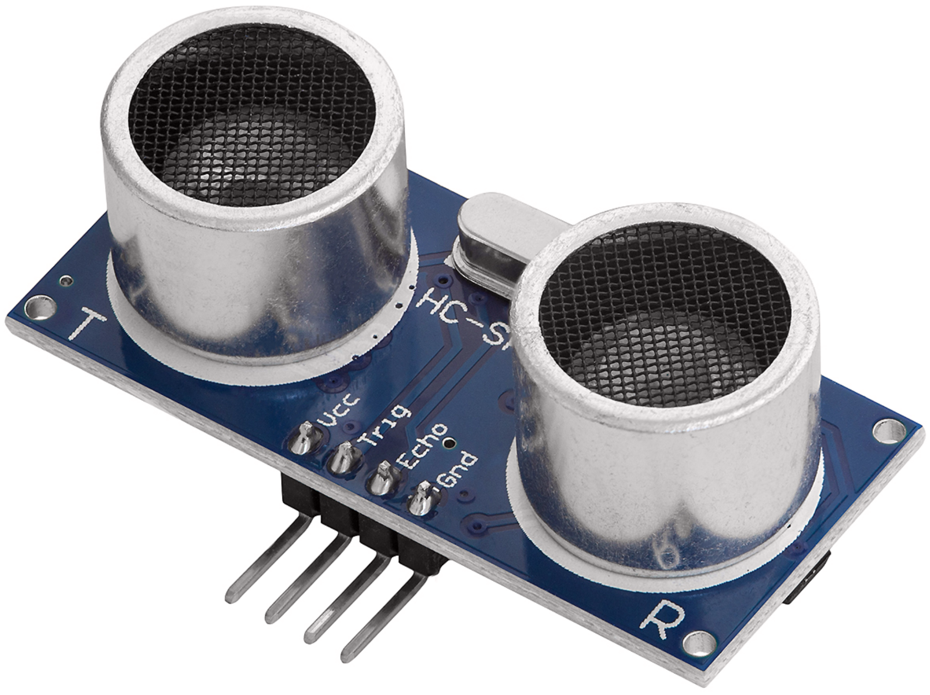
# Welcome!

Thank you for purchasing our *AZ-Delivery HC-SR04 Ultrasonic Sensor Module*. On the following pages, you will be introduced to how to use and set up this handy device.

**Have fun!**



## Table of Contents

## Introduction

The HC-SR04 ultrasonic sensor module is a device that can measure distance using ultrasonic sound waves. It can determine the distance between the module and other objects in vicinity with high accuracy.

The sensor is commonly used in obstacle avoiding devices, various distance measuring devices, automation projects, car parking systems, proximity alarms, etc.

The module consists of an ultrasonic transmitter, receiver, on-board electronic parts like amplifier chips, a crystal oscillator and other passive components like capacitors and resistors.

The operation principle of the module is the following. The ultrasonic sound waves are transmitted by the on-board transmitter. When there is an obstacle in front of the module, the ultrasonic sound waves get reflected by an obstacle back to the module. The reflected ultrasonic sound wave then gets picked up by the on-board receiver. This is then output on the ECHO pin as a digital signal with a frequency that is directly related to the time required for an ultrasonic sound wave to travel from the module to the obstacle and back to the receiver.

## Calculating the distance

The module emits ultrasonic sound waves on a frequency that is higher than the human hearing range (more than 20kHz). The speed of the sound waves traveling through the air is approximately *343m/*s at room temperature (*20°C*). This speed value depends on the environmental circumstances like temperature, humidity and variations in air pressure. The speed of sound in the air actually depends much more on the temperature and very little on humidity and the pressure of air. It increases its value approximately *0.6m/s* per degree of Celsius. In most cases at a temperature of *20°C*, the value of *343 m/s* can be used. To accurately calculate the speed of the ultrasonic sound wave in the air, use the following formula:

**V (m/s) = 331.3 + (0.606 × T)**

where V (m/s) is the speed of ultrasonic sound wave in the air and T (°C) is the temperature of the air.

When the speed of the ultrasonic sound wave is multiplied by the time that the ultrasonic sound wave traveled from the module to the obstacle and back, the result is the distance from the module to the obstacle and back:

**Distance = Speed x Time**

Because the sound waves travel *343* meters per one second, to measure distances from 20mm up to 4000mm (the range of the module) the measurements are done in microseconds. So, the speed of the ultrasonic sound waves should be converted from the *m/s* into *cm/μs*, which is:

| | | |
|---|---|---|
| **343m/s** | **= 0.0343 cm/μs** | **= 1/29 cm/μs** |
| **13503.9in/s** | **= 0.0135 in/μs** | **= 1/74 in/μs** (used in the sketch) |

Then, to get the actual distance between the module and obstacle, the previous result should be divided by two, because the ultrasonic sound wave travels from the module to the obstacle and back, so calculation goes like this:

**Distance (cm) = Speed of sound (cm/μs) × Time (μs) / 2**

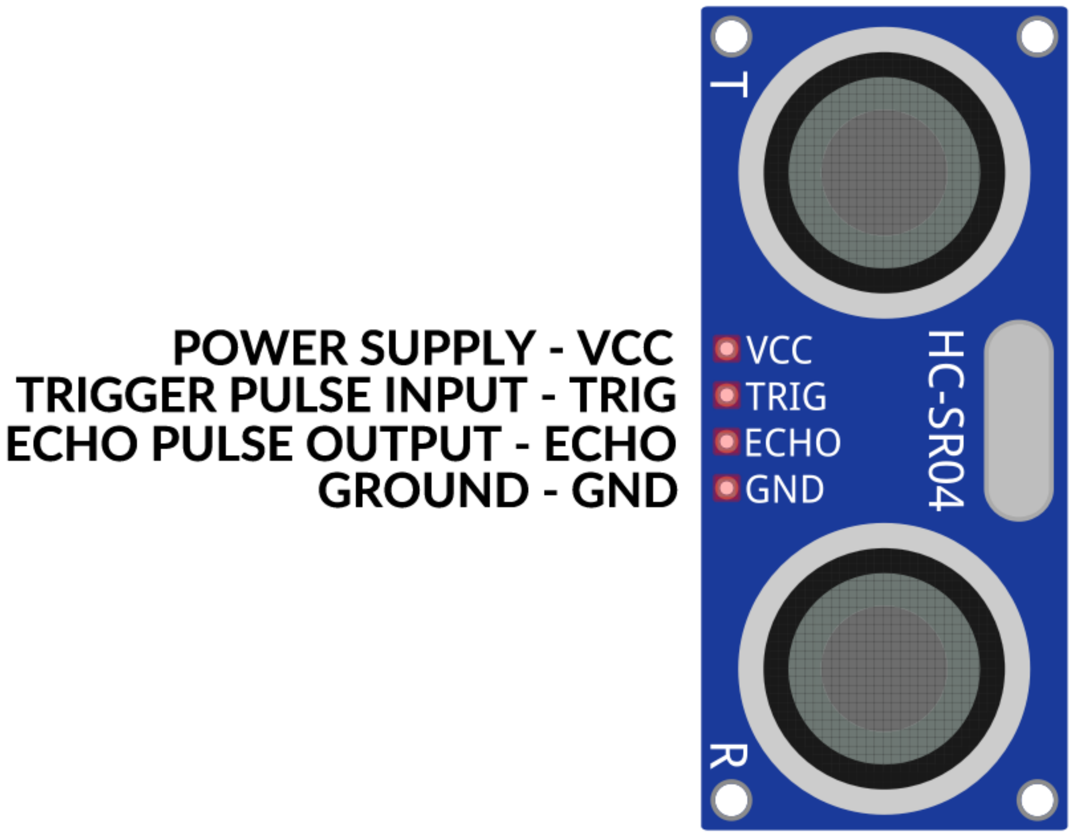
## Specifications

| | |
|---|---|
| Power supply voltage: | up to 5V |
| Operating voltage: | from 3V to 5V |
| Output voltage: | 5V! |
| Current consumption: | 15mA |
| Quiescent current: | less than 2mA |
| Ultrasonic frequency: | 40kHz |
| Trigger input signal: | 10µS TTL pulse |
| Measuring angle: | 30 degree |
| Effectual angle: | less than 15 degrees |
| Operating distance range: | from 20mm to 4000mm (1in to 13 feet) |
| Claimed precision: | 3mm, realistically 10mm |
| Dimensions: | 45 x 20 x 15mm (1.8 x 0.8 x 0.6in) |

The minimal distance on which the measurements are valid is *20mm*. Below this threshold the readings may become unpredictable.

## The pinout

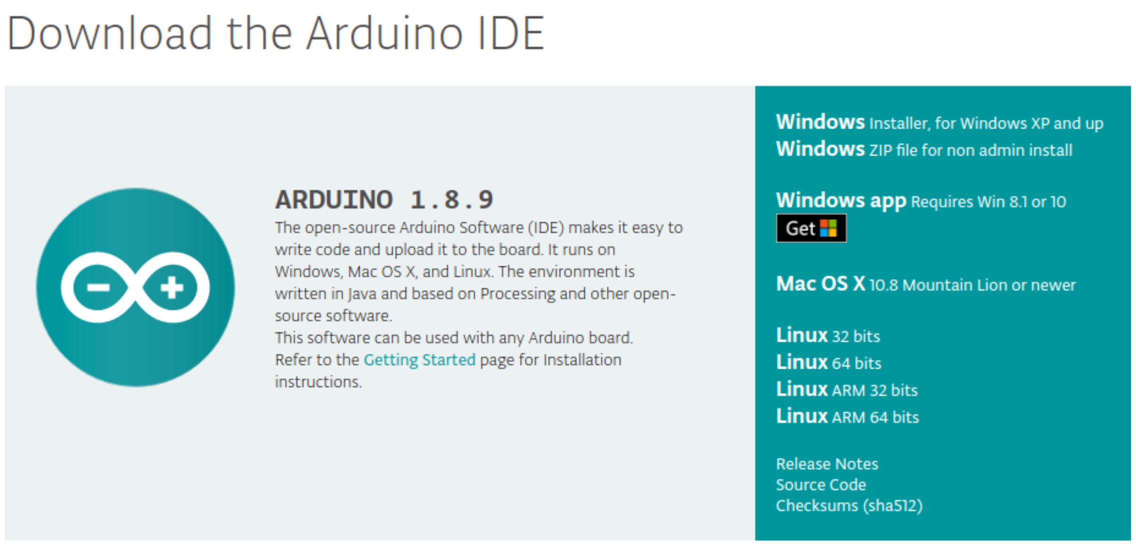
The HC-SR04 module has four pins. The pinout is shown on the following image:



**Note:** The output voltage of the module is in the 5V range. In order to use the module with the Raspberry Pi, the device called logic level converter should be used. Otherwise connecting 5V signals to the GPIO pins may cause damage to the Raspberry Pi. For this purpose use the device called *TXS0108E Logic Level Converter* that AZ-Delivery offers.

## How to set-up Arduino IDE

If the Arduino IDE is not installed, follow the *link* and download the installation file for the operating system of choice.



Download the Arduino IDE

ARDUINO 1.8.9

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

**Windows** Installer, for Windows XP and up
**Windows** ZIP file for non admin install

**Windows app** Requires Win 8.1 or 10
Get

**Mac OS X** 10.8 Mountain Lion or newer

**Linux** 32 bits
**Linux** 64 bits
**Linux** ARM 32 bits
**Linux** ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

For *Windows* users, double click on the downloaded *.exe* file and follow the instructions in the installation window.

For *Linux* users, download a file with the extension *.tar.xz*, which has to be extracted. When it is extracted, go to the extracted directory and open the terminal in that directory. Two *.sh* scripts have to be executed, the first called *arduino-linux-setup.sh* and the second called *install.sh*.

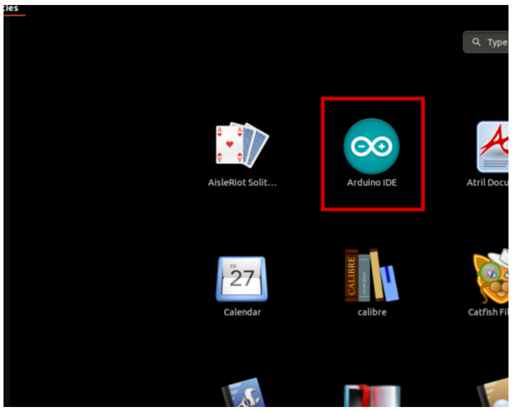
To run the first script in the terminal, open the terminal in the extracted directory and run the following command:

```
sh arduino-linux-setup.sh user_name
```

***user_name*** - is the name of a superuser in the Linux operating system. A password for the superuser has to be entered when the command is started. Wait for a few minutes for the script to complete everything.

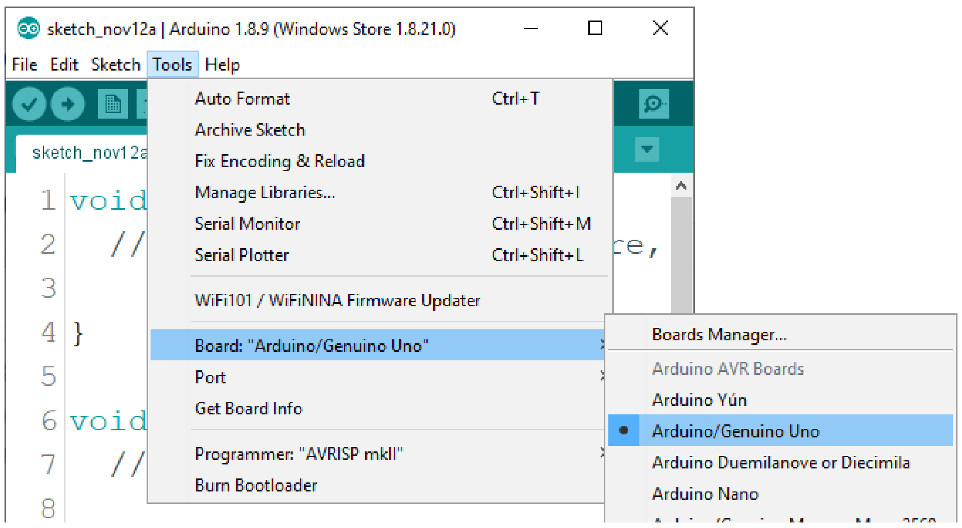
The second script, called *install.sh*, has to be used after the installation of the first script. Run the following command in the terminal (extracted directory): **`sh install.sh`**

After the installation of these scripts, go to the *All Apps*, where the *Arduino IDE* is installed.



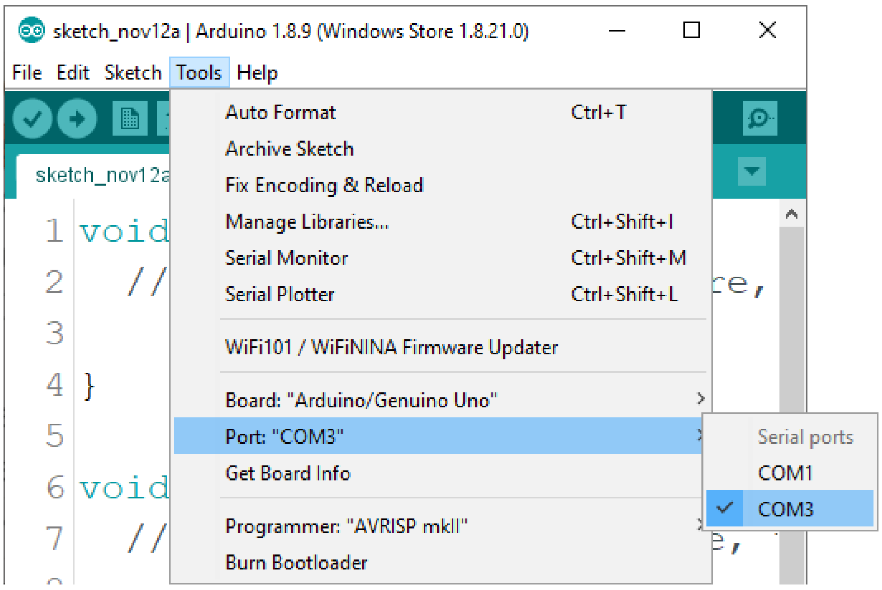
Almost all operating systems come with a text editor preinstalled (for example, `Windows` comes with `Notepad`, `Linux Ubuntu` comes with `Gedit`, `Linux Raspbian` comes with `Leafpad`, etc.). All of these text editors are perfectly fine for the purpose of the eBook.

Next thing is to check if your PC can detect an Arduino board. Open freshly installed Arduino IDE, and go to:
*`Tools > Board > {your board name here}`*
*`{your board name here}`* should be the *`Arduino/Genuino Uno`*, as it can be seen on the following image:


sketch_nov12a | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
sketch_nov12a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Yún
● Arduino/Genuino Uno
Arduino Duemilanove or Diecimila
Arduino Nano

The port to which the Arduino board is connected has to be selected. Go to:
*`Tools > Port > {port name goes here}`*
and when the Arduino board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

If the Arduino IDE is used on Windows, port names are as follows:


sketch_nov12a | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM3"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
✓ COM3

For *Linux* users, for example port name is */dev/ttyUSBx*, where *x* represents integer number between *0* and *9*.

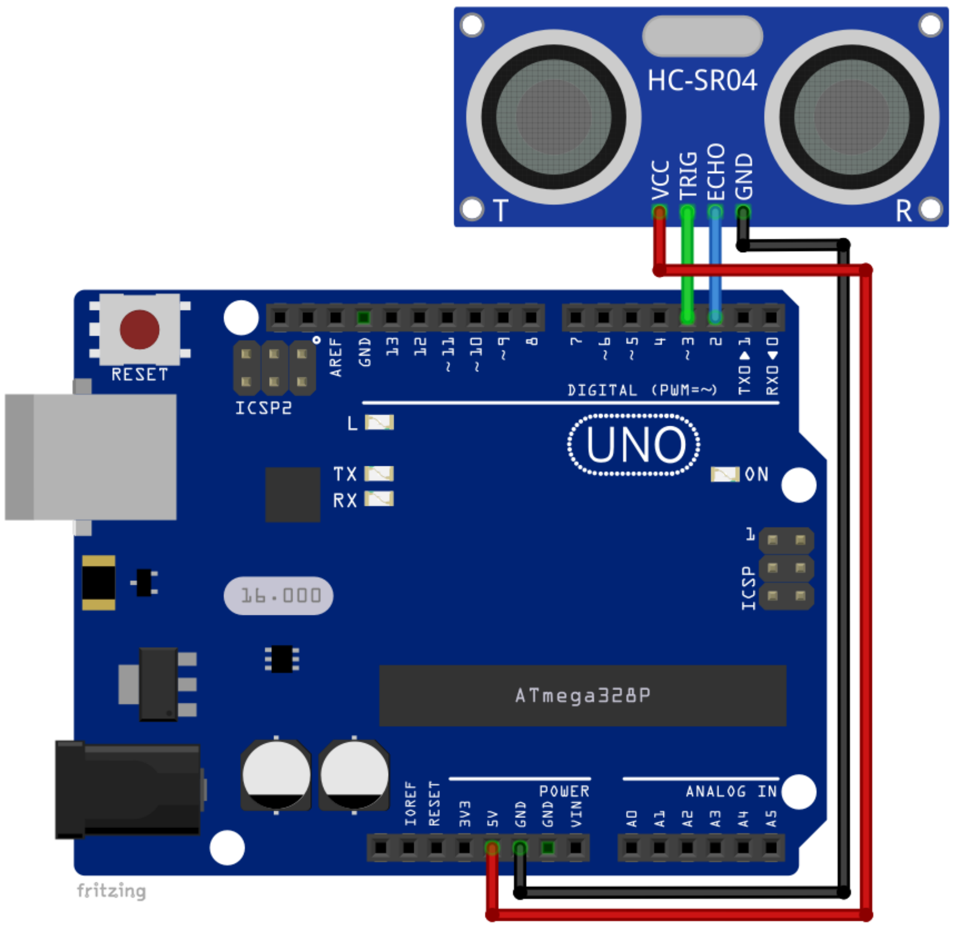
## How to set-up the Raspberry Pi and Python

For the Raspberry Pi, first the operating system has to be installed, then everything has to be set-up so that it can be used in the *Headless* mode. The *Headless* mode enables remote connection to the Raspberry Pi, without the need for a *PC* screen Monitor, mouse or keyboard. The only things that are used in this mode are the Raspberry Pi itself, power supply and internet connection. All of this is explained minutely in the free eBook:
*Raspberry Pi Quick Startup Guide*

The *Raspbian* operating system comes with *Python* preinstalled.

## Connecting the module with Uno

Connect the module with the Uno as shown on the following image:

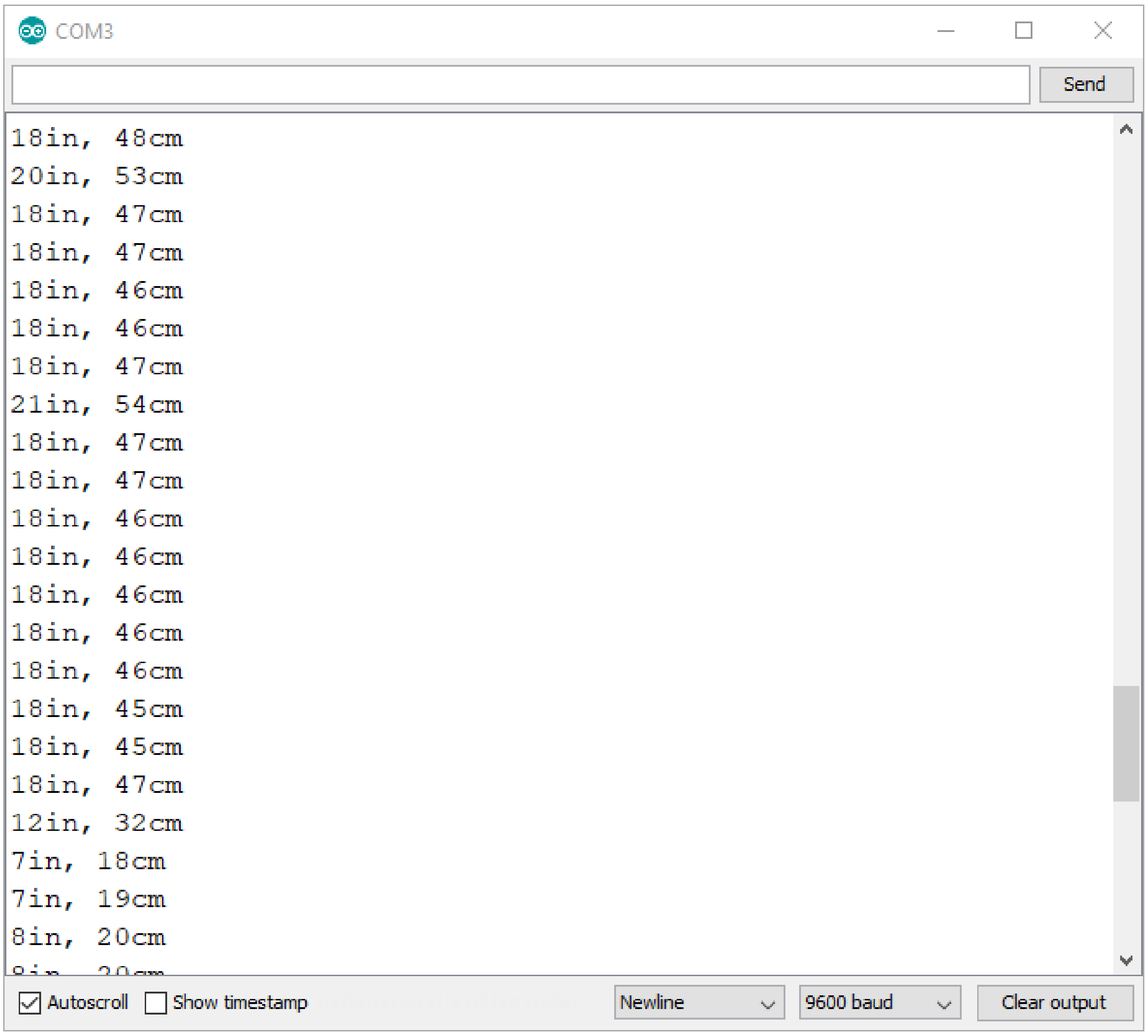


| Module pin | Uno pin | Wire color |
|---|---|---|
| VCC | 5V | **Red wire** |
| TRIG | D2 | **Green wire** |
| ECHO | D3 | **Blue wire** |
| GND | GND | **Black wire** |

## Sketch example

```
#define TRIG_PIN 2
#define ECHO_PIN 3
long duration, cm, inches;

void setup() {
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}
void loop() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  duration = pulseIn(ECHO_PIN, HIGH);

  cm = (duration / 2) / 29.1;   // Divide by 29.1 or multiply by 0.0343
  inches = (duration / 2) / 74; // Divide by 74 or multiply by 0.0135

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(500);
}
```

Upload the sketch to the Uno and run the Serial Monitor (*Tools* > *Serial Monitor*). The result should look like as on the following image:



```
18in, 48cm
20in, 53cm
18in, 47cm
18in, 47cm
18in, 46cm
18in, 46cm
18in, 47cm
21in, 54cm
18in, 47cm
18in, 47cm
18in, 46cm
18in, 46cm
18in, 46cm
18in, 46cm
18in, 46cm
18in, 45cm
18in, 45cm
18in, 47cm
12in, 32cm
7in, 18cm
7in, 19cm
8in, 20cm
```

The sketch starts with creating two macros *TRIG_PIN* and *ECHO_PIN*. These macros represent the digital pins of Uno to which pins of the module are connected.

Next, three *long* variables are created *duration*, *cm* and *inches*. In the *duration* variable the time interval of the measurement is saved. In the *cm* variable the calculated distance in the centimeters is stored. In the *inches* variable the calculated distance in the inches is stored.

In the *setup()* function, serial communication is started with a baud rate of *9600bps*. After this the modes for digital pins are set: *TRIG_PIN* as *OUTPUT* and *ECHO_PIN* as *INPUT*.

At the beginning of the *loop()* function, *TRIG_PIN* is activated in the following manner:
First, the state of the *TRIG_PIN* is held in *LOW* state for *5* microseconds, next it is held in the *HIGH* state for the 10 microseconds and then again it is set in the *LOW* state. This way, the ultrasonic sound wave is transmitted at the specific time interval so that the measurements can take place.

Next, the function *pulseln()* is used to measure the length of the pulse on the *ECHO_PIN*. The function has two arguments and returns a *long* value. The first argument is the name of the pin on which the measurement takes place.

The second argument of the *pulseIn()* function can have two values: *HIGH* or *LOW*. When the value of the second argument is *HIGH*, the function waits for the signal on the *ECHO_PIN* to change its state from *LOW* to *HIGH* to start the measurement. If the value of the second argument is *LOW*, then the function waits for the signal on the *ECHO_PIN* to change its state from *HIGH* to *LOW* to start the measurement. The return value is the *unsigned long* value, which represents the length of the pulse in microseconds.

The following line of code is used to measure the length of the pulse on the *ECHO_PIN*:

```
duration = pulseIn(ECHO_PIN, HIGH);
```

Where the return value from the *pulseIn()* function is stored in the *duration* variable.

Next, the distance calculation is done, with the following lines of code:

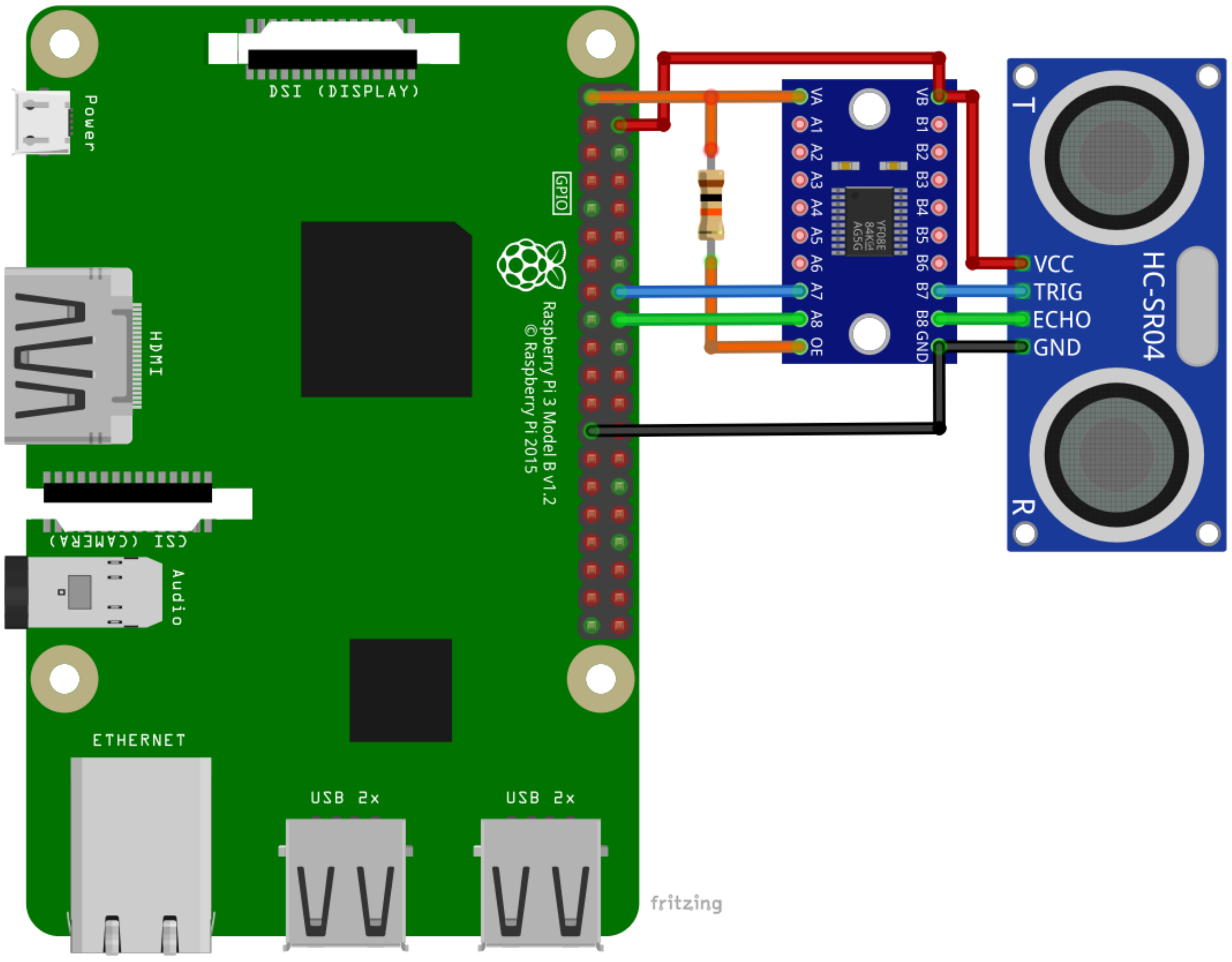
```
cm = (duration / 2) / 29.1;
inches = (duration / 2) / 74;
```

After this, the data is displayed in the Serial Monitor.

At the end of the *loop()* function, a delay pause of a half second is set. This pause represents the pause between two measurements.

## Connecting the module with Raspberry Pi

Connect the module with the Raspberry Pi as shown on the following image:



| Module pin | Logic Level Converter (LLC) pin | Wire color |
|---|---|---|
| VCC | VB | **Red wire** |
| TRIG | B7 | **Blue wire** |
| ECHO | B8 | **Green wire** |
| GND | GND | **Black wire** |

| LLC pin | Raspberry Pi pin | Physical pin | Wire color |
|---|---|---|---|
| VB | 5V | 4 | **Red wire** |
| VA | 3V3 | 1 | **Orange wire** |
| A7 | GPIO23 | 16 | **Blue wire** |
| A8 | GPIO24 | 18 | **Green wire** |
| GND | GND | 25 | **Black wire** |
| OE | 3V3 via 10kΩ * | 1 | **Orange wire** |

* Connect the OE pin of the LLC with 3V3 via 10kΩ resistor (pull-up resistor) to enable the LLC operation.

## Libraries and tools for Python

To use the module with the Raspberry Pi, the library RPi.GPIO has to be installed. If the library is not installed, open the terminal and run the following commands, one by one:

```
sudo apt-get update && sudo apt-get upgrade -y
sudo apt-get install python3-rpi.gpio
```

The example of the original script made by *Matt Hawkins* can be found on the following *link*.

## Python script

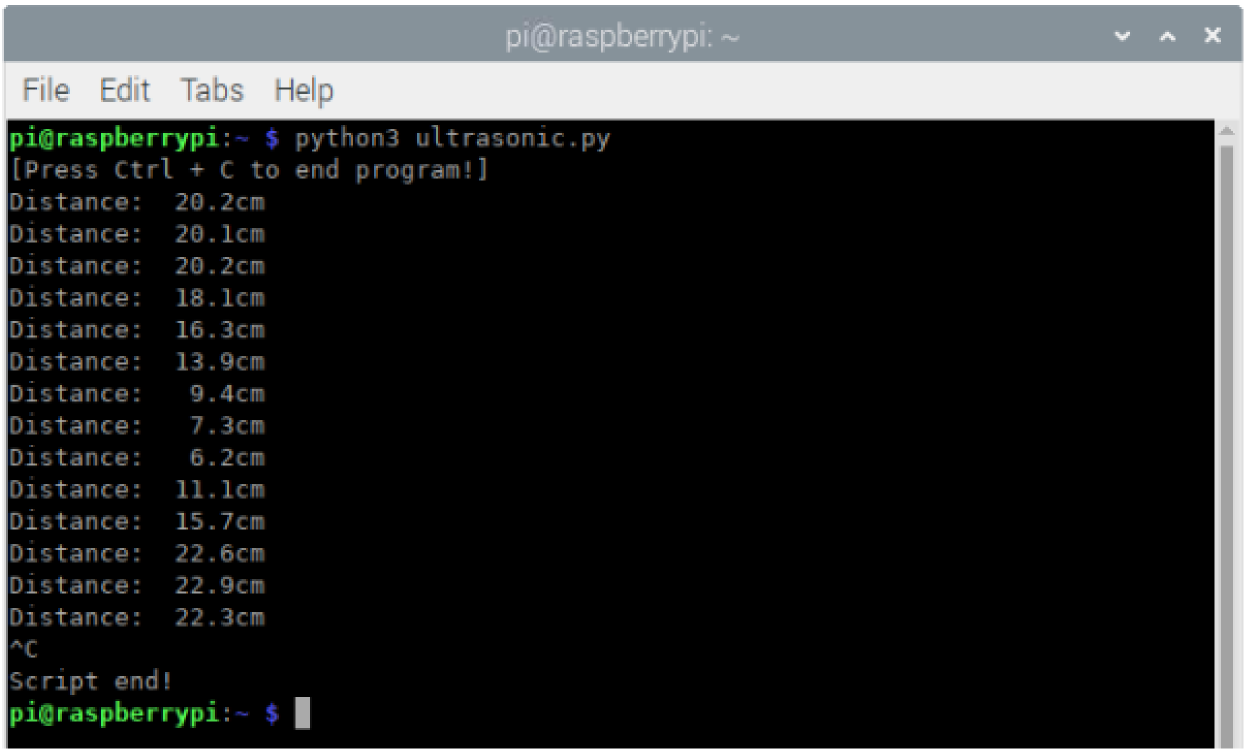
```python
import time
import RPi.GPIO as GPIO

def measure():
  speed_of_sound = 34300
  GPIO.output(TRIG_PIN, True)
  time.sleep(0.00001)
  GPIO.output(TRIG_PIN, False)
  start = time.time()
  while GPIO.input(ECHO_PIN) == 0:
    start = time.time()

  while GPIO.input(ECHO_PIN) == 1:
    stop = time.time()

  elapsed = stop - start
  distance = (elapsed * speed_of_sound) / 2
  return distance

def measure_average():
  distance1 = measure()
  time.sleep(0.1)
  distance2 = measure()
  time.sleep(0.1)
  distance3 = measure()
  distance = distance1 + distance2 + distance3
  distance = distance / 3
  return distance
```

```
GPIO.setmode(GPIO.BCM)

TRIG_PIN = 23
ECHO_PIN = 24
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

print('[Press Ctrl + C to end program!]')
try:
  GPIO.output(TRIG_PIN, False)
  time.sleep(0.5)

  while True:
    distance = measure_average()
    print('Distance: {:5.1f}cm'.format(distance))
    time.sleep(1)

except KeyboardInterrupt:
  print('\nScript end!')

finally:
  GPIO.cleanup()
```

Save the script by the name *`ultrasonic.py`*. To run the script, open the terminal in the directory where the script is saved and run the following command:

**`python3 ultrasonic.py`**

The result should look like as on the following image:



```
pi@raspberrypi:~ $ python3 ultrasonic.py
[Press Ctrl + C to end program!]
Distance:  20.2cm
Distance:  20.1cm
Distance:  20.2cm
Distance:  18.1cm
Distance:  16.3cm
Distance:  13.9cm
Distance:   9.4cm
Distance:   7.3cm
Distance:   6.2cm
Distance:  11.1cm
Distance:  15.7cm
Distance:  22.6cm
Distance:  22.9cm
Distance:  22.3cm
^C
Script end!
pi@raspberrypi:~ $ 
```

To stop the script press ‘CTRL + C’ on the keyboard.

The script starts with importing two libraries: *time* and *RPi.GPIO*.

Next, two functions are created: *measure()* and *measure_average()*. Both functions have no arguments and return the *double* value. The *measure()* function is used to do the distance measurement and calculation. The algorithm for measuring and calculating the distance is in this function. The *measure_average()* function is used to do three measurements and make an average value out of three measurements. The return value of these functions represents the calculated distance value.

Then, the GPIO pin naming is set to BCM, and the GPIO pin modes for *TRIG_PIN* and *ECHO_PIN* are set to output and input, respectively.

After this, the *try-except-finally* block of code is created. In the *try* block of code the state of *TRIG_PIN* is set to LOW, after which the indefinite loop block of code is created (*while True:*).

In the indefinite loop block of code, first, the *measure_average()* function is executed and the return value is stored to the distance variable. Next, the value in the distance variable is displayed in the terminal, with the following line of code:

```
print('Distance: {:5.1f}cm'.format(distance))
```

Where “`5.1f`”: “5” means the output contains 5 decimal places; “`.1`” means that there is one digit after the decimal point; and “`f`” means that the value is of type *`float`*.

To stop the script press ‘CTRL + C’ on the keyboard. This is called the keyboard interrupt. When the keyboard interrupt happens, the *`except`* block of code is executed, displaying message *`Script end!`* in the terminal.

The *`finally`* block of code is executed at the script end. When the *`finally`* block of code is executed, the function called *`cleanup()`* is executed. The *`cleanup()`* function disables all used GPIO interfaces and GPIO pin modes.

Now it is the time to learn and make your own projects. You can do that with the help of many example scripts and other tutorials, which can be found on the Internet.

**If you are looking for the high quality products for Arduino and Raspberry Pi, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.**

https://az-delivery.de

Have Fun!

Impressum

https://az-delivery.de/pages/about-us