Arduino & Environmental Monitoring:

Tapping into the Data Stream of Nature

NatureMetric

Zitong Zhou

zitong.zhou@mail.polimi.it

THEORY IN CONTEMPORARY ARCHITECTURAL DESIGN

Professor - Cattaneo Elisa Cristiana

Politecnico di Milano

Content

1. Introduction to Arduino

1.1 Overview of Arduino

1.1.1 History and Evolution of Arduino

1.1.2 Arduino System

1.1.3 Different Models of Arduino boards

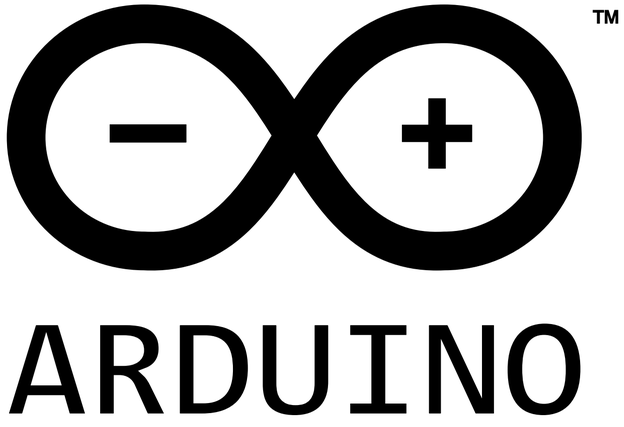
1.2 Advantages of Arduino

1.2.1 Why Arduino for Environmental Observation

1.2.2 What Arduino Can do

This chapter introduces the Arduino platform, detailing its history, evolution, and various models available. It explains why Arduino is particularly suited for environmental observation, highlighting its role in environmental monitoring with real-world examples and case studies. This sets the stage for understanding Arduino's capabilities and applications in environmental observation.

History and Evolution of Arduino



Arduino is an open-source electronics platform that has revolutionized the way people approach hardware and software integration, especially in the fields of DIY projects, educational purposes, and prototyping. Originating in Italy in 2005, the platform is designed with simplicity in mind, making it accessible for hobbyists, artists, and beginners, yet versatile enough for complex applications.

At the heart of the Arduino platform is the Arduino board, a microcontroller-based kit that can be programmed to interact with the physical world. These boards come in various models, each differing in size, capabilities, and complexity, catering to a wide range of applications. The most common Arduino board, the Arduino Uno, features digital and analog input/output (I/O) pins that can be connected to a variety of sensors, actuators, and other electronic components.

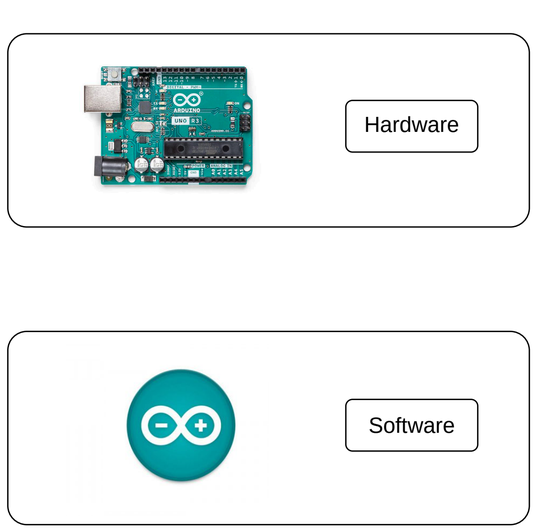
One of Arduino's most significant strengths is its user-friendly Integrated Development Environment (IDE), which offers a straightforward way to write and upload code to the board. The IDE uses a simplified version of C++, making programming more approachable for beginners. This software provides a perfect platform for learning the fundamentals of coding and electronics.

Arduino's versatility extends to its vast community and expansive library of open-source code. The community plays a pivotal role in the platform's development, offering support, sharing projects, and contributing to a rich repository of sketches (Arduino programs), libraries, and tutorials. This communal aspect not only accelerates learning but also fosters innovation and collaboration.

The applications of Arduino are incredibly diverse. In education, it serves as an invaluable tool for teaching programming and electronics. Its simplicity enables students to quickly grasp basic concepts and move on to more complex projects. In the realm of DIY projects, Arduino has been used for home automation, robotics, environmental monitoring, and much more. It's also found applications in professional settings, such as prototyping and product development, due to its low cost and flexibility.

What sets Arduino apart is its adaptability. With the addition of shields – expansion boards that plug into the main Arduino board – the capabilities of the Arduino can be significantly extended. These shields add functionalities like motor control, GPS, wireless communication, and even touch screen support.

Arduino System



Open source is source code that is made freely available for possible modification and redistribution.

This means that any person or company can produce Arduino products and improve and customize them on the original version.

The official original Arduino development board, all details and technical specifications are from the official Arduino

For Arduino development boards customized by third-party manufacturers, the technical details will be slightly adjusted based on the official ones. You need to check the specifications provided by the manufacturer.

Different Models of Arduino boards

Introduction to the Arduino Family

The Arduino family is diverse, with each board designed to suit different needs and skill levels. Key members include:

1. Arduino Uno:
   A great starter board, known for its user-friendliness.

2. Arduino Mega:
   Offers more I/O pins and memory, suitable for larger projects.

3. Arduino Leonardo:
   Unique in its ability to emulate a computer keyboard/mouse.

4. Arduino Nano:
   Compact form factor, ideal for space-limited applications.

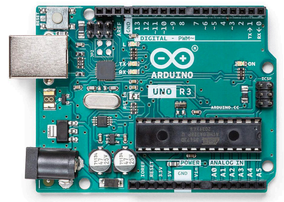
5. Arduino Micro:
   Similar to the Nano, but with a few differences in pin layout.

6. Arduino Due:
   Features a more powerful ARM processor, suited for intensive tasks.

7. Arduino Lilypad:
   Designed for wearable technology and e-textiles.

Each board caters to different aspects of project development, from size constraints to processing power. However, for environmental analysis, due to the need to cooperate with many environmental sensors, it's crucial to find a balance between functionality, ease of use, and community support.
w

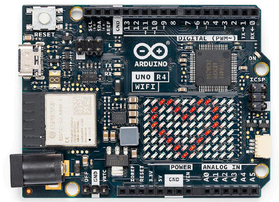
Newer version, but many libraries are incompatible



Arduino UNO R3

Most common and mature

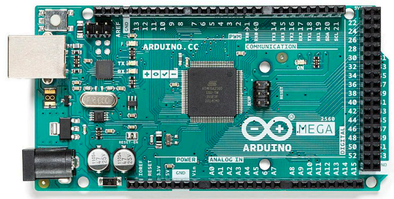
Released in 2012



Arduino UNO R4 / R4 WiFi

Nest Gen Arduino

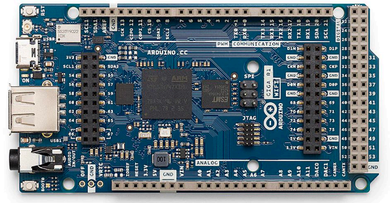
Released in 2023



Arduino Mega 2560

Bigger Arduino UNO R3

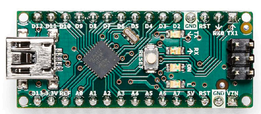
Released in 2010



Arduino GIGA R1 WiFi

The most powerful Arduino

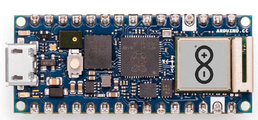
Released in 2023



Arduino Nano

Smaller Arduino UNO R3

Released in 2008



Arduino Nano 33 IoT

Smaller ArduinoUNO R4

Released in 2021

Processor: **AVR** architecture

Processor: **ARM** architecture

Why Choose Arduino for Environmental Observation

The Role of Arduino in Environmental Monitoring

Arduino has become an indispensable tool in the realm of environmental monitoring due to its versatility, accessibility, and ease of use. Here's an in-depth look at why Arduino stands out in this field:

1. Versatility and Flexibility: Arduino boards are capable of interfacing with a wide array of sensors used for measuring environmental parameters such as temperature, humidity, air quality, and more. This versatility allows for the development of a range of environmental monitoring systems tailored to specific needs.

2. Open-source Platform: Arduino's open-source nature fosters a collaborative environment where educators, students, and hobbyists can share their designs and code, accelerating innovation and learning in environmental observation.

3. Ease of Use: With its user-friendly programming environment and simple programming language based on C++, Arduino makes it possible for people with minimal programming or electronics experience to develop complex environmental monitoring systems.

4. Cost-Effectiveness: Arduino provides a low-cost solution compared to other microcontrollers and computing platforms, making it accessible for educational purposes, small-scale projects, and NGOs working in environmental conservation.

5. Real-Time Data Acquisition and Processing: Arduino's ability to process and analyze sensor data in real-time makes it ideal for monitoring environmental changes as they occur.

6. Portable and Low Power Consumption: Many Arduino boards are compact and can be run on batteries, making them suitable for field deployment in remote environmental monitoring stations.

Real-world Examples and Case Studie

Community Air Quality Monitoring: In many urban areas, citizen groups have used Arduino-based systems to collect data on air quality. These systems often include sensors for pollutants like NO2, CO2, and particulate matter, providing data that complements official monitoring stations and increases spatial coverage.

Wildlife Conservation Projects: Arduino has been employed in various wildlife conservation projects. For instance, sensors connected to Arduino boards have been used to monitor water quality in habitats critical to endangered species, providing vital data to support conservation efforts.

Weather Stations in Schools: Educational institutions have adopted Arduino to teach students about environmental science. Students build weather stations equipped with sensors for temperature, humidity, and atmospheric pressure, fostering hands-on learning about climate and weather patterns.

Soil Monitoring for Sustainable Agriculture: Arduino-based systems are being used by farmers and researchers to monitor soil moisture and temperature, helping in efficient water usage and understanding the microclimate of agricultural fields.

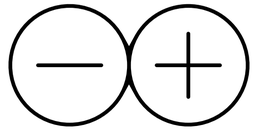
Forest Fire Detection Systems: In forest management, Arduino systems equipped with smoke and heat sensors provide early warnings of forest fires, allowing quicker response to prevent large-scale damage.

Urban Greenhouse Projects: In urban agriculture, Arduino-controlled environments regulate factors like light, temperature, and humidity, creating optimal conditions for plant growth in greenhouses.

Arduino's role in environmental observation is multifaceted and ever-expanding. Its application in real-world scenarios showcases its capability not only as a tool for learning and innovation but also as a driver of impactful environmental projects. From grassroots community initiatives to academic research and practical applications in conservation and agriculture, Arduino is proving to be an invaluable asset in our collective effort to monitor and protect the environment.

1.2 Advantages of Arduino

What Arduino can do



Arduino

Temperature

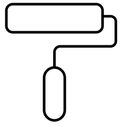
Humidity

Carbon monoxide
(CO)

Carbon dioxide
(CO2)

Ozone
(O3)

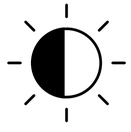
Ammonia
(NH4)



Formaldehyde
(H2CO)

Particulate matter
(PM)

Ultraviolet
(UV Index)

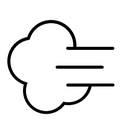


Brightness

Atmospheric pressure

Sound

Air Quality



Smoke

Motion

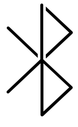
Internet of Things

Soil

Water

Geomagnetism

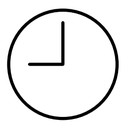
Storage



Bluetooth

Wi-Fi

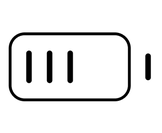
Internet of Things



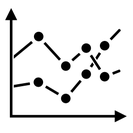
Time

Positioning
(GPS)

Display



Battery



Chart

Dashboard

Print

Here, readers are guided through the initial setup process, including a detailed list of required materials and tools, along with recommendations for sourcing these components. This chapter also includes a comprehensive step-by-step guide to installing the Arduino Integrated Development Environment (IDE), accompanied by an overview of its interface, to familiarize readers with the Arduino programming environment.

2. Setting Up Arduino Environment

2.1 Getting the Right Equipment

2.1.1 Arduino UNO R3

2.1.2 Arduino UNO R3 - 3rd Party

2.1.3 List of Required Materials and Tools

2.1.4 Recommendations for Purchasing

2.2 Installing the Arduino IDE

2.2.1 Step-by-step Installation Guide

2.2.2 Overview of the Arduino IDE Interface

2.2.3 Features of the Arduino IDE

Arduino UNO R3

Among all the Arduino variants, the Uno R3 emerges as the most well-rounded choice for several reasons:

Ease of Use: Its straightforward design makes it incredibly approachable for beginners. The Uno R3's layout and functionality are easy to grasp, making it ideal for those new to electronics and programming.

Compatibility: The Uno R3 enjoys widespread compatibility with a multitude of shields and components. This compatibility opens up a world of possibilities for projects without the need for complex adaptations.

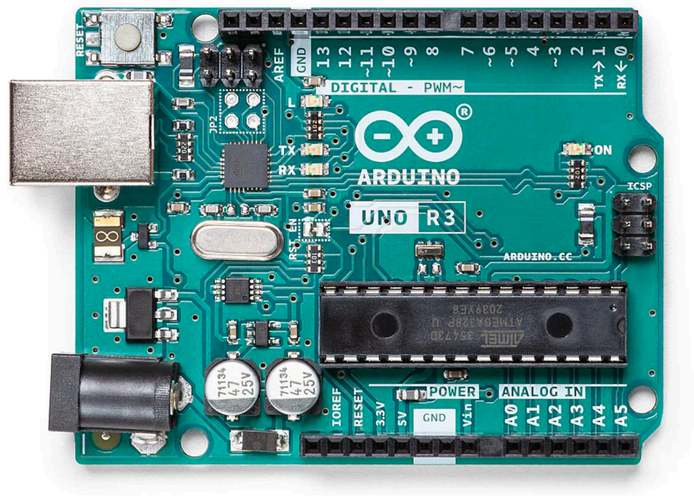
Community Support: The Arduino Uno R3 is the most popular model within the Arduino community. This popularity translates into an abundance of shared knowledge, tutorials, and troubleshooting guides, which are invaluable resources for students.

Educational Resource Availability: Given its popularity, the Uno R3 has a wealth of educational materials specifically tailored to it. From textbooks to online courses, the resources available make the learning process more structured and accessible.

Cost-Effectiveness: While offering a range of features, the Uno R3 remains affordable. This cost-effectiveness is crucial in an educational setting, where budgets may be a concern.

Versatility: The Uno R3 strikes a perfect balance between functionality and simplicity. It can handle a wide array of projects, from simple LED displays to more complex robotic controls, making it a versatile tool for learning and experimentation.

While the Arduino family offers a variety of boards suitable for different applications, the Arduino Uno R3 stands out for educational use. Its ease of use, broad compatibility, extensive community support, rich educational resources, cost-effectiveness, and versatility make it ideal for environmental monitoring and driving sensors. By choosing Uno R3, everyone will have access to a tool that can grow with their expanding skill sets and project complexity.



Arduino UNO R3

Arduino UNO R3 - 3rd Party

The emergence of third-party Arduino Uno R3 development boards is a direct consequence of the Arduino project's open-source nature. Arduino's designs, including its hardware (schematics and layout) and software (bootloader, IDE), are released under open-source licenses. This openness has encouraged a diverse range of manufacturers to produce their own versions of Arduino boards, including the popular Uno R3.

Reasons for the Emergence of 3rd Party Arduino Boards

1.Open-Source Philosophy: Arduino's open-source ethos allows anyone to modify and reproduce its designs. This has led to a proliferation of third-party boards that are either replicas or variations of the original.
2.Cost Reduction: Manufacturers often aim to provide more cost-effective solutions. By optimizing production processes or using different component suppliers, they can offer lower-priced alternatives to the official boards.
3.Innovation and Diversification: Some third-party boards include additional features or design tweaks to differentiate themselves from the original boards or to cater to specific user needs.

Advantages of 3rd Party Arduino Boards

1.Cost-Effectiveness: Often, these boards are less expensive than the official Arduino boards, making them more accessible, especially for students, hobbyists, or large-scale projects.
2.Variety and Specialization: Some third-party boards offer unique features such as additional hardware interfaces, different form factors, or built-in functionalities like Wi-Fi or Bluetooth.
3.Experimentation and Customization: They provide an opportunity for users to experiment with different hardware configurations and customizations.

Disadvantages of 3rd Party Arduino Boards

1.Variable Quality: The quality of third-party boards can be inconsistent. Some might match or exceed the quality of the official boards, while others might have issues like poor soldering, substandard components, or less rigorous quality control.
2.Compatibility Issues: While most aim for compatibility with official Arduino boards, there can be subtle differences that might cause issues with certain shields, accessories, or software.
3.Lesser Community and Official Support: The official Arduino Uno R3 benefits from extensive community support, documentation, and resources. Third-party boards might not have the same level of support, which can be a disadvantage for beginners.
4.Ethical Considerations: Some users might prefer to support the official Arduino project due to its role in fostering the open-source hardware movement, opting to purchase official boards to contribute to its sustainability.

3rd Party Arduino UNO R3 Boards List

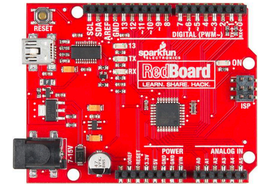
1.Elegoo UNO R3 Board

Features: Almost identical to the official Uno R3 but often more affordable. It uses the same ATmega328P chip and has the same I/O layout.
Differences: Primarily in branding and cost. Sometimes, these boards might have slight variations in component quality.

2.Adafruit Metro 328

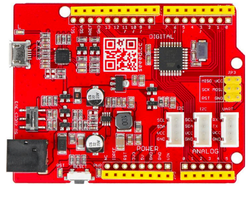
Features: Similar to the Arduino Uno R3 but with some quality-of-life improvements like labeled pins and a larger power regulator.
Differences: Adafruit adds its own touch to the design, ensuring it's a bit more user-friendly, especially for beginners. The build quality is often considered higher.

3.SparkFun RedBoard



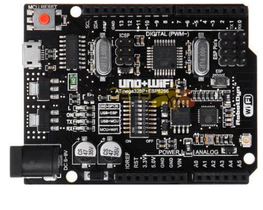
Features: Utilizes the same ATmega328P processor. It is designed to be as similar to the Arduino Uno R3 as possible in terms of software compatibility.
Differences: The major difference is in the USB interface used for programming. The RedBoard uses a mini-USB connector instead of the standard USB-B connector.

4.Seeeduino V4.2



Features: Offers a switch to select different operating voltages (3.3V or 5V), which is useful for different sensors and modules.
Differences: The voltage selection feature is not available on the standard Arduino Uno R3. It also has a micro-USB port instead of a USB-B port.

5.RobotDyn Uno R3



Features: Compatible with Arduino Uno R3 and often includes additional features like a micro-USB interface.
Differences: Sometimes, these boards include both an ATmega328P and an ESP8266 module, allowing for Wi-Fi capabilities, which is not a standard feature of the official Uno R3.

Key Considerations

Compatibility: Most third-party boards aim to maintain compatibility with Arduino Uno R3, meaning they can use the same shields and run the same sketches (programs).

Quality and Reliability: The build quality can vary. Some third-party boards match or even exceed the quality of the official ones, while others may cut corners to reduce costs.

Community Support: The official Arduino Uno R3 has the advantage of extensive community support, tutorials, and resources. However, many third-party boards also have strong communities and support networks.

In conclusion, third-party Arduino Uno R3 boards offer a range of features that might be appealing depending on the specific needs and preferences of the user, from cost-effectiveness to enhanced capabilities like Wi-Fi or voltage selection. Third-party Arduino Uno R3 boards appear as a result of Arduino's open-source model, offering a range of options in terms of cost, features, and availability. While they present certain advantages like affordability and variety, they also come with potential drawbacks in terms of quality, compatibility, and support, which users should consider when choosing a board for their projects.

List of required materials and tools

1. Arduino Board

Options: Start with an Arduino Uno R3, which is great for beginners. For more advanced projects, consider Arduino Mega.
Where to Buy: Purchase from authorized dealers or the official Arduino website or Amazon to ensure you get an original product.
Tips: You can try to find Arduino or sensor on second-hand websites, such as ebay or Subito in Italy.

2. Environmental Sensors

Types: Include a variety of sensors like temperature & humidity (e.g., DHT11, DHT22 (more higher accuracy)), air quality (e.g., MQ-135), and others based on the projects in the manual.
Sourcing: These can be bought from electronic component suppliers like SparkFun, Adafruit, or online marketplaces like Amazon or eBay.
Tips: MQ series sensors are not very accurate, but they are cheap enough. Or you can try buying from a dealer in China, but it might take a little longer of shipping.

3. Breadboard and Jump Wires

Purpose: Essential for prototyping without soldering.
Notice: Look for a breadboard with a clear layout and durable jump wires.

4. Resistors, Capacitors, LEDs, and Other Basic Components

Variety: Include a range of resistors and capacitors for different uses.
Supplier Suggestions: These basic components can be found in starter kits or can be bought in bulk from electronic stores.

5. Power Supply (According to demand)

Requirements: Arduino Uno typically requires a 5V USB power supply. For field projects, battery packs, powerbank or solar panels might be needed.
Selection: Ensure the power source matches the voltage and current requirements of your Arduino board and sensors.

6. Data Storage

Options: If data logging is part of your projects, include an SD card module or consider Arduino boards with built-in data logging capabilities.
Retailers: These can be found at the same suppliers as the Arduino boards or in specialized electronic shops.

7. Cables and Connectors

Types: USB cables for programming and connecting Arduino, along with various connectors for sensors. Arduino Uno: USB-B.
Source: Available at most electronics stores or online suppliers.

8. Enclosures and Mounting Hardware

Use: Protect the Arduino and sensors, especially for outdoor projects.
Customization: Can be bought pre-made or custom-designed using materials like acrylic or 3D-printed cases.

9. Tools (According to demand)

Include: Soldering iron, wire cutters, screwdrivers, and a multimeter.
Quality: Invest in good quality tools for safety and durability.

10. Starter Kits

Advantage: For beginners, consider purchasing a starter kit that includes the Arduino board, a collection of sensors, and basic components.
Recommended Kits: Look for kits specifically tailored to environmental monitoring if available.

Arduino Starter Kit -

Tips for Purchasing Components

Compare Prices: Shop around to get the best deals, but be wary of prices that seem too good to be true.
Quality over Quantity: It’s better to have fewer, reliable components than a large number of low-quality parts.
Check Reviews: Especially when buying from online marketplaces, check the reviews and ratings of the product and seller.
Community Recommendations: Arduino forums and communities are great places to get recommendations on where to buy components.

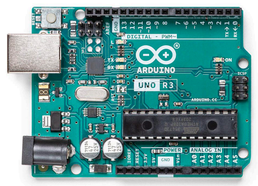
Amazon or the official Arduino store are the most convenient options, but that also means a high price.

You can try to find Arduino or sensor on second-hand websites, such as ebay or Subito in Italy.

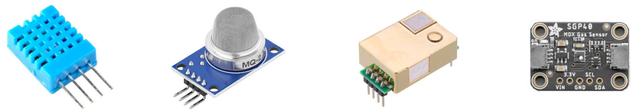
Looking for sellers from China may be a good choice, which often represents lower prices, but shipping time is always a mystery.

List of required materials and tools

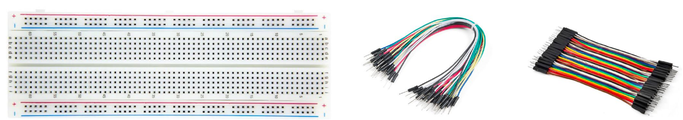
1. Arduino Board



2. Environmental Sensors



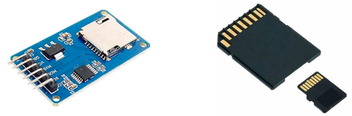
3. Breadboard and Jump Wires



4. Resistors, Capacitors, LEDs, and Other Basic Components

5. Power Supply (5V usb or 9V Adapter)

6. Data Storage (TF card or SD card)

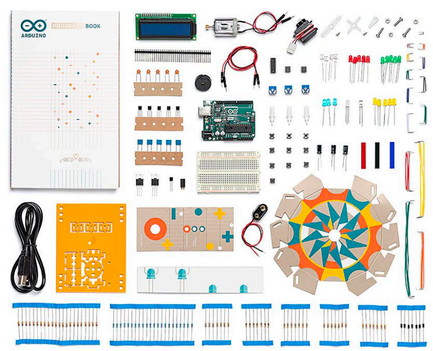


7. Cables and Connectors (USB-B)

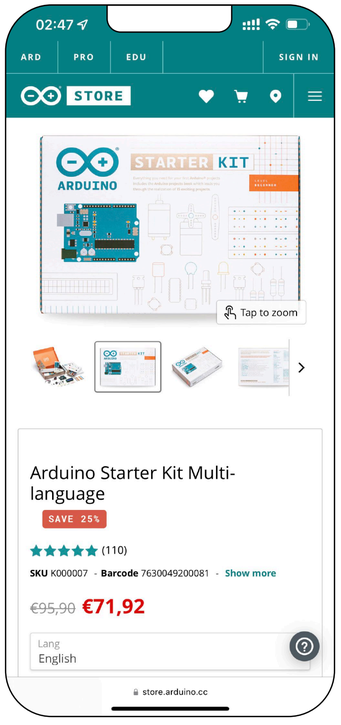
8. Enclosures and Mounting Hardware

9. Tools (According to demand)

10. Starter Kits (According to demand)

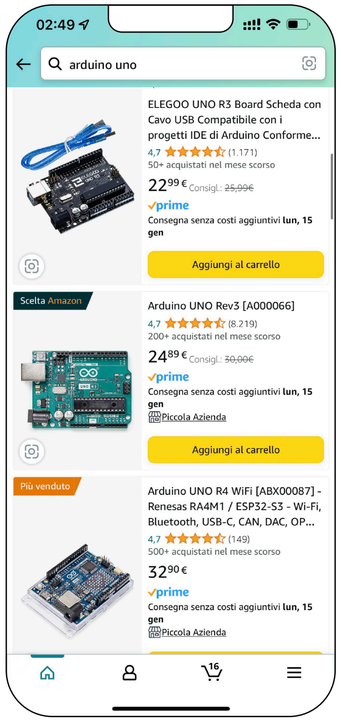


Recommendations for Purchasing



Arduino offical

But there are only official Arduino development boards and sensors, and there is a lack of third-party sensors.

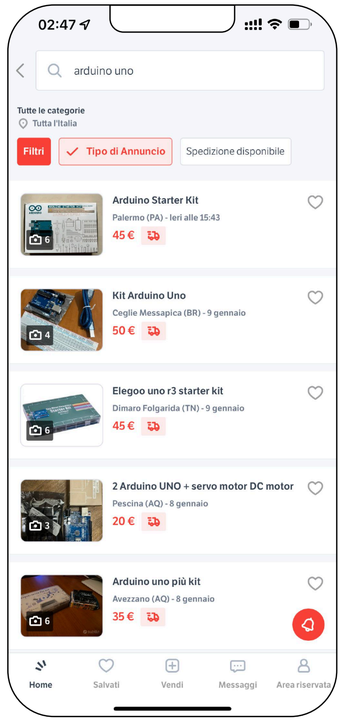


Amazon

Very fast, convenient, sometimes on sale, but some sensors are extremely expensive

ebay

At the same time, there are new or second-hand Arduino development boards or sensors, but be careful about where to send them.

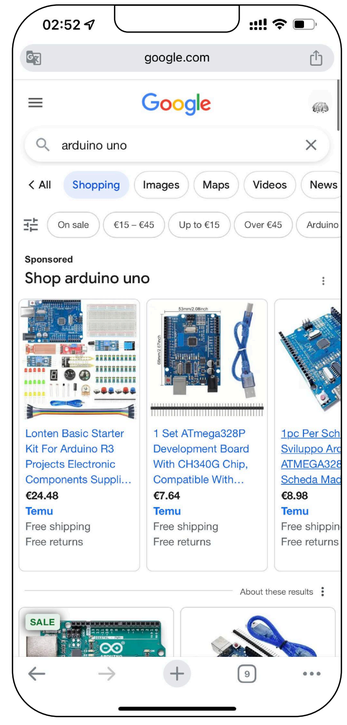


Subito

Italian second-hand website, good prices, and often Arduino kits for sale, but also lack of sensors,

AliExpress

Abundant sensors and low prices, but since most of them are shipped from China, you need to pay attention to the arrival time.



Google Shopping

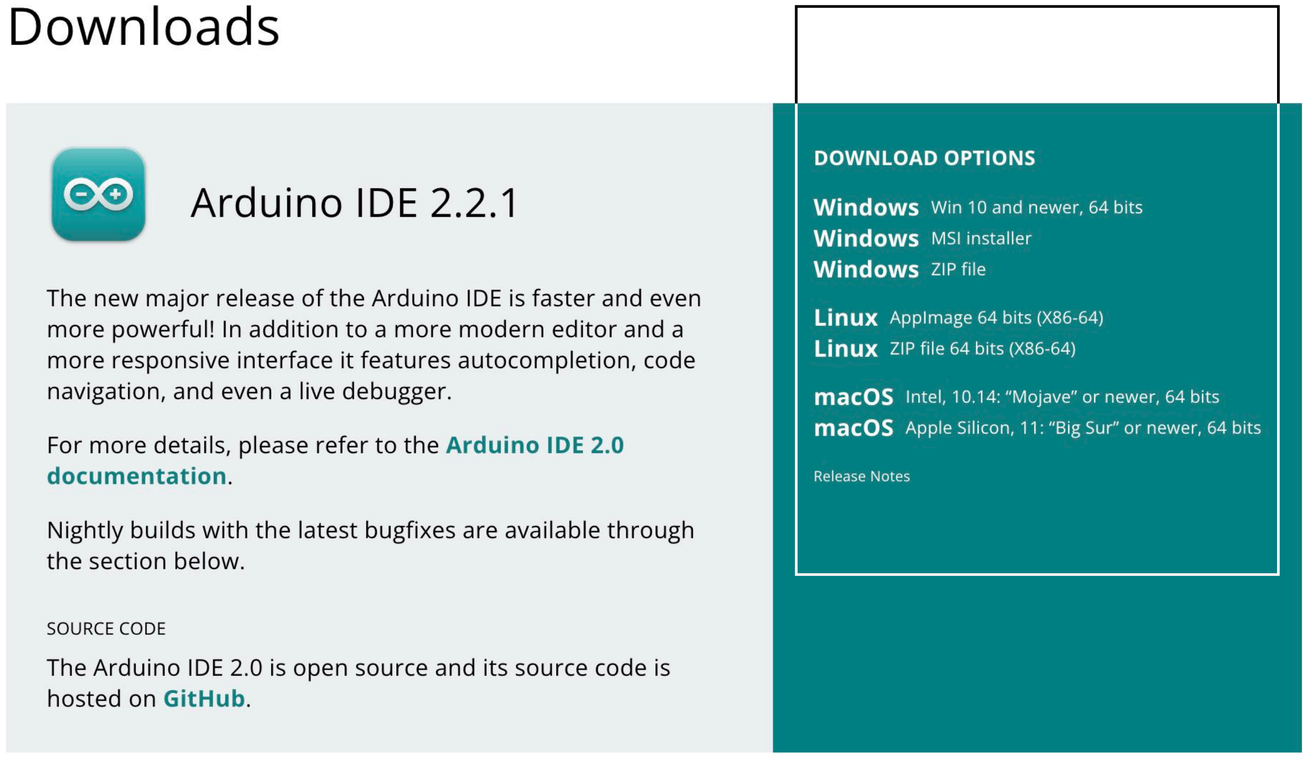
You can compare prices between different websites through the Google shopping tab.

Step-by-step Installation Guide

https://www.arduino.cc/en/software.

1. Downloading the Arduino IDE

- To the official Arduino website https://www.arduino.cc/en/software.
- Guide on selecting the correct version for their operating system (Windows, macOS, Linux).



2. Installation Process

For Windows:
Run the installer and accept the license agreement.
Emphasize selecting the right options, like drivers and additional software.

For macOS:
Drag and drop the Arduino IDE into the Applications folder.
Installation prompts for required drivers and additional software.

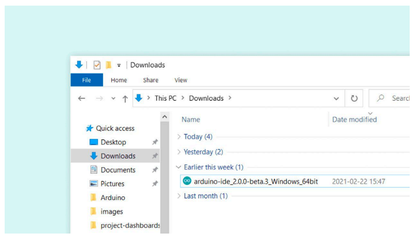
For Linux:
Follow the prompts to set permissions to allow access to the serial port.

3. Launching the Arduino IDE

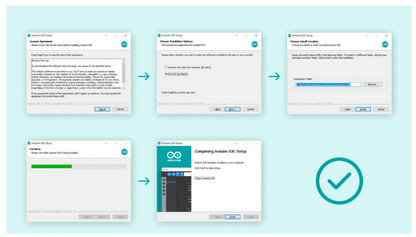
Opening the program.

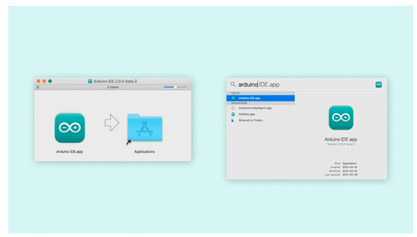
Requirements

Windows - Win 10 and newer, 64 bits
Mac OS X - Version 10.14: "Mojave" or newer, 64 bits
Linux - 64 bits



Windows





MacOS

Image sources: https://www.arduino.cc/en/software#future-version-of-the-arduino-ide

Overview of the Arduino IDE Interface

Verify / Upload:
Compile and upload your code to your Arduino Board.

Select Board & Port:
Detected Arduino boards automatically show up here, along with the port number.

Sketchbook:
Here you will find all of your sketches locally stored on your computer.

Boards Manager:
Browse through Arduino & third party packages that can be installed.

Library Manager:
Browse through thousands of Arduino libraries, made by Arduino & its community.

Debugger:
Test and debug programs in real time.

Search:
Search for keywords in your code.

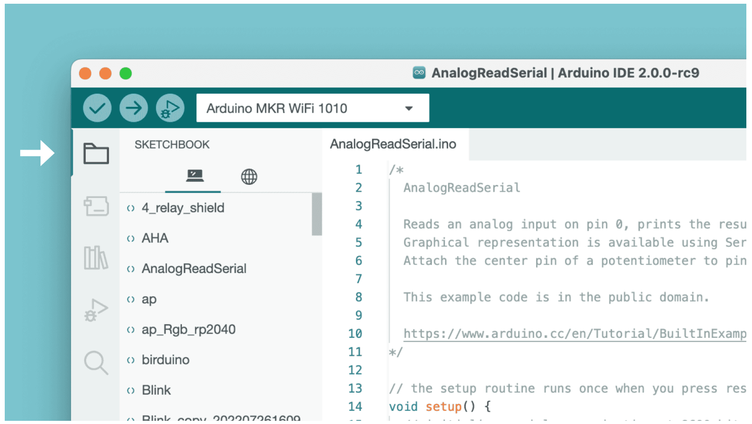
Open Serial Monitor:
Opens the Serial Monitor tool, as a new tab in the console.

Arduino IDE 2

Image sources: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2

Features of the Arduino IDE

1. Sketchbook



Your sketchbook is where your code files are stored. Arduino sketches are saved as .ino files, and must be stored in a folder of the exact name. For example, a sketch named my_sketch.ino must be stored in a folder named my_sketch.

Typically, your sketches are saved in a folder named Arduino in your Documents folder.

To access your sketchbook, click on the **folder** icon located in the sidebar.

2. Boards Manager

With the Boards Manager, you can browse and install board packages. A board package contains the "instructions" for compiling your code to the boards that are included in the board package.

There are several Arduino board packages available, such as avr, samd, megaavr and more.

There is **no need** to adjust this option when using Arduino development boards based on the AVR architecture, such as the **Arduino UNO R3** and Arduino Mega 2560

3. Library Manager

With the library manager you can browse and install thousands of libraries. Libraries are extensions of the Arduino API, and makes it easier to for example control a servo motor, read specific sensors, or use a Wi-Fi module.

Generally speaking, in this column, you can use their pre-made running codes shared by developers and sensor companies around the world, and you can directly drive various sensors through simple modifications.

Image sources: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2

Features of the Arduino IDE

4. Serial Monitor

The Serial Monitor is a tool that allows you to view data streaming from your board, via for example the **Serial.print()** command.

Historically, this tool has been located in a separate window, but is now integrated with the editor. This makes it easy to have multiple instances running at the same time on your computer.

To put it simply, all data read by the sensors, such as temperature and humidity, will be fed back to you from this window.

5. Serial Plotter

The Serial Plotter tool is great for visualizing data using graphs, and to monitor for example peaks in voltage.

You can monitor several variables simultaneously, with options to enable only certain types.

Data changes in the environment can be seen very intuitively and easily.

Image sources: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2

3. Basic Arduino Programming

3.1 Understanding Arduino Sketches

3.1.1 Structure of Arduino code

3.1.2 Basic Syntax and Conventions

3.1.3 Different models of Arduino boards

3.2 First Arduino Project: Blinking LED

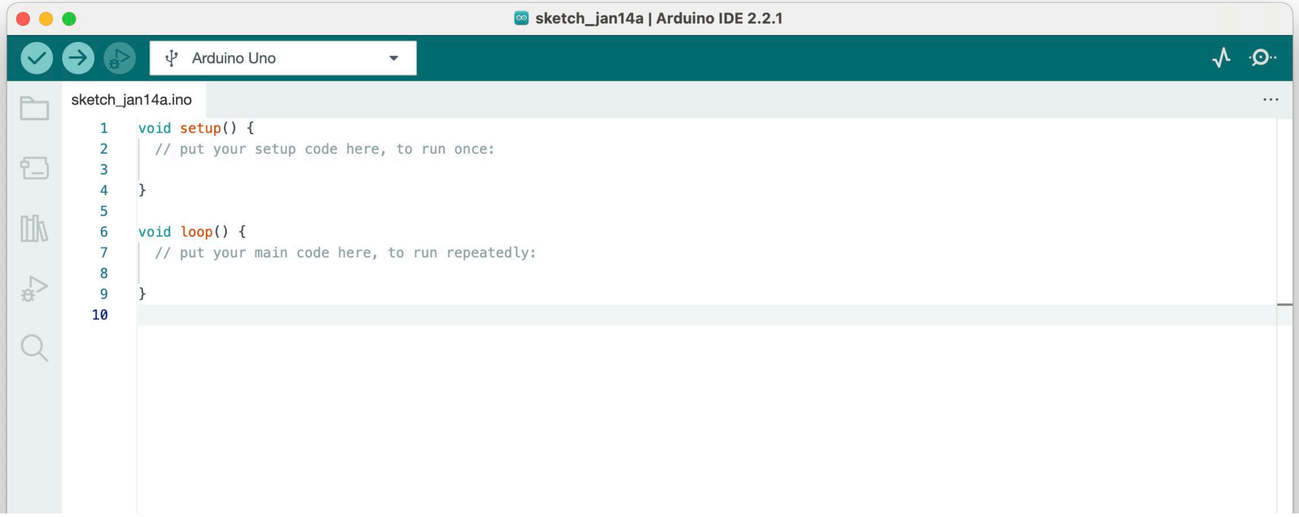
Uploading the Program

This chapter demystifies Arduino programming by breaking down the structure of Arduino code, focusing on the setup and loop functions, and explaining basic syntax and conventions. It includes a hands-on project: creating a blinking LED circuit, which serves as an introduction to wiring and programming in Arduino, providing readers with their first practical Arduino experience.

Structure of Arduino code (setup and loop functions)

The Setup() and Loop() functions represent the main core of every program developed using the Arduino controller. In fact, each program starts from the code defined within these functions. Specifically, the two functions are characterized as follows:

As shown in the picture on the right, this is the default interface when we create a new Arduino program. There are only two functions: void Setup() { and void Loop() {. This is the basic structure for all Arduino operations.



Setup() Function:

- This function is called when an Arduino board first powers up or resets.

- Common uses include setting pin modes (e.g., pinMode(13, OUTPUT);) to define pins as inputs or outputs and initiating serial communication (Serial.begin(9600);).

- It runs only once, making it ideal for initialization settings.

loop() Function:

- This function is where the bulk of a program's work is done. It loops consecutively, allowing the board to change and respond.

- Use cases include reading sensor data, controlling motors, updating lights, etc.

- It is essential to understand that this function repeats indefinitely until the board is powered off or reset.

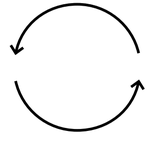
It is important to consider that the body of a function is defined using curly brackets. Specifically, the open brace { represents the beginning of the function, while the close brace } indicates its end.

Let's take a simple example: if we compare the Arduino running programme to a restaurant, void Setup() is equivalent to opening the door of the restaurant every morning, to clean up, turn on the lights, prepare the ingredients, open the door to welcome the customers, and all these things only need to be executed once every time the machine is switched on.

After the execution of Setup, it comes to the stage of void Loop(), guests order food - make meals - sell orders - wipe the table - continue to serve the next customer, just like this cycle.

Setup - run once

Loop - run repeatedly



Basic Syntax and Conventions

Arduino C/C++ Based Language

Variables and Data Types:

int: For integers. Example: int ledPin = 13;
float: For floating-point numbers. Example: float temperature;
char: For characters. Example: char myChar = 'A';
boolean: For true/false values. Example: boolean isOn = false;

Control Structures:

if statement for decision making. Example: if (temperature > 30) {...}
Loops like for and while. Example: for(int i = 0; i < 10; i++) {...}

Functions:

Describe creating and calling functions. Example: void blinkLed() {...}

Comments and Readability:

Single-line comments (//) and multi-line comments (/* */).

For specific analysis of all functions, please refer to the Arduino official website:
https://www.arduino.cc/reference/en/

https://www.arduino.cc/reference/en/

Language Reference

Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

Functions

For controlling the Arduino board and performing computations.

Digital I/O	Math	Bits and Bytes
digitalRead()	abs()	bit()
digitalWrite()	constrain()	bitClear()

Variables

Arduino data types and constants.

Constants	Data Types	Variable Scope & Qualifiers
Floating Point Constants	array	const
HIGH	LOW	bool	scope

Structure

The elements of Arduino (C++) code.

Sketch	Arithmetic Operators	Pointer Access Operators
loop()	% (remainder)	& (reference operator)
setup()	* (multiplication)	* (dereference operator)

Uploading the Program

Arduino boards come with a little utility: the built-in LED.

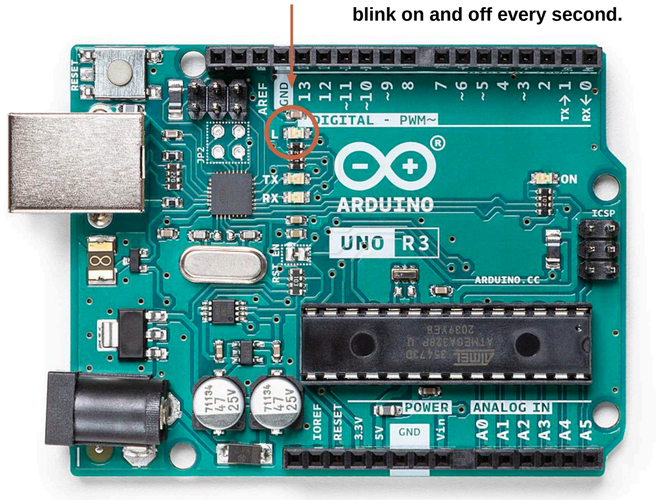
It is identified by the letter L next to it. On the Arduino Uno, it is near pin #13:

This LED is connected to the digital I/O pin #13 in most boards (Arduino UNO). In some boards, like the Arduino MKR series, it's linked to the pin #6.

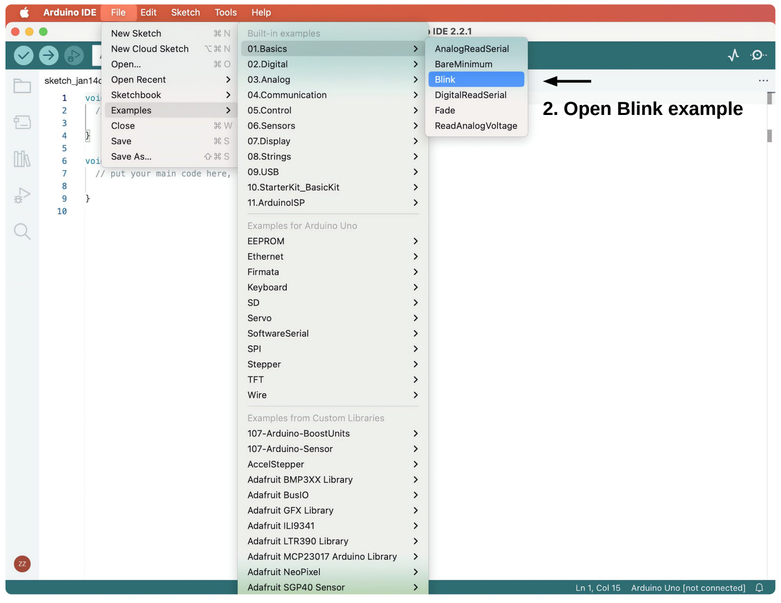
In any case you can reference the exact pin using the LED_BUILTIN constant, that is always correctly mapped by the Arduino IDE to the correct pin, depending on the board you are compiling for.

To light up the LED, first you need to set the pin to be an output in setup():

5. Once uploaded, the LED should blink on and off every second.



1. Connect your Arduino Board and computer



Explain:

Setting Up:

- In setup(), declare the LED pin as an output: pinMode(13, OUTPUT);

Blinking the LED:

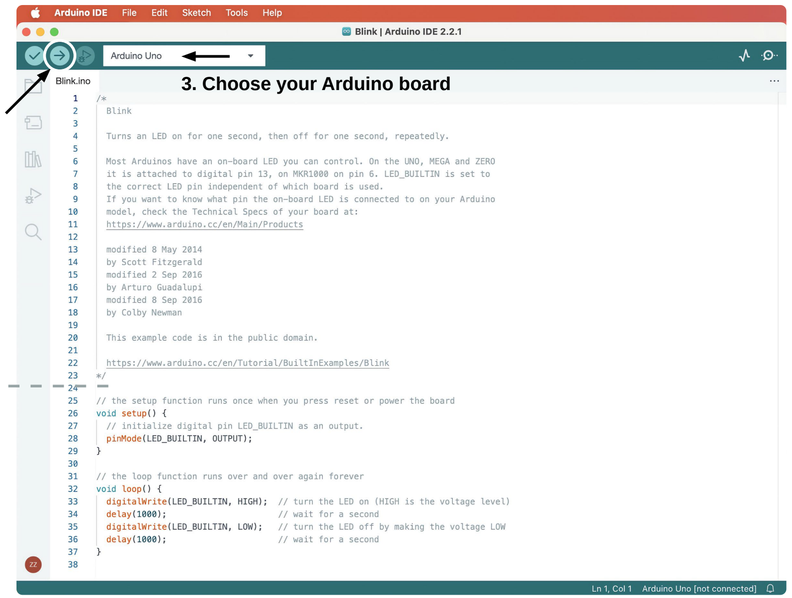
- In loop(), turn the LED on and off using digitalWrite(13, HIGH); and digitalWrite(13, LOW);.
- Introduce the delay() function to control the blink interval: delay(1000); for a 1-second delay.

Code:

```
// the setup function runs once when you press reset or power the
board

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  // turn the LED on (HIGH is the voltage level

  delay(1000);
  // wait for a second

  digitalWrite(LED_BUILTIN, LOW);
  // turn the LED off by making the voltage LOW

  delay(1000);
  // wait for a second
}
```

4. Upload programer to Arduino

Code



An exploration of various environmental sensors such as temperature, humidity, and air quality sensors is presented. This chapter assists readers in selecting the right sensor for their projects and provides detailed instructions on connecting these sensors to Arduino, including wiring diagrams and steps to read sensor data effectively.

4. Apply Environmental Sensors

4.1 Types of Environmental Sensors

4.1.1 Overview of sensors

4.1.2 List of gas sensors

4.1.3 Other useful modules

4.1.4 Selecting the Right Sensor for Your Project

4.1.5 Recommendations for Purchasing

4.2 Connecting Sensors to Arduino

4.2.1 Structure of Arduino UNO R3

4.2.2 Connection Methods of Arduino & Sensor

4.2.3 Breadboard and Jump Wires

4.2.4 Wiring Diagrams and Setup Instructions

Overview of Sensors

Arduino is compatible with a wide array of sensors, each providing different functionalities. This versatility allows it to be used in numerous applications, from simple DIY projects to complex scientific instruments. Here are some common sensors and the functions they can achieve:

1. Temperature Sensor (e.g., DHT11, DHT22)
Function: Measures ambient temperature.
Applications: Weather stations, temperature monitoring systems.

2. Humidity Sensor (e.g., SHT30, SHT40)
Function: Measures air humidity.
Applications: Environmental monitoring, home automation systems.

3. Motion/Pir Sensor (e.g., HC-SR501)
Function: Detects motion in an environment.
Applications: Security systems, automatic lighting.

4. Ultrasonic Distance Sensor (e.g., HC-SR04)
Function: Measures distance to an object using ultrasonic waves.
Applications: Robotics, obstacle avoidance systems.

5. Light Sensor (e.g., Photoresistor)
Function: Measures light intensity.
Applications: Automatic lighting, light level monitoring.

6. UV Sensor (e.g., MCU-GUVA-S12SD)
Function: Measures sunlight, ultraviolet, infrared and visible light.
Applications: Sunlight intensity and UV index.

7. IR Sensor
Function: Detects infrared light, often used for proximity or line detection.
Applications: Line-following robots, object detection.

8. Pressure Sensor (e.g., BMP180)
Function: Measures atmospheric pressure.
Applications: Weather stations, altitude measurements.

9. Color Sensor (e.g., TCS3200)
Function: Detects color of objects.
Applications: Color sorting machines, quality control in manufacturing.

10. Accelerometer and Gyroscope (e.g., MPU6050)
Function: Measures acceleration and rotational motion.
Applications: Motion detection, gesture control.

11. Sound Sensor (e.g., Electret microphone)
Function: Detects sound levels.
Applications: Noise level monitoring, audio recording.

12. Heart Rate Sensor (e.g., MAX30100)
Function: Measures heart rate.
Applications: Fitness devices, patient monitoring.

13. Flex Sensor
Function: Measures the amount of bend or flex.
Applications: Wearable devices, robotics.

14. Touch Sensor (e.g., Capacitive touch sensor)
Function: Detects touch or proximity.
Applications: Interactive projects, user interfaces.

15. Soil Moisture Sensor
Function: Measures moisture level in soil.
Applications: Agriculture, smart gardening.

16. Water Sensor
Function: Measuring changes in water level height.
Applications: Agriculture, Evaporation, environmental changes.

17. Hall Effect Sensor
Function: Detects magnetic fields.
Applications: Proximity switching, speed detection.

18. RFID Reader
Function: Reads RFID tags.
Applications: Access control, inventory management.

19. GPS Module
Function: Provides location data.
Applications: Tracking systems, navigation.

20. Gas Sensor
Function: Detects various gases like smoke, methane, LPG.
Applications: Safety devices for gas leakage, air quality monitoring.

20.1 CO2 Sensor (e.g., MH-Z19, MG-811)
Function: Measures Carbon Dioxide (CO2) levels in the environment.
Applications: Indoor air quality monitoring, HVAC systems, environmental monitoring.

20.2 HCHO Sensor (Formald2ehyde)
Function: Detects concentration of formaldehyde, a harmful volatile organic compound (VOC).
Applications: Indoor air quality monitoring, industrial safety, environmental testing.

20.3 VOC Sensor (e.g., SGP41)
Function: Detects various Volatile Organic Compounds (VOCs) including harmful gases like benzene, alcohol, smoke.
Applications: Air purifiers, smart ventilation systems, environmental monitoring.

20.4 Air Quality Sensor (e.g., MQ-135, CCS811)
Function: Provides a general measure of air quality by detecting multiple gases like CO2, VOCs, and NOx.
Applications: Urban air quality monitoring, smart home systems, environmental impact studies.

20.5 Ozone Sensor (e.g., MQ-131)
Function: Measures concentration of ozone (O3) in the air.
Applications: Environmental monitoring, industrial safety, air purification systems.

20.6 MQ Series Sensors
Function: The MQ series sensors are a family of gas sensors detecting various gases like methane, propane, carbon monoxide, and more.
Applications: Depending on the specific model, applications include leak detection in homes and industries, air quality monitoring, and safety systems in mining and chemical industries.

General Considerations:
Compatibility: Always ensure the sensor is compatible with Arduino in terms of voltage and communication protocol.

Libraries and Drivers: Many sensors have specific libraries available for Arduino, making programming easier.

Applications: The choice of sensor depends largely on the application and the environmental conditions where it will be used.

Arduino's ability to connect with these sensors and more, combined with its easy-to-use programming environment, makes it an ideal platform for a myriad of projects, from simple to complex.

Overview of Sensors (* means tested)

1 & 2 - Temperature & Humidity

* DHT 11

* DHT 22

* SHT 31

* SHT 40/41/45

3 - Motion/PIR

* HC SR501

* MH SR602

4 - Ultrasonic Distance

* HC SR04

TOF 200C

VL 6180X

5 - Light Sensors

HC SR04

* TSL 2591

6 - UV Sensors

* S12SD

* GY 8511

UVM 30A

7 - IR Sensors

UV Sensor

* LTR 390

QRE 1113

IR Sensor

8. Pressure Sensors

BMP 180

* BMP 280

* BMP 390

* BMP 680

9. Color Sensors

TCS 3200

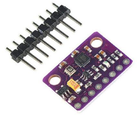
GY 33

TCS 3472

10. Accelerometer & Gyroscope

MPU 6050

MMA 7361



GY 91

MCU 6DOF

11. Sound Sensors

Sound Sensor

LM 393

SPW 2430

MAX 9814

12. Heart Rate Sensors

MAX 30100

XD 58C

13. Flex Sensor

Flex Sensor

14. Touch Sensor

TTP 223

15. Soil Moisture

Soil Moisture Sensor

16. Water Sensor

Water Sensor

17. Hall Effect Sensor

US 188

18. RFID Reader

RC 522

19. GPS Modules

GT U7

* NEO 6M

* NEO M9N

4.1 Types of Environmental Sensor

List of Gas Sensors (* means tested)

In an era where environmental health is paramount, monitoring air quality and other environmental factors has become crucial. Arduino, a versatile open-source microcontroller, has emerged as a valuable tool in this endeavor. Its ability to interface with a wide range of sensors, coupled with its simplicity and accessibility, makes it ideal for air quality and environmental monitoring projects.

Arduino's primary strength lies in its simplicity and flexibility. It serves as a bridge between various sensors and the digital world, enabling even hobbyists and students to build sophisticated monitoring systems. The platform's adaptability allows it to be customized for a variety of environmental monitoring needs.

Roles of Arduino in Air Quality and Environmental Monitoring:

Data Collection: Arduino can gather real-time data from various gas sensors.

Data Logging: Storing sensor data over time for analysis.

Alert Systems: Triggering alarms or notifications when gas concentrations reach unsafe levels.

Data Transmission: Sending data to a computer or cloud service for further analysis.

Control Systems: Automatically controlling air purifiers or ventilation systems based on air quality.

Environmental Assessment: Helping in research and study of environmental pollution patterns.

Educational Tools: Used in schools and universities for teaching environmental science and electronics.

Community Monitoring: Citizen science projects for urban air quality monitoring.

Industrial Safety: Monitoring air quality in manufacturing and chemical plants to ensure worker safety.

Healthcare Applications: Monitoring air quality in hospitals and health care facilities.

Agricultural Monitoring: Checking air quality in agricultural settings, especially in greenhouses.

Urban Planning: Data gathered can assist in urban development planning, focusing on pollution control.

Smart Home Integration: Integrating with home automation.

20. Gas Sensors

CO2

***** PMS 5003 / 7003 / 9003

***** MH Z19

***** SCD 30

***** SCD 40

MG 811

HCHO

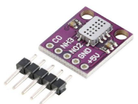
***** Dart WZ-S

Voc

***** SGP 30

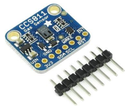
***** SGP 41

Air Quality



MCU 6814

MICE 5524



CCS 811

Ozone

***** MQ 131

Smoke

MQ 2

Alcohol

MQ 3

Methane

MQ 4

Natural gas

MQ 5

Carbon Monoxide

MQ 7

Hydrogen

MQ 8

Combustible gas

MQ 9

Air Quality

MQ 135

Hydrogen Sulfide

MQ 136

Ammonia

MQ 137

Aldehydes

MQ 138

Freon

MQ 139

Other Useful Modules (* means tested)

Arduino offers a wide range of expansion modules, commonly referred to as "shields," which enhance its capabilities in various areas. Here's a detailed introduction to some of the key modules:

1. Storage Modules:

SD Card Module: This allows Arduino to read and write data to an SD card, ideal for data logging or storing large amounts of data.
EEPROM Module: EEPROM, or Electrically Erasable Programmable Read-Only Memory, provides additional memory space for storing data that must be preserved between power cycles.

2. Time Modules:

Real-Time Clock (RTC) Module: RTC modules like the DS3231 provide accurate timekeeping and can maintain time even when the Arduino is powered off, thanks to a backup battery.

3. Communication Modules:

Bluetooth Module (HC-05/HC-06): These modules enable wireless communication via Bluetooth, allowing Arduino to interact with smartphones, computers, and other Bluetooth-enabled devices.
WiFi Module (ESP8266): This module allows Arduino to connect to the internet via WiFi, making IoT (Internet of Things) projects feasible.

4. Display Modules:

LCD Display (16x2 or 20x4): Liquid Crystal Displays are used for showing text or numbers, useful for user interfaces in projects.
OLED Display: Offers higher contrast and can display graphics, ideal for more complex displays.
E-Paper Display (Wavehare 4.2): Has extremely low power consumption and simulates the display method of natural paper. It consumes no power during static display and has a good display effect, but it does not emit light and has a low refresh rate. It is suitable for use as supermarket price tags, thermometers, etc.

5. Motor Control Modules:

Servo Motor Controller: Useful for precision control of servo motors, often used in robotics.
Stepper Motor Driver (A4988/DRV8825): These are used to control stepper motors, essential in CNC machines, 3D printers, etc.

6. Power Modules:

Battery Shield: Provides a convenient way to power Arduino projects through batteries.
Solar Charger Shield: Enables solar-powered projects, useful for outdoor applications.

Each of these modules interfaces with the Arduino through its digital and analog I/O pins, and many of them communicate using standard protocols like SPI, I2C, or UART, making them relatively easy to program and integrate into a wide range of projects. The availability of libraries and example codes for these modules also makes the development process more accessible for beginners and hobbyists.

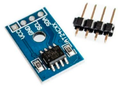
1. Storage modules

***** TF/Micro SD card module

***** TF/Micro SD card module

SD card module

AT24C EEPROM Module



AT24C EEPROM Module

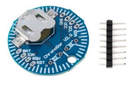
W25Q16 EEPROM Module

2. Time modules

***** DS3231 RTC module

DS1307 RTC module

DS1302 RTC module



DS3231SN RTC module

3.1 Bluetooth Modules

***** HC-06 bluetooth module

***** HM-10 BLE module

3.2 WiFi moudle

***** ESP-01S WiFi moudle

4. Display modules

***** TM1637 4-Digit Display

***** 1602 16x2 LCD moudle

***** 0.96' OLED moudle

3.5' TFT moudle

***** 4.2' e-paper moudle

5. Motor Control Modules

***** 5v stepper motor

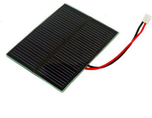
***** ULN 2003 Motor Driver

A4988 Motor Driver

6. Power modules

***** AAA Battery Shield

***** 9V Battery Shield



Solar Charger Shield

Selecting the Right Sensor for Your Project

Understanding Project Objectives

Define the Environmental Parameter: Identify the specific environmental element you need to monitor, such as temperature, humidity, air quality, or specific gases.

Project Scope: Determine whether the project requires monitoring a single parameter or multiple parameters simultaneously.

Compatibility with Arduino

Voltage Compatibility: Ensure the sensor's operating voltage matches that of the Arduino board (commonly 5V or 3.3V).

Communication Protocol: Check if the sensor communicates via analog, digital, I2C, SPI, or UART, and ensure your Arduino board supports this protocol.

Library Availability: Look for sensors with existing Arduino libraries, which makes programming and integration easier.

Ethical and Safety Considerations

Safety: Ensure the sensor does not pose any safety risks, especially if used by beginners or in educational settings.

Environmental Impact: Consider the environmental impact of the sensor, including its manufacturing process, lifespan, and disposal.

Accuracy and Precision Requirements

Accuracy Needs: Consider the level of accuracy required for your data. High-precision sensors are essential for professional or scientific projects, while for educational or DIY projects, standard sensors might suffice.

Precision and Resolution: Understand the difference between accuracy (closeness to the true value) and precision (repeatability of measurements). Also, consider the resolution, which is the smallest change the sensor can detect.

Ease of Use and Community Support

Beginner-Friendly: If the target audience is beginners, choose sensors that are easier to set up and use.

Community and Documentation: Sensors with a strong community following and good documentation (such as tutorials and forums) are preferable, as they provide better learning support.

Cost and Availability

Budget Constraints: Balance between the cost of the sensor and the features it offers. While some high-end sensors can be expensive, there are many affordable options that offer decent performance for educational purposes.

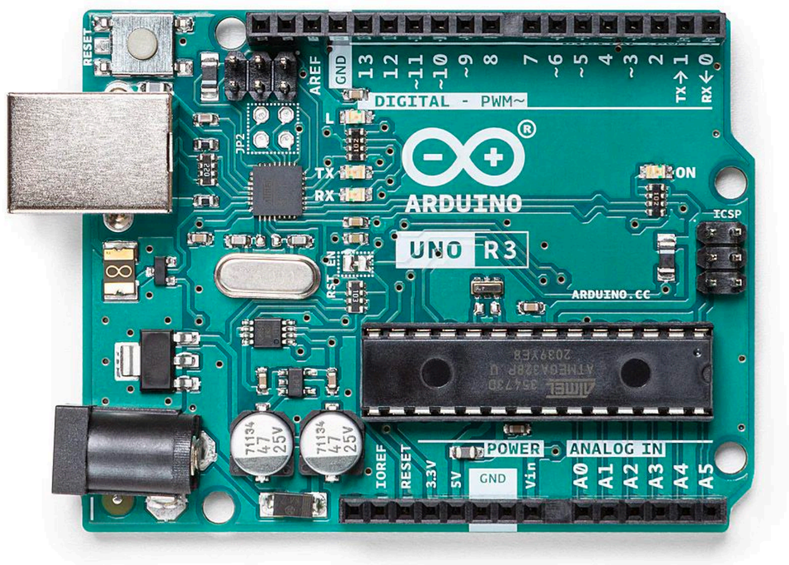
Availability: Choose sensors that are readily available and have reliable supply chains, especially if the project will be replicated or scaled.

Environmental Considerations

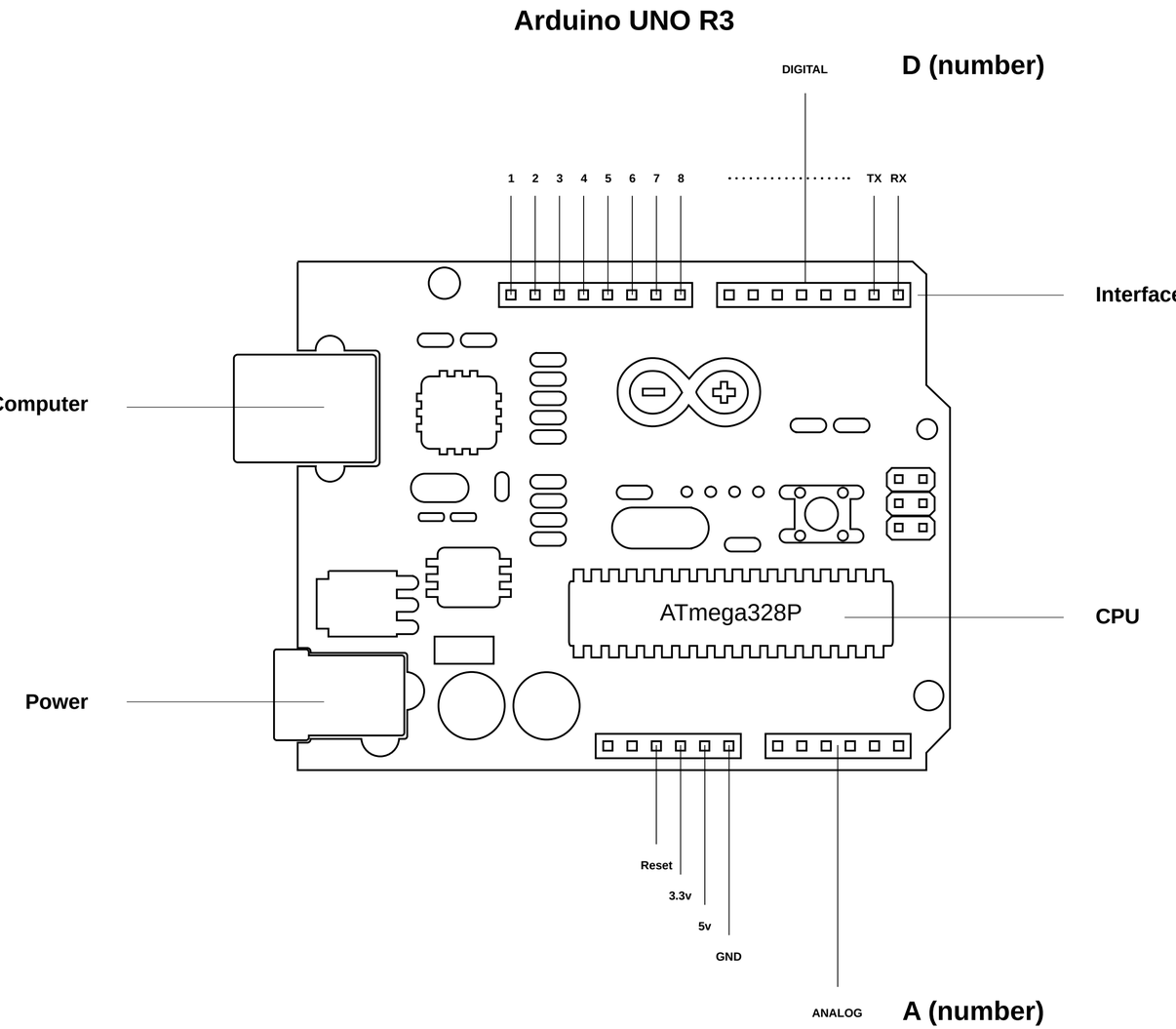
Operating Conditions: Evaluate the environmental conditions in which the sensor will operate, such as temperature range, humidity levels, and potential exposure to elements like water or sunlight.

Durability and Longevity: For outdoor or harsh environments, choose sensors that are robust and can withstand such conditions over time.

Structure of Arduino UNO R3



Arduino UNO R3



Structure of Arduino UNO R3

The Arduino Uno R3 is a popular microcontroller board based on the ATmega328P. It's equipped with a variety of pins, each serving a specific purpose. Let's go through the major categories of pins:

Digital I/O Pins (0-13):
Digital pins 0-13: These can be used for both digital input (like reading a button) and digital output (like driving an LED).
PWM: Some of these pins (usually marked with a ~, like 3, 5, 6, 9, 10, and 11 on the Uno) support PWM (Pulse Width Modulation), useful for analog-like control for dimming LEDs or controlling motor speed.

Analog Input Pins (A0-A5):
Analog pins A0-A5: These are used to read analog voltages. They are handy when you need to read sensor data that varies continuously (like a temperature sensor).

Power Pins:
5V: This pin outputs a regulated 5V from the regulator on the board.
3.3V: Provides a 3.3V supply generated by the on-board regulator.
GND: Ground pins.
Vin: The input voltage to the Arduino board when it's using an external power source.
Reset: Can be used to reset the microcontroller.

Communication Pins:
RX and TX (pins 0 and 1): These are used for serial communication.
A4 (SDA) and A5 (SCL): Used for I2C communication.
ICSP Header: Used for SPI communication.

Other Pins:
AREF: Used to provide reference voltage for analog inputs.
Reset Button: Resets the microcontroller.

Each pin on the Arduino can be configured via the Arduino programming environment using the Arduino language, which is based on C/C++. The flexibility of these pins allows for a wide range of applications, from simple LED control to complex robotics.

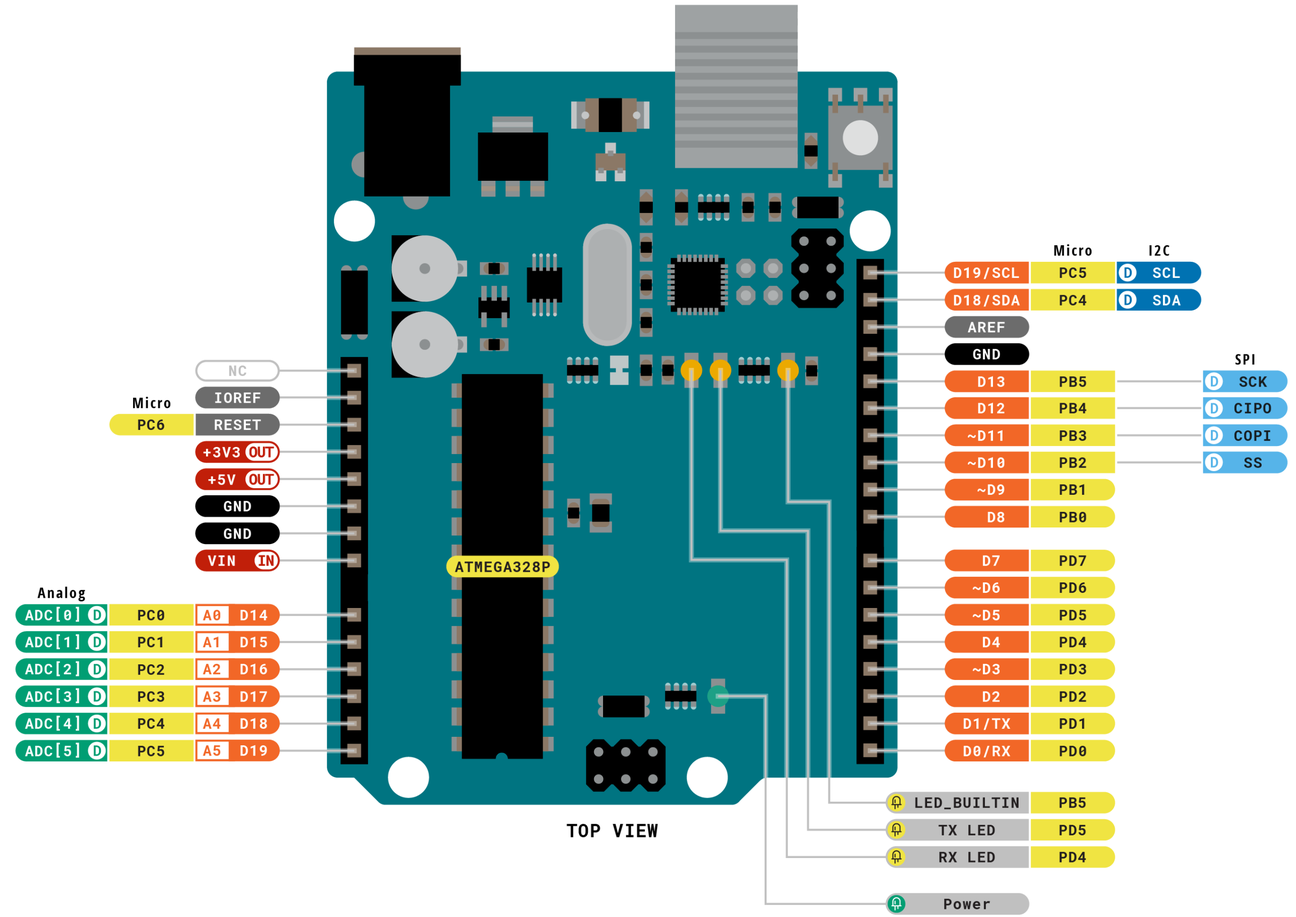


Image sources: https://docs.arduino.cc/hardware/uno-rev3

Connection Methods of Arduino & Sensor

Connecting sensors to an Arduino involves providing them with the necessary power. The power connections are crucial for the sensor's operation. Here's how to connect Arduino's power to sensors in the three scenarios you've mentioned:

General Tips:

- Always start by checking the sensor's datasheet for power requirements.

- Be careful not to reverse the power connections as it can damage both the Arduino and the sensor.

- In case of higher power requirements, consider using a separate power supply to prevent damage to the Arduino.

- Ensure that the ground of the Arduino and the ground of any external power supply are connected together to provide a common ground.

- Following these guidelines will help you connect a variety of sensors to your Arduino, ensuring proper power supply and functionality.

Power -	Power +
GND	5V / 3V
GND	VIN
GND	VCC

1. Sensor with GND and 5V/3V
This type of sensor can operate either on 5V or 3.3V.

Connection:
GND on the sensor to GND on the Arduino. This creates a common ground.
For 5V operation, connect the 5V pin on the sensor to the 5V output on the Arduino.
For 3.3V operation, connect the 3V pin on the sensor to the 3.3V output on the Arduino.

Considerations:
Ensure that the sensor can operate at the voltage you are providing (either 5V or 3.3V).
Some sensors can work with both voltages, but others are restricted to one.

2. Sensor with GND and VIN
VIN usually refers to an input voltage that is higher than the typical 5V or 3.3V.

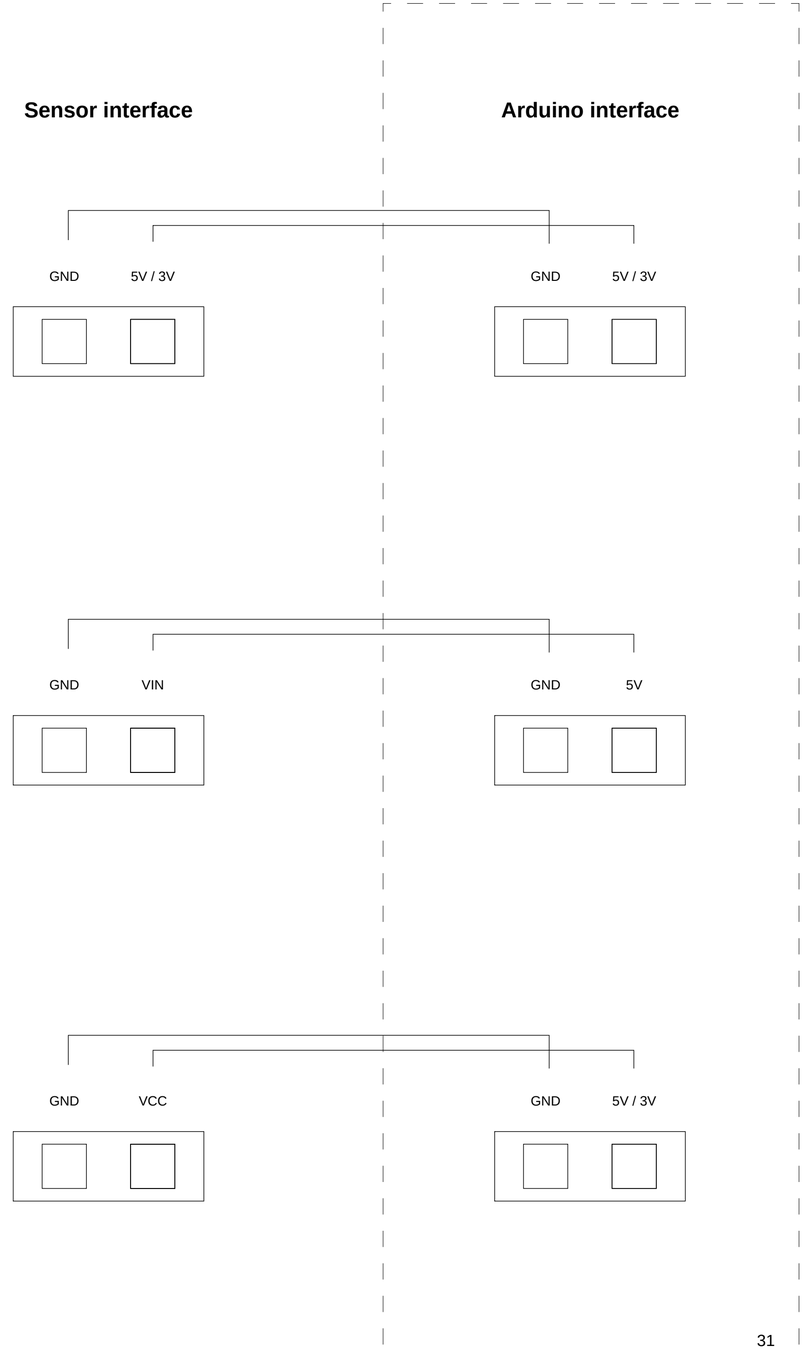
Connection:
GND on the sensor to GND on the Arduino.
VIN on the sensor to an external power source if the sensor requires more voltage than what the Arduino can provide. If the sensor works within the Arduino's voltage range, you can connect VIN to the Arduino's 5V pin.

Considerations:
The voltage requirements depend on the sensor. Check the sensor's datasheet.
If the sensor requires more than 5V, you'll need an external power source.

3. Sensor with GND and VCC
VCC generally refers to the power supply voltage.

Connection:
GND on the sensor to GND on the Arduino.
VCC on the sensor to either the 5V or 3.3V output on the Arduino, depending on the sensor's voltage requirements.

Considerations:
It's important to check the sensor's datasheet to understand the correct operating voltage.
Supplying a higher voltage than what the sensor can handle can damage it.



Connection Methods of Arduino & Sensor

Arduino sensors can use a variety of digital interfaces to communicate with an Arduino board. Each interface has its characteristics and is suitable for different applications. The most common digital interfaces for Arduino sensors include:

One-Wire:

Info:

- A communication bus system that uses, as the name suggests, a single wire for data transmission.
- Suitable for low-speed communication with devices like temperature sensors.
- Allows multiple devices to be connected to a single wire, each with a unique address.

Usage: Suitable for situations where minimal wiring is crucial and data transfer rates are low, such as the DHT 11 & DHT 22).

Speed: Low. Designed for lower data rate applications.

Complexity: Low to moderate. Requires management of device addresses.

I2C (Inter-Integrated Circuit):

Info:

- A multi-master, multi-slave, packet-switched, single-ended, serial communication bus.
- Uses two wires: SCL (clock line) and SDA (data line), allowing multiple devices to be connected to the same bus.
- Suitable for short-distance communication within a single device.

Usage: Ideal for connecting multiple sensors with minimal wiring, especially when pin availability is limited.

Speed: Moderate speed, typically up to 400 kHz in the standard mode, and up to 3.4 MHz in the high-speed mode.

Complexity: Moderate. Requires understanding of device addresses and bus arbitration.

PWM (Pulse Width Modulation):

Info:

- Not a communication protocol
- But a technique used to control devices like motors and LEDs by varying the width of digital pulses.
- Widely used in controlling the speed of motors or dimming LEDs.

Usage: Widely used in controlling actuators like servo motors, dimming LEDs, or any application requiring variable output.

Speed: N/A. It's about controlling the duration of the digital signal, not the data transfer rate.

Complexity: Low. Relatively straightforward to implement in Arduino sketches.

UART (Universal Asynchronous Receiver/Transmitter):

Info:

- Used for asynchronous serial communication.
- Involves only 2 wires transmitting (TX) and receiving (RX) data.
- Commonly used for communication between the Arduino and a computer, GPS modules, and other serial devices.

Usage: Commonly used for serial communication between the Arduino and computers, GPS modules, or other serial devices.

Speed: Moderate. Speed is set by a baud rate, which must be matched on both communicating devices.

Complexity: Low. Simple to implement and understand, especially for point-to-point communication.

SPI (Serial Peripheral Interface):

Info:

- A synchronous serial communication interface used for short-distance communication, primarily in embedded systems.
- Utilizes a master-slave architecture with separate lines for data (MOSI and MISO), a clock (SCK), and often a separate select line for each slave (SS).
- Faster than I2C, but requires more pins.

Usage: Preferred for high-speed data transfer between the Arduino and peripheral devices like SD cards or display screens.

Speed: High speed, often faster than I2C, making it suitable for applications requiring rapid data transfer.

Complexity: Higher complexity due to more lines (MOSI, MISO, SCK, SS) and the need for managing multiple slave devices.

Digital GPIO (General Purpose Input/Output):

- Simple digital pins on Arduino that can be set to input or output.
- Used for reading digital sensors (like push buttons) or controlling digital devices (like LED lights).

Usage: Basic digital communication, suitable for binary input devices like switches or output devices like LED indicators.

Speed: N/A. Depends on the Arduino's processing capability.

Complexity: Very low. Suitable for beginners and simple projects.

Conclusion

Each interface has its advantages and is chosen based on factors like speed requirements, distance between the sensor and the microcontroller, the complexity of the network (number of devices), and resource availability (like the number of pins on the microcontroller). When selecting a digital interface for an Arduino sensor or device, consider the following factors:

Speed Requirements: SPI is best for high-speed data transfer, while I2C and UART are suitable for moderate speeds.

Complexity and Resource Constraints: I2C uses fewer pins but is more complex than digital GPIO. SPI requires more pins and is more complex in handling multiple devices.

Distance and Environment: CAN is robust for longer distances and noisy environments, while RS-232 is ideal for industrial applications.

Number of Devices: I2C and One-Wire are suitable for multiple devices on a single bus, whereas SPI and UART are typically used for point-to-point communication.

Specific Application Needs: PWM for controlling actuators, Digital GPIO for simple binary input/output tasks, and specialized protocols like CAN for automotive applications.

Ultimately, the choice depends on the specific requirements of your project, including speed, complexity, distance, and the number and type of devices you need to interface with.

Connection Methods of Arduino & Sensor

Connecting various sensors to an Arduino Uno R3 involves understanding the type of interface each sensor uses. Here's a detailed introduction for each of the specified scenarios:

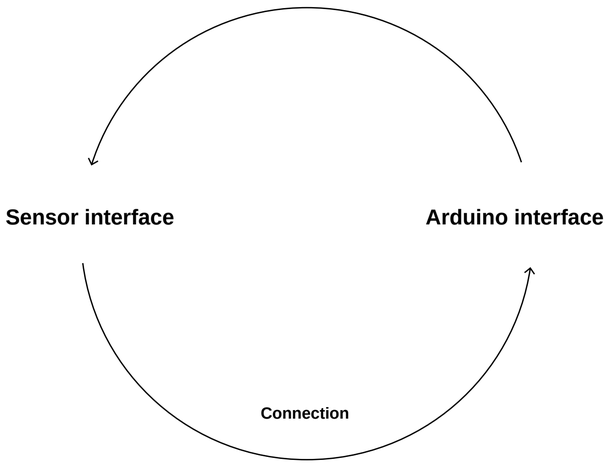
General Tips:

- Always check the sensor's datasheet for voltage and current requirements to avoid damaging the sensor or the Arduino.

- Some sensors may require additional components like resistors or capacitors for proper operation.

- For coding, include necessary libraries and understand the sensor's protocol to effectively gather and interpret data.

- By following these guidelines, you can successfully connect and program a variety of sensors with your Arduino Uno R3.



1. Sensor with Digital, GND, and 5V Interfaces
This type of sensor typically sends a high or low digital signal to the Arduino.
Connection:

5V on the sensor to 5V on the Arduino.
GND on the sensor to GND on the Arduino.
Digital on the sensor to any of the Digital I/O pins (0-13) on the Arduino.
Example: A motion sensor (PIR sensor).
Programming: Use digitalRead() in the Arduino code to read the digital signal.

2. Sensor with Analog, GND, and 5V Interfaces
Analog sensors provide a variable voltage that the Arduino reads as a value between 0 and 1023. Connection:

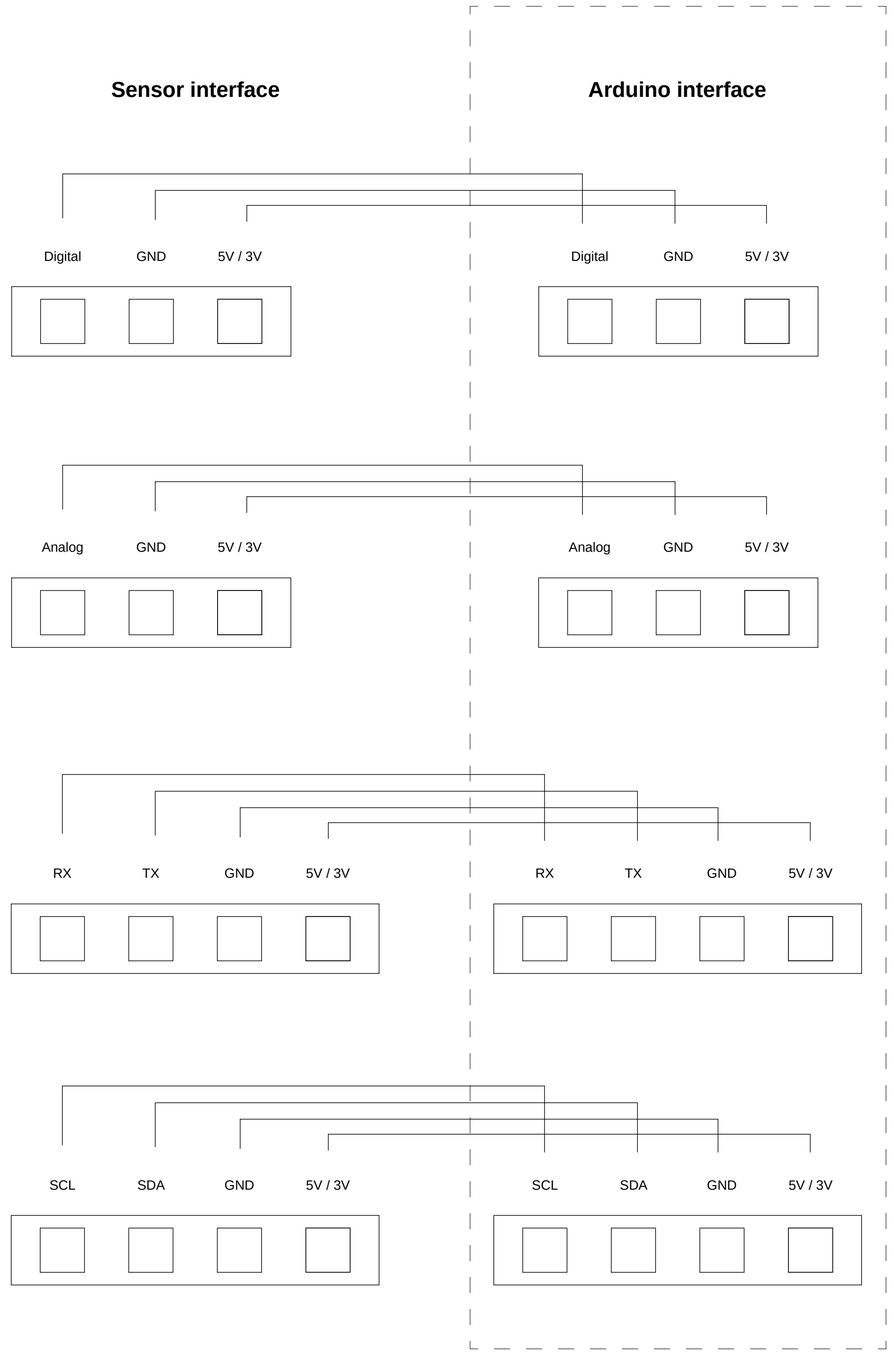
5V on the sensor to 5V on the Arduino.
GND on the sensor to GND on the Arduino.
Analog on the sensor to any of the Analog Input pins (A0-A5) on the Arduino.
Example: A temperature sensor (like LM35).
Programming: Use analogRead() in the Arduino code to read the analog value.

3. Sensor with TX, RX, GND, and 5V Interfaces
This is a serial communication interface. TX and RX stand for Transmit and Receive, respectively. Connection:

5V on the sensor to 5V on the Arduino.
GND on the sensor to GND on the Arduino.
TX on the sensor to RX on the Arduino.
RX on the sensor to TX on the Arduino.
Example: A GPS module.
Programming: Use the Serial library in Arduino to communicate with the sensor.

4. Sensor with SCL, SDA, GND, and 5V Interfaces
These are for I2C communication, a two-wire communication protocol. Connection:

5V on the sensor to 5V on the Arduino.
GND on the sensor to GND on the Arduino.
SCL on the sensor to A5 (SCL) on the Arduino.
SDA on the sensor to A4 (SDA) on the Arduino.
Example: An accelerometer like the MPU6050.
Programming: Use the Wire library for I2C communication in Arduino.



Breadboard and Jump Wires

Understanding Breadboards

A breadboard is a device for constructing a temporary prototype of an electronic circuit without soldering. It's rectangular and made of plastic with numerous holes (contact points) arranged in rows. The holes are connected electrically in specific patterns:

- Terminal Strips: The central area, typically consisting of two sets of columns (a-e and f-j), are electrically connected in rows. Each row is independent of the others.
- Bus Strips: Usually on both sides of the breadboard, longer rows typically marked with red (+) and blue (-) lines, indicating positive and negative power lines.

Using a Breadboard

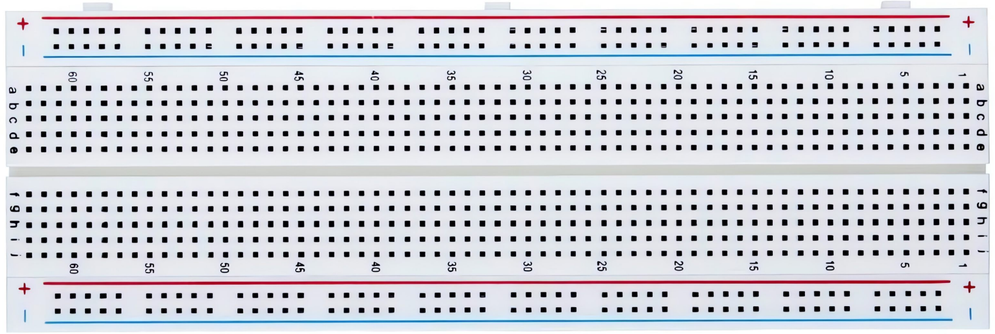
1. Planning Your Circuit: Before inserting any components, plan your circuit. This involves determining where to place your Arduino, sensors, resistors, and other components.

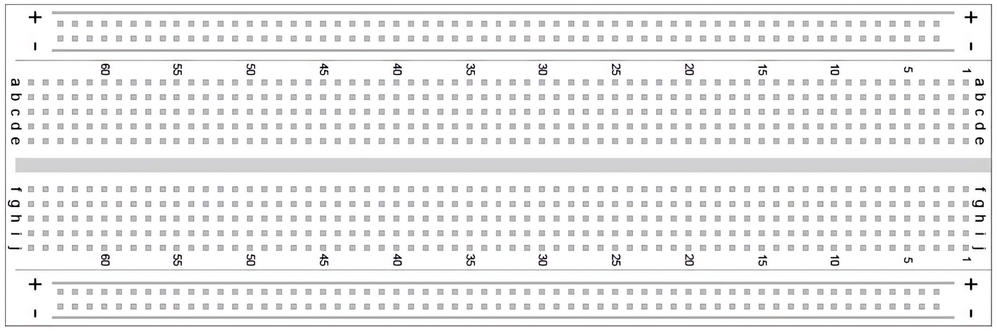
2. Inserting Components: Insert components into the breadboard holes. Each component leg should be in a different row to avoid short circuits.

3. Creating Connections:
- Use jumper wires to make connections. These wires come in various lengths and colors, which can be helpful for keeping track of different parts of your circuit.
- To connect the Arduino to the breadboard, you'll generally use male-to-male jumper wires.

4. Powering the Breadboard:
- You can power the breadboard from the Arduino. The Arduino has 5V and GND pins, which can be connected to the breadboard's power bus strips using jumper wires.
- Ensure that the power lines on the breadboard are correctly connected to the corresponding power and ground lines of the Arduino.

5. Testing Connections: After setting up, it's always a good practice to double-check all connections before powering the circuit.





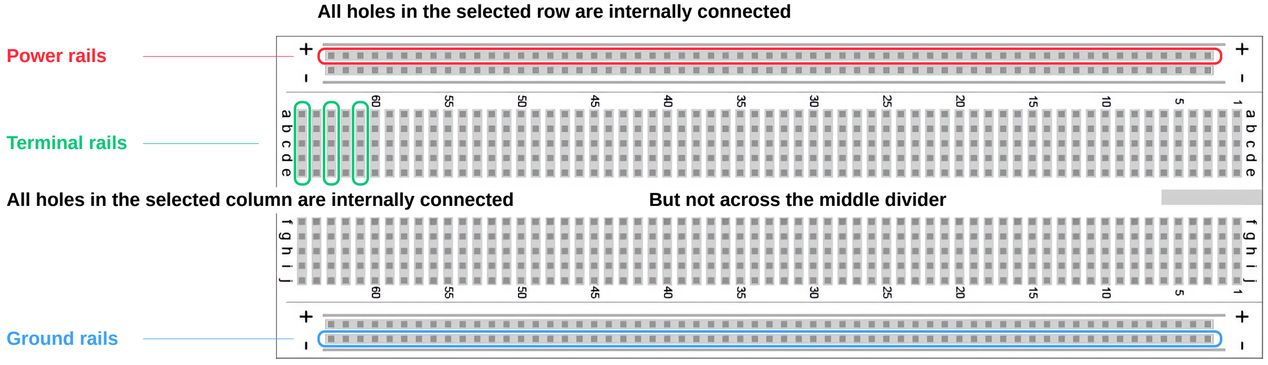


Image sources: https://wiring.org.co/learning/tutorials/breadboard/

Wiring Diagrams and Setup Instructions - DHT 22

About the DHT-22 sensor

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed).

Connections are simple, the first pin on the left to 3-5V power, the second pin to your data input pin and the right most pin to ground.

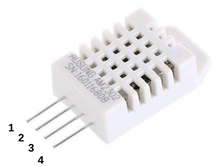
Technical details:
Power: 3-5V
Max Current: 2.5mA
Humidity: 0-100%, 2-5% accuracy
Temperature: -40 to 80°C, ±0.5°C accuracy

What you will need - Hardware

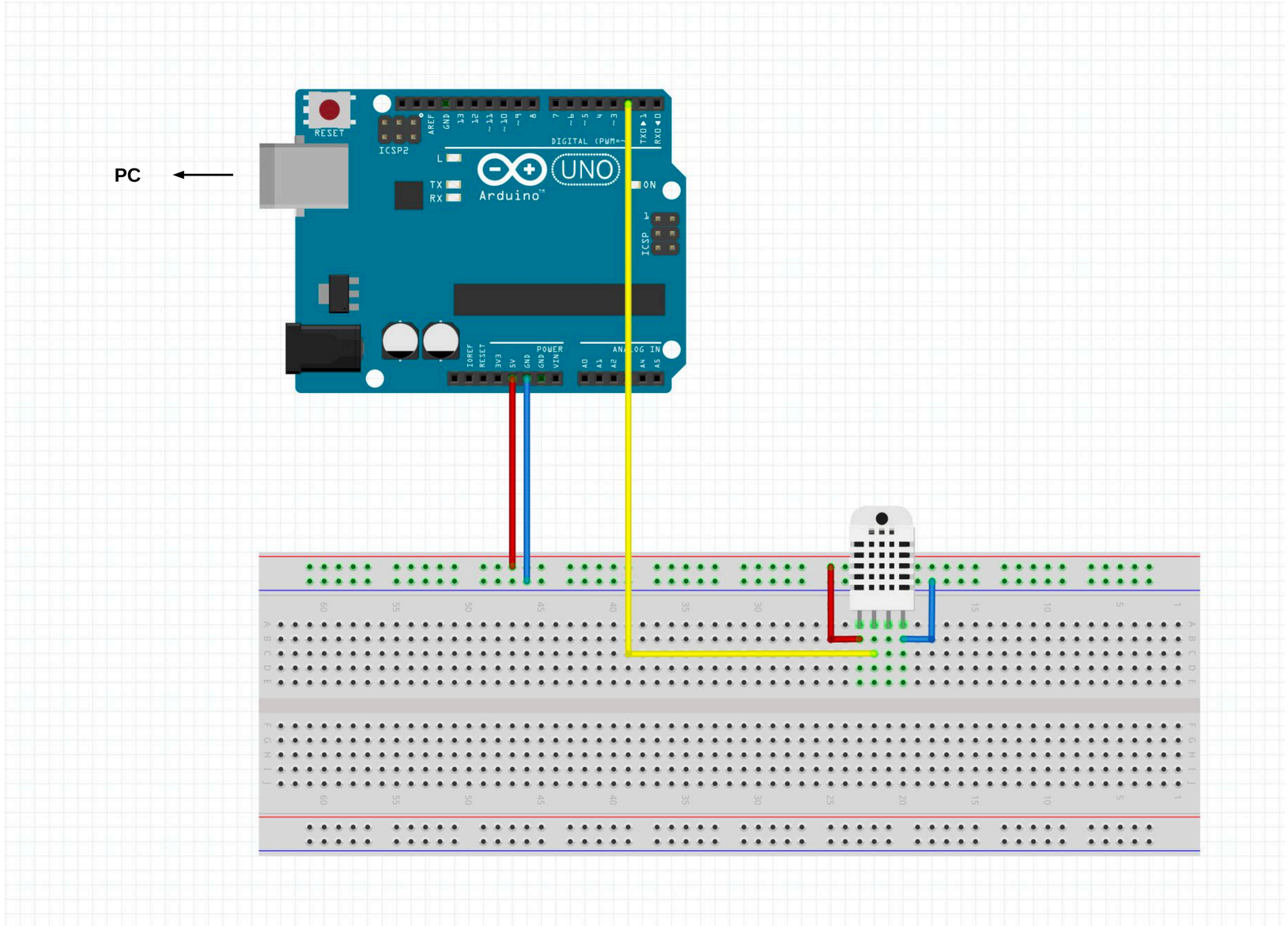
Arduino uno
Breadboard
DHT 22
Jump Wires

DHT 22 pins

1	VCC
2	DATA
3	NC
4	GND



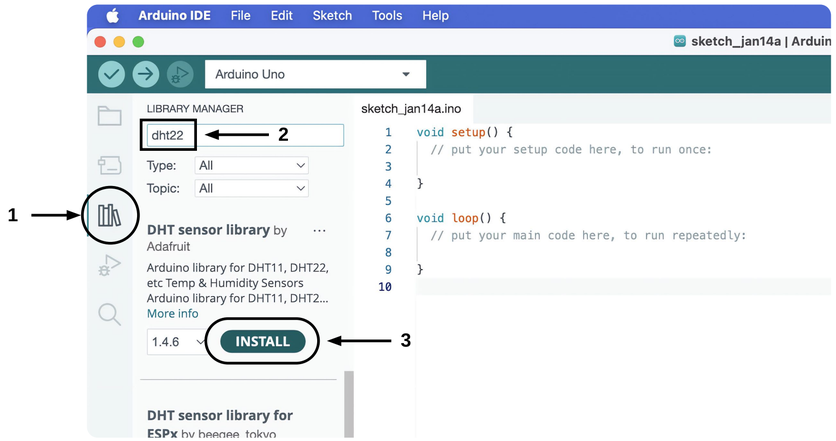
Connection method



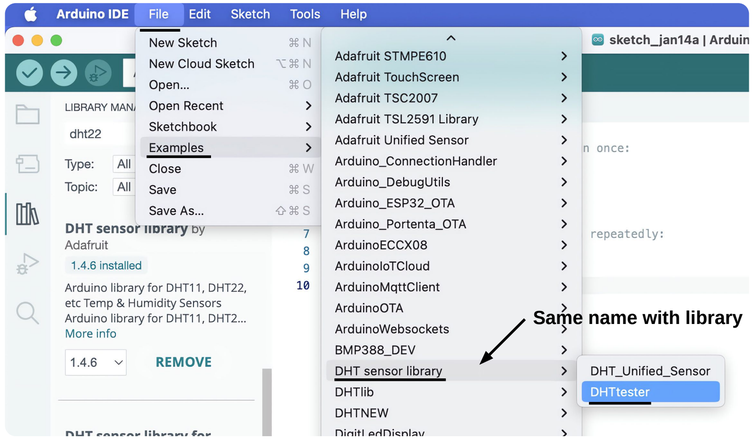
Wiring Diagrams and Setup Instructions
- DHT 22

Setup instructions

1. Open Arduino IDE and search for your sensor model (DHT22) in the Library Manager column, and than install the DHT sensor library (also you can choose other DHT libraries).

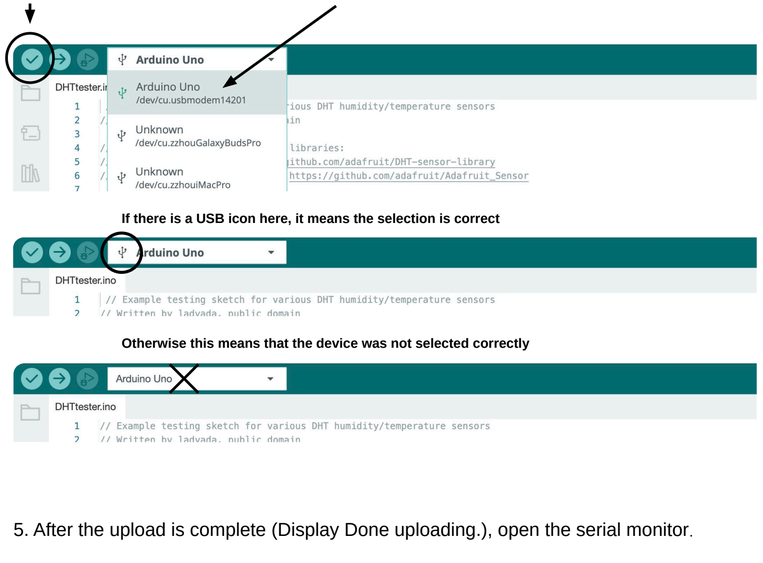


2. After installing the DHT library, open the file example. This is the DHT22 code shared by developers or enthusiasts. It can run successfully after simple modifications.

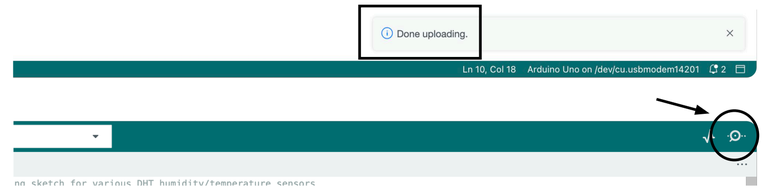


3. Now a new window pops up automatically. This is the code that drives DHT22. You just need to modify the defined DHT22 pin to your pin.

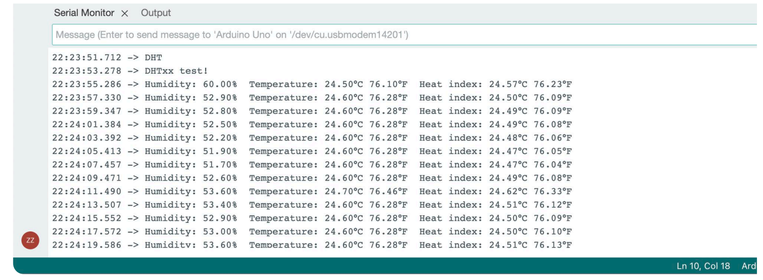
4. Upload the code to your Arduino. (Don't forget to choose your Arduino device correctly.)



5. After the upload is complete (Display Done uploading.), open the serial monitor.



6. Result



Wiring Diagrams and Setup Instructions

- DHT 22 (Serial Monitor - Output text data)

//Code explanation

```ino
#include "DHT.h"

#define DHTPIN 2                          // Define the digital pin connected to the DHT sensor.
#define DHTTYPE DHT22                     // Specify the DHT sensor type. In this case, DHT22.

DHT dht(DHTPIN, DHTTYPE);                 // Initialize the DHT sensor.

void setup() {
  Serial.begin(9600);                     // Initialize serial communication at 9600 bits per second.
  Serial.println(F("DHTxx test!"));       // Print characters "DHTxx test!".

  dht.begin();                            // Initialize the DHT sensor.
}

void loop() {
  delay(2000);                            // Wait for 2000 milliseconds between each measurement.

  float h = dht.readHumidity();           // Read the humidity value.
  float t = dht.readTemperature();        // Read the temperature in Celsius.
  float f = dht.readTemperature(true);    // Read the temperature in Fahrenheit.

  // Check for failed readings from the DHT sensor.
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  float hif = dht.computeHeatIndex(f, h);          // Compute heat index in Fahrenheit.
  float hic = dht.computeHeatIndex(t, h, false);   // Compute heat index in Celsius.

  // Print the results to the serial monitor.
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print("%  Temperature: ");
  Serial.print(t);
  Serial.print("°C ");
  Serial.print(f);
  Serial.print("°F  Heat index: ");
  Serial.print(hic);
  Serial.print("°C ");
  Serial.print(hif);
  Serial.println("°F");
}
```

println means newline

```
Humiditv:
53.60
% Temperature:
24.60
°C
76.28
°F Heat index:
24.51
°C
76.13
°F
```

(This code is available directly copy and use in the Arduino IDE)

Code breakdown

1. Library Inclusion: The DHT.h library is included to interact with the DHT sensor.

2. Pin Configuration: DHTPIN defines the pin to which the DHT sensor is connected.

3. Sensor Type: DHTTYPE specifies the type of DHT sensor (DHT22 in this case).

4. Sensor Initialization: The DHT object is initialized with the pin number and sensor type.

5. Setup Function: The setup() function initializes serial communication and the sensor.

6. Main Loop: The loop() function contains the main logic to read and process sensor data.

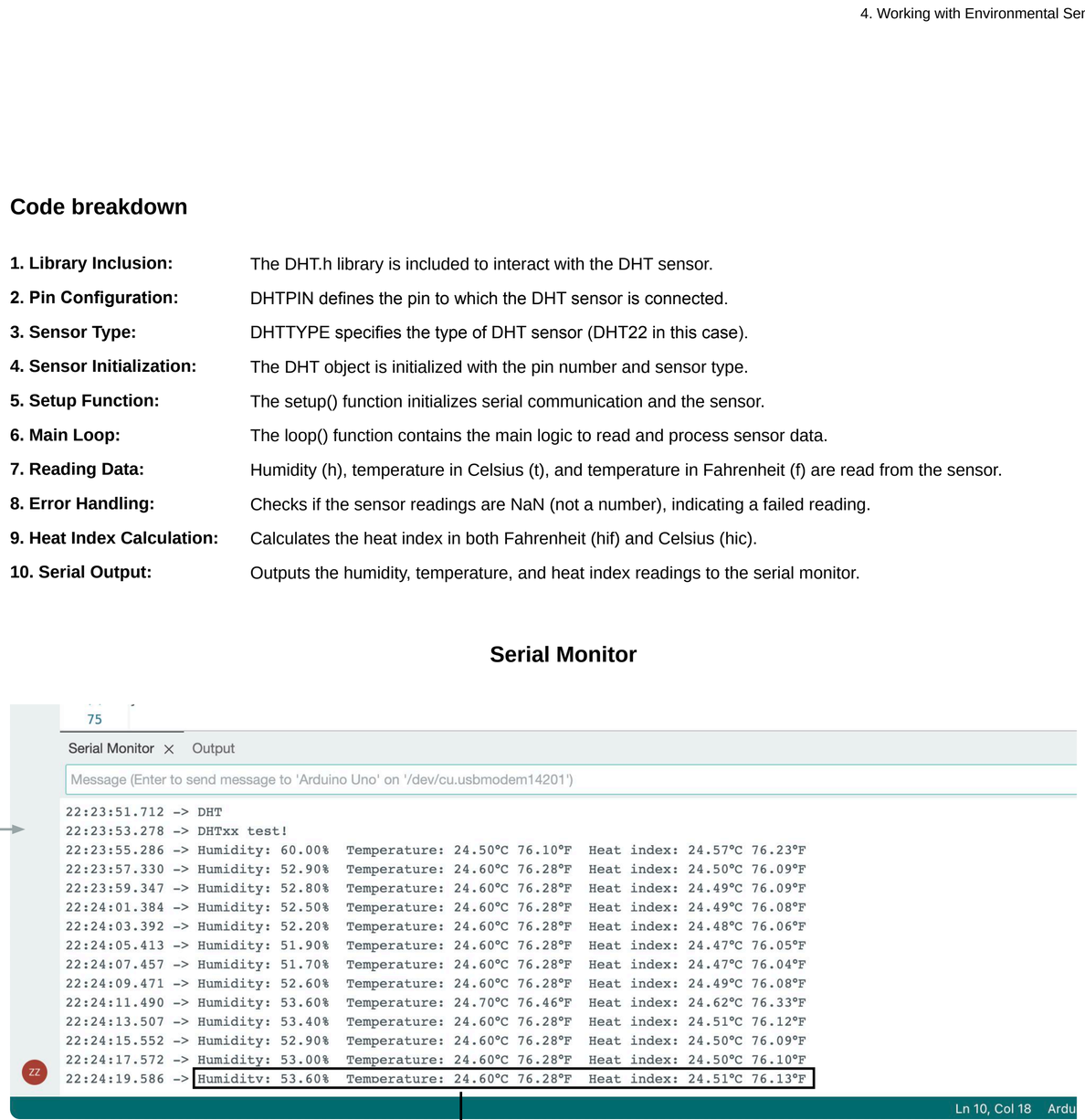
7. Reading Data: Humidity (h), temperature in Celsius (t), and temperature in Fahrenheit (f) are read from the sensor.

8. Error Handling: Checks if the sensor readings are NaN (not a number), indicating a failed reading.

9. Heat Index Calculation: Calculates the heat index in both Fahrenheit (hif) and Celsius (hic).

10. Serial Output: Outputs the humidity, temperature, and heat index readings to the serial monitor.

Serial Monitor



Humiditv: 53.60% Temperature: 24.60°C 76.28°F Heat index: 24.51°C 76.13°F

Wiring Diagrams and Setup Instructions

- DHT 22 (Serial Plotter - Output graphics data)

//Code explanation

```ino
#include "DHT.h"

#define DHTPIN 2                          // Define the digital pin connected to the DHT sensor.
#define DHTTYPE DHT22                     // Specify the DHT sensor type. In this case, DHT22.

DHT dht(DHTPIN, DHTTYPE);                 // Initialize the DHT sensor.

void setup() {
  Serial.begin(9600);                     // Initialize serial communication at 9600 bits per second.
  Serial.println(F("DHTxx test!"));       // Print characters "DHTxx test!".

  dht.begin();                            // Initialize the DHT sensor.
}

void loop() {
  delay(2000);                            // Wait for 2000 milliseconds between each measurement.

  float h = dht.readHumidity();           // Read the humidity value.
  float t = dht.readTemperature();        // Read the temperature in Celsius.
  float f = dht.readTemperature(true);    // Read the temperature in Fahrenheit.

  // Check for failed readings from the DHT sensor.
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  float hif = dht.computeHeatIndex(f, h);          // Compute heat index in Fahrenheit.
  float hic = dht.computeHeatIndex(t, h, false);   // Compute heat index in Celsius.

  // Print the results to the serial plotter.
  Serial.print("Humidity:");
  Serial.print(h);
  Serial.print(",");
  Serial.print("Temperature:");
  Serial.print(t);
  Serial.print(",");
  Serial.print("TemperatureF:");
  Serial.print(f);
  Serial.print(",");
  Serial.print("Heat_indexC:");
  Serial.print(hic);
  Serial.print(",");
  Serial.print("Heat_indexF:");
  Serial.print(hif);
  Serial.println("°F");
}
```

//New Code

```ino
Serial.print("Humidity: ");
Serial.print(h);
Serial.print("%  Temperature: ");
Serial.print(t);
Serial.print("°C ");
Serial.print(f);
Serial.print("°F  Heat index: ");
Serial.print(hic);
Serial.print("°C ");
Serial.print(hif);
Serial.println("°F");
```

Original Code

The Serial Plotter tool is a versatile tool for tracking different data that is sent from your Arduino board. It functions similarly to your standard Serial Monitor tool which is used to print data "terminal style", but is a greater visual tool that will help you understand and compare your data better.

After the previous Serial Monitor tutorial, if we want to use Serial Plotter, we must change the format of Serial.print() to be recognized by Serial Monitor. There are two points to note here: The first is to set the correct number of variables, and the second is to set the label name of each variable.

```ino
Serial.print("Name:");
Serial.print(variable);
Serial.print(",");
......

Serial.println("°F");
```

1st: Maintain format – ("Name" + ":")
2ed: Maintain format – (variable)
3rd: Maintain format – ","

last line: use println make newline

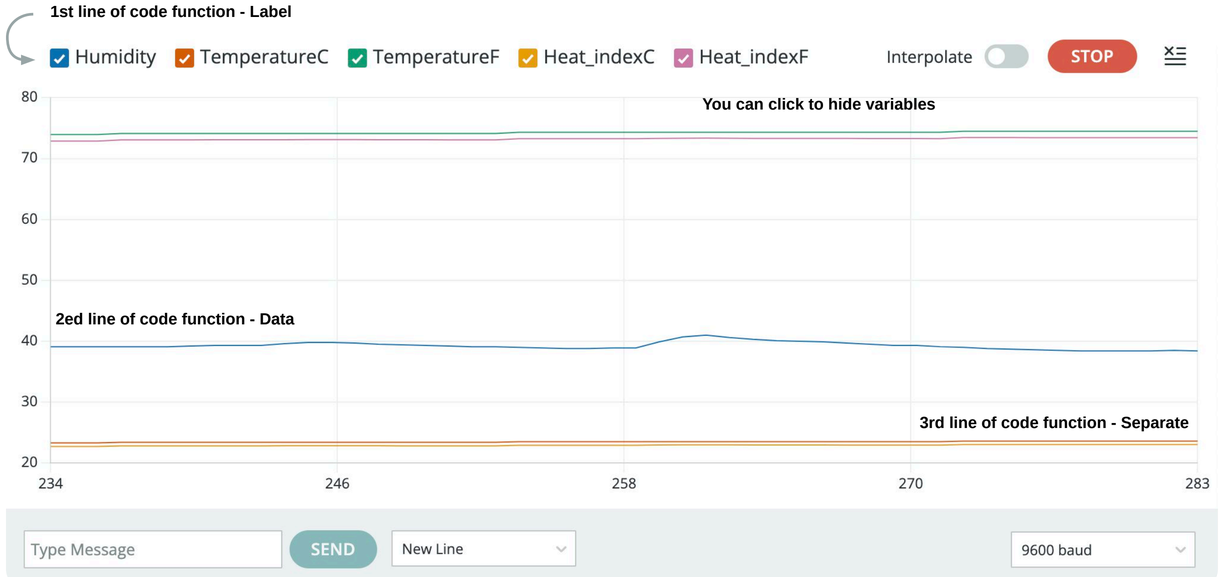
```ino
Serial.print("Humidity:");
Serial.print(h);
Serial.print(",");
```

example

```ino
Serial.print("Humidity:");
Serial.print(h);
Serial.print(",");
Serial.print("Temperature:");
Serial.print(t);
Serial.print(",");
Serial.print("TemperatureF:");
Serial.print(f);
Serial.print(",");
Serial.print("Heat_indexC:");
Serial.print(hic);
Serial.print(",");
Serial.print("Heat_indexF:");
Serial.print(hif);
Serial.println("°F");
```

All code of serial plotter needed

Serial Plotter



5. Data Collection and Analysis

Focusing on the critical aspect of data collection, this chapter covers methods for data logging using SD cards and serial output. It emphasizes organizing and storing data efficiently and introduces basic techniques for analyzing environmental data, including simple visualization methods like spreadsheets or basic plotting in Arduino.

Data Logging

What is Data Logging?

Definition:

Data logging is the systematic process of recording and storing data points over a specified period. In environmental monitoring, this means capturing data like temperature, humidity, or air quality at regular intervals.

Components of a Data Logger:

- Sensors: These are devices that detect and respond to some type of input from the physical environment. For instance, a temperature sensor like the LM35 measures temperature and converts it into a readable digital signal.

- Microcontroller (Arduino): The Arduino acts as the brain of the data logger. It reads the input from the sensors and processes this data. For example, it converts the analog signal from a temperature sensor into a digital value that can be understood and stored.

- Storage Medium: This is where the data is stored. Common storage mediums in Arduino-based projects include SD cards for large data sets or internal memory for smaller ones. The choice of storage depends on the size and nature of the data being collected.

Importance of Data Logging in Environmental Monitoring

Time-Series Data Analysis:

Data logging enables the collection of data at regular intervals, creating a time series. This is vital for observing trends and patterns over time, like the gradual increase in temperature in a particular area.

Predictive Analysis and Trend Identification:

By analyzing the collected data, we can predict future environmental conditions. For example, consistent increases in CO_2 levels in an area may predict poorer air quality in the future.

Decision Making and Policy Formation:

Data collected through logging can be instrumental in informing policy decisions. For example, if data shows a significant increase in pollution levels in a city, it could lead to the implementation of stricter environmental regulations.

How Arduino Fits into Data Logging

Arduino as a Data Collector:

Arduino can be connected to a variety of sensors to collect environmental data. For instance, connecting a CO_2 sensor to Arduino can help monitor air quality.

Arduino’s Flexibility:

The versatility of Arduino lies in its ability to work with a wide range of sensors and its user-friendly programming environment. It can be programmed to collect data as frequently as needed and in the format required.

Storage Options with Arduino:

Arduino can store data in various ways. Using an SD card module, you can write data to an SD card. Alternatively, you can use Arduino's serial communication capabilities to send data to a computer for storage.

Challenges in Data Logging

Data Storage Limitations:

Handling large amounts of data can be challenging. Discuss the limitations of various storage methods and how to optimize storage space, like using data compression techniques.

Data Integrity and Reliability:

Stress the importance of accurate and reliable data collection. Discuss potential sources of error, like sensor inaccuracies or data corruption, and ways to mitigate them.

Power Management:

For remote data logging, managing power consumption is crucial. Introduce Arduino's low-power modes and strategies for efficient power use in remote monitoring setups.

Real-world Applications

Urban Air Quality Monitoring

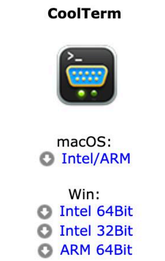
Utilizing sensors to measure pollutants in urban areas, providing data for public health assessments.

Methods of Data Logging
- Computer serial monitor

Using Software - CoolTerm

Without purchasing any additional hardware, it is undoubtedly the easiest way to read and record the data collected by Arduino directly through the computer. However, due to the limitations of the Arduino IDE 2 serial display, we need to use the CoolTerm tool to replace the serial display. Line display, export data into a file format that can be recognized by excel.

https://freeware.the-meiers.org/



Installing CoolTerm

1. Download: Open link above and choose correct version download.

2. Install: After downloading, unzip the file and install.

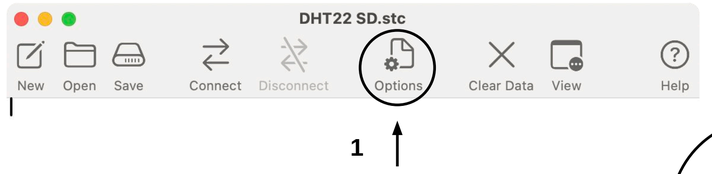
Setting Up CoolTerm with Arduino

1. Connect Your Arduino: Make sure your Arduino is connected to your computer via USB.

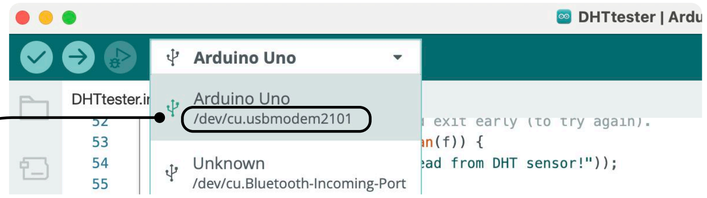
2. Launch CoolTerm: Open CoolTerm from your Applications list.

3. Configure CoolTerm:
   - Select the Serial Port: Go to the "Options" menu. Under the "Serial Port" tab, you should see a list of available ports. Select the port that your Arduino is connected to (usually named something like /dev/tty.usbmodem... or /dev/tty.usbserial...).
   - Set Baud Rate: Set the baud rate to match the rate set in your Arduino sketch (commonly 9600, but it can be different depending on your Arduino code).
   - Other Settings: Leave other settings (like data bits, parity, stop bits, and flow control) to their default values, unless your specific Arduino setup requires different settings.

4. Save Configuration: Once you have configured the settings, save them by clicking "OK".

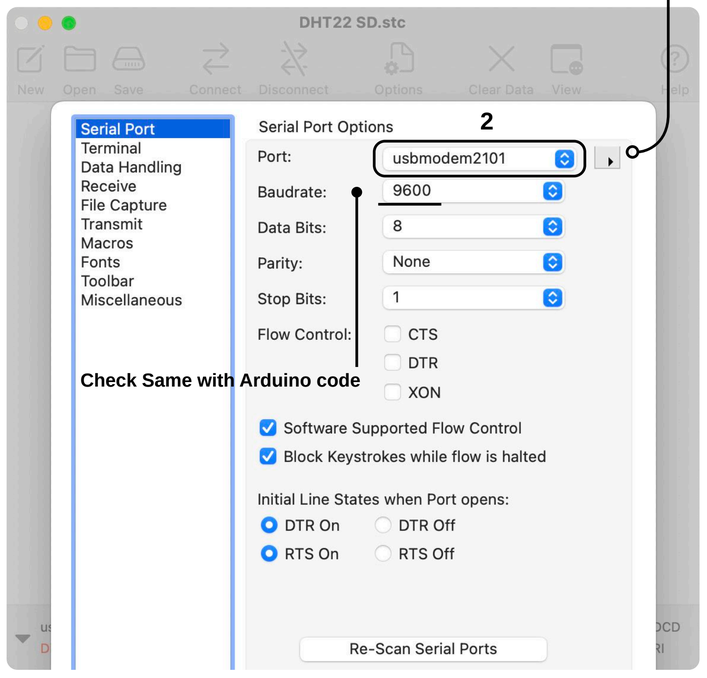


Check Same with Arduino IDE port

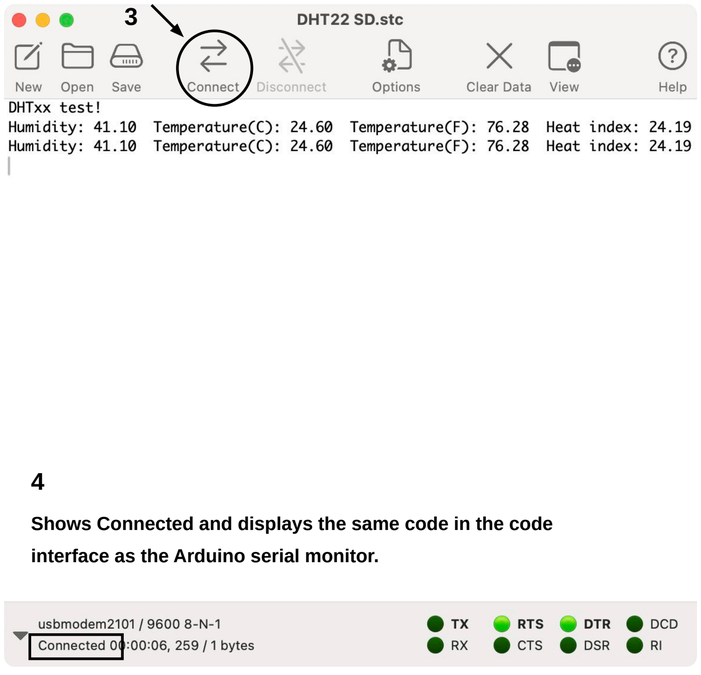


Capturing Data

1. Open the Connection: Click the "Connect" button in CoolTerm. This establishes a connection with your Arduino.
2. Check Data Stream: You should now see data streaming in the CoolTerm window if your Arduino is sending data to the serial port.



Check Same with Arduino code



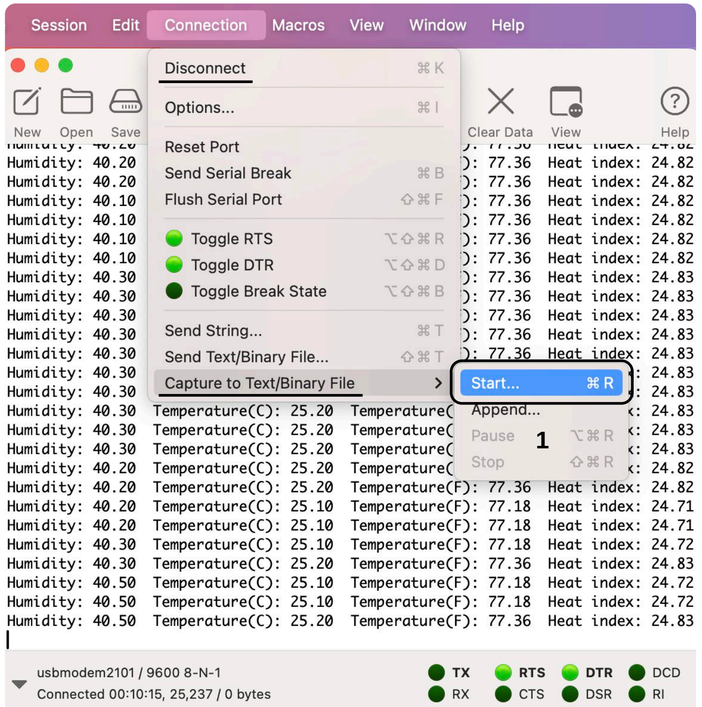
4

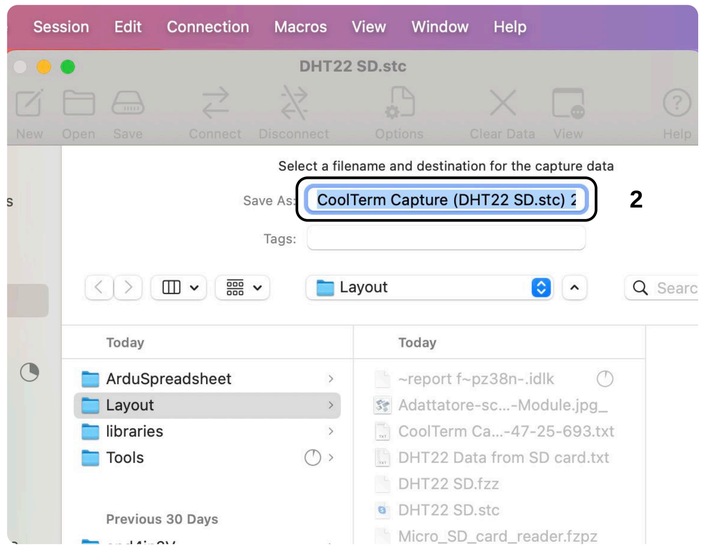
Shows Connected and displays the same code in the code interface as the Arduino serial monitor.

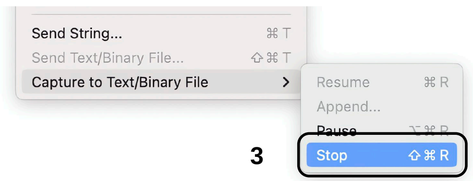
Methods of Data Logging
- Computer serial monitor

Exporting Data to a File by CoolTerm

1. Start Capture: Go to the “Capture” menu in CoolTerm and select “Start Capture To File”.
2. Name Your File: A dialog box will appear asking where to save the file. Choose your desired location and give the file a name.
3. Capture Data: CoolTerm will now start capturing all the data displayed in its window into the file you specified.
4. Stop Capture: Once you have captured enough data, go back to the “Capture” menu and click “Stop Capture”.W







Final Steps

1. Close Connection: Click the "Disconnect" button in CoolTerm to safely close the connection to your Arduino.
2. Open and Analyze Data: Navigate to the location where you saved the file. You can open this file with a text editor or import it into Excel for further analysis.

Additional Tips

- Real-Time Data Monitoring: You can monitor data in real-time in CoolTerm while it is being captured.

- Formatting Data: If you plan to import the data into Excel, consider formatting the data in your Arduino sketch for easier parsing. For example, separate sensor readings with commas to create CSV (Comma Separated Values) format.

- Automated Timestamps: If you need timestamps for each reading, modify your Arduino code to include a timestamp with each data packet.

- Data Consistency: Ensure that your Arduino is consistently sending data at a rate that CoolTerm can handle. If data is being sent too quickly, some of it may be missed.

- Troubleshooting: If you don't see any data in CoolTerm, make sure that no other applications (like the Arduino IDE Serial Monitor) are using the same serial port, as this can block CoolTerm from accessing the port.

- CoolTerm Alternatives: If for some reason CoolTerm doesn't meet your needs, other applications like SerialTools or goSerial can also be used in a similar manner.

+ Since the USB port can only be used by one software at the same time, you cannot use Arduino IDE to upload code at the same time when using CoolTerm. You need to disconnect CoolTerm from Arduino first, and vice versa.

+ Since the data needs to be imported into Excel for further processing, need pay attention to the format of the Arduino data output,

Which format is generally:

"Name" + ":" + "Data" + ";" "Name" + ":" + "Data" + ";" "Name" + ":" + "Data" + ";"

Humiditv: 53.60; Temperature(c): 24.60; Temperature(f): 76.28 Heat index: 24.51

- Finally, you will get a file with Arduino data in .txt format

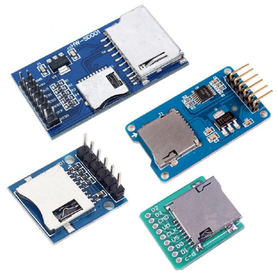
Methods of Data Logging
- MicroSD Cards Module

Using MicroSD Cards

Compared with the previous method of using CoolTerm to export data, the biggest advantage of using an sd card is that it is free from the constraints of a computer, especially when collecting environmental data outdoors or when there is a need for mobility. At this time, the Arduino does not need to be connected to the computer, just It needs to be connected to a small power bank to work.

Equipment Needed:

Arduino board (e.g., Arduino Uno)
SD card module (e.g., Catalex MicroSD Card Module)
MicroSD card (formatted to FAT16 or FAT32)
Breadboard
Jumper wires



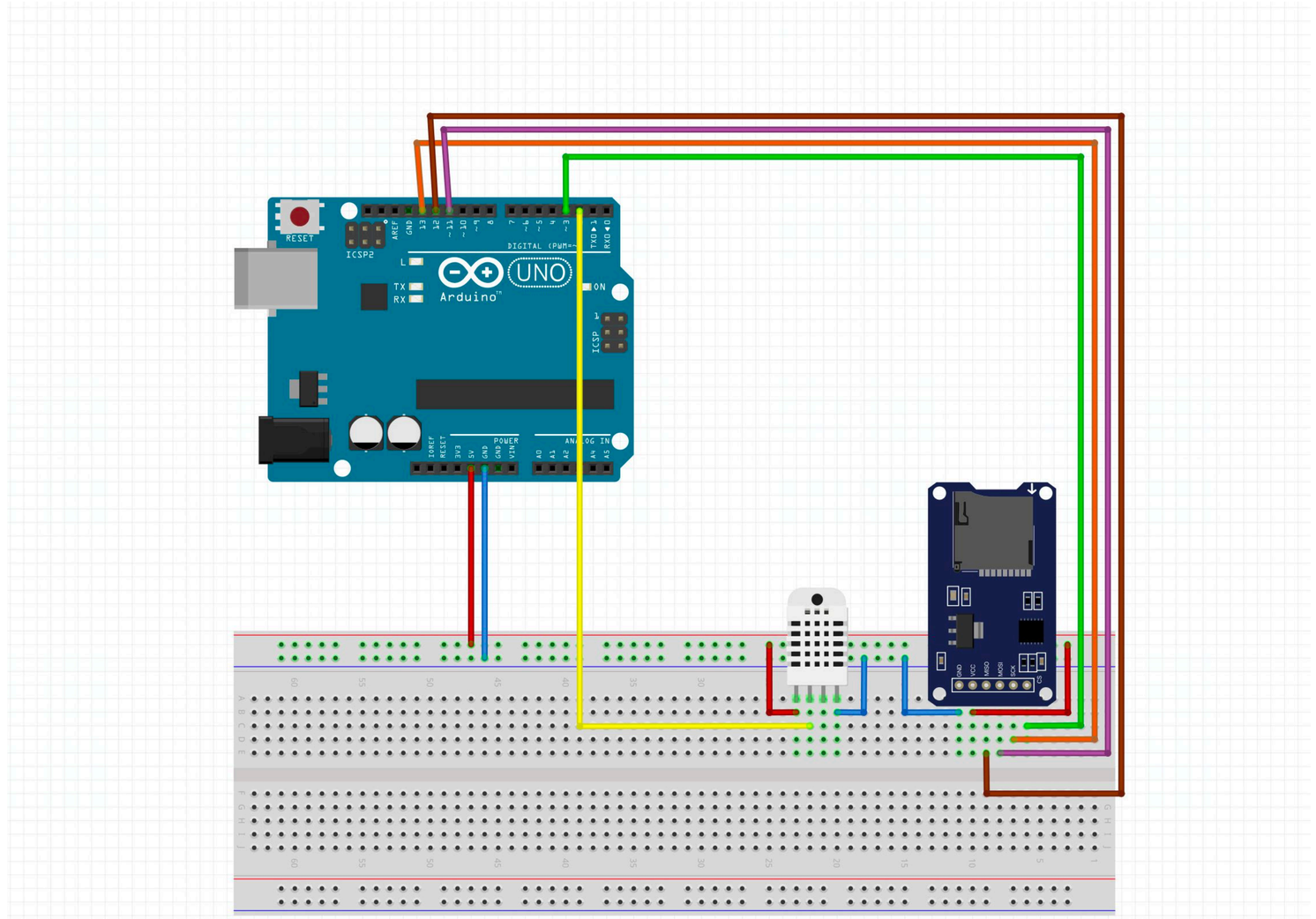
Step-by-Step Instructions:

1. Connecting the SD Card Module to Arduino

- Power Connections:
  - Connect the VCC pin of the SD card module to the 5V pin on the Arduino.
  - Connect the GND pin of the SD card module to one of the GND pins on the Arduino.
- Data Connections:
  - Connect MOSI (Master Out Slave In) pin of the SD card module to digital pin 11 on the Arduino.
  - Connect MISO (Master In Slave Out) pin of the SD card module to digital pin 12 on the Arduino.
  - Connect SCK (Serial Clock) pin of the SD card module to digital pin 13 on the Arduino.
  - Connect CS (Chip Select) pin of the SD card module to digital pin 4 on the Arduino (this pin can be changed in the code if necessary).

2. Preparing the SD Card

- Use a computer to format the MicroSD card to FAT16 or FAT32.
- Insert the SD card into the SD card module.



Methods of Data Logging
- MicroSD Cards Adapter

3. Arduino IDE Setup

- Open the Arduino IDE.
- Include the SD library by adding #include <SD.h> at the beginning of your code.

4. Initializing the SD Card

In the setup() function, initialize the SD card with SD.begin(CS_PIN), where CS_PIN is the chip select pin used in your connection (e.g., 4).

5. Writing Data to the SD Card

- Open a file in write mode: File dataFile = SD.open("datalog.txt", FILE_WRITE);
- Write sensor data to the file: dataFile.println(sensorData);
- Close the file after writing: dataFile.close();

6. Error Checking

- Implement error checking to ensure the file opens. If it fails to open, print an error message to the Serial Monitor.

//Program

```
#include <SD.h>             // Including SD Card Library
#include <SPI.h>            // Including SPI Library
#include "DHT.h"            // Including DHT Sensor Library

#define DHTPIN 2            // Declaring the pin to which DHT data pin is connected
#define DHTTYPE DHT22       // Selecting the type of DHT sensor used in this project

long seconds=00;            // For storing second
long minutes=00;            // For storing minute
long hours=00;              // For storing hour

int CS_pin = 10;            // Chip select pin of SD card module

DHT dht(DHTPIN, DHTTYPE);   // Declaring DHT object
File sd_file;               // Declaring File object

void setup()  {
  Serial.begin(9600);       // Start serial communication at 9600 baud rate
  pinMode(CS_pin, OUTPUT);  // Set the chip select pin as an output
  dht.begin();              // Initialize the Arduino for reading data from DHT sensor

  // SD Card Initialization
  if (SD.begin())  {
    Serial.println("SD card is initialized. Ready to go");
  }
  else  {
    Serial.println("Failed");
    return;                 // Exit setup if SD card initialization fails
  }

  sd_file = SD.open("data.txt", FILE_WRITE);     // Open a file on the SD card in write mode

  if (sd_file)  {                                // Check if the file is available and write the headers
    Serial.print("Time");                       // Write headers to serial monitor
    Serial.print(",");
    Serial.print("Humidity");
    Serial.print(",");
    Serial.print("Temperature_C");
    Serial.print(",");
    Serial.print("Temperature_F");
    Serial.print(",");
    Serial.println("Heat_index");

    sd_file.print("Time");                       // Write headers to the file
    sd_file.print(",");
    sd_file.print("Humidity");
    sd_file.print(",");
    sd_file.print("Temperature_C");
    sd_file.print(",");
    sd_file.print("Temperature_F");
    sd_file.print(",");
    sd_file.println("Heat_index");
  }
  sd_file.close();                               //closing the file
}
```

Explanation

First of all, we have added libraries required for this project. Then we defined the DHT sensor type and pin to which it is connected.

In the setup function, we initialized the serial communication at 9600 baudrate. **Here serial communication is used only for displaying logs in the Arduino Serial Monitor.** The dht.begin() function will initialize the Arduino for reading data from DHT sensor. Similarly sd.begin() function will initialize the Arduino for write data to SD card module. After that we will open the file “data.txt” from SD card using sd.open() a. Then we will write titles ‘Time’, ‘Humidity’, ‘Temperature_C’, ‘Temperature_F’ and ‘Heat_index’ in it. After that we will close the connection to save the file.

(This code is available directly copy and use in the Arduino IDE)

Information sources: https://electrosome.com/temperature-humidity-data-logger-arduino/

5.1 Storing Sensor Data

Methods of Data Logging
- MicroSD Cards Adapter

Tips and Best Practices

- Regular Data Backup: Regularly back up data from the SD card to prevent data loss.

- Power Supply Considerations: Ensure a stable power supply to Arduino during long-term data logging to prevent data corruption.

- SD Card Capacity: Choose an appropriately sized SD card based on your data logging frequency and duration.

//Program

```
void loop()  {
  sd_file = SD.open("data.txt", FILE_WRITE);    // Reopen the file in write mode
  if (sd_file)  {          // If the file opened successfully, call senddata function
    senddata();
  }
  else  {                  // if the file didn't open, print an error:
    Serial.println("error opening file");
  }
  delay(1000);             // Delay for a second before next iteration
}

void senddata()  {
  // Loop to collect data every 2 seconds for a minute
  for(long seconds = 00; seconds < 60; seconds=seconds+2)  {
    float temp = dht.readTemperature(); //Reading the temperature as Celsius and storing in temp
    float hum = dht.readHumidity();     //Reading the humidity and storing in hum
    float fah = dht.readTemperature(true); // Read temperature in Fahrenheit and store in fah
    float heat_index = dht.computeHeatIndex(fah, hum);

    sd_file.print(hours);               // Write data to the SD file sd_file.print(hours);
    sd_file.print(":");
    sd_file.print(minutes);
    sd_file.print(":");
    sd_file.print(seconds);
    sd_file.print(",  ");
    sd_file.print(hum);
    sd_file.print(",    ");
    sd_file.print(temp);
    sd_file.print(",      ");
    sd_file.print(fah);
    sd_file.print(",      ");
    sd_file.println(heat_index);

    Serial.print(hours);               // Also print the same data to the serial monitor
    Serial.print(":");
    Serial.print(minutes);
    Serial.print(":");
    Serial.print(seconds);
    Serial.print(",  ");
    Serial.print(hum);
    Serial.print(",    ");
    Serial.print(temp);
    Serial.print(",       ");
    Serial.print(fah);
    Serial.print(",      ");
    Serial.println(heat_index);

    if(seconds>=58)  {                 // Update the minute every 60 seconds
      minutes= minutes + 1;
    }

    if (minutes>59)  {                 // Reset minutes and increment hours after 59 minutes
      hours = hours + 1;
      minutes = 0;
    }

    sd_file.flush(); //saving the file

    delay(2000);
  }
  sd_file.close();   //closing the file
}
```

Explanation

In the loop function, we will open the SD card “data.txt” file again. If it is opened successfully we will call senddata() function, otherwise it will show ‘error opening file’ in the serial monitor.

In the senddata() function, we have made a loop which will count the time. Then we have read the temperature, humidity from DHT22 sensor and heat index is calculated. After storing these values in the variables, we will write these values to the SD card file. After that we will call sd_file.flush() to ensure that all data is stored in the SD card. These steps are repeated with a delay of 2 seconds.

(This code is available directly copy and use in the Arduino IDE)

Information sources: https://electrosome.com/temperature-humidity-data-logger-arduino/

Efficient Data Storage

Choosing the Right Data Format

Creating a CSV File for Data Logging

What is CSV: Briefly explain that CSV stands for Comma-Separated Values, a simple file format used to store tabular data.
Advantages of CSV:
- Compatibility: CSV files are compatible with many applications, including spreadsheet software like Excel and data analysis tools.
- Simplicity: Easy to read and write with minimal coding required.
- Efficiency: CSV files can be more space-efficient than other text formats like JSON or XML.

Steps to Implement CSV Data Logging

Step 1: Define the Data Structure:
- Decide on the data points to log (e.g., temperature, humidity, timestamp).
- Plan the CSV header accordingly (e.g., timestamp,temperature,humidity).

Step 2: Writing Code for CSV Logging:
- Use the File object to create and open a CSV file on the SD card.
- Write the header to the file: dataFile.println("timestamp,temperature,humidity");.
- For each data entry, format the string in CSV format and write to the file: dataFile.println(String(millis()) + "," + String(temperature) + "," + String(humidity));.
- Remember to close the file after each write operation to save the data.

Maintaining Data Integrity

Handling Power Failures and Safe Write Operations:
- Use the flush() method after each write operation to ensure data is written to the SD card.
- Implement a check to verify if the file is open before writing: if (dataFile) { // write data }.
- Consider a battery backup system for the arduino.

Reducing Data Size

Implementing Data Precision and Selective Logging

Step 1: Data Precision:
- Understand the required precision for each sensor data point.
- For example, reduce the decimal places for float values:
  float temperature = round(sensorValue * 100.0) / 100.0; // two decimal places.

Step 2: Selective Logging:
- Implement a condition to log data only when there is a significant change.
- Example: if(abs(lastTemp - currentTemp) > threshold) { // log data }.
- Define threshold based on the required sensitivity.

Writing Efficient Code for Data Storage

Using Buffers and Efficient String Operations:
- Implement a buffer to temporarily store multiple data points.
- Write the buffered data to the SD card in a single operation to reduce write cycles.
- Optimize string operations
  (use string concatenation carefully to avoid excessive memory use)

Arduino to prevent data loss during power outages.

Proper File Closing:
- Always close the file after writing: dataFile.close();.
- In case of an error or an interrupt, ensure file closure in the error handling routine.

Efficient Data Retrieval

Indexing and Optimized Reading:
- Create a simple index for large data files, which could be as simple as noting file positions of data entries at regular intervals.
- For optimized reading, load data in chunks instead of line by line, especially when dealing with large files.

Data Compression Techniques

Implementing Basic Compression in Arduino

Step 1: Understanding Run-Length Encoding (RLE)
- Concept: RLE compresses data by reducing repeated values to a single value and a count. For example, AAAABBBCCDAA becomes 4A3B2C1D2A.
- Suitability: It's most effective for data with lots of repetition.

Step 2: Writing RLE Code for Arduino
- Initial Setup: Create variables to store the current character, count, and the previous character in your data stream.
- Loop Through Data: Iterate through the sensor data string. If the current character is the same as the previous one, increment the count. If it's different, write the count and previous character to the file, then reset the count.
- Finalize: After the loop, ensure the last set of characters and count are written to the file.

Regular Maintenance and Monitoring

SD Card Health Check
- Routine Checks: Regularly check the SD card for errors or corruptions, especially after unexpected shutdowns or errors.
- Format and Maintenance: Occasionally formatting the SD card (after backing up data) can maintain its health and performance.

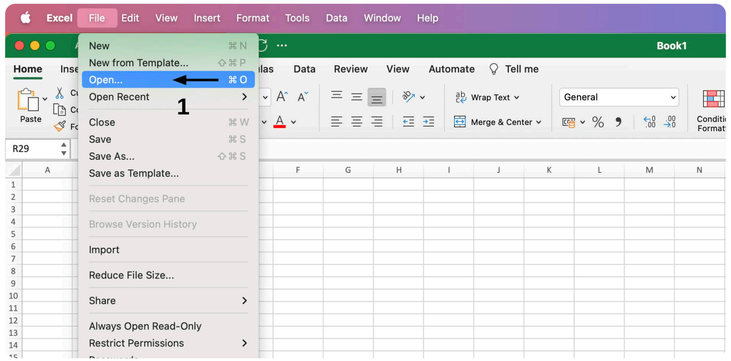
Monitoring Storage Capacity
- Capacity Check: Implement code to check the remaining capacity of the SD card. This can prevent data loss due to the card being full.
- Storage Management: Provide instructions on how to transfer data from the SD card to a computer or cloud storage for long-term archiving, and then clear the card for new data.

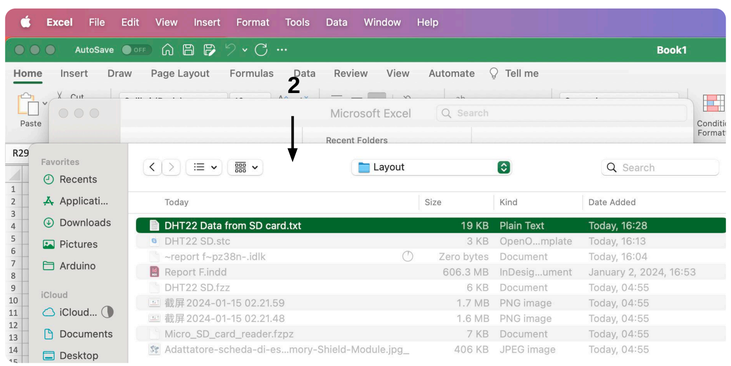
Visualizing Data

Importing Logs to Excel

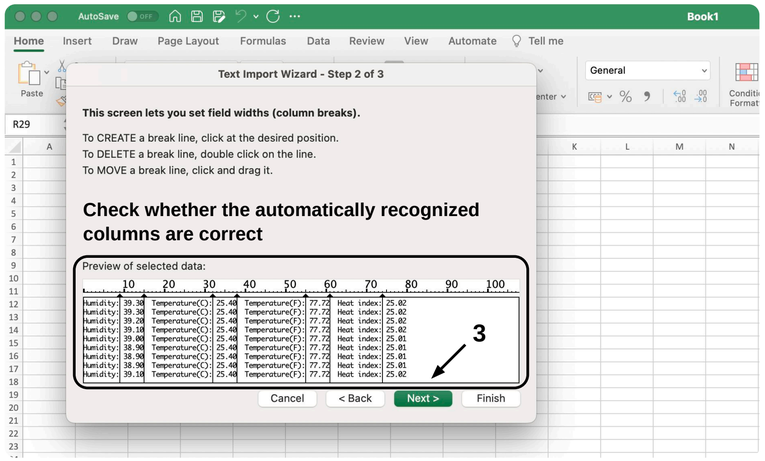
For viewing or evaluating the logged data, you can take the SD card from the module and connect it to a PC via a SD card reader.

1. Open Microsoft Excel
2. Press Office button or File
3. Click Open
4. Select the file from the CoolTerm or SD card and open it.

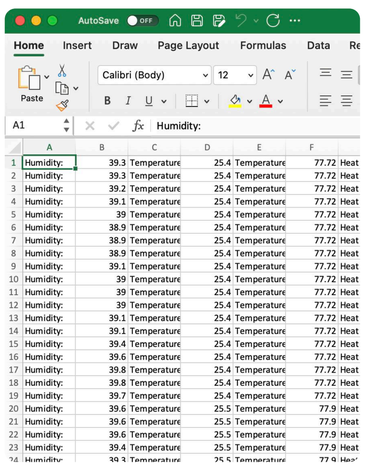


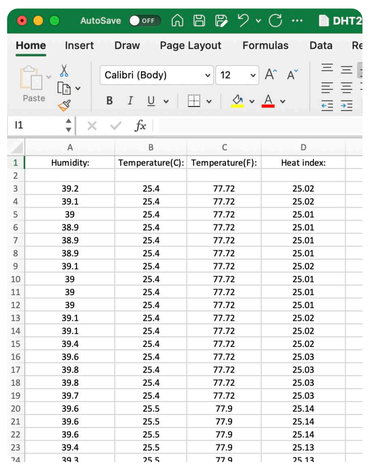


5. Check in the pop-up window whether the vertical separator is correct.
6. Adjust the format slightly.



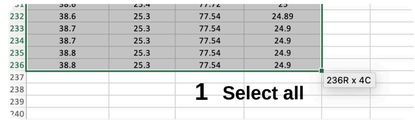
Manual adjustment and optimization

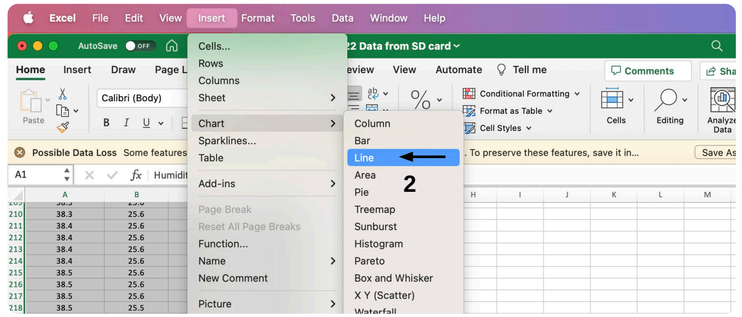




Generating a Graph

1. Select all data
1. Go to Insert tab
2. Click on Line option
3. Select 2D Line2. Press Office button or File

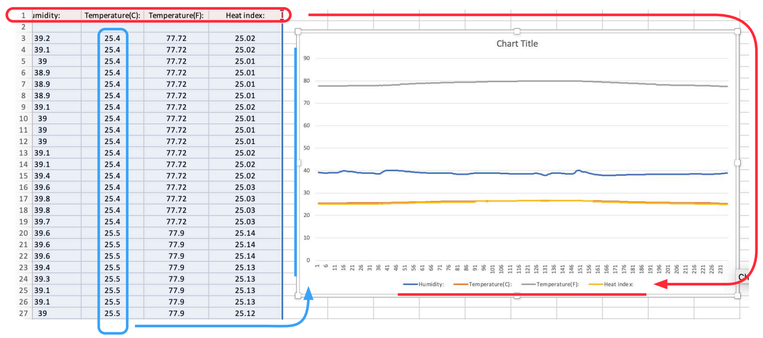




The graph will look like this. You can also customize this graph by using other options.

X-axis: Excel horizontal first row elements (Humidity, Temperature etc.)

Y-axis: Excel A, B, C..... (Data of Arduino)



Basic Data Analysis

Data processing

Understanding Your Data:

- Open your CSV file in a spreadsheet program (like Excel or Google Sheets). Here, each column corresponds to different data types (e.g., Column A: Temperature, Column B: Humidity, Column C: Timestamp).
- Scroll through your data to identify any anomalies, such as missing values, outliers, or evident errors (like a temperature reading that is outside the possible range).

Calculating Basic Statistics:

Average (Mean):
- In a spreadsheet, you can calculate the average of a dataset. For instance, if your temperature data is in column A, from cell A2 to A100, you would use the formula **'=AVERAGE(A2:A100)'** . This formula computes the mean value of all temperature readings in the specified range.

Minimum and Maximum Values:
- Determine the lowest and highest readings using spreadsheet formulas. **'=MIN(A2:A100)'** gives you the lowest temperature, while **'=MAX(A2:A100)'** gives you the highest in the specified range.

Standard Deviation:
- Standard deviation shows the variation or dispersion of a set of values. Use **'=STDEV.P(A2:A100)'** to calculate it for your temperature data. A high standard deviation indicates that the readings are spread out over a wider range, whereas a low standard deviation indicates that the readings are more clustered together.

Trend Analysis:

- Apply a trendline in your spreadsheet to understand the direction of your data over time. For example, in a temperature vs. time graph, adding a linear trendline can help visualize whether the temperature is generally increasing, decreasing, or staying constant over the observed period.

Text to graphics

Using Spreadsheets for Data Visualization:

1. Creating a Line Chart for Trends:
   - Highlight the data range including dates/times (e.g., Column C) and corresponding sensor readings (e.g., Column A for temperature).
   - Navigate to 'Insert' > 'Chart', select 'Line' as your chart type. This will plot your selected data in a line graph, ideal for observing trends over time.

2. Customizing Your Chart:
   - Add a meaningful title (e.g., 'Temperature Trends Over Time').
   - Label your axes for clarity, with the X-axis representing time and the Y-axis representing the sensor reading (e.g., temperature in Celsius).
   - Adjust the color and style of your chart for better readability and visual appeal. You might change the line color or thickness, adjust the scale of your axes, or add gridlines for easier reading.

Basic Plotting in Arduino:

1. Using the Serial Plotter:
   - In your Arduino code, ensure you have **'Serial.begin(9600);'** in the setup() function to start serial communication.
   - Where you read your sensor, add **'Serial.println(sensorValue);'**. For instance, **'Serial.println(temperature);'** after reading the temperature sensor.

2. Observing Data with Serial Plotter:
   - Upload the code to your Arduino and open the Serial Plotter via 'Tools' in the Arduino IDE.
   - The plotter will display real-time graphs based on your sensor readings. Each point on the graph represents a sensor reading taken at a particular instance.

3. Interpreting the Plot:
   - Watch for trends or sudden changes in the graph. For instance, a sudden spike in the temperature graph could indicate an external heat source affecting the sensor.

Interpreting the Results

Contextualizing Data:

- Relate the observed data trends to real-world events or environmental conditions. For example, correlating a rise in humidity readings with known rainfall events in the area.

Error and Quality Analysis:

- Reflect on potential errors in data collection. These could be from sensor inaccuracies, environmental factors affecting the sensor (like a temperature sensor in direct sunlight), or data logging issues.

Predictive Insights:

- Utilize observed data trends to make informed predictions. For example, if pollution levels consistently rise during certain hours, predict similar trends under similar conditions in the future.

Summary

In this extension, a brief summary of the previous content is provided to provide a comprehensive understanding of the process of analyzing environmental data collected with an Arduino. By walking through the detailed steps of data analysis, from understanding raw data to extracting meaningful insights through statistical analysis and visualization, the tools necessary to effectively interpret and utilize data are introduced. These skills are crucial for anyone looking to use Arduino in environmental observation and monitoring.

An exploration of various environmental sensors such as temperature, humidity, and air quality sensors is presented. This chapter assists readers in selecting the right sensor for their projects and provides detailed instructions on connecting these sensors to Arduino, including wiring diagrams and steps to read sensor data effectively.

6. Building Environmental Projects

6.1 Project 1: DIY Weather Station

Basic Data Analysis

Objective

The objective of this project is to design and build a comprehensive DIY Weather Station using Arduino, capable of monitoring and displaying key environmental parameters including temperature, humidity, light intensity, and atmospheric pressure. By integrating a DHT22 sensor for precise temperature and humidity readings, a TSL2591 sensor for measuring light intensity, and a BMP390 sensor for atmospheric pressure detection, students will learn about sensor interfacing, data acquisition, and real-time data display using a 1602A LCD screen. This project aims to provide hands-on experience in environmental monitoring technology, foster problem-solving skills, and inspire an interest in atmospheric sciences and electronics.

Components Needed

- Arduino board (e.g., Arduino Uno)
- DHT22 sensor (for temperature and humidity)
- TSL2591 sensor (for light intensity)
- BMP390 sensor (for atmospheric pressure)
- 1602A LCD display (with I2C module for easier connection)
- Breadboard
- Jumper wires

Libraries Needed

DHT sensor library
Adafruit TSL2591 library
Adafruit BMP3XX library
LiquidCrystal_I2C library

Assembly Instructions

1. Connecting the DHT22 Sensor:

VCC to 5V on Arduino.
GND to Ground.
DATA to a digital pin (e.g., D2).

2. Connecting the TSL2591 Light Sensor:

VCC to 3.3V on Arduino.
GND to Ground.
SCL to A5.
SDA to A4.

3. Connecting the BMP390 Pressure Sensor:

VCC to 3.3V on Arduino.
GND to Ground.
SCL to A5 (shared with TSL2591).
SDA to A4 (shared with TSL2591).

4. Connecting the 1602A LCD Display:

GND to Ground.
VCC to 5V on Arduino.
SDA to A4 (shared with TSL2591 and BMP390).
SCL to A5 (shared with TSL2591 and BMP390).
Programming the Weather Station:

7. Troubleshooting and Practices

Addressing common challenges faced in Arduino projects, this chapter offers solutions and troubleshooting tips. It also discusses best practices in Arduino programming and circuit design, emphasizing coding standards, conventions, and safety guidelines.

The final chapter encourages further exploration in advanced Arduino topics, such as IoT applications and wireless connectivity. It also provides guidance on getting involved in the Arduino community and utilizing online resources to continue learning and improving your skills.

8. Advancing Your Skills

8.1 Exploring Advanced Topics

8.1.1 Introduction to IoT with Arduino

8.1.2 Using Arduino with wireless modules

8.2 Community and Further Resources

8.2.1 Joining the Arduino community

8.2.2 Online resources and forums for continuous learning

9. More sensor code examples

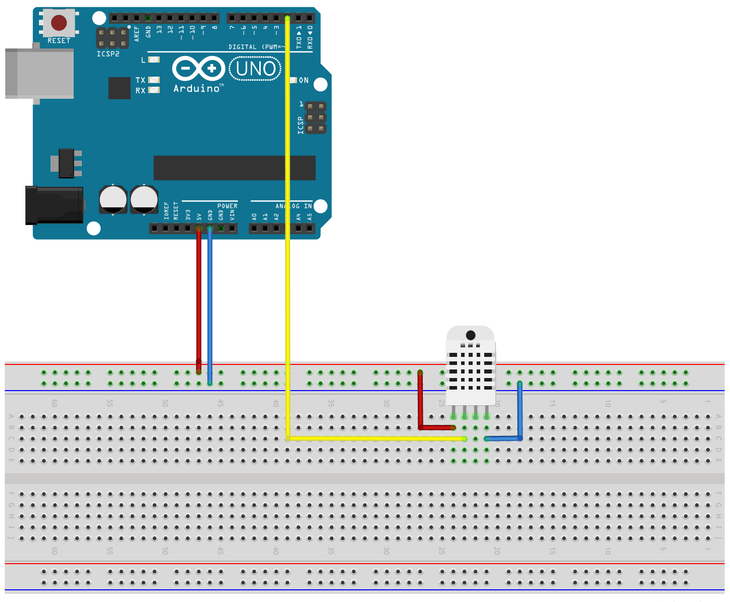
DHT 11 & DHT 22

About the DHT 11 & 22 sensor

The DHT 11 & 22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed).

Connections are simple, the first pin on the left to 3-5V power, the second pin to your data input pin and the right most pin to ground.

Technical details:	**DHT 11**	**DHT 22**
Power:	3.3V 5V	3.3V 5V
Max Current:	2.5mA	2.5mA
Humidity:	20-80%, ±5%	0-100%, ±2-5%
Temperature:	0 to 80°C, ±2°C	-40 to 80°C, ±0.5°C
Protocol:	One-Wire	One-Wire



Hardware Prepare

Arduino uno

Breadboard

Jump Wires

DHT 22

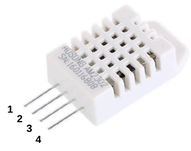
Wiring method

DHT 22 pins		Arduino pins	
1	VCC	-	5v
2	DATA	-	D2
3	NC		
4	GND	-	GND

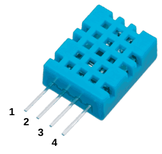
Arduino library

DHT sensor library by Adafruit

or other



DHT 22



DHT 11

//Code explanation

```ino
#include "DHT.h"

#define DHTPIN 2                                // Define the digital pin connected to the DHT sensor.
#define DHTTYPE DHT22                           // Specify the DHT sensor type. In this case, DHT22.

DHT dht(DHTPIN, DHTTYPE);                       // Initialize the DHT sensor.

void setup() {
  Serial.begin(9600);                           // Initialize serial communication at 9600 bits per second.
  Serial.println(F("DHTxx test!"));             // Print characters "DHTxx test!".

  dht.begin();                                  // Initialize the DHT sensor.
}

void loop() {
  delay(2000);                                  // Wait for 2000 milliseconds between each measurement.

  float h = dht.readHumidity();                 // Read the humidity value.
  float t = dht.readTemperature();              // Read the temperature in Celsius.
  float f = dht.readTemperature(true);          // Read the temperature in Fahrenheit.

  // Check for failed readings from the DHT sensor.
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  float hif = dht.computeHeatIndex(f, h);       // Compute heat index in Fahrenheit.
  float hic = dht.computeHeatIndex(t, h, false);   // Compute heat index in Celsius.

  // Print the results to the serial plotter.
  Serial.print("Humidity:");
  Serial.print(h);
  Serial.print(",");
  Serial.print("Temperature:");
  Serial.print(t);
  Serial.print(",");
  Serial.print("TemperatureF:");
  Serial.print(f);
  Serial.print(",");
  Serial.print("Heat_indexC:");
  Serial.print(hic);
  Serial.print(",");
  Serial.print("Heat_indexF:");
  Serial.print(hif);
  Serial.println("°F");
}
```

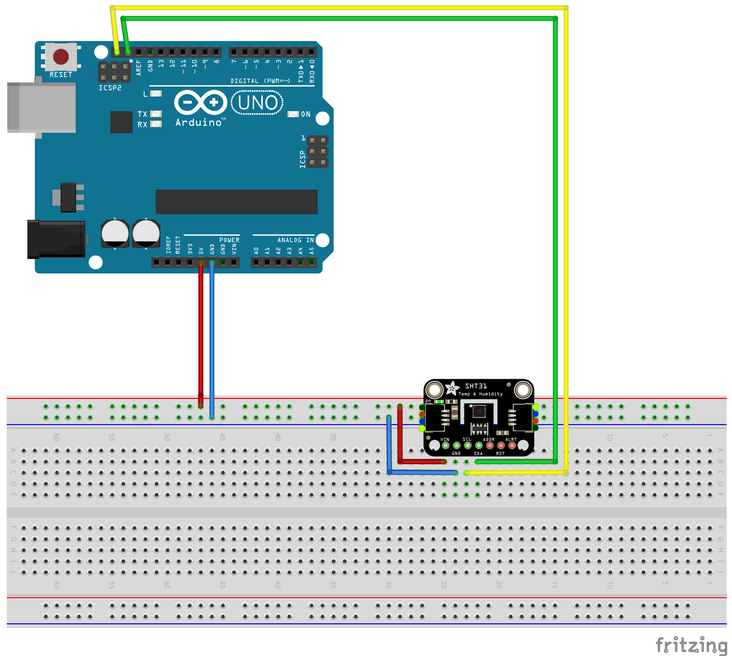
SHT 31

About the SHT 31 sensor

Sensirion Temperature/Humidity sensors are some of the finest & highest-accuracy devices you can get. The SHT31-D sensor has an excellent ±2% relative humidity and ±0.3°C accuracy for most uses.

Unlike earlier SHT sensors, this sensor has a true I2C interface, with two address options. It also is 3V or 5V compliant, so you can power and communicate with it using any microcontroller or microcomputer.

Technical details:	**SHT 31**
Power:	3.3V 5V
Humidity:	0-100%, ±2%
Temperature:	-40 to 125°C, ±0.3°C
Protocol:	I2C
I2C address:	0x44



Hardware Prepare

Arduino uno
Breadboard
Jump Wires
SHT 31

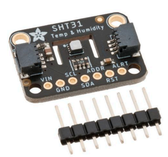
Wiring method

SHT 31 pins		Arduino pins	
1	VCC	-	5v
2	GND	-	GND
3	SCL	-	SCL
4	SDA	-	SDA
5	ADDR		
6	RST		
7	ALRT		

Arduino library

Adafruit SHT31 Library by Adafruit

or other



//Code explanation

```ino
#include <Arduino.h>                          // Include the main Arduino library
#include <Wire.h>                             // Include the Wire library for I2C communication
#include "Adafruit_SHT31.h"                   // Include the Adafruit SHT31 library for the SHT31 sensor

bool enableHeater = false;                    // A flag to control the heater state
uint8_t loopCnt = 0;                          // To keep track of loop iterations for heater toggling

Adafruit_SHT31 sht31 = Adafruit_SHT31();      // Create an instance of the SHT31 sensor class

void setup() {
  Serial.begin(9600);                         // Start serial communication at 9600 baud rate

  while (!Serial)                             // Wait for the serial console to open
    delay(10);

  Serial.println("SHT31 test");               // Print a message to the serial console

  if (!sht31.begin(0x44)) {                   // Set to 0x45 for alternate I2C address
    Serial.println("Couldn't find SHT31");    // Initialize the SHT31 sensor, check if it's connected
    while (1) delay(1);                       // Infinite loop if sensor is not found
  }

    Serial.print("Heater Enabled State: ");   // Check and display the initial state of the heater
  if (sht31.isHeaterEnabled())
    Serial.println("ENABLED");
  else
    Serial.println("DISABLED");
}

void loop() {
  float t = sht31.readTemperature();          // Read temperature and humidity from the sensor
  float h = sht31.readHumidity();

  if (!isnan(t)) {                            // Check if the temperature reading is valid and print it
    Serial.print("Temp *C = "); Serial.print(t); Serial.print("\t\t");
  } else {
    Serial.println("Failed to read temperature");
  }

  if (!isnan(h)) {                            // Check if the humidity reading is valid and print it
    Serial.print("Hum. % = "); Serial.println(h);
  } else {
    Serial.println("Failed to read humidity");
  }

  delay(1000);                                // Wait for 1 second before the next reading

  if (loopCnt >= 30) {                        // Toggle the heater state every 30 seconds
    enableHeater = !enableHeater;             // Toggle the heater state
    sht31.heater(enableHeater);               // Enable/disable the heater on the sensor
    Serial.print("Heater Enabled State: ");
    if (sht31.isHeaterEnabled())
      Serial.println("ENABLED");
    else
      Serial.println("DISABLED");

    loopCnt = 0;                              // Reset the counter
  }
  loopCnt++;                                  // Increment the loop counter
}
```

9.1 Sensor: Temperature and Humidity

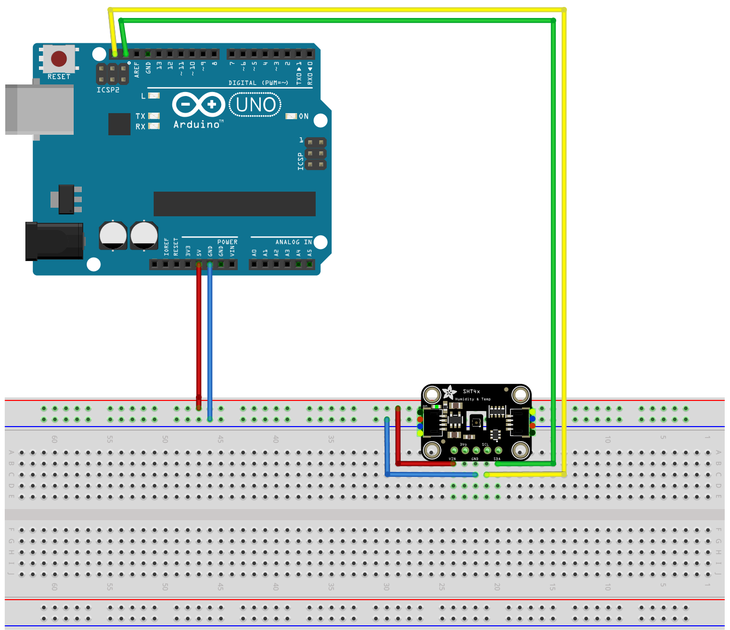
SHT 40 / 41 / 45

About the SHT 4x sensor

Sensirion Temperature/Humidity sensors are some of the finest & highest-accuracy devices you can get. The SHT40, SHT41, and SHT45 sensors are the fourth generation.

Other than these differences in relative humidity and temperature accuracy, there is no discernible difference between these three chips. The I2C addresses and code are identical for all.

Technical details:	**SHT 40**	**SHT 41**	**SHT 45**
Power:	3.3V 5V	3.3V 5V	3.3V 5V
Humidity:	0-100%, ±3%	0-100%, ±1.8%	0-100%, ±1%
Temperature:	-40 to 125°C, ±0.2°C	-40 to 125°C, ±0.2°C	-40 to 125°C, ±0.1°C
Protocol:	I2C	I2C	I2C
I2C address:	0x44	0x44	0x44



Hardware Prepare

Arduino uno
Breadboard
Jump Wires
SHT 40 / 41 / 45

Wiring method

SHT 4x pins		Arduino pins		
1	VIN	-	5v	Choose one
2	3v3	-	3v	
3	GND	-	GND	
4	SCL	-	SCL	
5	SDA	-	SDA	

Arduino library

Adafruit SHT4x Library by Adafruit

or other

DHT 40 DHT 41 DHT 45

//Code explanation

```ino
#include "Adafruit_SHT4x.h"                          // Include the Adafruit SHT4x library

Adafruit_SHT4x sht4 = Adafruit_SHT4x();              // Create an instance of the SHT4x sensor class

void setup() {
  Serial.begin(115200);                              // Start serial communication at 115200 baud rate

  while (!Serial)
    delay(10);                                       // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit SHT4x test");             // Print a message to the serial console
  if (! sht4.begin()) {                              // Initialize the SHT4x sensor, check if it's connected
    Serial.println("Couldn't find SHT4x");           // Print an error message if the sensor is not found
    while (1) delay(1);                              // Infinite loop if sensor is not found
  }
  Serial.println("Found SHT4x sensor");              // Sensor found message
  Serial.print("Serial number 0x");                  // Print the serial number of the sensor
  Serial.println(sht4.readSerial(), HEX);            // Read and print the serial number in hexadecimal format

  // You can have 3 different precisions, higher precision takes longer
  sht4.setPrecision(SHT4X_HIGH_PRECISION);           // Set to high precision mode
  switch (sht4.getPrecision()) {
    case SHT4X_HIGH_PRECISION: Serial.println("High precision");
      break;
    case SHT4X_MED_PRECISION: Serial.println("Med precision");
      break;
    case SHT4X_LOW_PRECISION: Serial.println("Low precision");
      break;
  }

  // You can have 6 different heater settings, higher heat and longer times uses more power
  sht4.setHeater(SHT4X_NO_HEATER);                   // Set heater to off
  switch (sht4.getHeater()) {                        // Display the current heater setting
    case SHT4X_NO_HEATER: Serial.println("No heater");
      break;
    case SHT4X_HIGH_HEATER_1S: Serial.println("High heat for 1 second");
      break;
    case SHT4X_HIGH_HEATER_100MS: Serial.println("High heat for 0.1 second");
      break;
    case SHT4X_MED_HEATER_1S: Serial.println("Medium heat for 1 second");
      break;
    case SHT4X_MED_HEATER_100MS: Serial.println("Medium heat for 0.1 second");
      break;
    case SHT4X_LOW_HEATER_1S: Serial.println("Low heat for 1 second");
      break;
    case SHT4X_LOW_HEATER_100MS: Serial.println("Low heat for 0.1 second");
      break;
  }
}
void loop() {
  sensors_event_t humidity, temp;                    // Create sensor event objects for humidity and temperature

  uint32_t timestamp = millis();                     // Record the current time
  sht4.getEvent(&humidity, &temp);                   // Read temperature and humidity data
  timestamp = millis() - timestamp;                  // Calculate the time taken for the sensor read operation

  // Print temperature and humidity readings to the serial console
  Serial.print("Temperature: "); Serial.print(temp.temperature); Serial.println(" degrees C");
  Serial.print("Humidity: "); Serial.print(humidity.relative_humidity); Serial.println("% rH");

  Serial.print("Read duration (ms): ");              // Print the duration of the sensor read operation
  Serial.println(timestamp);

  delay(1000);                                       // Wait for 1 second before repeating the loop
}
```

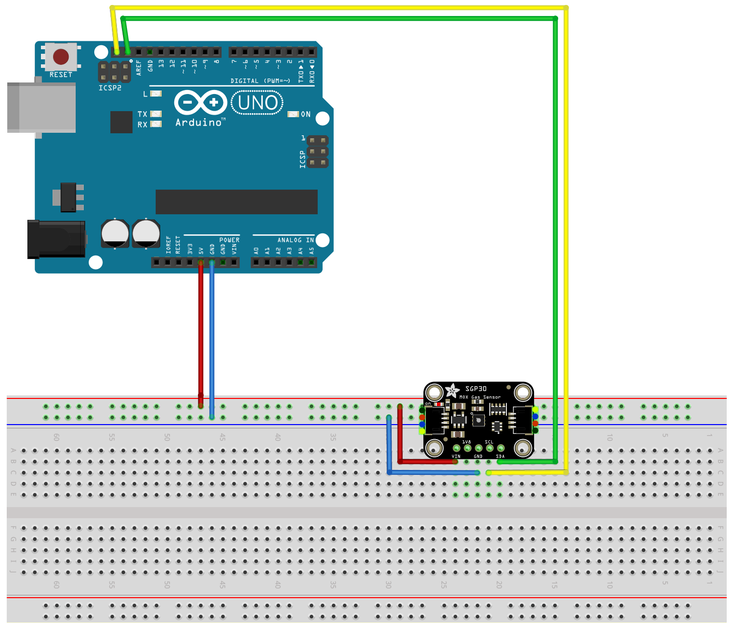
SGP 30

About the SGP 30 sensor

This is a gas sensor that can detect a wide range of Volatile Organic Compounds (VOCs) and H2 and is intended for indoor air quality monitoring. It will return a Total Volatile Organic Compound (TVOC) reading and an equivalent carbon dioxide reading (eCO2) over I2C.

The SGP combines multiple metal-oxide sensing elements on one chip to provide more detailed air quality signals.

Technical details:	**SGP 30**
Power:	3.3V 5V
eCo2:	400 - 60,000 ppm
TVOC:	0 - 60,000 ppm
Protocol:	I2C
I2C address:	0x58



Hardware Prepare

Arduino uno
Breadboard
Jump Wires
SGP 30

Wiring method

SGP 30 pins		Arduino pins		
1	VIN	-	5v	Choose one
2	3v3	-	3v	
3	GND	-	GND	
4	SCL	-	SCL	
5	SDA	-	SDA	

Arduino library

Adafruit SGP30 Sensor by Adafruit
or **SGP30** by Rob Tillaart
or other

//Code explanation

```ino
#include <Wire.h>                          // Include Wire library for I2C communication
#include "Adafruit_SGP30.h"                // Include Adafruit SGP30 library for air quality sensor

Adafruit_SGP30 sgp;                        // Create an instance of the SGP30 class

// Function to calculate absolute humidity in mg/m^3
uint32_t getAbsoluteHumidity(float temperature, float humidity) {
    // Approximation formula from Sensirion SGP30 Driver Integration chapter 3.15
    const float absoluteHumidity = 216.7f * ((humidity / 100.0f) * 6.112f * exp((17.62f * temperature) / (243.12f + temperature)) / (273.15f + temperature)); // Calculate absolute humidity in g/m^3
    const uint32_t absoluteHumidityScaled = static_cast<uint32_t>(1000.0f * absoluteHumidity); // Convert to mg/m^3
    return absoluteHumidityScaled;
}

void setup() {
  Serial.begin(115200);                            // Start serial communication at 115200 baud rate
  while (!Serial) { delay(10); }                   // Wait for serial console to open
  Serial.println("SGP30 test");                    // Print test message to serial

  if (!sgp.begin()) {                              // Initialize SGP30 sensor
    Serial.println("Sensor not found :(");         // Sensor not found message
    while (1);                                     // Infinite loop if sensor not found
  }
  Serial.print("Found SGP30 serial #");            // Print found sensor message
  Serial.print(sgp.serialnumber[0], HEX);          // Print sensor's serial number in hexadecimal format
  Serial.print(sgp.serialnumber[1], HEX);
  Serial.println(sgp.serialnumber[2], HEX);
}

int counter = 0;                                   // Counter for baseline readings

void loop() {
  if (!sgp.IAQmeasure()) {                         // Perform IAQ measurement
    Serial.println("Measurement failed");          // Print error if measurement fails
    return;
  }
  // Print Total Volatile Organic Compounds (TVOC) and equivalent CO2 (eCO2) levels
  Serial.print("TVOC "); Serial.print(sgp.TVOC); Serial.print(" ppb\t");
  Serial.print("eCO2 "); Serial.print(sgp.eCO2); Serial.println(" ppm");

  if (!sgp.IAQmeasureRaw()) {                      // Perform raw signal measurement (H2 and Ethanol)
    Serial.println("Raw Measurement failed");      // Print error if raw measurement fails
    return;
  }
  Serial.print("Raw H2 "); Serial.print(sgp.rawH2); Serial.print(" \t");           // Print raw Hydrogen
  Serial.print("Raw Ethanol "); Serial.print(sgp.rawEthanol); Serial.println("");  // Print raw Ethanol values

  delay(1000);                                     // Wait for 1 second

  counter++;                                       // Increment counter each loop
  if (counter == 30) {                             // Every 30 readings, get baseline values for eCO2 and TVOC
    counter = 0; // Reset counter

    uint16_t TVOC_base, eCO2_base;
    if (!sgp.getIAQBaseline(&eCO2_base, &TVOC_base)) {      // Get baseline values
      Serial.println("Failed to get baseline readings");    // Print error if unable to get baseline
      return;
    }
    // Print baseline values for eCO2 and TVOC in hexadecimal format
    Serial.print("****Baseline values: eCO2: 0x"); Serial.print(eCO2_base, HEX);
    Serial.print(" & TVOC: 0x"); Serial.println(TVOC_base, HEX);
  }
}
```

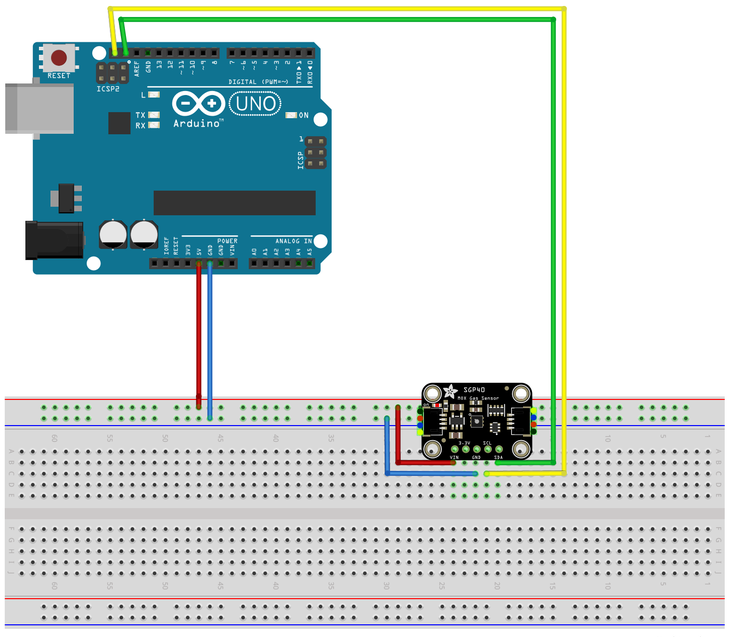
SGP 40

About the SGP 40 sensor

This is a gas sensor that can detect a wide range of Volatile Organic Compounds (VOCs) and H2 and is intended for indoor air quality monitoring. The SGP40 is the next generation after the SGP30, but does not give TVOC/eCO2 values out like the SGP30. Instead, raw signal from the sensor is processed using their software algorithm to give an overall 'air quality' value form 0 to 500.

The SGP series air quality sensor is best used with the SHT series temperature and humidity sensor for temperature and humidity calibration.

Technical details:	**SGP 40**
Power:	3.3V 5V
Voc Index:	0 - 500 Voc Index points
SRAW:	0 - 65,535 ticks
Protocol:	I2C
I2C address:	0x59



Hardware Prepare

Arduino uno
Breadboard
Jump Wires
SGP 40

Wiring method

SGP 40 pins		Arduino pins		
1	VIN	-	5v	Choose one
2	3v3	-	3v	
3	GND	-	GND	
4	SCL	-	SCL	
5	SDA	-	SDA	

Arduino library

Adafruit SGP40 Sensor by Adafruit
or **BlueDot SGP40 SHT40** by BlueDot
or **Sensirion I2C SGP40**
or other

//Code explanation

```
#include "Adafruit_SGP40.h"   // Include the library for the SGP40 VOC sensor
#include "Adafruit_SHT31.h"   // Include the library for the SHT31 temperature and humidity sensor

Adafruit_SGP40 sgp;           // Create an instance of the SGP40 class for the VOC sensor
Adafruit_SHT31 sht31;         // Create an instance of the SHT31 class for the temperature and humidity sensor

void setup() {
  Serial.begin(115200);             // Start serial communication at 115200 baud rate
  while (!Serial) { delay(10); }    // Wait for the serial console to open

  Serial.println("SGP40 test with SHT31 compensation");  // Print a startup message

  if (!sgp.begin()) {                                    // Initialize the SGP40 sensor
    Serial.println("SGP40 sensor not found :(");         // Print an error message if the sensor is not found
    while (1);                                           // Infinite loop if the sensor is not found
  }

  if (!sht31.begin(0x44)) {                       // Initialize the SHT31 sensor at I2C address 0x44 (default)
    Serial.println("Couldn't find SHT31");        // Print an error message if the sensor is not found
    while (1);                                    // Infinite loop if the sensor is not found
  }

  Serial.print("Found SHT3x + SGP40 serial #");  // Print a message indicating both sensors are found
  // Print the SGP40 sensor's serial number in hexadecimal format
  Serial.print(sgp.serialnumber[0], HEX);
  Serial.print(sgp.serialnumber[1], HEX);
  Serial.println(sgp.serialnumber[2], HEX);
}

int counter = 0;              // Counter variable for tracking loop iterations (unused in this code)

void loop() {
  uint16_t sraw;              // Variable to store raw VOC sensor reading
  int32_t voc_index;          // Variable to store calculated VOC index

  float t = sht31.readTemperature();                                  // Read temperature from SHT31
  Serial.print("Temp *C = "); Serial.print(t); Serial.print("\t\t");  // Print temperature
  float h = sht31.readHumidity();                                     // Read humidity from SHT31
  Serial.print("Hum. % = "); Serial.println(h);                       // Print humidity

  sraw = sgp.measureRaw(t, h);            // Measure raw VOC signal with temperature and humidity compensation
  Serial.print("Raw measurement: ");
  Serial.println(sraw);                   // Print raw VOC measurement

  voc_index = sgp.measureVocIndex(t, h);  // Calculate VOC index using temperature and humidity compensation
  Serial.print("Voc Index: ");
  Serial.println(voc_index);              // Print VOC index

  delay(1000);                            // Wait for 1 second before repeating the loop
}
```

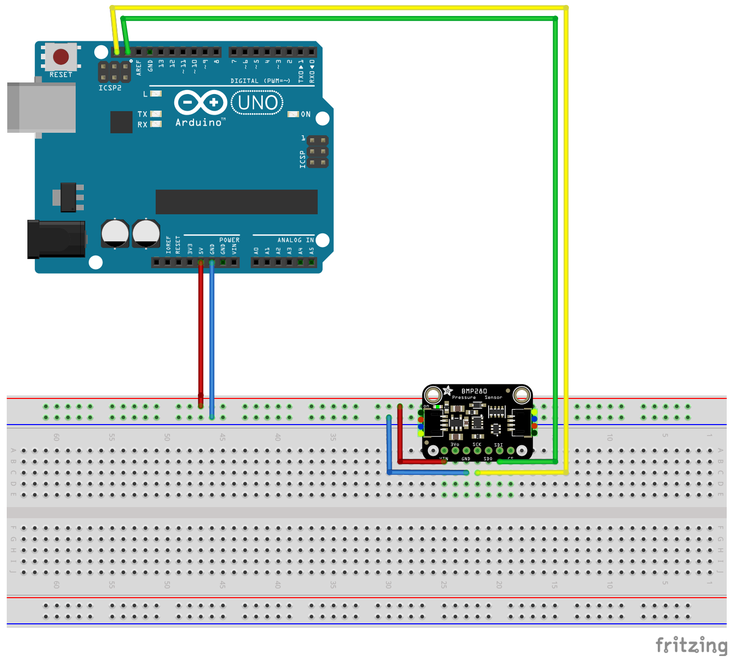
BMP 280

About the BMP 280 sensor

Bosch has stepped up their game with their new BMP280 sensor, an environmental sensor with temperature, barometric pressure that is the next generation upgrade to the BMP085/BMP180/BMP183. This sensor is great for all sorts of weather sensing.

This precision sensor from Bosch is the best low-cost, precision sensing solution for measuring barometric pressure with ±1 hPa absolute accuracy, and temperature with ±1.0°C accuracy. Because pressure changes with altitude, and the pressure measurements are so good, you can also use it as an altimeter with ±1 meter accuracy.

Technical details:	**BMP 280**
Power:	3.3V 5V
Pressure:	± 12 Pascals
Altimeter:	± 1 meters
Protocol:	I2C
I2C address:	0x76



Hardware Prepare

Arduino uno

Breadboard

Jump Wires

BMP 280

Wiring method

BMP 280 pins		Arduino pins		
1	VIN	-	5v	Choose one
2	3v3	-	3v	
3	GND	-	GND	
4	SCK	-	SCL	
5	SDO			
6	SDI	-	SDA	
7	CS			

Arduino library

Adafruit BMP280 Library by Adafruit

or **BMP280** by dvarrel

or other

//Code explanation

```
#include <Wire.h>                                   // Include Wire library for I2C communication
#include <Adafruit_BMP280.h>                        // Include Adafruit BMP280 library

// Create an instance of the Adafruit_BMP280 class to represent the BMP280 sensor
Adafruit_BMP280 bmp;                                // use I2C interface
// Obtain pointers to the temperature and pressure sensor objects within the BMP280
Adafruit_Sensor *bmp_temp = bmp.getTemperatureSensor();
Adafruit_Sensor *bmp_pressure = bmp.getPressureSensor();

void setup() {
  Serial.begin(9600);                               // Initialize serial communication at 9600 baud rate
  while ( !Serial ) delay(100);                     // wait for native usb
  Serial.println(F("BMP280 Sensor event test"));

  unsigned status;                                  // Variable to store the status of sensor initialization
  // Initialize the BMP280 sensor
  status = bmp.begin();
  if (!status) {                                    // Check if the sensor is not initialized properly
    // Print an error message if the sensor is not found
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
                     "try a different address!"));
    Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16);   // Print the sensor ID
    // Print suggestions for troubleshooting based on the sensor ID
    Serial.print("        ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
    Serial.print("   ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("        ID of 0x60 represents a BME 280.\n");
    Serial.print("        ID of 0x61 represents a BME 680.\n");
    while (1) delay(10);                            // Infinite loop if the sensor is not found
  }

  /* Configure sensor with default settings from the datasheet */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode. */
                  Adafruit_BMP280::SAMPLING_X2,     /* Temp. oversampling */
                  Adafruit_BMP280::SAMPLING_X16,    /* Pressure oversampling */
                  Adafruit_BMP280::FILTER_X16,      /* Filtering. */
                  Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */

  bmp_temp->printSensorDetails();                   // Print details of the temperature sensor
}

void loop() {
  sensors_event_t temp_event, pressure_event;       // Create event objects for temperature and pressure
  bmp_temp->getEvent(&temp_event);                  // Get the temperature event
  bmp_pressure->getEvent(&pressure_event);          // Get the pressure event

  Serial.print(F("Temperature = "));                // Print temperature data
  Serial.print(temp_event.temperature);
  Serial.println(" *C");

  Serial.print(F("Pressure = "));                   // Print pressure data
  Serial.print(pressure_event.pressure);
  Serial.println(" hPa");

  Serial.println();                                 // Print a newline for readability
  delay(2000);                                      // Wait for 2 seconds before next read
}

```

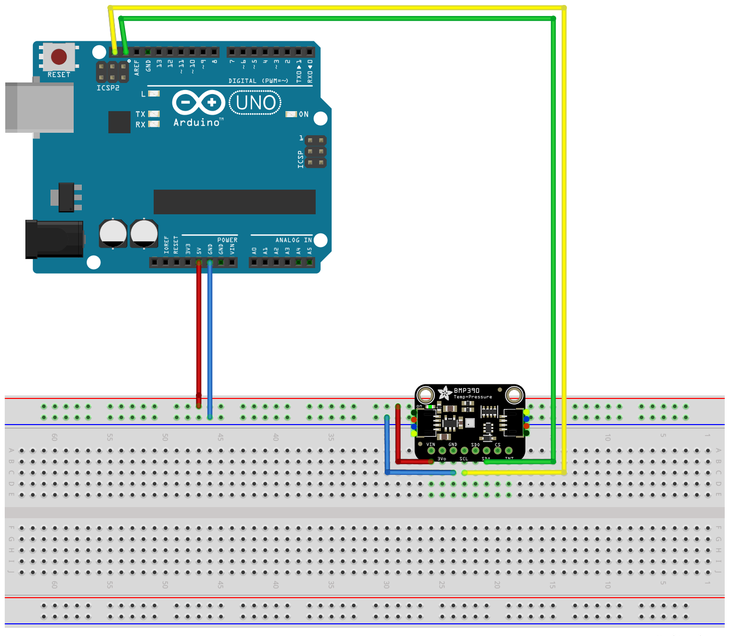
BMP 388 / 390

About the BMP 390 sensor

The BMP3xx is the next-generation of sensors from Bosch and is the upgrade to the BMP280 - with a low altitude noise as low as 0.1m and the same fast conversion time. 388 has a relative accuracy of ± 8 Pascals, which translates to about ± 0.5 meters of altitude (compare to the BMP280's 12 Pascal/ ±1 meter). 390 has a relative accuracy of ± 3 Pascals, which translates to about ± 0.25 meters of altitude.

Note that for absolute height you'll still need to enter in the barometric pressure at sea level if the weather changes.

Technical details:	**BMP 388**	**BMP 390**
Power:	3.3V 5V	3.3V 5V
Pressure:	± 8 Pascals	± 3 Pascals
Altimeter:	± 0.5 meters	± 0.25 meters
Protocol:	I2C	I2C
I2C address:	0x77	0x77



Hardware Prepare

Arduino uno

Breadboard

Jump Wires

BMP 388 / 390

Wiring method

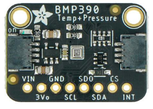
BMP 390 pins		Arduino pins		
1	VIN	-	5v	Choose one
2	3v3	-	3v	
3	GND	-	GND	
4	SCL	-	SCL	
5	SDO			
6	SDA	-	SDA	
7	CS			
8	INT			

Arduino library

Adafruit BMP3XX Library by Adafruit

or **DFRobot_BMP3XX** by DFRobot

or other



//Code explanation

```arduino
#include <Wire.h>                                       // Include the Wire library for I2C communication
#include <Adafruit_Sensor.h>                            // Include the Adafruit Unified Sensor library
#include "Adafruit_BMP3XX.h"                            // Include the library for the BMP3XX sensor

// Define the sea-level pressure in hPa as a reference for altitude calculation
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BMP3XX bmp;                                    // Create an instance of the BMP3XX class

void setup() {
  Serial.begin(115200);                                 // Start serial communication at 115200 baud rate
  while (!Serial);                                      // Wait for serial console to become ready
  Serial.println("Adafruit BMP388 / BMP390 test");      // Print a startup message

  // Initialize the BMP3XX sensor using I2C
  if (!bmp.begin_I2C()) {                               // If initialization fails, print an error message
    Serial.println("Could not find a valid BMP3 sensor, check wiring!");
    while (1);                                          // Infinite loop if the sensor is not found
  }

  // Configure sensor settings
  bmp.setTemperatureOversampling(BMP3_OVERSAMPLING_8X);  // Set temperature oversampling
  bmp.setPressureOversampling(BMP3_OVERSAMPLING_4X);     // Set pressure oversampling
  bmp.setIIRFilterCoeff(BMP3_IIR_FILTER_COEFF_3);        // Set the IIR filter coefficient
  bmp.setOutputDataRate(BMP3_ODR_50_HZ);                 // Set the output data rate
}

void loop() {
  if (!bmp.performReading()) {                          //Perform reading. If failed, print error message
    Serial.println("Failed to perform reading :(");
    return;                                             // Exit the loop on failure
  }
  // Print the temperature reading in Celsius
  Serial.print("Temperature = ");
  Serial.print(bmp.temperature);
  Serial.println(" *C");

  // Print the pressure reading in hPa
  Serial.print("Pressure = ");
  Serial.print(bmp.pressure / 100.0);                   // Convert pressure from Pa to hPa
  Serial.println(" hPa");

  // Print the approximate altitude in meters
  Serial.print("Approx. Altitude = ");

 // Calculate altitude using the specified sea-level pressure
  Serial.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");

  Serial.println();                                     // Print a new line for readability
  delay(2000);                                          // Wait for 2 seconds before next read
}
```

9.1 Sensor: Ambient light

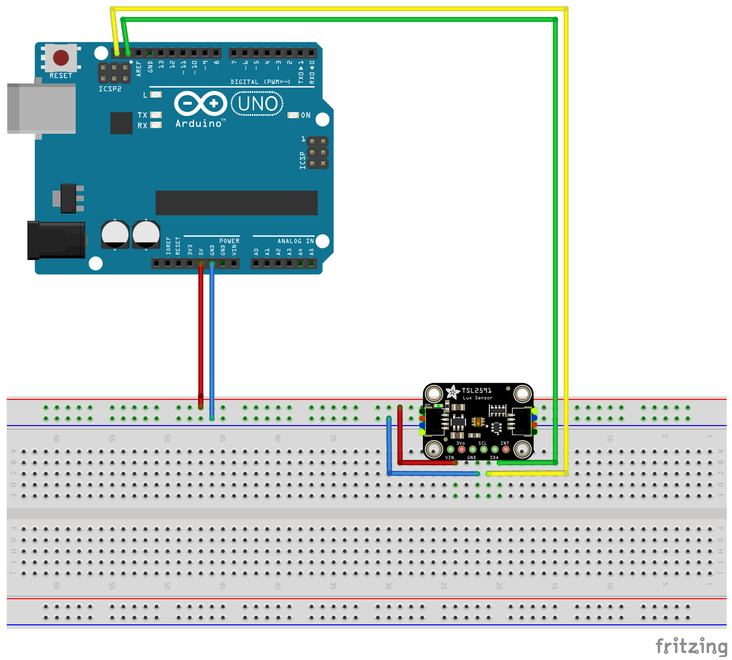
TSL 2591

About the TSL 2591 sensor

The TSL2591 luminosity sensor is an advanced digital light sensor, ideal for use in a wide range of light situations. Compared to low cost CdS cells, this sensor is more precise, allowing for exact lux calculations and can be configured for different gain/timing ranges to detect light ranges from up to 188uLux up to 88,000 Lux on the fly.

The best part of this sensor is that it contains both infrared and full spectrum diodes! That means you can separately measure infrared, full-spectrum or human-visible light.

Technical details:	**TSL 2591**
Power:	3.3V 5V
Light:	188 - 88,000 Lux
Dynamic:	600,000,000:1
Protocol:	I2C
I2C address:	0x29



Hardware Prepare

Arduino uno
Breadboard
Jump Wires
TSL 2591

Wiring method

TSL 2591 pins		Arduino pins		
1	VIN	-	5v	Choose one
2	3v3	-	3v	
3	GND	-	GND	
4	SCL	-	SCL	
5	SDA	-	SDA	
6	INT			

Arduino library

Adafruit TSL2591 Library by Adafruit

or other

//Code explanation

```ino
#include <Wire.h>                                       // Include Wire library for I2C communication
#include <Adafruit_Sensor.h>                            // Include Adafruit Unified Sensor library
#include "Adafruit_TSL2591.h"                           // Include Adafruit TSL2591 library

Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591);   // Create an instance of the TSL2591 class with a sensor ID

// Function to display basic sensor details
void displaySensorDetails(void) {
  sensor_t sensor;
  tsl.getSensor(&sensor);                                                   // Retrieve sensor details
  Serial.print(F("Sensor:       ")); Serial.println(sensor.name);           // Display sensor name
  Serial.print(F("Driver Ver:   ")); Serial.println(sensor.version);        // Display driver version
  Serial.print(F("Unique ID:    ")); Serial.println(sensor.sensor_id);      // Display sensor ID
  delay(500);
}

// Function to configure the sensor's gain and integration time
void configureSensor(void) {
  tsl.setGain(TSL2591_GAIN_MED);                                            // Set gain to medium
  tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS);                             // Set integration time to 300ms
}

void setup(void) {
  Serial.begin(9600);                                                       // Initialize serial communication
  Serial.println(F("Starting Adafruit TSL2591 Test!"));

  if (tsl.begin()) {
    Serial.println(F("Found a TSL2591 sensor"));                            // Sensor found
  } else {
    Serial.println(F("No sensor found ... check your wiring?"));            // Sensor not found
    while (1);
  }

  displaySensorDetails();                                                   // Display sensor details
  configureSensor();                                                        // Configure the sensor
}

// Function to perform a simple read and display the luminosity
void simpleRead(void) {
  uint16_t x = tsl.getLuminosity(TSL2591_VISIBLE);                          // Get visible light reading
  Serial.print(F("[ ")); Serial.print(millis()); Serial.print(F(" ms ] "));
  Serial.print(F("Luminosity: "));
  Serial.println(x, DEC);
}

// Function to read IR, Full Spectrum, and calculate Lux
void advancedRead(void) {
  uint32_t lum = tsl.getFullLuminosity();           // Get full luminosity (32 bits with IR and full spectrum)
  uint16_t ir = lum >> 16;                                                  // Upper 16 bits are IR
  uint16_t full = lum & 0xFFFF;                                             // Lower 16 bits are full spectrum
  Serial.print(F("[ ")); Serial.print(millis()); Serial.print(F(" ms ] "));
  Serial.print(F("IR: ")); Serial.print(ir);  Serial.print(F("  "));
  Serial.print(F("Full: ")); Serial.print(full); Serial.print(F("  "));
  Serial.print(F("Visible: ")); Serial.print(full - ir); Serial.print(F("  "));
  Serial.print(F("Lux: ")); Serial.println(tsl.calculateLux(full, ir), 6); // Calculate and display Lux
}

void loop(void) {
  // Choose one of the readings
  // simpleRead(); // Uncomment for simple read
  advancedRead(); // Uncomment for advanced read
  // unifiedSensorAPIRead(); // Uncomment for Unified Sensor API read

  delay(500);                                                               // Delay between readings
}
```