

AZ-Delivery

Welcome!

Thank you for purchasing our *AZ-Delivery MQ-2 Gas Sensor Module*. On the following pages, you will be introduced to how to use and set up this handy device.

Have fun!

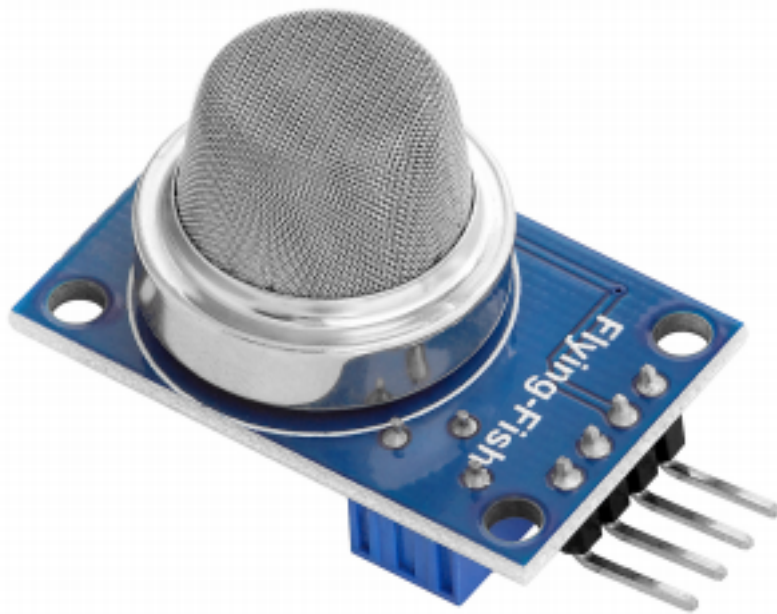




Table of Contents

Introduction	3
Specifications	4
The pinout	5
How to set-up the Raspberry Pi and Python	10
Connecting the module with ATmega328p	11
Sketch example	12
Connecting Nano V3.0 as ADC for Raspberry Pi	17
Connecting the module with Raspberry Pi	21
Python script for MQ-2 module	28



Introduction

The MQ-2 gas sensor module is a device that is used for sensing and measuring the concentration of gases in the air. It can detect such gases as: LPG, propane, methane, hydrogen, alcohol, smoke and carbon monoxide. Though it can detect those gases, it is not able to distinguish the difference between them.

The MQ-2 is a Metal Oxide Semiconductor (MOS), also known as a chemiresistor. The sensor contains a sensing material which resistance changes with different gas concentrations. This change of the resistance is used for gas detection. The sensor also has a built-in potentiometer, with which we can adjust its sensitivity.

The sensor is enclosed within two layers of fine stainless steel mesh called *Anti-explosion network*. As a result of that, it is able to detect flammable gases without incidents. Likewise, it provides protection for the sensor, and it filters out suspended particles. That way, only gases are able to pass inside the sensing chamber.

The module has an on-board LM393 comparator chip which converts the readings into digital and analog signals. There is also a potentiometer which is used to calibrate detection sensitivity.



Specifications

Operating voltage:	5V
Operating current:	150mA
Power consumption:	900mW
Load resistance:	20k Ω
Heater resistance:	33 Ω +5%
Sensing resistance	10k Ω - 60k Ω
Preheat time:	24h
Concentration scope:	200 – 10000ppm (parts per million)
Output:	analog, digital
Dimensions:	33x21x22mm (1.3x0.8x0.9in)

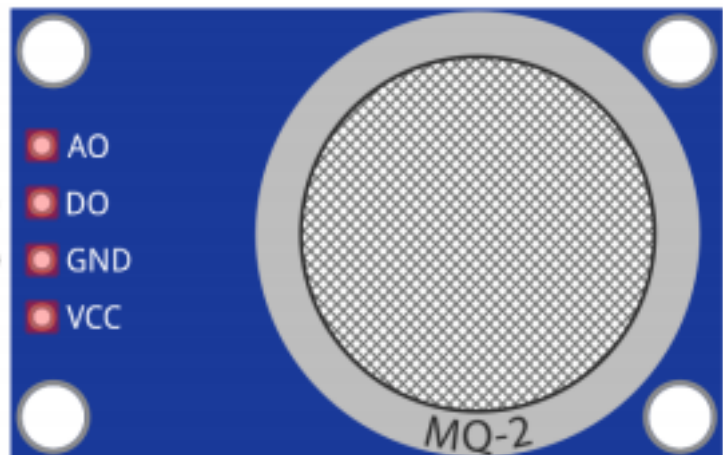
For the best detecting results, gas sensor has to be preheated. The best preheat time for the sensor is above 24 hours. For the detailed information about the sensor specifications, refer to the datasheet.

The module sensitivity can be adjusted with an on-board potentiometer. Moving the potentiometer shaft into the clockwise direction increases the sensitivity. Moving the shaft of the potentiometer in the counterclockwise direction decreases the sensitivity of the module.

The pinout

The gas sensor module has four pins. The pinout is shown on the following image:

ANALOG OUTPUT - A0
DIGITAL OUTPUT - D0
GROUND - GND
POWER SUPPLY - VCC



NOTE: The Raspberry Pi does not have a digital-analog converter and can not be used to read analog voltages.

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



The screenshot shows the Arduino IDE download page. On the left, there is a large teal circle containing the Arduino logo (an infinity symbol with a minus sign on the left and a plus sign on the right). To the right of the logo, the text reads: **ARDUINO 1.8.9**. Below this, it says: "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." On the right side of the page, there is a teal sidebar with the following links: **Windows** installer, for Windows XP and up; **Windows** ZIP file for non-admin install; **Windows app** Requires Win 8.1 or 10; **Get** (with a download icon); **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**; and **Checksums (sha512)**.

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

Az-Delivery

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*.

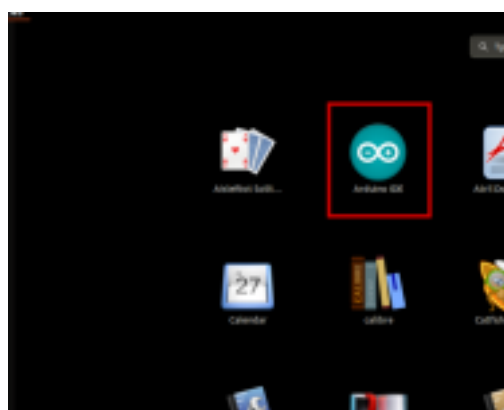
To run the first script in the terminal, open the terminal in the extracted directory and run the following command:

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.



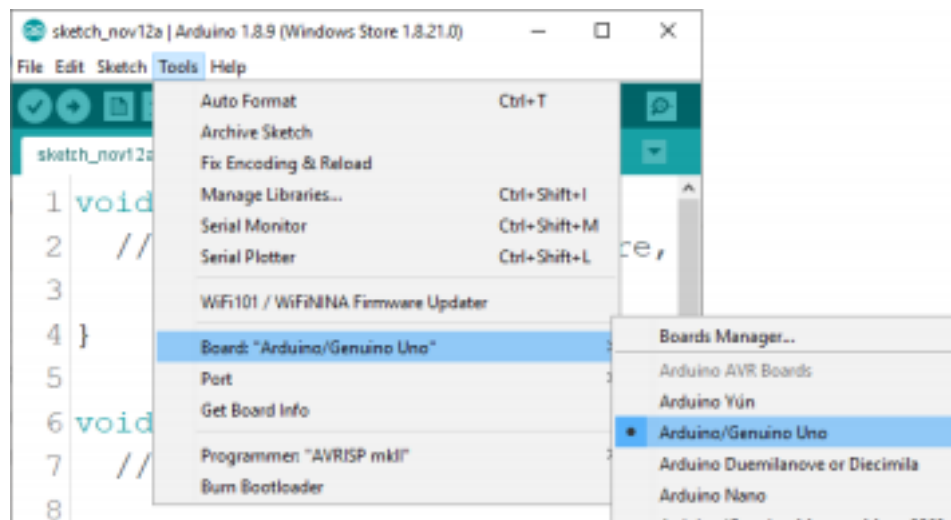
Az-Delivery

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 the microcontroller board. Open freshly installed Arduino IDE, and go to:

Tools > Board > {your board name here}

{your board name here} should be the *Microcontroller/Genuino Uno*, as it can be seen on the following image:



The port to which the microcontroller board is connected has to be selected.

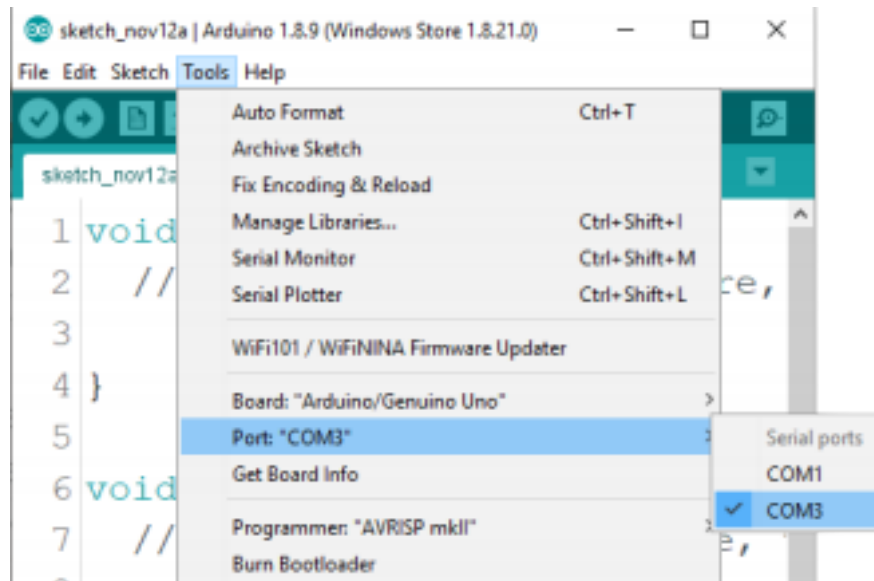
Go to: *Tools > Port > {port name goes here}*

and when the microcontroller board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

Az-Delivery

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

follows:



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:

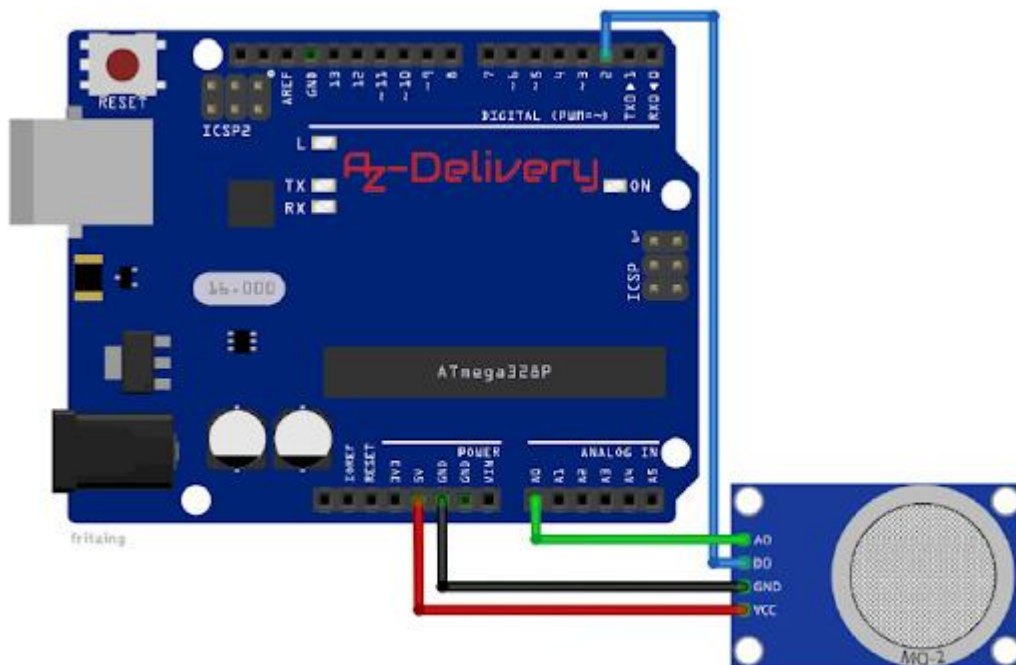
[Raspberry Pi Quick Startup Guide](#)

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

Az-Delivery

Connecting the module with ATmega328p

Connect the module with the ATmega328p as shown on the following image:



Module pin	ATmega 328p pin	Wire color
VCC	5V	Red wire
GND	GND	Black wire
D0	D2	Blue wire
A0	A0	Green wire

Az-Delivery

Sketch example

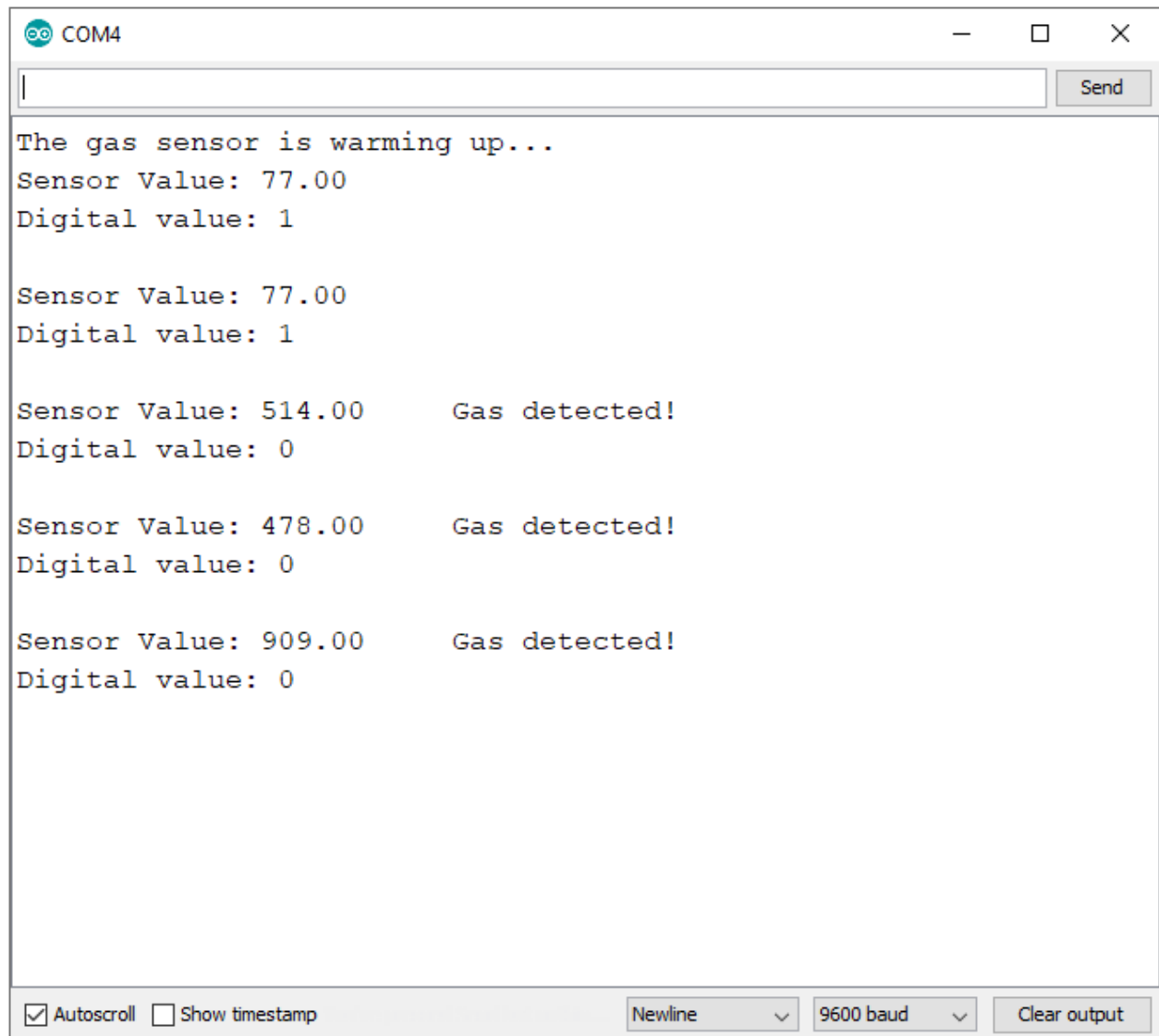
```
#define DIGITAL_PIN 2
#define ANALOG_PIN 0
uint16_t gasVal;
boolean isgas = false;
String gas;
void setup() {
  Serial.begin(9600);
  Serial.println("The sensor is warming up...");
  delay(30000);
  pinMode(DIGITAL_PIN, INPUT);
}
void loop() {

  gasVal = analogRead(ANALOG_PIN);
  isgas = digitalRead(DIGITAL_PIN);

  if (isgas) {
    gas = "No";
  }
  else {
    gas = "Yes";
  }
  gasVal = map(gasVal, 0, 1023, 0, 100);
  Serial.print("Gas detected: ");
  Serial.println(gas);
  Serial.print("Gas percentage: ");
  Serial.print(gasVal);
  Serial.print("%\n");
  delay(2000);
}
```

Az-Delivery

Upload the sketch to the ATmega328p and open Serial Monitor (*Tools > Serial Monitor*). The result should look like as on the following image:





The sketch starts with defining and creating two macros called *DIGITAL_PIN*, *ANALOG_PIN*.

The *DIGITAL_PIN* represents the digital pin of ATmega328p that is used for connecting the digital output pin of the sensor.

The *ANALOG_PIN* represents the analog input pin of ATmega328p that is used for connecting the analog output pin of the sensor.

The module data can be read in two ways. The one is by reading the analog output pin of the module, and the other is by reading the digital output pin of the module. To read the analog output pin of the module, the variable called *gasVal* is used to store return value from the *analogRead()* function. The return value is an integer number in the range from 0 to 1023. To convert it into a percentage, the *map()* function is used. This is a built-in function of the Arduino IDE. It has five arguments and returns an integer value.

Az-Delivery

For example:

```
gasVal = map(input, in_min, in_max, out_min, out_max)
```

First argument is the *input* value, which is in the range from the *in_min* to *in_max*. The return value is an integer number in the range from *out_min* to *out_max*. This function maps one number in the input range, to other number which is in the different range.

To read the digital output pin of the module, the *isGas* variable is used to store the return value of the *digitalRead()* function.

At the end of the *loop()* function, the data is displayed in the Serial Monitor. Between two measurements there is 2 seconds pause: `delay(2000);`



Connecting Nano V3.0 as ADC for Raspberry Pi

Because the Raspberry Pi does not have Analog to Digital Converter (ADC), the task is to make the Raspberry Pi able to read analog voltages. For this purpose ATmega328p or Nano V3.0 can be used. In order to do so, Nano V3.0 has to be connected to the Raspbian operating system. Nano V3.0 can read analog voltages, and it can use Serial Interface via USB port to send data to the Raspberry Pi.

First, the Arduino IDE has to be installed on the Raspbian. Second, the firmware for the microcontroller board needs to be uploaded to Nano V3.0 and Python library has to be downloaded.

To do this, start the Raspbian, open the terminal, and run the following command to update the Raspbian:

`sudo apt-get update && sudo apt-get upgrade -y`

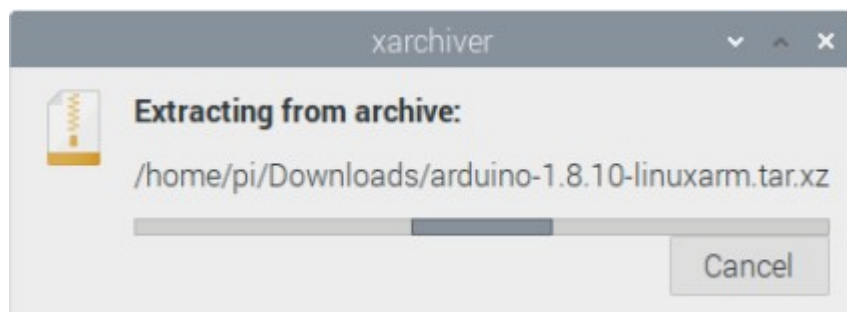
Az-Delivery

To download and install the Arduino IDE, go to the [Arduino site](#): and download the *tar.xz* file of Arduino IDE for **Linux ARM 32 bits** as shown on the following image:

Download the Arduino IDE



Then, extract the *tar.xz* file. Open file explorer in directory where *tar.xz* file is downloaded, right click on it, and run the option Extract Here. Wait for a few minutes for the extracting process to be completed.

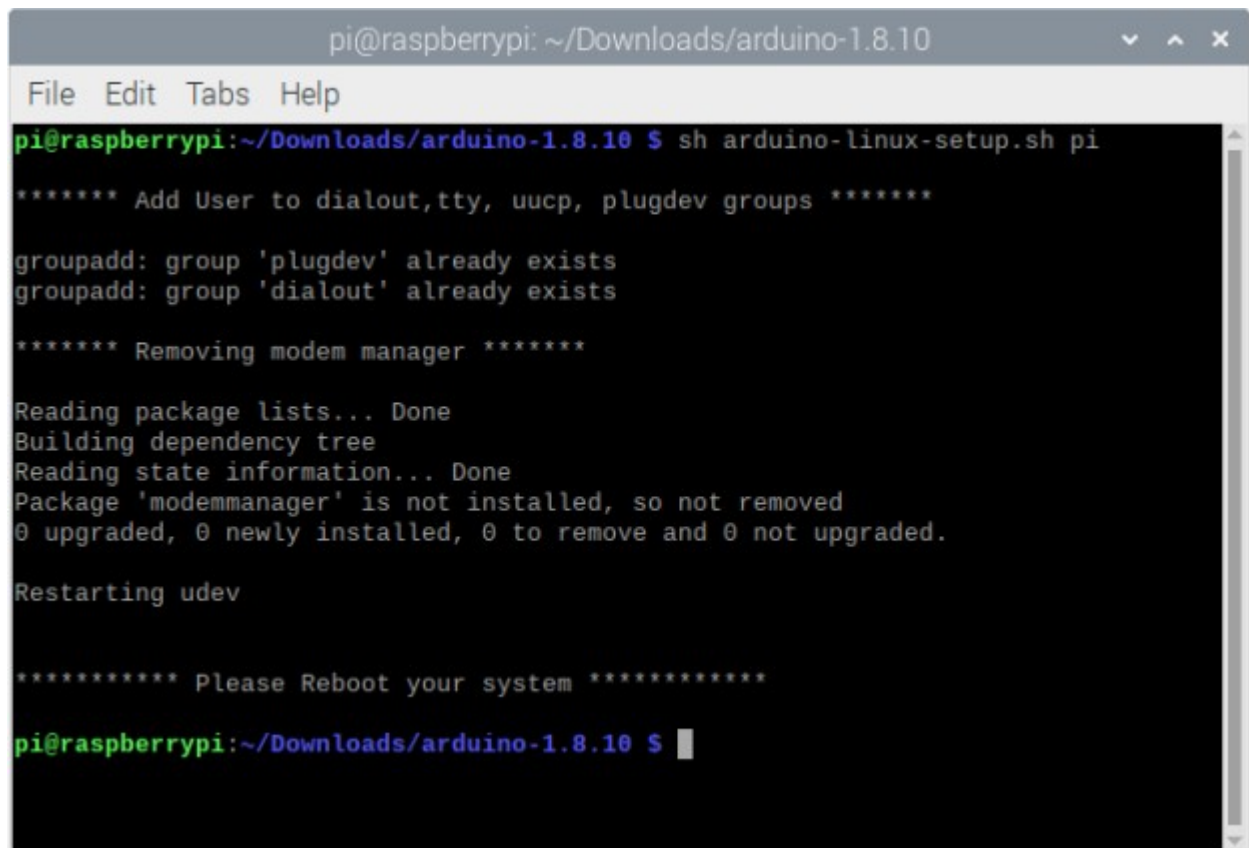


Az-Delivery

Open the terminal in the directory where installation files are extracted and run the following command:

sh arduino-linux-setup.sh pi

where *pi* is the name of the superuser in Raspbian.



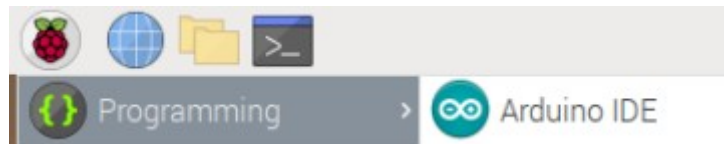
```
pi@raspberrypi: ~/Downloads/arduino-1.8.10
File Edit Tabs Help
pi@raspberrypi:~/Downloads/arduino-1.8.10 $ sh arduino-linux-setup.sh pi
***** Add User to dialout, tty, uucp, plugdev groups *****
groupadd: group 'plugdev' already exists
groupadd: group 'dialout' already exists
***** Removing modem manager *****
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'modemmanager' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Restarting udev

***** Please Reboot your system *****
pi@raspberrypi:~/Downloads/arduino-1.8.10 $
```

After this, to install the Arduino IDE, run the following command: **sudo sh install.sh**

Az-Delivery

The Arduino IDE is now installed. To run Arduino IDE, open the app:
Applications Menu > Programming > Arduino IDE



Before the next steps, first the *pip3* and *git* apps have to be installed;
Open the terminal and run the following command.

sudo apt install python3-pip git -y

The library for Python is called *nanpy*. To install it, open terminal and run the following command: **pip3 install nanpy**

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ pip3 install nanpy
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting nanpy
  Downloading https://www.piwheels.org/simple/nanpy/nanpy-0.9.6-py3-none-any.whl
    (47kB)
    100% |#####| 51kB 209kB/s
Requirement already satisfied: pyserial in /usr/lib/python3/dist-packages (from
nanpy) (3.4)
Installing collected packages: nanpy
Successfully installed nanpy-0.9.6
pi@raspberrypi:~/Scripts $
```

Az-Delivery

After installation of the *nanpy* library, download a firmware by running the following command:

```
git clone https://github.com/nanpy/nanpy-firmware.git
```

Change the directory to *nanpy-firmware* by running the following command:

```
cd nanpy-firmware
```

And run the following command:

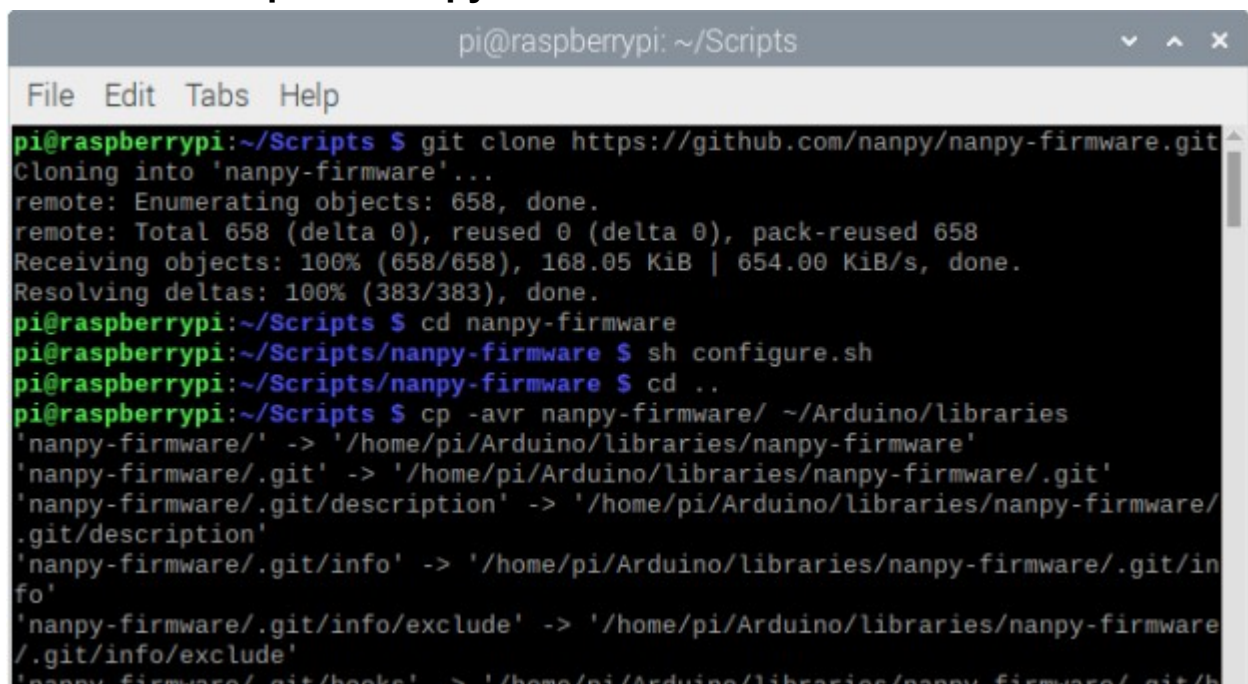
```
sh configure.sh
```

Next, copy the *nanpy-firmware* directory into:

Arduino/libraries directory.

To do so, run the following command:

```
cp -avr nanpy-firmware/ ~/Arduino/libraries
```

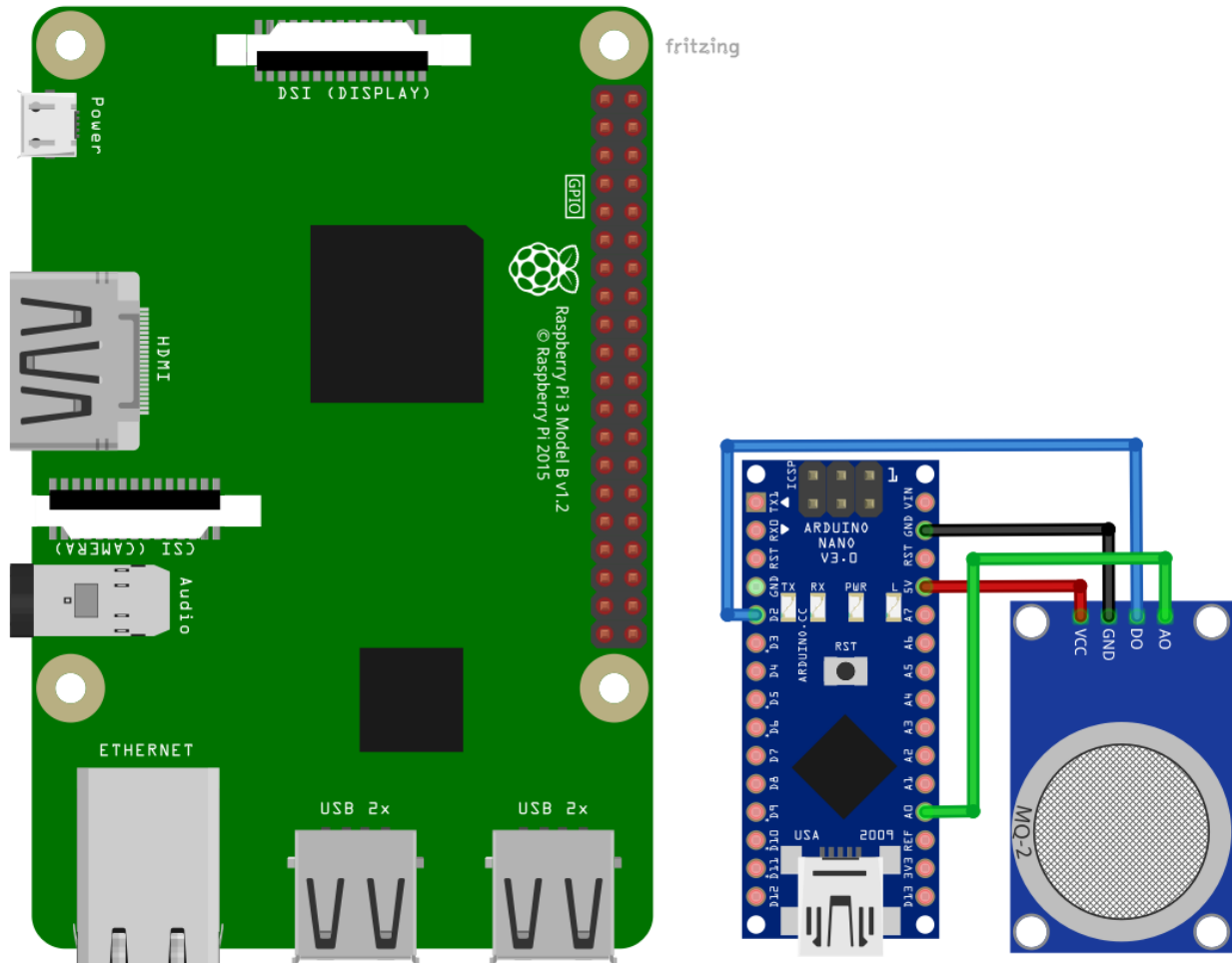
A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window shows the execution of several commands to install the nanpy-firmware. The commands and their outputs are: 1. 'git clone https://github.com/nanpy/nanpy-firmware.git' which clones the repository into a directory named 'nanpy-firmware'. 2. 'cd nanpy-firmware' which changes the current directory to the newly created one. 3. 'sh configure.sh' which runs a shell script. 4. 'cd ..' which changes the directory back to the parent directory. 5. 'cp -avr nanpy-firmware/ ~/Arduino/libraries' which copies the entire nanpy-firmware directory and its contents into the Arduino libraries directory. The output of the last command shows a recursive copy of the directory structure, including .git files and folders like 'description', 'info', and 'info/exclude'.

The *nanpy-firmware* is now installed and ready to be used.

Az-Delivery

Connecting the module with Raspberry Pi

Connect the module with the Nano V3.0 as shown on the following image:



Module pin	Nano V3.0 pin	Wire color
VCC	5V	Red wire
GND	GND	Black wire
D0	D2	Blue wire
A0	A0	Green wire



Next, connect the Nano V3.0 via USB cable to the Raspberry Pi and open the Arduino IDE in the Raspbian operating system. Check if Arduino IDE can detect the USB port on which the Nano V3.0 is connected: *Tools > Port > dev/ttyUSB0*

Then, go to: *Tools > Board > {board name}* and select *Nano V3.0* board.

After that, to open a sketch for the *nanpy-firmware*, go to: *File > Examples > nanpy-firmware > Nanpy*

Upload the sketch to the Nano V3.0. To test if everything works properly, the simple *Blink* script has to be created, where the on-board LED of the Nano V3.0 is used to blink.

Create the *Blink.py* script, and open it in the default text editor.

Az-Delivery

In the *Blink.py* script write the following lines of code:

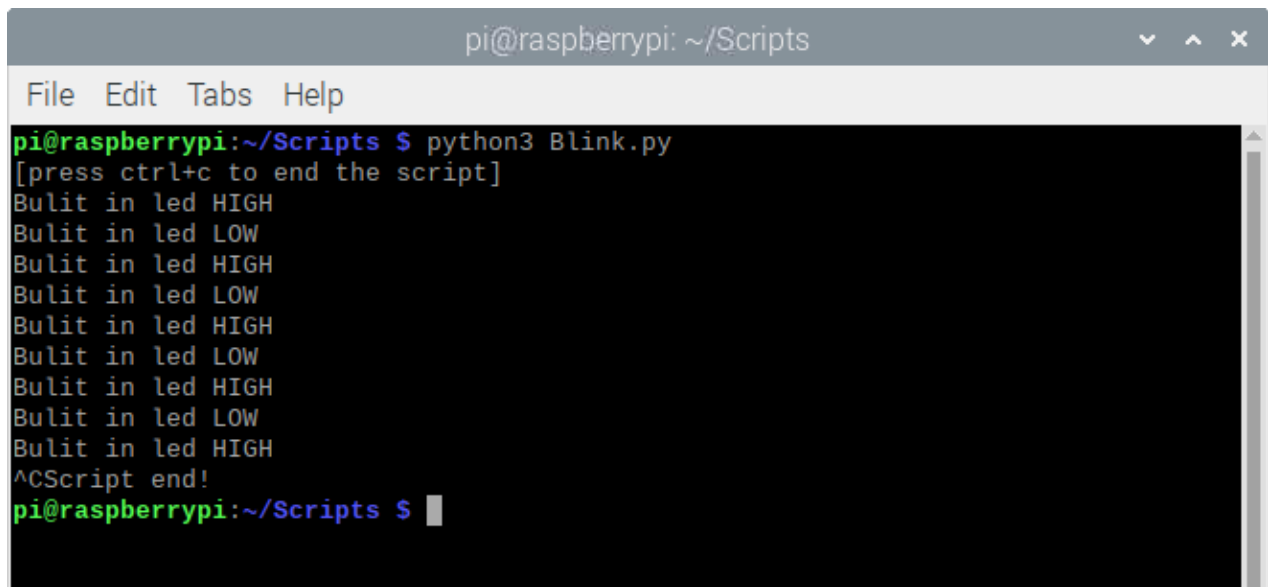
```
from nanpy import (ArduinoApi, SerialManager) from time
import sleep
ledPin = 13
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1) except:
    print('Failed to connect to the Arduino') print('[Press CTRL
+ C to end the script!!]') a.pinMode(ledPin, a.OUTPUT) #
Setup Arduino try:
    while True:
        a.digitalWrite(ledPin, a.HIGH)
        print('Bulit in led HIGH')
        sleep(1)
        a.digitalWrite(ledPin, a.LOW)
        print('Bulit in led LOW')
        sleep(1)

except KeyboardInterrupt:
    print("\nScript end!")
    a.digitalWrite(ledPin, a.LOW)
```

Az-Delivery

Save the script by the name *Blink.py*. To run the script, open the terminal in the directory where the script is saved and run the following command:
python3 Blink.py

The result should look like as on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 Blink.py
[press ctrl+c to end the script]
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press 'CTRL + C' on the keyboard.

The LED connected to the digital pin 13 of the Nano V3.0 should start blinking every second.

Az-Delivery

The script starts with importing two libraries, the *nanpy* library functions, and the *time*.

Then, the variable called *ledPin* is created and initialized with number 13. The number 13 represents the number of the digital pin on which LED is connected (on-board LED of the Nano V3.0).

After that, the *try-except* block of code is used to try and connect to the Nano V3.0. If connection is not successful, message: *Failed to connect to the* microcontroller board is displayed in the terminal.

If connection is successful, a communication object called "a" is created and initialized. The object "a" represents the Nano V3.0 board. Any function used in the Arduino IDE can be used with the "a" object, as it can be seen in the code.

With the following line of code, the pin mode is set-up for the digital pin 13: `a.pinMode(ledPin, a.OUTPUT)`

Then, in the indefinite loop block (*while True:*) the function *digitalWrite()* is used to set the state of the digital pin 13 (HIGH or LOW state). With the *digitalWrite()* function the LED connected to the pin 13 can be turned ON or OFF.

Az-Delivery

In the indefinite loop block, the LED is first turned *ON* for a second, and then turned *OFF* for a second. This is called *blinking the LED*. The time interval of a single blink can be changed in the following line of code: *sleep(1)*

Where number *1* represents the number of seconds for the duration of the time interval.

To end the indefinite loop, press 'CTRL + C' on the keyboard. This is called the keyboard interrupt, which is set in the *except* block (*except KeyboardInterrupt*). In the *except* block the on-board LED is turned OFF.



Python script for MQ-2 module

```
from nanpy import (ArduinoApi, SerialManager)
import time

try:
    connection_1 = SerialManager()
    a = ArduinoApi(connection=connection_1)
except:
    print('Failed to connect to the Arduino')
    DIGITAL_PIN = 2
    ANALOG_PIN = 0
    time.sleep(2)
    print('Sensor is warming up...')
    print('[Press CTRL+C to end the script]')
    time.sleep(5) # Sensor warming up...

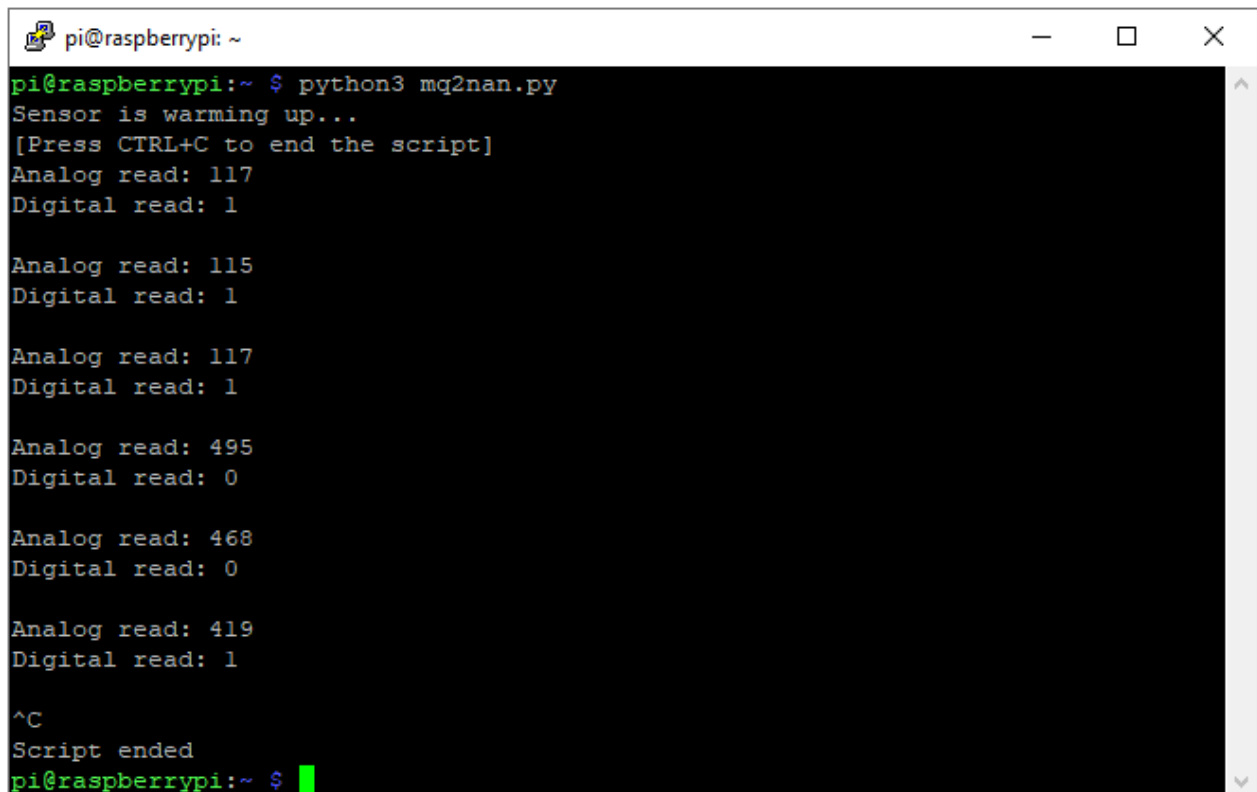
try:
    while True:
        analogReading = a.analogRead(ANALOG_PIN)
        digitalReading = a.digitalRead(DIGITAL_PIN)
        print("Analog read: {}\nDigital read: {}".format(analogReading, digitalReading))
        time.sleep(2)
except KeyboardInterrupt:
    print("\nScript end!")
```

Az-Delivery

Save the script by the name *mq2nan.py*. To run the script, open the terminal in the directory where the script is saved and run the following command:

python3 mq2nan.py

The result should look like as on the following image:

A terminal window titled 'pi@raspberrypi: ~' with standard window controls. The terminal shows the execution of 'python3 mq2nan.py'. The output includes a warming-up message, a prompt to press CTRL+C, and several pairs of 'Analog read' and 'Digital read' values. The script ends with '^C' and 'Script ended', returning to the shell prompt.

```
pi@raspberrypi:~ $ python3 mq2nan.py
Sensor is warming up...
[Press CTRL+C to end the script]
Analog read: 117
Digital read: 1

Analog read: 115
Digital read: 1

Analog read: 117
Digital read: 1

Analog read: 495
Digital read: 0

Analog read: 468
Digital read: 0

Analog read: 419
Digital read: 1

^C
Script ended
pi@raspberrypi:~ $
```

To stop the script press 'CTRL + C' on the keyboard.



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 microelectronics and accessories, 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>