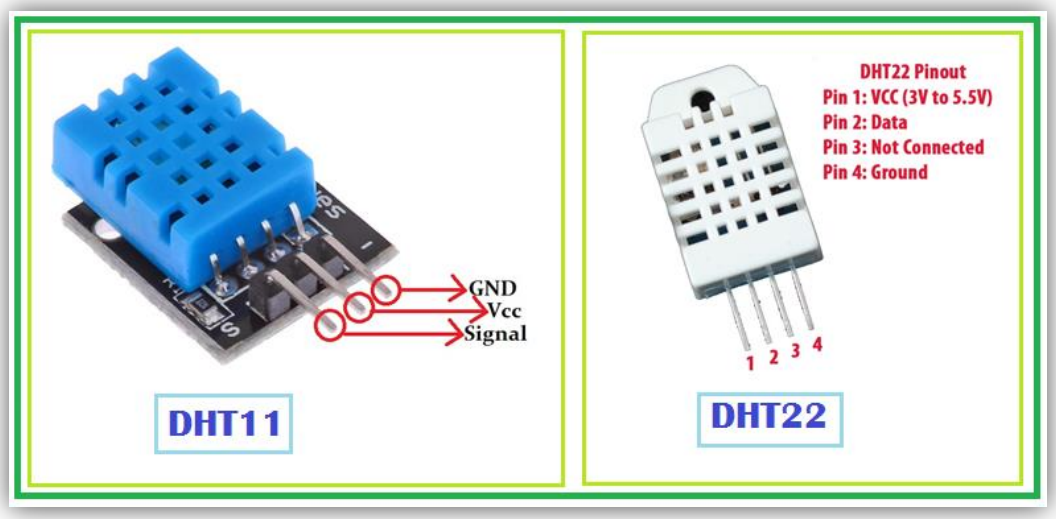
For more details click on this link:
https://programmingelectronics.com/an-easy-way-to-make-noise-with-arduino-using-tone/

Active Buzzer

Example1.

```
int buzzer = 8; // set the buzzer control digital IO pin

void setup() {
  pinMode(buzzer, OUTPUT); // set pin 8 as output
}

void loop() {
   digitalWrite(buzzer, LOW); // send low signal to buzzer
   delay(100);
}
```

Temperature and humidity Sensor



- The often used temperature and humidity sensor are DHT11 and DHT22.

DHT11

- DHT11 digital temperature and humidity sensor is a composite Sensor contains a calibrated digital signal output of the temperature and humidity. Application of a dedicated digital modules collection technology and the temperature and humidity sensing technology, to ensure that the product has high reliability and excellent long-term stability. The sensor includes a resistive sense of wet components and NTC temperature measurement devices, and connected with a high-performance 8-bit microcontroller.
- Applications of DHT11 are HVAC, dehumidifier, testing and inspection equipment, consumer goods, automotive, automatic control, data loggers, weather stations, home appliances, humidity regulator, medical and other humidity measurement and control.
- **Serial communication instructions (single-wire bi-directional)**
  - Single bus Description :
    - DHT11 uses a simplified single-bus communication. Single bus that only one data line, the system of data exchange, control by a single bus to complete. Device (master or slave)

through an open-drain or tri-state port connected to the data line to allow the device does not send data to release the bus, while other devices use the bus; single bus usually require an external one about 5.1kΩ pull-up resistor, so that when the bus is idle, its status is high. Because they are the master-slave structure, and only when the host calls the slave, the slave can answer, the host access devices must strictly follow the single-bus sequence, if the chaotic sequence, the device will not respond to the host.

- Single bus to transfer data defined:
  - DATA For communication and synchronization between the microprocessor and DHT11, single-bus data format, a transmission of 40 data, the high first-out.

  - Data format:
    The 8bit humidity integer data + 8bit the Humidity decimal data +8 bit temperature integer data + 8bit fractional temperature data +8 bit parity bit.
- Parity bit data definition
  - "8bit humidity integer data + 8bit humidity decimal data +8 bit temperature integer data + 8bit temperature fractional data" 8bit checksum is equal to the results of the last eight.

  - Example 1: 40 data is received:

0011 0101	**0000 0000**	**0001 1000**	**0000 0000**	**0100 1101**
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate：
0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101
Received data is correct：
Humidity：0011 0101=35H=53%RH
Temperature：0001 1000=18H=24℃

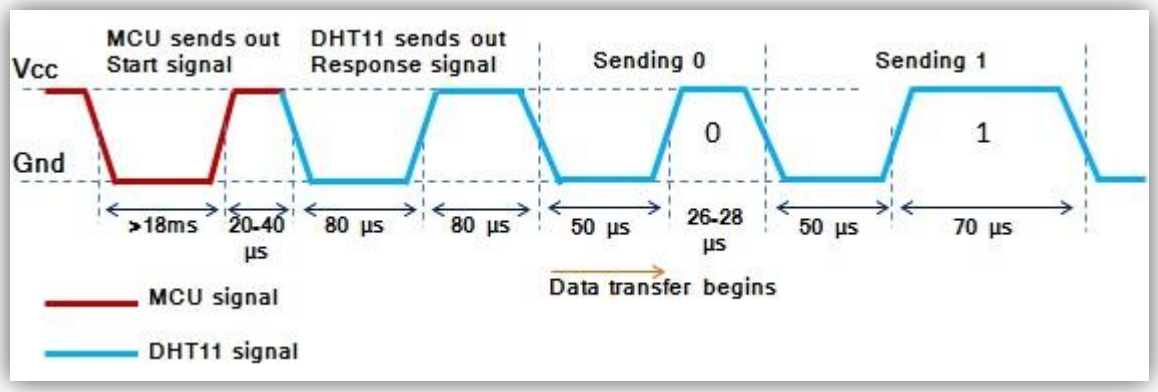
- Example 2: 40 data is received:

0011 0101	**0000 0000**	**0001 1000**	**0000 0000**	**0100 1001**
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate：
0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101
01001101 ≠ 01001001
Received data is not correct, give up, to re receive data.

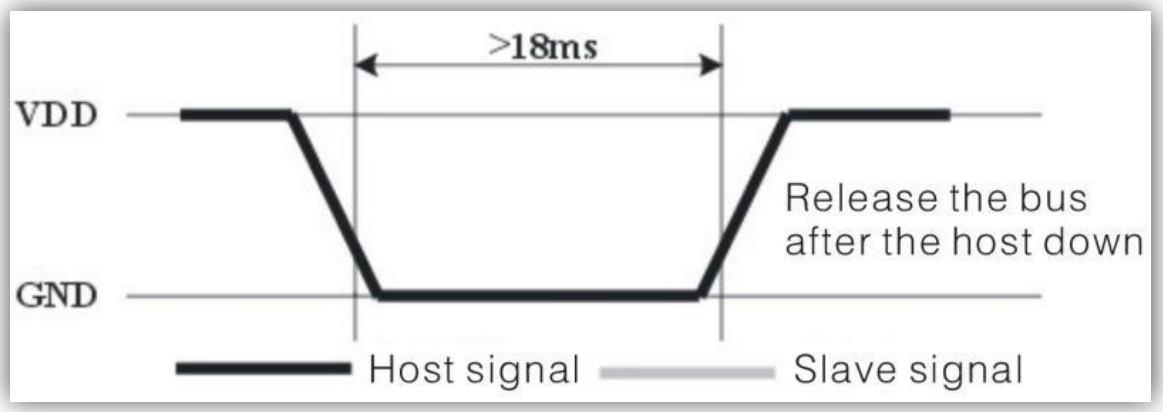
Humidity：0011 0101=35H=53%RH
Temperature：0001 1000=18H=24℃

- Data Timing Diagram
  - User host (MCU) to send a signal, DHT11 converted from low-power mode to high-speed mode, until the host began to signal the end of the DHT11 send a response signal to send 40bit data, and trigger a letter collection. The signal is sent as shown.



- Peripherals read steps
  - Communication between the master and slave can be done through the following steps (peripherals (such as microprocessors) read DHT11 the data of steps).

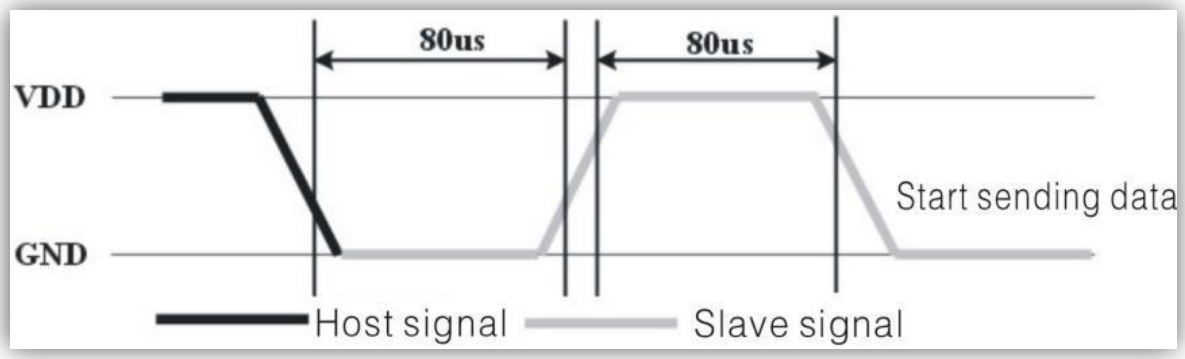
Step 1:
After power on DHT11 (DHT11 on after power to wait 1S across the unstable state during this period cannot send any instruction), the test environment temperature and humidity data, and record the data, while DHT11 the DATA data lines pulled by pull-up resistor has been to maintain high; the DHT11 the DATA pin is in input state, the moment of detection of external signals.
Step 2:
Microprocessor I / O set to output at the same time output low, and low hold time cannot be less than 18ms, then the microprocessor I / O is set to input state, due to the pull-up resistor, a microprocessor/ O DHT11 the DATA data lines also will be high, waiting DHT11 to answer signal, send the signal as shown:



HOST SENDS A START SIGNAL

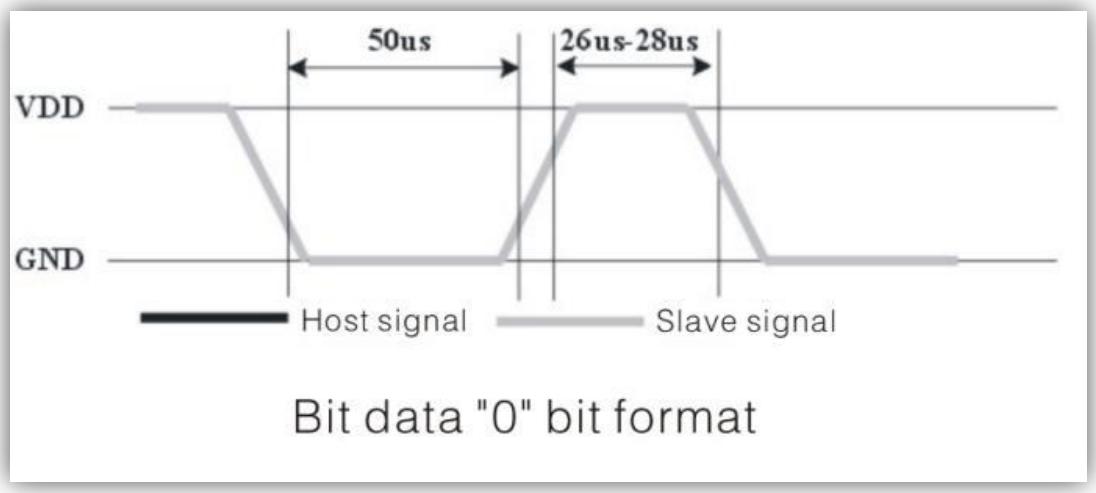
Step 3:

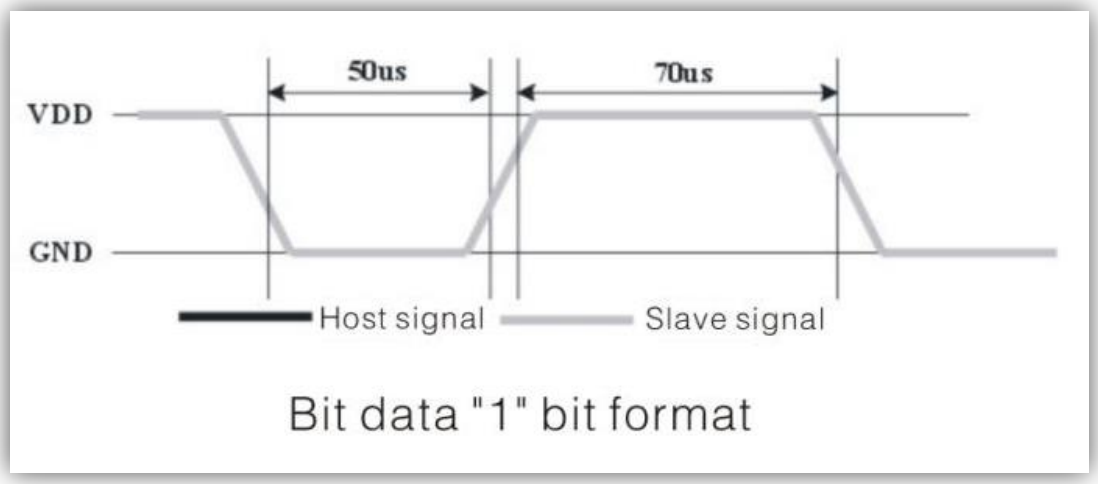
DATA pin is detected to an external signal of DHT11 low, waiting for external signal low end the delay DHT11 DATA pin in the output state, the output low of 80 microseconds as the response signal, followed by the output of 80 micro-seconds of high notification peripheral is ready to receive data, the microprocessor I / O at this time in the input state is detected the I / O low (DHT11 response signal), wait 80 microseconds highdata receiving and sending signals as shown:



Step 4:

Output by DHT11 the DATA pin 40, the microprocessor receives 40 data bits of data "0" format: the low level of 50 microseconds and 26-28 microseconds according to the changes in the I/O level, bit data "1" format: the high level of low plus, 50 microseconds to 70 microseconds. Bit data "0", "1" signal format as shown:





End signal:

- Continue to output the low 50 microseconds after DHT11 the DATA pin output 40 data, and changed the input state, along with pull-up resistor goes high. But DHT11 internal re-test environmental temperature and humidity data, and record the data, waiting for the arrival of the external signal.

Note: For detailed study refer to the link below:
https://akizukidenshi.com/download/ds/aosong/DHT11.pdf

- # DHT22
  - **DHT22** is better than DHT11 because it has a wider range of measurement: 0 to 100% for humidity and -40°C to +125°C for temperature.
  - Also it has a digital output (Single-bus) that provides higher data accuracy.

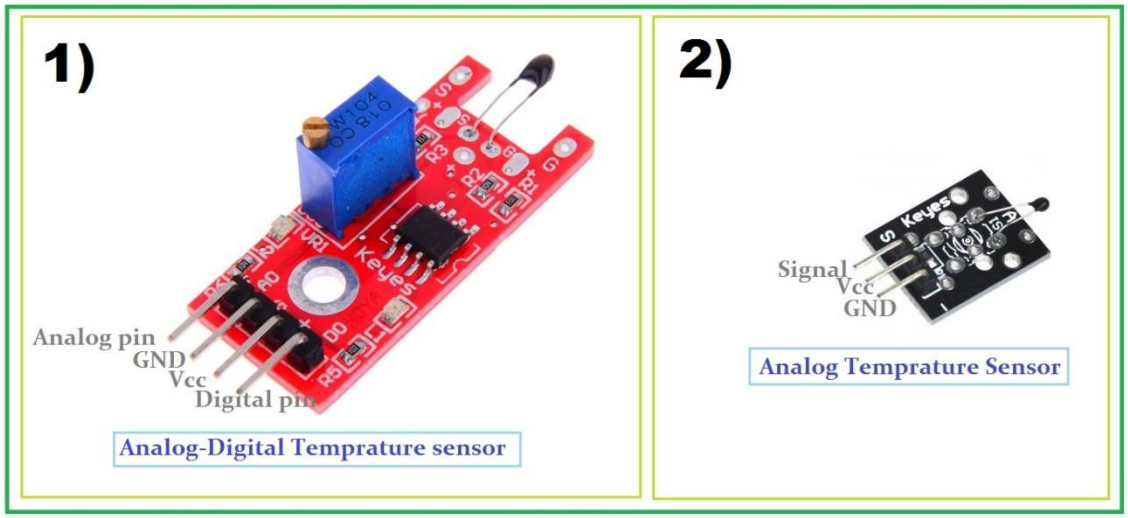
- **PROGRAM:**

```
int read(int pin, byte* ptemperature, byte* phumidity, byte pdata[40]);
int confirm(int pin, int us, byte level);
byte bits2byte(byte data[8]);
int sample(int pin, byte data[40]);
int parse(byte data[40], byte* ptemperature, byte* phumidity);
int pinDHT11 = 2;
void setup(){
Serial.begin(115200);
}
int confirm(int pin, int us, byte level) {
// wait one more count to ensure.
int cnt = us / 10 + 1;
bool ok = false;
for (int i = 0; i < cnt; i++) {
if (digitalRead(pin) != level) {
ok = true;
break;
}
delayMicroseconds(10);
}
if (!ok) {
return -1;
}
return 0;
}
byte bits2byte(byte data[8]) {
byte v = 0;
for (int i = 0; i < 8; i++) {
v += data[i] << (7 - i);
}
return v;
}
int sample(int pin, byte data[40]) {
// empty output data.
memset(data, 0, 40);
// notify DHT11 to start:
//    1. PULL LOW 20ms.
//    2. PULL HIGH 20-40us.
//    3. SET TO INPUT.
pinMode(pin, OUTPUT);
digitalWrite(pin, LOW);
delay(20);
digitalWrite(pin, HIGH);
delayMicroseconds(30);
pinMode(pin, INPUT);
```

```
// DHT11 starting:
//    1. PULL LOW 80us
//    2. PULL HIGH 80us
if (confirm(pin, 80, LOW)) {
return 100;
}
if (confirm(pin, 80, HIGH)) {
return 101;
}
// DHT11 data transmite:
//    1. 1bit start, PULL LOW 50us
//    2. PULL HIGH 26-28us, bit(0)
//    3. PULL HIGH 70us, bit(1)
for (int j = 0; j < 40; j++) {
if (confirm(pin, 50, LOW)) {
return 102;
}
// read a bit, should never call method,
// for the method call use more than 20us,
// so it maybe failed to detect the bito.
bool ok = false;
int tick = 0;
for (int i = 0; i < 8; i++, tick++) {
if (digitalRead(pin) != HIGH) {
ok = true;
break;
}
delayMicroseconds(10);
}
if (!ok) {
return 103;
}
data[j] = (tick > 3? 1:0);
}
// DHT11 EOF:
//    1. PULL LOW 50us.
if (confirm(pin, 50, LOW)) {
return 104;
}
return 0;
}
int parse(byte data[40], byte* ptemperature, byte* phumidity) {
byte humidity = bits2byte(data);
byte humidity2 = bits2byte(data + 8);
byte temperature = bits2byte(data + 16);
byte temperature2 = bits2byte(data + 24);
byte check = bits2byte(data + 32);
```

```
byte expect = humidity + humidity2 + temperature + temperature2;
if (check != expect) {
return 105;
}
*ptemperature = temperature;
*phumidity = humidity;
return 0;
}
int read(int pin, byte* ptemperature, byte* phumidity, byte pdata[40]) {
int ret = 0;
byte data[40] = {0};
if ((ret = sample(pin, data)) != 0) {
return ret;
}
byte temperature = 0;
byte humidity = 0;
if ((ret = parse(data, &temperature, &humidity)) != 0) {
return ret;
}
if (pdata) {
memcpy(pdata, data, 40);
}
if (ptemperature) {
*ptemperature = temperature;
}
if (phumidity) {
*phumidity = humidity;
}
return ret;
}
void loop(){
Serial.println("==================================");
Serial.println("Sample DHT11...");
// read without samples.
byte temperature = 0;
byte humidity = 0;
if (read(pinDHT11, &temperature, &humidity, NULL)) {
Serial.print("Read DHT11 failed.");
return;
}
Serial.print("Sample OK: ");
Serial.print((int)temperature); Serial.print(" *C, ");
Serial.print((int)humidity); Serial.println(" %");
// DHT11 sampling rate is 1HZ.
delay(1000);
}
```

Digital and Analog Temperature Sensor



- ## NTC

- A temperature sensor measures the hotness or coolness of an object. The sensor's working base is the voltage that's read across the diode. The temperature rises whenever the voltage increases. The sensor records any voltage drop between the transistor base and emitter. When the difference in voltage is amplified, the device generates an analogue signal that's proportional to the temperature.
- In fig1) Analog-Digital temperature sensor works as both analog and digital temperature sensor. If the output pin is connected to analog pin, it will work as a analog temperature sensor and if the output pin is connected to the digital pin, it will work as a digital temperature sensor.
- In fig2) Analog temperature sensor will give output as an analog signal.
- **Digital Temperature Sensor**
  - The biggest benefit to using this type of sensor is the accuracy and the ability to add multiple sensors to 1 wire and easily gather a reading from a specific sensor.
  - This temperature sensor gives output as an digital signal.

- **Analog Temperature Sensor**

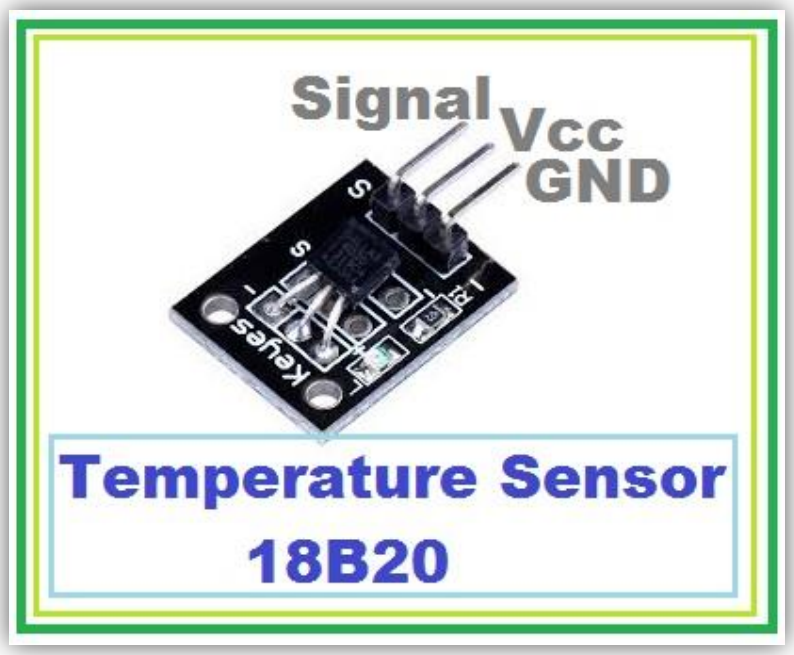
- These sensors are much more accurate and easier to code.
- This temperature sensor gives output as an analog signal.

PROGRAM

```
#include <math.h>

int sensorPin = A5; // select the input pin for the potentiometer

double Thermistor(int RawADC) {
  double Temp;
  Temp = log(10000.0*((1024.0/RawADC-1)));
  Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp ))*
Temp );
  Temp = Temp - 273.15;            // Convert Kelvin to Celcius
   //Temp = (Temp * 9.0)/ 5.0 + 32.0; // Convert Celcius to Fahrenheit
   return Temp;
}

void setup() {
 Serial.begin(9600);
}

void loop() {
 int readVal=analogRead(sensorPin);
 double temp =  Thermistor(readVal);

 Serial.println(temp);  // display tempature
 //Serial.println(readVal);  // display tempature

 delay(500);
}
```

For more details refer to the link below:
http://www.instructables.com/id/Thermistor-Sensor-Module-Interface-With-Arduino/

Temperature Sensor 18b20



- The device uses OneWire protocol.
- **1-Wire protocol : 1-Wire** is a device communications bus system designed by Dallas Semiconductor Corp. that provides low-speed data, signaling, and power over a single conductor. **1-Wire** is similar in concept to I^2C, but with lower data rates and longer range.
- The DS18B20 can be powered by between 3.0V and 5.5V so you can simply connect its GND pin to 0V and the VDD pin to +5V from the Arduino. However, the DS18B20 can also extract its power from the data line which means we only effectively need two wires to connect it up. This makes it great for use as an external sensor.

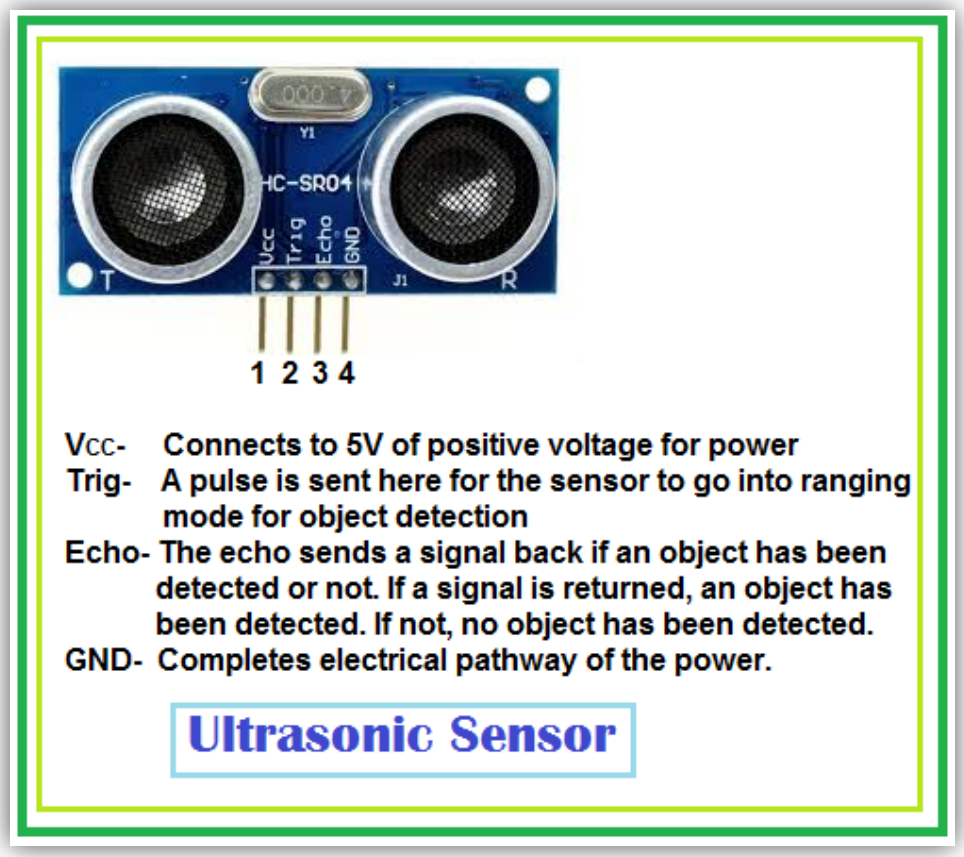
- **PROGRAM**

```
#include <OneWire.h>
 // DS18S20 Temperature chip i/o
OneWire ds(10);  // on pin 10

void setup(void) {
  // initialize inputs/outputs
  // start serial port
  Serial.begin(9600);
}

void loop(void) {

  //For conversion of raw data to C
  int HighByte, LowByte, TReading, SignBit, Tc_100, Whole, Fract;

  byte i;
  byte present = 0;
  byte data[12];
  byte addr[8];

  if ( !ds.search(addr)) {
     Serial.print("No more addresses.\n");
     ds.reset_search();
     return;
  }

  Serial.print("R=");
  for( i = 0; i < 8; i++) {
    Serial.print(addr[i], HEX);
    Serial.print(" ");
  }

  if ( OneWire::crc8( addr, 7) != addr[7]) {
     Serial.print("CRC is not valid!\n");
     return;
  }

  if ( addr[0] == 0x10) {
     Serial.print("Device is a DS18S20 family device.\n");
  }
  else if ( addr[0] == 0x28) {
     Serial.print("Device is a DS18B20 family device.\n");
  }
  else {
     Serial.print("Device family is not recognized: 0x");
```

```
    Serial.println(addr[0],HEX);
    return;
  }

  ds.reset();
  ds.select(addr);
  ds.write(0x44,1);        // start conversion, with parasite power on at the end

  delay(1000);     // maybe 750ms is enough, maybe not
  // we might do a ds.depower() here, but the reset will take care of it.

  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);         // Read Scratchpad

  Serial.print("P=");
  Serial.print(present,HEX);
  Serial.print(" ");
  for ( i = 0; i < 9; i++) {           // we need 9 bytes
   data[i] = ds.read();
   Serial.print(data[i], HEX);
   Serial.print(" ");
  }
  Serial.print(" CRC=");
  Serial.print( OneWire::crc8( data, 8), HEX);
  Serial.println();

  //Conversion of raw data to C
  LowByte = data[0];
  HighByte = data[1];
  TReading = (HighByte << 8) + LowByte;
  SignBit = TReading & 0x8000;  // test most sig bit
  if (SignBit) // negative
  {
   TReading = (TReading ^ 0xffff) + 1; // 2's comp
  }
  Tc_100 = (6 * TReading) + TReading / 4;    // multiply by (100 * 0.0625) or
6.25

  Whole = Tc_100 / 100;  // separate off the whole and fractional portions
  Fract = Tc_100 % 100;


  if (SignBit) // If its negative
  {
    Serial.print("-");
  }
```

```
  Serial.print(Whole);
  Serial.print(".");
  if (Fract < 10)
  {
    Serial.print("0");
  }
  Serial.print(Fract);

  Serial.print("\n");
  //End conversion to C
}
```

Ultrasonic Sensor



- This ultrasonic distance sensor provides steady and **accurate distance measurements** within the range of 2cm to a whooping 450cm. It has a focus of less than 15 degrees and an accuracy of about 2mm.
- The Ultrasonic sound waves has an extremely high pitch that humans cannot hear and is also free from external noises from passive or active sources.
- This particular sensor transmits an ultrasonic sound that has a frequency of about 40 kHz.
- The sensor has two main parts- transducer that creates an **ultrasonic sound wave** while the other part listens to its echo.
- **Ultrasonic distance measurement principle :**
    - Ultrasonic transmitter emitted an ultrasonic wave in one direction, and started timing when it launched. Ultrasonic

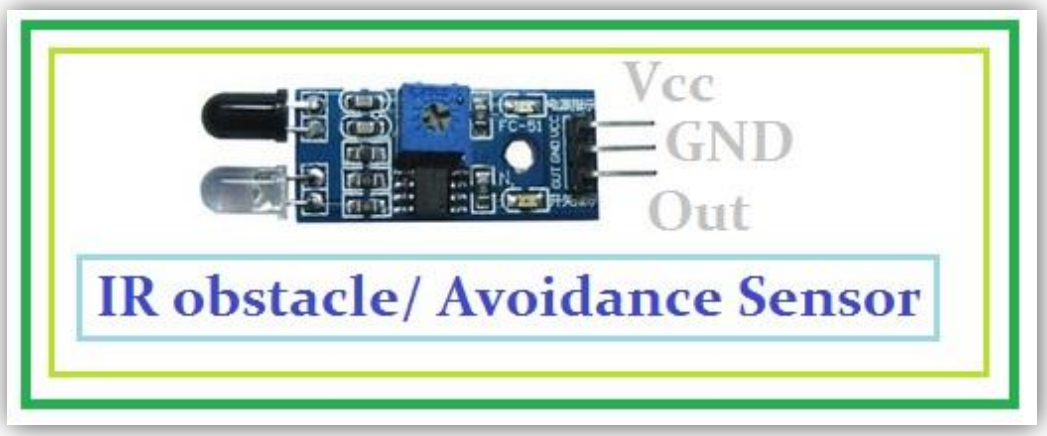
spread in the air, and would return immediately when it encountered obstacles on the way.

- At last, the ultrasonic receiver would stop timing when it received the reflected wave.
- As Ultrasonic spread velocity is 340m / s in the air, based on the timer record t, we can calculate the distance (s) between the obstacle and transmitter, namely: s = 340t / 2, which is so-called time difference distance measurement principle .
- The principle of ultrasonic distance measurement used the already-known air spreading velocity, measuring the time from launch to reflection when it encountered obstacle, and then calculate the distance between the transmitter and the obstacle according to the time and the velocity.
- Thus, the principle of ultrasonic distance measurement is the same with radar. Distance Measurement formula is expressed as: L = C X T In the formula, L is the measured distance, and C is the ultrasonic spreading velocity in air, also, T represents time (T is half the time value from transmitting to receiving).

- **PROGRAM**

```
#include <NewPing.h>
#define TRIGGER_PIN  12
#define ECHO_PIN     11
#define MAX_DISTANCE 300

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
  Serial.begin(115200);
}

void loop() {
  delay(50);
  int uS = sonar.ping();
  Serial.print("Ping: ");
  Serial.print(uS / US_ROUNDTRIP_CM);
  Serial.println("cm");
```

IR Obstacle/Avoidance Sensor



- An IR Obstacle Sensor works in accordance with the infrared reflection principle to detect obstacles. When there is no object, the infrared receiver receives no signals; when there is an object ahead which blocks and reflects the infrared light, the infrared receiver will receive signals.

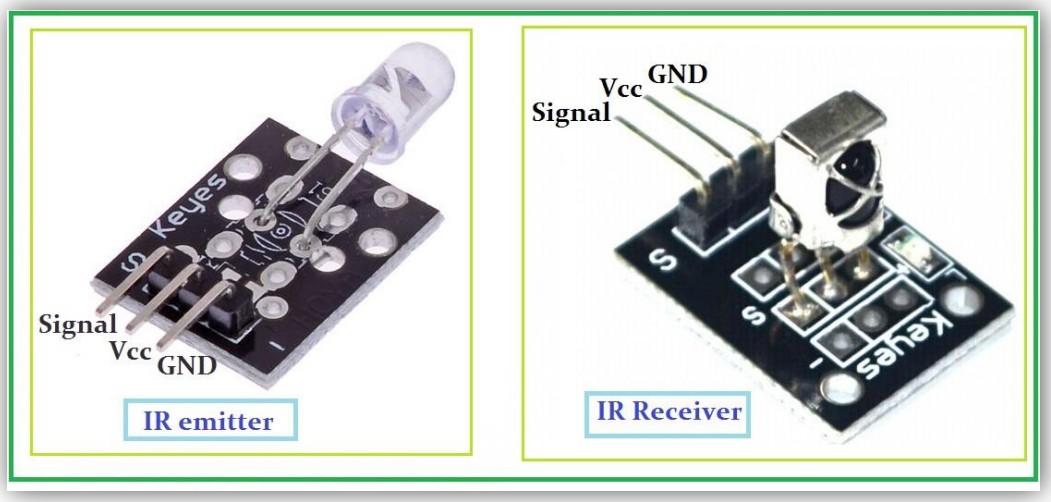
WORKING

- An obstacle avoidance sensor mainly consists of an infrared transmitter, an infrared receiver and a potentiometer.
- According to the reflecting character of an object, if there is no obstacle, the emitted infrared ray will weaken with the distance it spreads and finally disappear. If there is an obstacle, when the infrared ray encounters it, the ray will be reflected back to the infrared receiver.
- Then the infrared receiver detects this signal and confirms an obstacle in front.

- **PROGRAM**

```
void setup() {
  // Connect the wire as following:-
  //GND to GND //Vcc to 5V
  //Out to Ao pin
  //On bringing you hand or any obstacle near the sensor you will find a low
signal
pinMode(0,INPUT);
Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
int sen = analogRead(0);
Serial.println(sen);
 }
```

IR Emitter and IR Receiver



- This is a LED that emits a IR (infrared) Light that can be used to for example control your TV or as a light source for a IR security camera.
- The Infrared Emitter is used to transmit infrared signals through an infrared LED, while there is an **Infrared receiver** to get the signals on the other side.
- **IR Emitter**
  - R LED emits infrared light, means it emits light in the range of Infrared frequency. We cannot see Infrared light through our eyes, they are invisible to human eyes. The wavelength of Infrared (700nm – 1mm) is just beyond the normal visible light. Everything which produces heat and emits infrared like our human body. Infrared have the same properties as visible light, like it can be focused, reflected and polarized like visible light.
  - Other than emitting invisible infrared light, IR LED looks like a normal LED and also operates like a normal LED, means it consumes 20mA current and 3vots power. IR LEDs have light emitting angle of approx. 20-60 degree and range of approx. few centimeters to several feets; it depends upon the type of IR transmitter and the manufacturer. Some transmitters have the range in kilometers.
- **IR Receiver**
  - An infrared receiver, or IR receiver, is hardware that sends information from an infrared remote control to another device by receiving and decoding signals. In general, the receiver outputs a code to uniquely identify the infrared signal that it receives. This

code is then used in order to convert signals from the remote control into a format that can be understood by the other device. It is the part of a device that receives infrared commands from a remote control. Poorly placed IR receivers can result in what is called "tunnel vision", where the operational range of a remote control is reduced because they are set so far back into the chassis of a device.

- **PROGRAM(IR Receiver)**

```
#include <IRremote.h> //Note you will need to delete a built in library that comes with the arduino IDE that has a conflict with this library.
//Please delete the ir robot library.
//Variables:
 IRrecv irrecv(7); //Set up the IR Reciever, call it irrecv and attach it to the correct pin
decode_results results; //Set up a variable to hold the results
 void setup() {
// put your setup code here, to run once:
 Serial.begin(9600);
 irrecv.enableIRIn(); // Start the receiver
 }
void loop() {
// put your main code here, to run repeatedly:
 if (irrecv.decode(&results)) {
String tmp = (String)results.value;
//Read and store the incoming code from the IR receiver
 if (tmp == "4294967295") //This value tells the Arduino that the previous button is still active.
  {
Serial.println("Button still active."); //Report to the serial monitor the last button was still held down
 }
else {
 Serial.println(results.value, HEX); //A new button has been pressed with the value shown.
 } irrecv.resume(); // Receive the next value
 }
 }
```

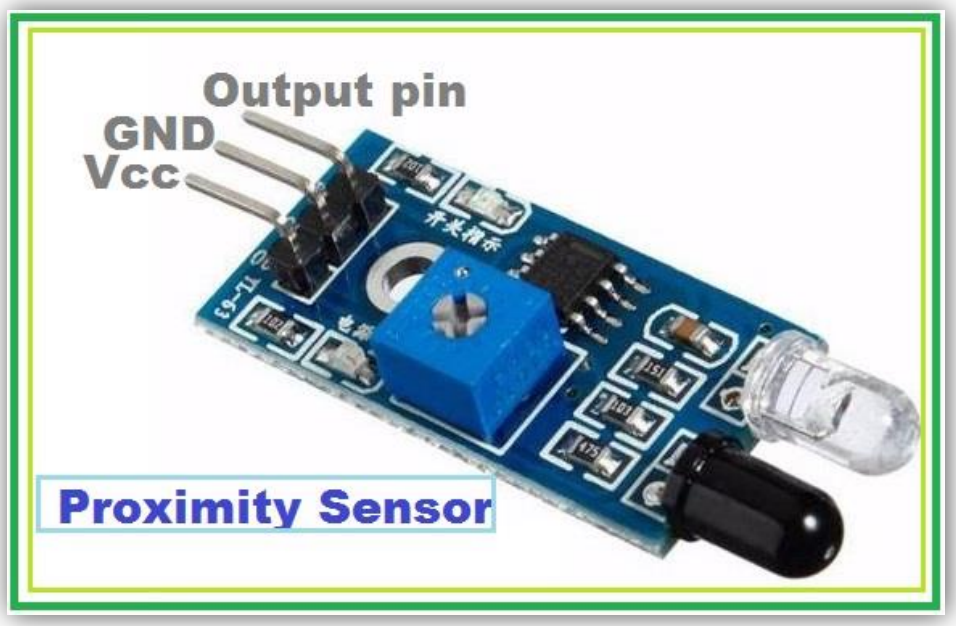
Tracking Sensor

- IR light reflection switch, useful for obstacle avoidance or line following on models that move around the floor. An obstacle in front of the sender/receiver diodes will cause the 'out' pin to be pulled low (active low). A pot allows adjustment of the circuit's sensitivity. The detection distance can be up to approximately 1 cm.
- A Line Tracker mostly consists of an infrared light sensor and an infrared LED. It functions by illuminating a surface with infrared light; the sensor then picks up the reflected infrared radiation and, based on its intensity, determines the reflectivity of the surface in question.
- Application of a line tracker can be Edge detection.

◆ PROGRAM

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println(digitalRead(2)); // print the data from the sensor
  delay(500);
}
```

Proximity Sensor



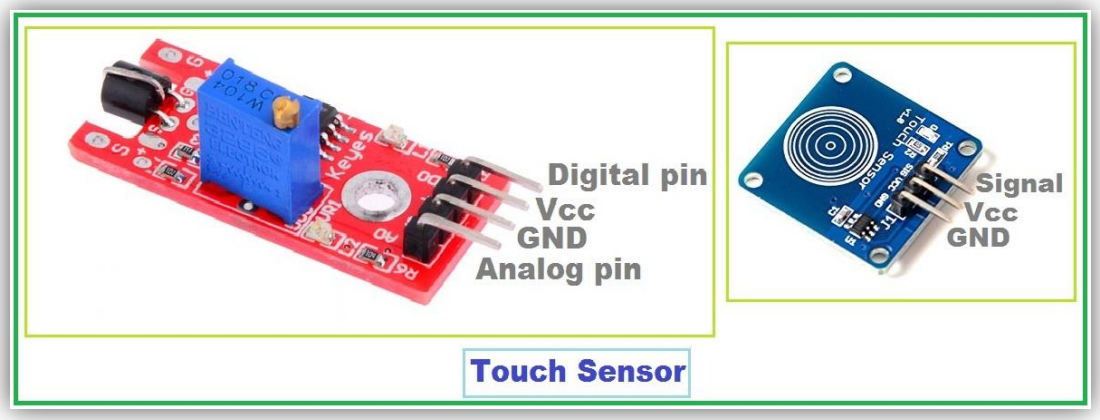
- A **proximity sensor** is a sensor able to detect the presence of nearby objects without any physical contact.
- A proximity sensor often emits an electromagnetic field or a beam of electromagnetic radiation (infrared, for instance), and looks for changes in the field or return signal. The object being sensed is often referred to as the proximity sensor's target. Different proximity sensor targets demand different sensors. For example, a capacitive or photoelectric sensor might be suitable for a plastic target; an inductive proximity sensor always requires a metal target.
- The maximum distance that this sensor can detect is defined "nominal range". Some sensors have adjustments of the nominal range or means to report a graduated detection distance. Some know these processes as "thermosensation".
- Proximity sensors can have a high reliability and long functional life because of the absence of mechanical parts and lack of physical contact between sensor and the sensed object.
- Proximity sensors are commonly used on smartphones to detect (and skip) accidental touchscreen taps when held to the ear during a call. They are also used in machine vibration monitoring to measure the variation in distance between a shaft and its support

bearing. This is common in large steam turbines, compressors, and motors that use sleeve-type bearings.

- **PROGRAM**

```
void setup() {
  // put your setup code here, to run once:
pinMode(0,INPUT);
Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
int sen = analogRead(0);
Serial.println(sen);
delay(100);
}
```

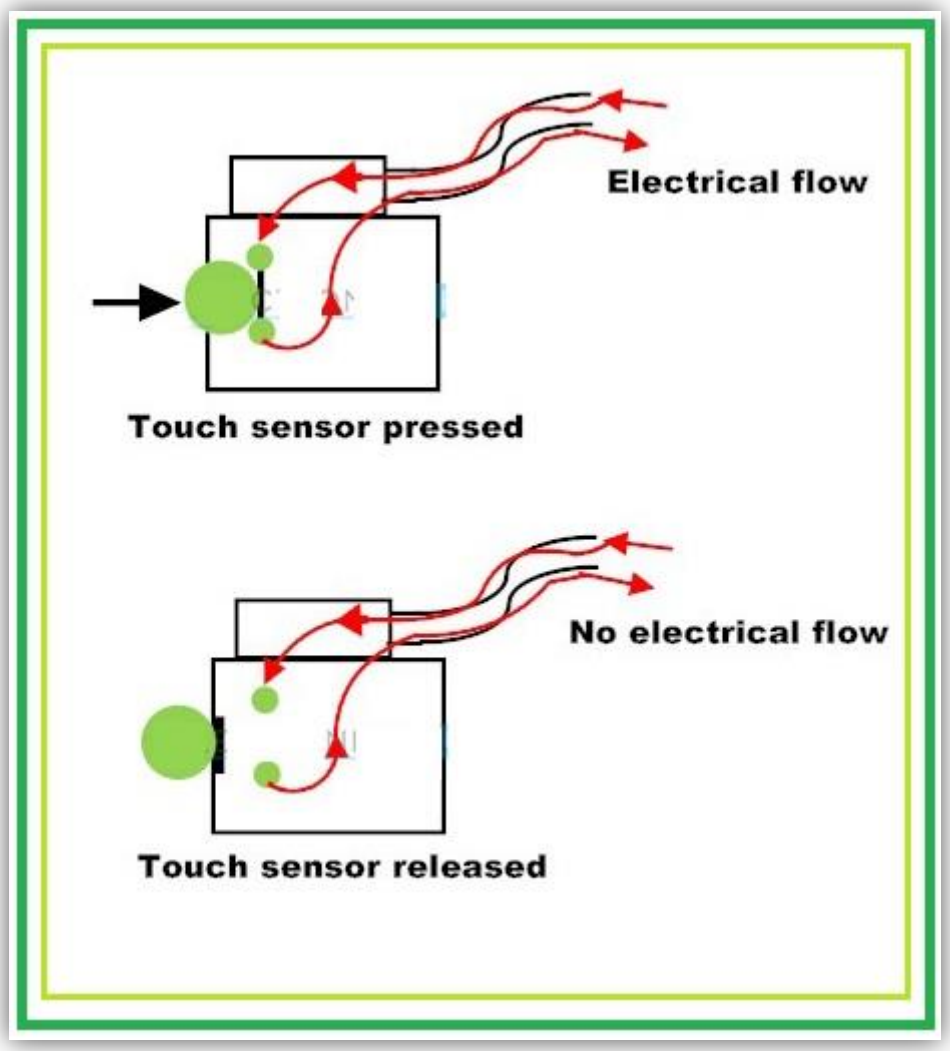
Touch Sensor



- The sense of touch is an important sensory channel in many animals and some plants. Our senses inform to us when our hands touch something. Computer input devices are indifferent to human contact as there is no reaction from software in the event of making, maintaining or breaking physical contact like touches or releases. Thus, touch sensing input devices offers numerous possibilities for novel interaction techniques.
- A touch sensor detects touch or near proximity without relying on physical contact. Touch sensors are making their way into many applications like mobile phones, remote controls, control panels, etc. Present day touch sensors can replace mechanical buttons and switches. Touch sensors with simple rotational sliders, touch pads and rotary wheels offer significant advantages for more intuitive user interfaces. Touch sensors are more convenient and more reliable to use without moving parts. The use of touch sensors provides great freedom to the system designer and help in reducing the overall cost of the system. The overall look of the system can be more appealing and contemporary

Working

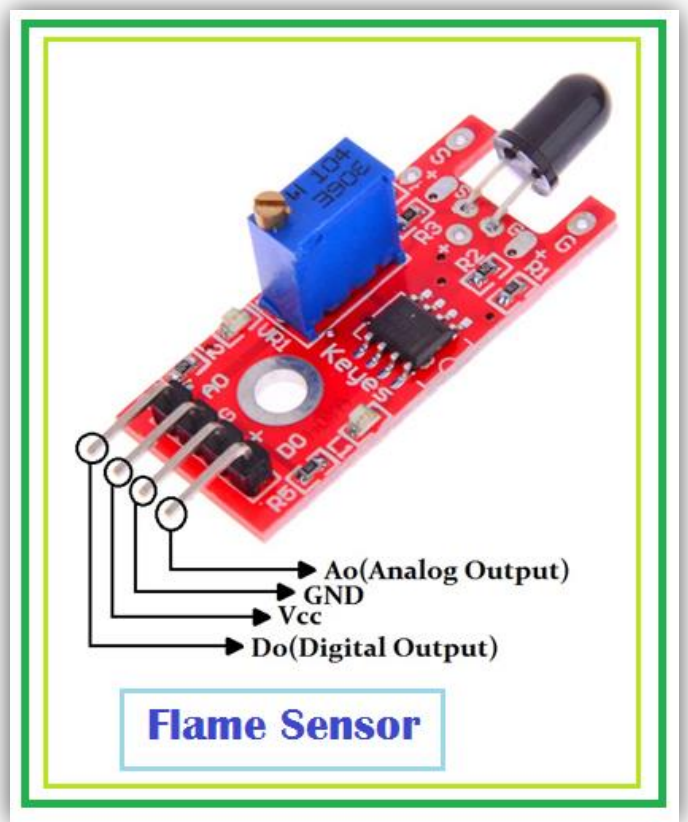
- Touch sensors are also called as tactile sensors and are sensitive to touch, force or pressure. They are one of the simplest and useful sensors. The working of a touch sensor is similar to that of a simple switch. When there is contact with the surface of the touch sensor, the circuit is closed inside the sensor and there is a flow of current. When the contact is released, the circuit is opened and no current flows.The pictorial representation of working of a touch sensor is shown below.



- **PROGRAM**

```
int Led = 13 ; // define LED Interface
int buttonpin = 3; // define Metal Touch Sensor Interface
int val ; // define numeric variables val
void setup ()
{
  pinMode (Led, OUTPUT) ; // define LED as output interface
  pinMode (buttonpin, INPUT) ; // define metal touch sensor output interface
}
void loop ()
{
  val = digitalRead (buttonpin) ; // digital interface will be assigned a value of 3 to
read val
  if (val == HIGH) // When the metal touch sensor detects a signal, LED flashes
  {
    digitalWrite (Led, HIGH);
  }
  else
  {
    digitalWrite (Led, LOW);
  }
}
```

Flame Sensor

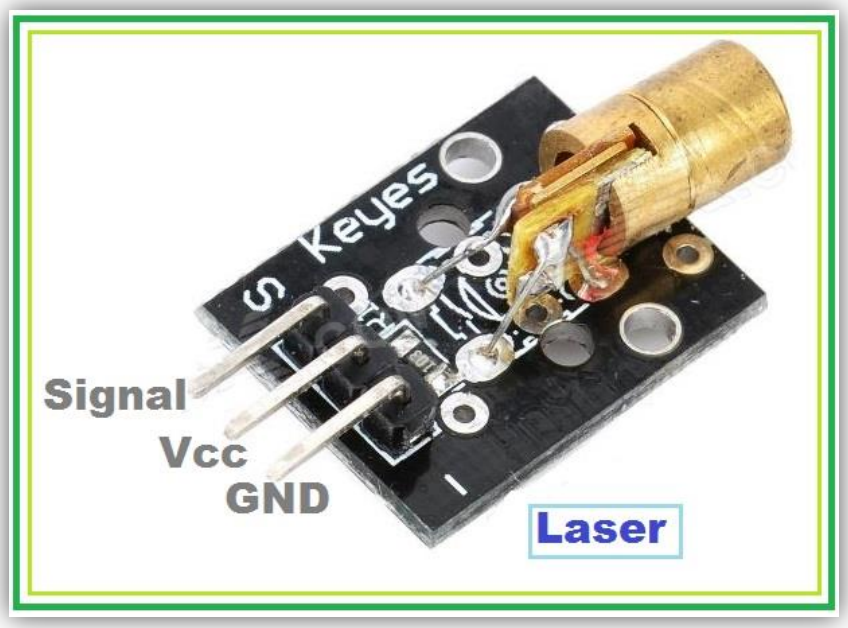


- A **flame detector** is a sensor designed to detect and respond to the presence of a flame or fire.
- The spectral sensitivity of the sensor is optimized to detect emissions from naked flames.
- These types of sensors are used for short range fire detection and can be used to monitor projects or as a safety precaution to cut devices off / on.
- This unit is mostly accurate up to about 3 feet.
- The flame sensor is very sensitive to IR wavelength at 760 nm ~ 1100 nm light.
- Analog output (Ao): Real-time output voltage signal on the thermal resistance.
- Digital output (Do): When the temperature reaches a certain threshold, the output high and low signal threshold adjustable via potentiometer.

◆ PROGRAM

```
// lowest and highest sensor readings:
const int sensorMin = 0;     // sensor minimum
const int sensorMax = 1024;  // sensor maximum

void setup() {
  // initialize serial communication @ 9600 baud:
  Serial.begin(9600);
}
void loop() {
  // read the sensor on analog A0:
      int sensorReading = analogRead(A0);
  // map the sensor range (four options):
  // ex: 'long int map(long int, long int, long int, long int, long int)'
      int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

  // range value:
  switch (range) {
  case 0:    // A fire closer than 1.5 feet away.
    Serial.println("** Close Fire **");
    break;
  case 1:    // A fire between 1-3 feet away.
    Serial.println("** Distant Fire **");
    break;
  case 2:    // No fire detected.
    Serial.println("No Fire");
    break;
  }
  delay(1);  // delay between reads
}
```

Laser Emitter



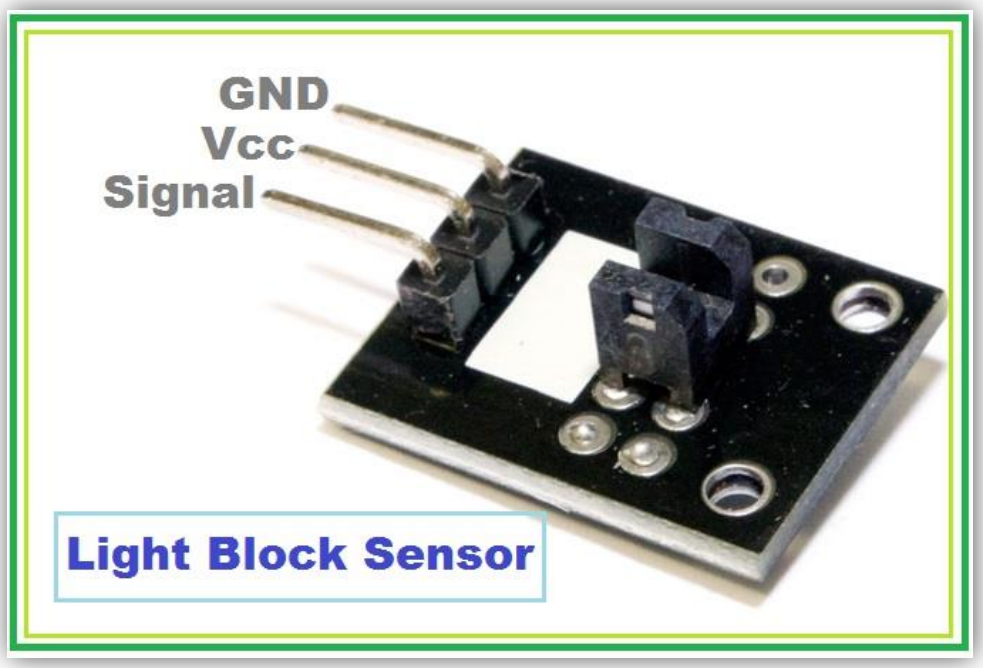
- Laser sensors are used where small objects or precise positions are to be detected. They are designed as through-beam sensors, retro-reflective sensors or diffuse reflection sensors.
- Laser light consists of light waves of the same wave length with a fixed phase ratio (coherence). This results in an important feature of laser systems that is the almost parallel light beam.
- The result: Long ranges can be achieved thanks to the small angle of divergence. The laser spot which is also clearly visible in daylight simplifies the alignment of the system.

- **APPLICATION**
  - Detection of tiny objects.
  - Clearly visible red light for easy adjustment to the object.
  - Automatic switch point setting by pressing a pushbutton.
  - Application sensors for special application areas.
  - System components for fine adjustment.

- **PROGRAM**

```
void setup ()
{
  pinMode (13, OUTPUT); // define the digital output interface 13 feet
}
void loop () {
  digitalWrite (13, HIGH); // open the laser head
  delay (1000); // delay one second
  digitalWrite (13, LOW); // turn off the laser head
  delay (1000); // delay one second
}
```

Light Blocking Sensor



- This sensor is widely used in motor speed detection, pulse count, the position limit, etc. The DO output interface can be directly connected to a micro-controller IO port, if there is a block detection sensor, such as the speed of the motor encoder can detect. DO modules can be connected to the relay, limit switch, and other functions, it can also with the active buzzer module, compose alarm.
- **PROGRAM**

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2; // the number of the pushbutton pin
const int ledPin = 13; // the number of the LED pin
// variables will change:
int buttonState = 0; // variable for reading the pushbutton status
void setup() {
// initialize the LED pin as an output:
pinMode(ledPin, OUTPUT);
// initialize the pushbutton pin as an input:
pinMode(buttonPin, INPUT);
}
```

```
void loop(){
// read the state of the pushbutton value:
buttonState = digitalRead(buttonPin);
// check if the pushbutton is pressed.
// if it is, the buttonState is HIGH:
if (buttonState == HIGH) {
// turn LED on:
digitalWrite(ledPin, HIGH);
}
else {
// turn LED off:
digitalWrite(ledPin, LOW);
}
}
```

Light Cup

- Minimum two light cup modules is required to build any application.
- **Working**:
  - When the light cup is tilted from its original (straight) position, the mercury ball completes the circuit and the LED turns on. When the light cup is kept in its straight position, the mercury ball comes back to its position and disconnects the circuit, therefore the LED turns off.
- Two modules are used to perform a Magic Light Cup trick, where light is "poured" from one cup to another, and the brightness changes to show how much light has been poured. PWM (pulse width modulation) is used to change the brightness of the light in each cup. Mercury switches provide a digital signal that triggers the PWM regulator on the Arduino, which changes the brightness of the attached LED. Light is poured back and forth between two cups. This requires two modules.

- **PROGRAM**

Example1. (Using two light cups)

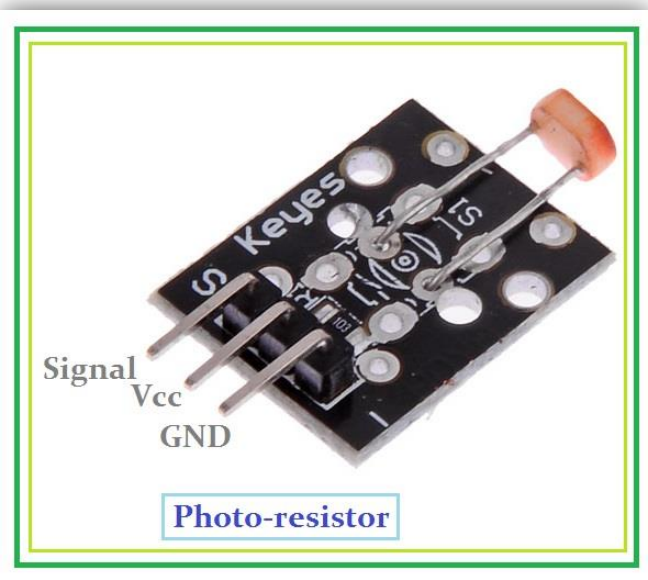
```
int LedPinA = 5;
int LedPinB = 6;
int ButtonPinA = 7;
int ButtonPinB = 4;
int buttonStateA = 0;
int buttonStateB = 0;
int brightness = 0;
void setup ()
{
  pinMode (LedPinA, OUTPUT);
  pinMode (LedPinB, OUTPUT);
  pinMode (ButtonPinA, INPUT);
  pinMode (ButtonPinB, INPUT);
}
void loop ()
{
  buttonStateA = digitalRead (ButtonPinA);
  if (buttonStateA == HIGH && brightness! = 255)
  {
    brightness + +;
  }
  buttonStateB = digitalRead (ButtonPinB);
  if (buttonStateB == HIGH && brightness! = 0)
  {
    brightness -;
  }
  analogWrite (LedPinA, brightness); // A few Guan Yuan (ii) ?
  analogWrite (LedPinB, 255 - brightness);// B Yuan (ii) a few Bang ?
  delay (25);
}
```

Example2. (Using one light cups)

```
int SwitchPIN=2;
int LightPIN=4;
int Switch_STATUS=0;

void setup() {
  // put your setup code here, to run once:
  pinMode(SwitchPIN, INPUT);
```

```
  pinMode(LightPIN, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Switch_STATUS=digitalRead(SwitchPIN);
  if(Switch_STATUS==0)
  {
    digitalWrite(LightPIN, LOW);
    Serial.println("Hg Switch: OFF");
  }
  else
  {
    digitalWrite(LightPIN, HIGH);
    Serial.println("Hg Switch: ON");
  }
}
```

Photo resistor



- Photo resistors, also known as light dependent resistors (LDR), are light sensitive devices most often used to indicate the presence or absence of light, or to measure the light intensity. In the dark, their resistance is very high, sometimes up to 1MΩ, but when the LDR sensor is exposed to light, the resistance drops dramatically, even down to a few ohms, depending on the light intensity.
- LDRs have a sensitivity that varies with the wavelength of the light applied and are nonlinear devices. They are used in many applications but are sometimes made obsolete by other devices such as photodiodes and phototransistors.
- **WORKING**
  - Based on the materials used, photo resistors can be divided into two types; intrinsic and extrinsic. Intrinsic photo resistors use undoped materials such as silicon or germanium.
  - Photons that fall on the device excite electrons from the valence band to the conduction band, and the result of this process are more free electrons in the material, which can carry current, and therefore less resistance.
  - Extrinsic photo resistors are made of materials doped with impurities, also called dopants.
  - The dopants create a new energy band above the existing valence band, populated by electrons. These electrons need less energy to make the transition to the conduction band

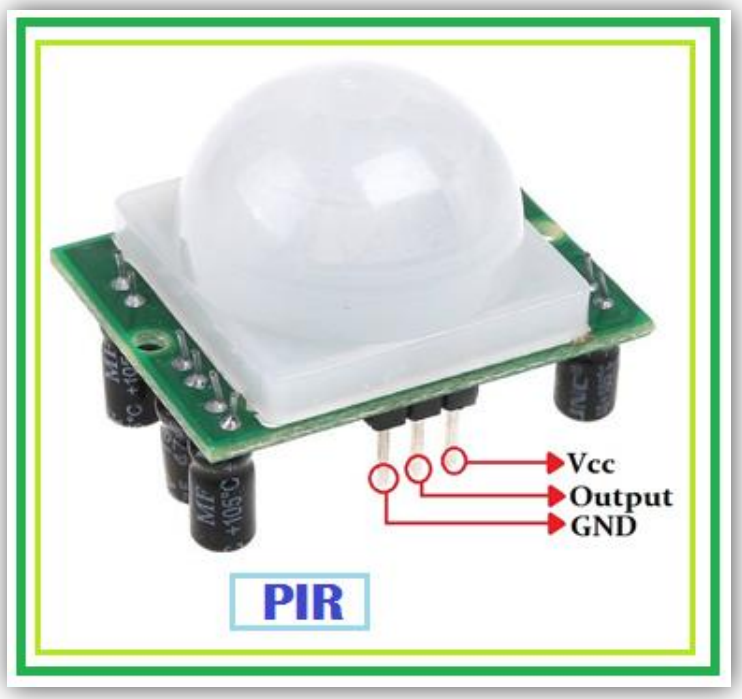
thanks to the smaller energy gap. The result is a device sensitive to different wavelengths of light. Regardless, both types will exhibit a decrease in resistance when illuminated.

- The higher the light intensity, the larger the resistance drop is. Therefore, the resistance of LDRs is an inverse, nonlinear function of light intensity.

- **PROGRAM**

```
int sensorPin = A0; // select the input pin for LDR
int sensorValue = 0; // variable to store the value coming from the sensor
void setup() {
Serial.begin(9600); //sets serial port for communication
}
void loop() {
sensorValue = analogRead(sensorPin); // read the value from the sensor
Serial.println(sensorValue); //prints the values coming from the sensor on
the screen
delay(100);
}
```

PIR

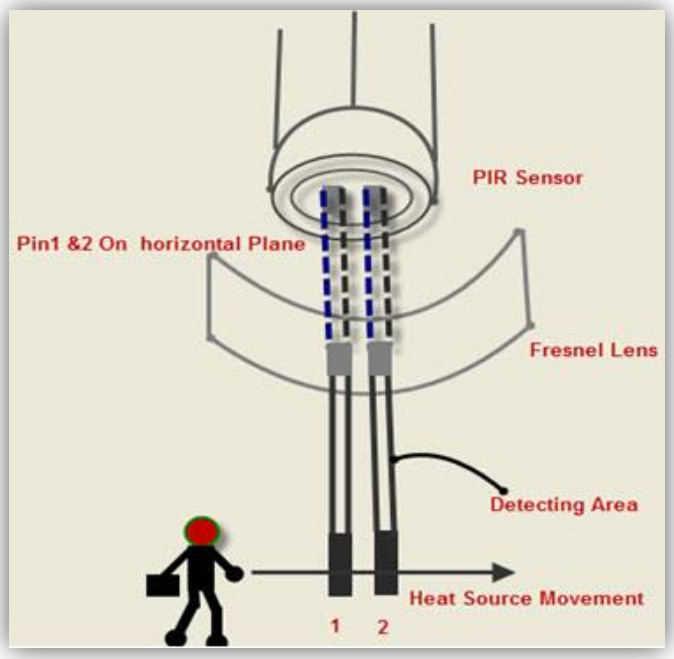


- A passive infrared **sensor** (**PIR sensor**) is an electronic **sensor** that measures infrared (IR) light radiating from objects in its field of view. They are most often used in **PIR**-based motion detectors.
- PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensors range.
- They are small, inexpensive, low-power, easy to use and don't wear out. For that reason they are commonly found in appliances and gadgets used in homes or businesses.
- They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors.
- PIRs are basically made of a pyroelectric sensor (which you can see above as the round metal can with a rectangular crystal in the center), which can detect levels of infrared radiation. Everything emits some low level radiation, and the hotter something is, the more radiation is emitted.
- The sensor in a motion detector is actually split in two halves. The reason for that is that we are looking to detect motion (change) not average IR levels.

- The two halves are wired up so that they cancel each other out. If one half sees more or less IR radiation than the other, the output will swing high or low.

WORKING:

- The PIR sensors are more complicated than the other sensors as they consist of two slots. These slots are made of a special material which is sensitive to IR.
- The Fresnel lens is used to see that the two slots of the PIR can see out past some distance. When the sensor is inactive, then the two slots sense the same amount of IR.The ambient amount radiates from the outdoors, walls or room,etc.
- When a human body or any animal passes by, then it intercepts the first slot of the PIR sensor. This causes a positive differential change between the two bisects.
- When a human body leaves the sensing area, the sensor generates a negative differential change between the two bisects.
- The infrared sensor itself is housed in a hermetically sealed metal to improve humidity/temperature/noise/immunity.
- There is a window which is made of typically coated silicon material to protect the sensing element.



- **PROGRAM**

Example1.

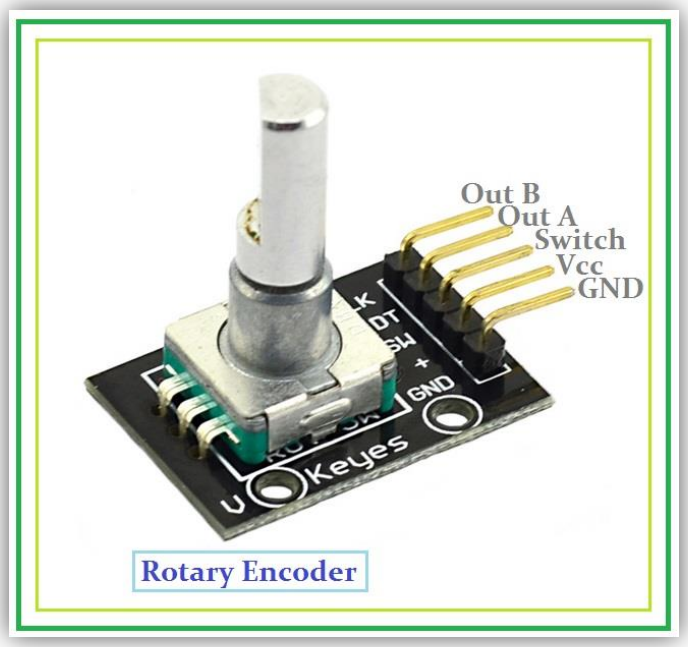
```
int pirpin = A0;
int pirvalue;
void setup() { // put your setup code here, to run once:
pinMode(pirpin, INPUT);
Serial.begin(9600);
}
void loop() {
// put your main code here, to run repeatedly:
pirvalue= analogRead(pirpin);
Serial.println(pirvalue);
delay(100);
}
```

Example2.

```
/* PIR sensor tester
 */

int ledPin = 13;                // choose the pin for the LED
int inputPin = 2;               // choose the input pin (for PIR sensor)
int pirState = LOW;             // we start, assuming no motion detected
int val = 0;                    // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);      // declare LED as output
  pinMode(inputPin, INPUT);     // declare sensor as input

  Serial.begin(9600);
}

void loop(){
  val = digitalRead(inputPin);  // read input value
  if (val == HIGH) {            // check if the input is HIGH
    digitalWrite(ledPin, HIGH);  // turn LED ON
    if (pirState == LOW) {
      // we have just turned on
      Serial.println("Motion detected!");
      // We only want to print on the output change, not state
      pirState = HIGH;
```

```
  }
} else {
  digitalWrite(ledPin, LOW); // turn LED OFF
  if (pirState == HIGH){
    // we have just turned of
    Serial.println("Motion ended!");
    // We only want to print on the output change, not state
    pirState = LOW;
  }
}
```

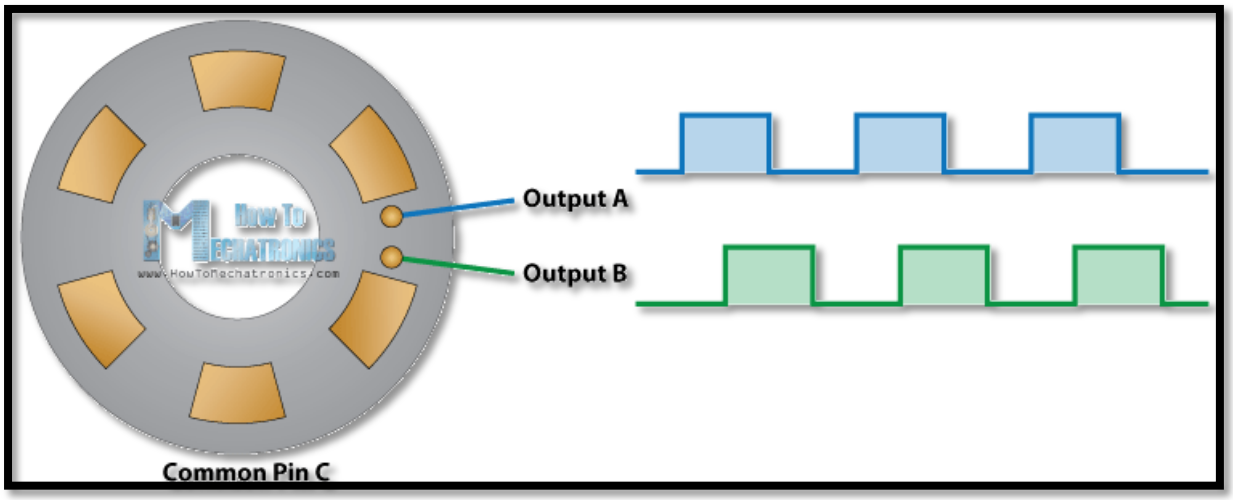
Rotary Encoder



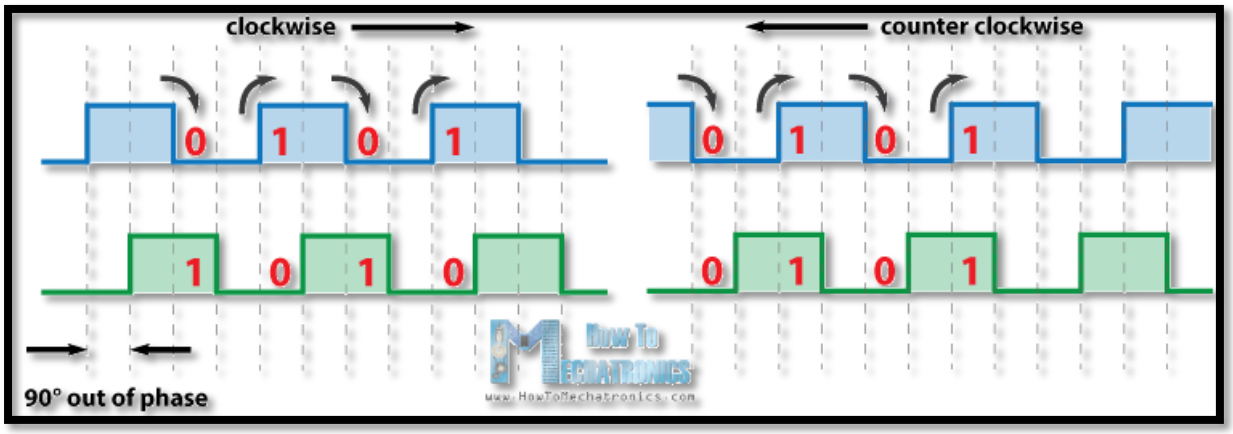
- A **rotary encoder**, also called a **shaft encoder**, is an electro-mechanical device that converts the angular position or motion of a shaft or axle to an analog or digital code.

Working

- Let's take a closer look at the encoder and see its working principle. Here's how the square wave pulses are generated: The encoder has a disk with evenly spaced contact zones that are connected to the common pin C and two other separate contact pins A and B, as illustrated below.



- When the disk will start rotating step by step, the pins A and B will start making contact with the common pin and the two square wave output signals will be generated accordingly.
- Any of the two outputs can be used for determining the rotated position if we just count the pulses of the signal. However, if we want to determine the rotation direction as well, we need to consider both signals at the same time.
- We can notice that the two output signals are displaced at 90 degrees out of phase from each other. If the encoder is rotating clockwise the output A will be ahead of output B.



- So if we count the steps each time the signal changes, from High to Low or from Low to High, we can notice at that time the two output signals have opposite values. Vice versa, if the encoder is rotating counter clockwise, the output signals have equal values. So considering this, we can easily program our controller to read the encoder position and the rotation direction.

- **PROGRAM**

EXAMPLE1.

```
/*******Interrupt-based Rotary Encoder Sketch*******
by Simon Merrett, based on insight from Oleg Mazurov, Nick Gammon, rt,
Steve Spence
*/

static int pinA = 2; // Our first hardware interrupt pin is digital pin 2
static int pinB = 3; // Our second hardware interrupt pin is digital pin 3
volatile byte aFlag = 0; // let's us know when we're expecting a rising edge
on pinA to signal that the encoder has arrived at a detent
volatile byte bFlag = 0; // let's us know when we're expecting a rising edge
on pinB to signal that the encoder has arrived at a detent (opposite direction
```

```
to when aFlag is set)
volatile byte encoderPos = 0; //this variable stores our current value of encoder position. Change to int or uin16_t instead of byte if you want to record a larger range than 0-255
volatile byte oldEncPos = 0; //stores the last encoder position value so we can compare to the current reading and see if it has changed (so we know when to print to the serial monitor)
volatile byte reading = 0; //somewhere to store the direct values we read from our interrupt pins before checking to see if we have moved a whole detent

void setup() {
  pinMode(pinA, INPUT_PULLUP); // set pinA as an input, pulled HIGH to the logic voltage (5V or 3.3V for most cases)
  pinMode(pinB, INPUT_PULLUP); // set pinB as an input, pulled HIGH to the logic voltage (5V or 3.3V for most cases)
  attachInterrupt(0,PinA,RISING); // set an interrupt on PinA, looking for a rising edge signal and executing the "PinA" Interrupt Service Routine (below)
  attachInterrupt(1,PinB,RISING); // set an interrupt on PinB, looking for a rising edge signal and executing the "PinB" Interrupt Service Routine (below)
  Serial.begin(115200); // start the serial monitor link
}

void PinA(){
  cli(); //stop interrupts happening before we read pin values
  reading = PIND & 0xC; // read all eight pin values then strip away all but pinA and pinB's values
  if(reading == B00001100 && aFlag) { //check that we have both pins at detent (HIGH) and that we are expecting detent on this pin's rising edge
    encoderPos --; //decrement the encoder's position count
    bFlag = 0; //reset flags for the next turn
    aFlag = 0; //reset flags for the next turn
  }
  else if (reading == B00000100) bFlag = 1; //signal that we're expecting pinB to signal the transition to detent from free rotation
  sei(); //restart interrupts
}

void PinB(){
```

```
  cli(); //stop interrupts happening before we read pin values
  reading = PIND & oxC; //read all eight pin values then strip away all but
pinA and pinB's values
  if (reading == Booooo1oo && bFlag) { //check that we have both pins at
detent (HIGH) and that we are expecting detent on this pin's rising edge
    encoderPos ++; //increment the encoder's position count
    bFlag = o; //reset flags for the next turn
    aFlag = o; //reset flags for the next turn
  }
  else if (reading == Booooo1ooo) aFlag = 1; //signal that we're expecting pinA
to signal the transition to detent from free rotation
  sei(); //restart interrupts
}

void loop(){
  if(oldEncPos != encoderPos) {
    Serial.println(encoderPos);
    oldEncPos = encoderPos;
  }
}
```

EXAMPLE2.

```
/*    Arduino Rotary Encoder Tutorial
 *
 *  by Dejan Nedelkovski, www.HowToMechatronics.com
 *
 */

 #define outputA 6
 #define outputB 7
 int counter = o;
 int aState;
 int aLastState;
 void setup() {
   pinMode (outputA,INPUT);
   pinMode (outputB,INPUT);

   Serial.begin (9600);
```

```
  // Reads the initial state of the outputA
  aLastState = digitalRead(outputA);
 }
 void loop() {
  aState = digitalRead(outputA); // Reads the "current" state of the outputA
  // If the previous and the current state of the outputA are different, that means a
Pulse has occured
  if (aState != aLastState){
    // If the outputB state is different to the outputA state, that means the encoder
is rotating clockwise
    if (digitalRead(outputB) != aState) {
      counter ++;
    } else {
      counter --;
    }
    Serial.print("Position: ");
    Serial.println(counter);
  }
  aLastState = aState; // Updates the previous state of the outputA with the current
state
 }
```

Description of the above code: So first we need to define the pins to which our encoder is connected and define some variables needed for the program. In the setup section we need to define the two pins as inputs, start the serial communication for printing the results on the serial monitor, as well as read the initial value of the output A and put the value into the variable aLastState.

Then in the loop section we read the output A again but now we put the value into the aState variable. So if we rotate the encoder and a pulse is generated, these two values will differ and the first “if” statement will become true. Right after that using the second “if” statement we determine the rotation direction. If the output B state differ from the output A state the counter will be increased by one, else it will be decreased. At the end, after printing the results on the serial monitor, we need to update the aLastState variable with aState variable.

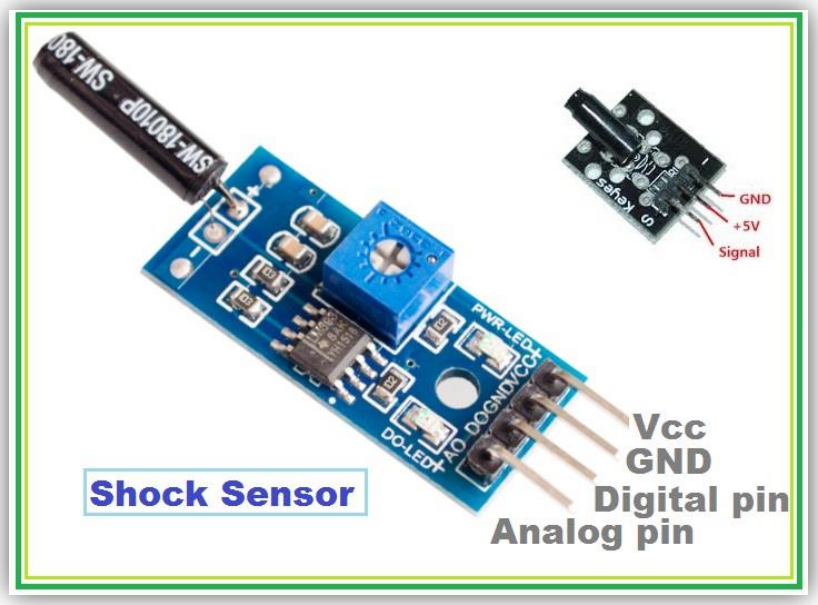
That’s all we need for this example. If upload the code, start the Serial Monitor and start rotating the encoder we will start getting the values in the serial monitor. The particular module that I have makes 30 counts each full cycle.

Furthermore go to the below link for Controlling a Stepper Motor Using a Rotary Encoder:

1)http://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/

2)https://github.com/mathertel/RotaryEncoder

Shock Sensor



- A **shock detector** or **impact monitor** is a device which indicates whether a physical shock or impact has occurred. These usually have a binary output (go/no-go) and are sometimes called *shock overload devices*. Shock detectors can be used on shipments of fragile valuable items to indicate whether a potentially damaging drop or impact may have occurred.

Working

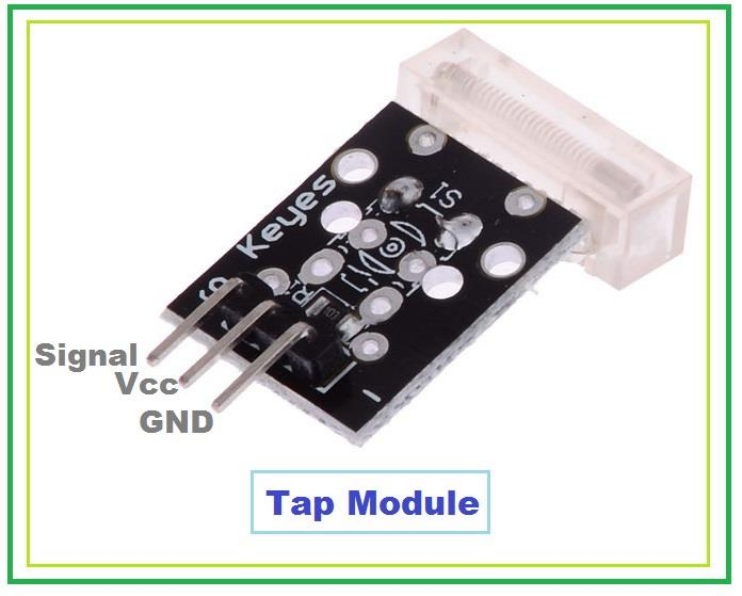
- The sensor for vibration detection is a vibration detector (or shock sensor), the detector must have a mechanical displacement to generate the alarm signal; vibration detection equipment is not only best suitable for file cabinets, vaults, strongrooms, safes and Automated Teller Machines (ATM), confidential protection special objects, but also suitable for other systems in combination, to prevent intruders break in from wall. How to use the vibration detector in correct application is very important. It is often used to provide protection in a special object where protected area that with staff's activities.

- There are two major detection methods for vibration detector; the one is mechanical detection, it works as a ON/OFF switch using the mechanical movement of metal contact, the other is acoustic sound detection. Compare to acoustic sound detection, adopts mechanical detection vibration detector only detect the true physic vibration with low false alarm. The vibration detector that is based on acoustic sound detection (with microphone), it may trigger false alarm by high noise from car, thunder in summer.

- **PROGRAM**

```
//KY-002 Shock Sensor Tutorial

int shockPin = 10; // Use Pin 10 as our Input
int shockVal = HIGH; // This is where we record our shock measurement
boolean bAlarm = false;

unsigned long lastShockTime; // Record the time that we measured a shock

int shockAlarmTime = 100; // Number of milli seconds to keep the shock alarm
high


void setup ()
{
 Serial.begin(9600);
 pinMode (shockPin, INPUT) ; // input from the KY-002
}
void loop ()
{
 shockVal = digitalRead (shockPin) ; // read the value from our sensor

 if (shockVal == LOW) // If we're in an alarm state
 {
    Serial.println("Shock Alarm");
 }
 else
 {
    Serial.println("no alarm");
 }
 delay(30);
}
```

Tap Module



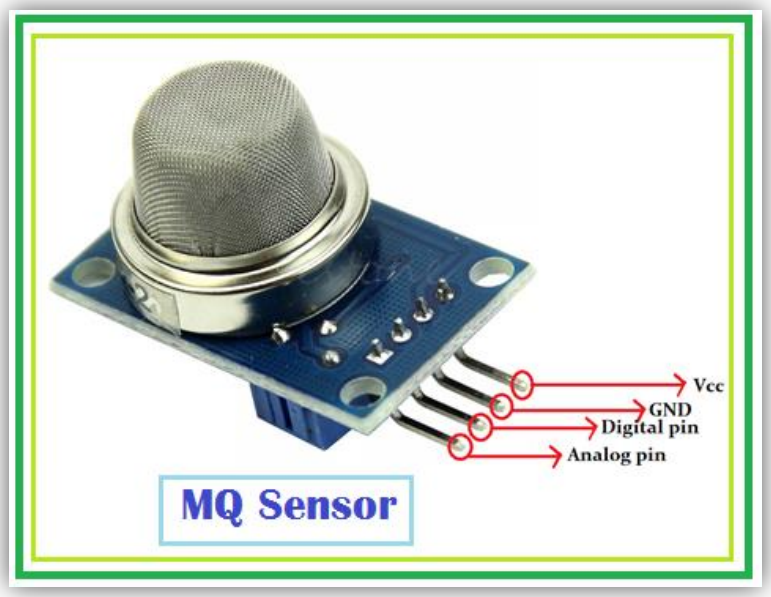
- The tap sensor, detects the knocks and the taps. It can work like a switch. The sensor sends data momentarily to the board. To keep the LED on, the button state change codes should be used. So the sensor will work as a switch.

◆ PROGRAM

```
const int  buttonPin = 3;    // the pin that the pushbutton is attached to
const int ledPin = 13;       // the pin that the LED is attached to

// Variables will change:
int buttonPushCounter = 0;   // counter for the number of button presses
int buttonState = 0;         // current state of the button
int lastButtonState = 0;     // previous state of the button

void setup() {
  // initialize the button pin as a input:
  pinMode(buttonPin, INPUT);
  // initialize the LED as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communication:
  Serial.begin(9600);
}


void loop() {
```

```
  // read the pushbutton input pin:
  buttonState = digitalRead(buttonPin);

  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {
    // if the state has changed, increment the counter
    if (buttonState == HIGH) {
      // if the current state is HIGH then the button
      // wend from off to on:
      buttonPushCounter++;
      Serial.println("on");
      Serial.print("number of button pushes:  ");
      Serial.println(buttonPushCounter);
    }
    else {
      // if the current state is LOW then the button
      // wend from on to off:
      Serial.println("off");
    }
  }
  // save the current state as the last state,
  //for next time through the loop
  lastButtonState = buttonState;


  // turns on the LED every four button pushes by
  // checking the modulo of the button push counter.
  // the modulo function gives you the remainder of
  // the division of two numbers:
  if (buttonPushCounter % 4 == 0) {
    digitalWrite(ledPin, HIGH);
  } else {
   digitalWrite(ledPin, LOW);
  }
}
```

MQ Sensor(MQ2)



- In current technology scenario, monitoring of gases produced is very important. From home appliances such as air conditioners to electric chimneys and safety systems at industries monitoring of gases is very crucial.
- **Gas sensors** are very important part of such systems. Small like a nose, gas sensors spontaneously react to the gas present, thus keeping the system updated about any alterations that occur in the concentration of molecules at gaseous state.

Working:

- The voltage that the sensor outputs changes accordingly to the smoke/gas level that exists in the atmosphere. The sensor outputs a voltage that is proportional to the concentration of smoke/gas.
- In other words, the relationship between voltage and gas concentration is the following:
  - **The greater** the gas concentration, **the greater** the output voltage
  - **The lower** the gas concentration, **the lower** the output voltage
- The output can be an analog signal (Ao) that can be read with an analog input of the Arduino or a digital output (Do) that can be read with a digital input of the Arduino.

- # PROGRAM

```
const int gasPin = A0; //GAS sensor output pin to Arduino analog A0 pin
void setup()
{
  Serial.begin(9600); //Initialize serial port - 9600 bps
}
void loop()
{
  Serial.println(analogRead(gasPin));
  delay(1000); // Print value every 1 sec
}
```

Rainfall Sensor

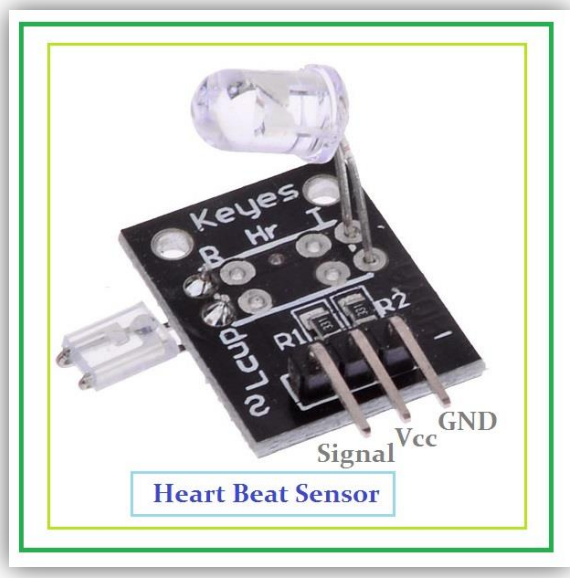
- The rain sensor module is an easy tool for rain detection. It can be used as a switch when raindrop falls through the raining board and also for measuring rainfall intensity.
- The module features, a rain board and the control board that is separate for more convenience, power indicator LED and an adjustable sensitivity though a potentiometer. The analog output is used in detection of drops in the amount of rainfall.
- Connected to 5V power supply, the LED will turn on when induction board has no rain drop, and DO output is high.
- When dropping a little amount water, DO output is low, the switch indicator will turn on. Brush off the water droplets, and when restored to the initial state, outputs high level.

◆ PROGRAM

```
int nRainIn = A1;
int nRainDigitalIn = 2;
int nRainVal;
boolean bIsRaining = false;
String strRaining;
void setup() {
  Serial.begin(9600);
```

```
  pinMode(2,INPUT);
}
void loop() {
  nRainVal = analogRead(nRainIn);
  bIsRaining = !(digitalRead(nRainDigitalIn));

  if(bIsRaining){
    strRaining = "YES";
  }
  else{
    strRaining = "NO";
  }
    Serial.print("Raining?: ");
  Serial.print(strRaining);
  Serial.print("\t Moisture Level: ");
  Serial.println(nRainVal);
   delay(200);
}
```

Heart Beat Sensor



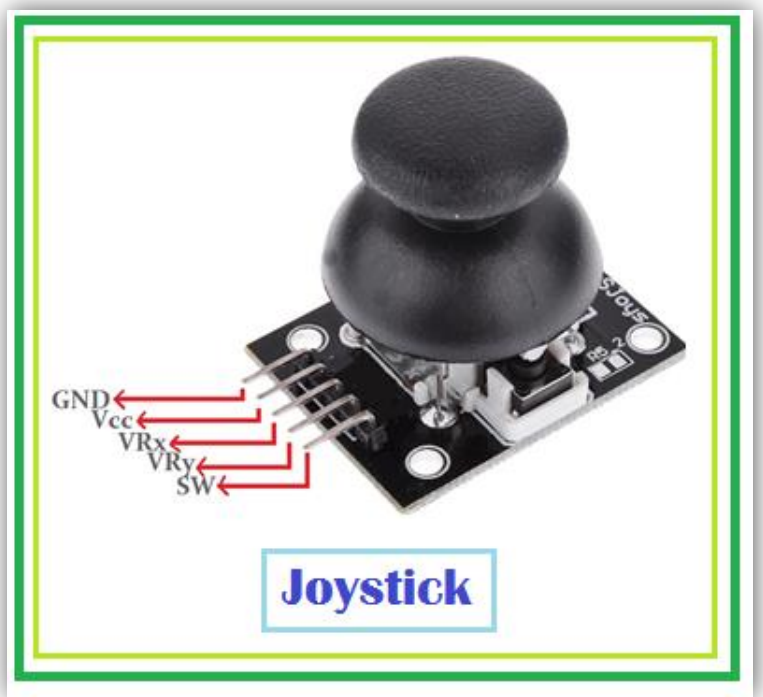
- The heartbeat sensor is based on the principle of photo phlethysmography.
- **Photo phlethysmography**: Measurement of blood flow or blood pressure by optical means (typically involving measurement of changes in the transmission or scattering of light created by blood flow in a part of the body).

  Working
  - The basic heartbeat sensor consists of a light emitting diode and a detector like a light detecting resistor or a photodiode.
  - The heart beat pulses causes a variation in the flow of blood to different regions of the body. When a tissue is illuminated with the light source, i.e. light emitted by the led, it either reflects (a finger tissue) or transmits the light. Some of the light is absorbed by the blood and the transmitted or the reflected light is received by the light detector.
  - The amount of light absorbed depends on the blood volume in that tissue. The detector output is in form of electrical signal and is proportional to the heart beat rate.

- **PROGRAM**

```
// Pulse Monitor Test Script
int sensorPin = 0;
double alpha = 0.75;
int period = 100;
double change = 0.0;
double minval = 0.0;
void setup ()
{
  Serial.begin (9600);
}
void loop ()
{
    static double oldValue = 0;
    static double oldChange = 0;

    int rawValue = analogRead (sensorPin);
    double value = alpha * oldValue + (1 - alpha) * rawValue;

    Serial.print (rawValue);
    Serial.print (",");
    Serial.println (value);
    oldValue = value;

    delay (period);
}
```

Dual Axis XY Joystick



- A **joystick** is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling.
- Joysticks are often used to control video games, and usually have one or more push-buttons whose state can also be read by the computer. A popular variation of the joystick used on modern video game consoles is the analog stick. Joysticks are also used for controlling machines such as cranes, trucks, underwater unmanned vehicles, wheelchairs, surveillance cameras, and zero turning radius lawn mowers. Miniature finger-operated joysticks have been adopted as input devices for smaller electronic equipment such as mobile phones.
- Specialist joysticks, classed as an assistive technology pointing device, are used to replace the computer mouse for people with fairly severe physical disabilities. Rather than controlling games, these joysticks control the pointer. They are often useful to people with athetoid conditions, such as cerebral palsy, who find them easier to grasp than a standard mouse.
- Miniature joysticks are available for people with conditions involving muscular weakness. They are also used on electric powered

wheelchairs for control since they are simple and effective to use as a control method.

- **PROGRAM**

```
// Arduino pin numbers
const int SW_pin = 2; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output

void setup() {
  pinMode(SW_pin, INPUT);
  digitalWrite(SW_pin, HIGH);
  Serial.begin(115200);
}

void loop() {
  Serial.print("Switch:  ");
  Serial.print(digitalRead(SW_pin));
  Serial.print("\n");
  Serial.print("X-axis: ");
  Serial.print(analogRead(X_pin));
  Serial.print("\n");
  Serial.print("Y-axis: ");
  Serial.println(analogRead(Y_pin));
  Serial.print("\n\n");
  delay(500);
}
```