Arduino Lab

The proposed practical works aim to introduce the use of the Arduino board and develop applications based on this board.

1. Installation and configuration of the Arduino UNO R3 board

1.1. Description of the Arduino board

We will use an Arduino UNO R3 board. It uses an ATMEGA328P microcontroller powered by 5V. It has 14 digital input/output pins, 6 of which can be used for PWM (Pulse Width Modulation). There are 6 analog inputs. The microcontroller has a 10-bit resolution ADC. The board includes a circuit that allows easy USB management and can power the board. It has 32 KB of Flash memory, 2 KB of RAM, 1 KB of EEPROM, a 16 MHz clock frequency, and a max I/O current of 40 mA.

Besides the number of ports, the board features hardware-implemented SPI protocol, hardware-implemented I2C protocol, and PWM signals on six ports (11, 10, 9, 6, 5, and 3). It also has two interrupts (ports 2 and 3) and a hardware-implemented UART/Serial protocol (ports 0 and 1), as shown in Figure 1.

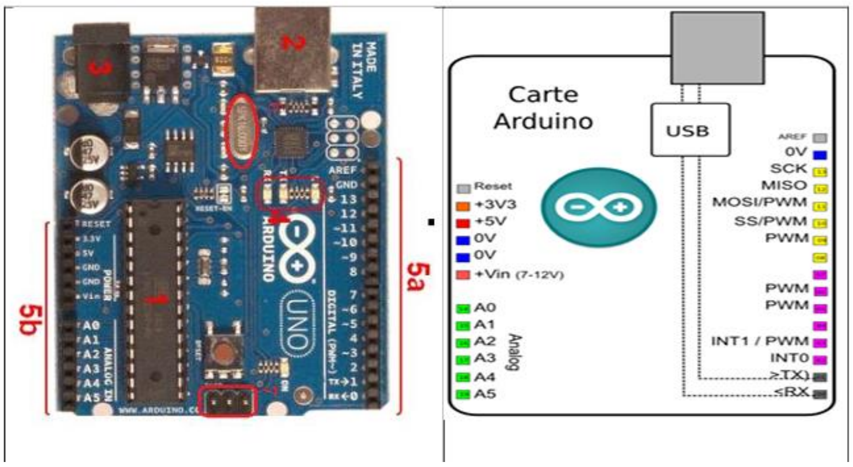


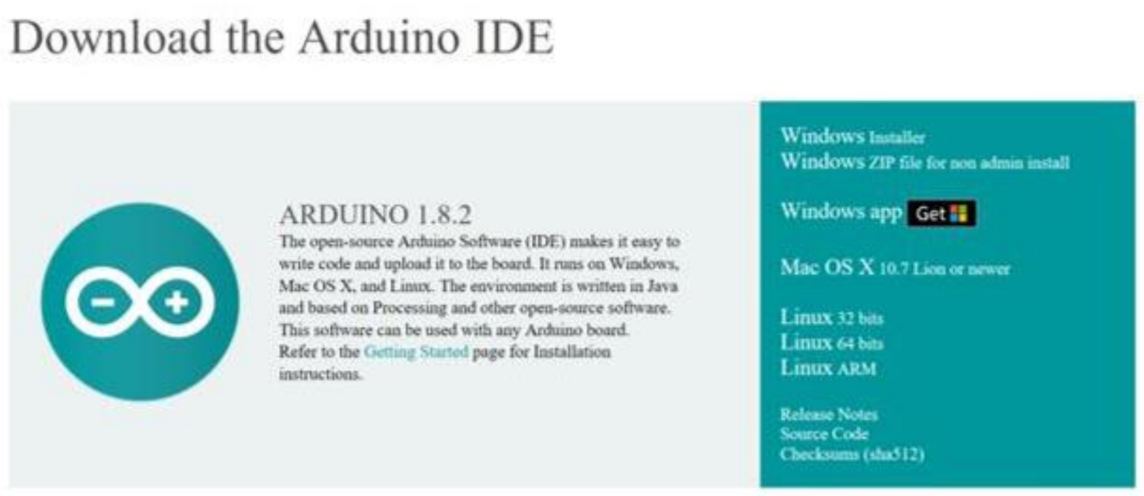
Figure 1: Arduino UNO R3 board

1.2 Discovering the Arduino development platform

The development platform consists of a hardware part (an electronic board) and a software part, both of which are open source. The role of the Arduino board is to receive a program, store it in permanent memory, and execute it to interact with the physical world (i.e., sensors and actuators).

1.2.1 Software installation

The installation of this tool depends on your computer's operating system. It is open source. For Windows systems, Figure 2 shows the "Software" tab section on the websites www.arduino.cc or www.arduino.org, allowing you to choose the IDE according to your operating system.


Download the Arduino IDE

ARDUINO 1.8.2

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

Windows Installer
Windows ZIP file for non admin install

Windows app Get

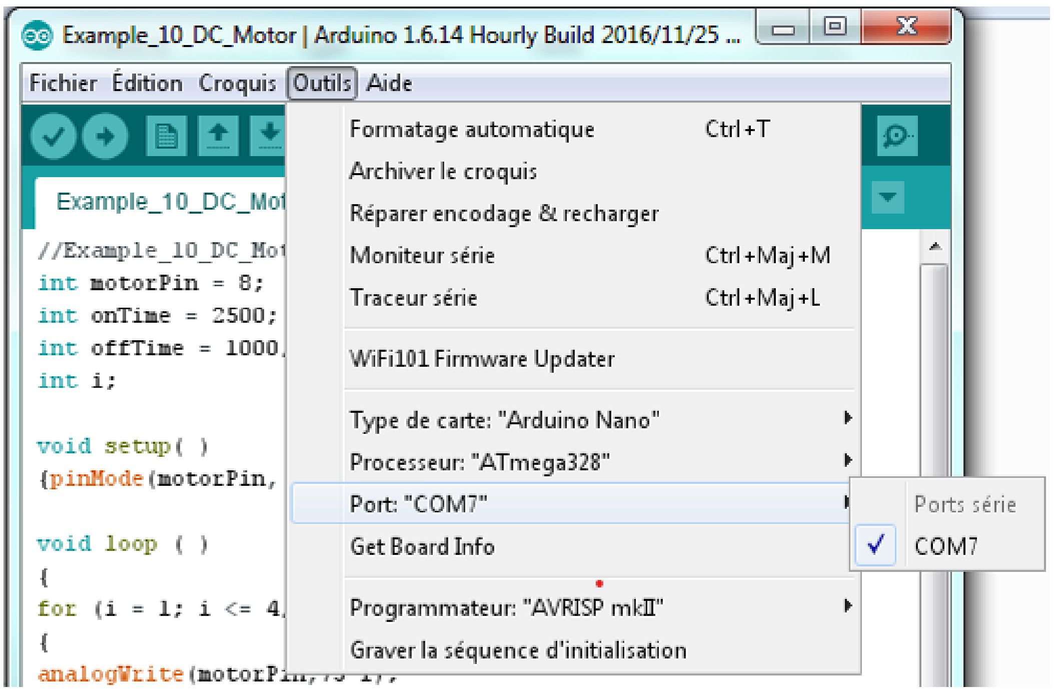
Mac OS X 10.7 Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM

Release Notes
Source Code
Checksums (sha512)

Figure 2: Arduino IDE

Simply follow the different steps, and the drivers for the Arduino boards will also be installed. Launch the newly installed IDE. In the File menu, find the Blink example program (File -> Examples -> 01. Basics -> Blink). Connect your Arduino board to your computer via USB. If your board is a UNO, you should see "Arduino/Nano ATmega328 on COM7" at the bottom right of the IDE (Nano and 7 are examples). Before uploading this program to your Nano board, check in Tools that the board type is Nano and the port is COM7 (Figure 3); otherwise, change the options. Upload the program and observe the LED blinking at a frequency of 2 Hz (one second on and one second off). If this is the case, the IDE is perfectly installed, and you can start programming your boards.


Example_10_DC_Motor | Arduino 1.6.14 Hourly Build 2016/11/25 ...
Fichier Édition Croquis Outils Aide
Example_10_DC_Mot
Formatage automatique Ctrl+T
Archiver le croquis
Réparer encodage & recharger
Moniteur série Ctrl+Maj+M
Traceur série Ctrl+Maj+L
WiFi101 Firmware Updater
Get Board Info
Graver la séquence d'initialisation
Ports série

```
//Example_10_DC_Mot
int motorPin = 8;
int onTime = 2500;
int offTime = 1000,
int i;

void setup( )
{pinMode(motorPin,

void loop ( )
{
for (i = 1; i <= 4,
{
analogWrite(motorPin,
```

Type de carte: "Arduino Nano"
Processeur: "ATmega328"
Port: "COM7"
Programmateur: "AVRISP mkII"
✓ COM7

Figure 3: Connecting the Arduino board

The main functions of the Arduino IDE software are: • To write and compile programs for the Arduino board • To connect to the Arduino board to transfer the programs • To communicate with the Arduino board This integrated development environment (IDE) is dedicated to the Arduino language and Arduino board programming.

1.2.2. Test 1:

Our first experiment is to control an LED with the Arduino. We use an Arduino, a 3mm or 5mm LED (of any color), prototype wires, a breadboard, and a 220 Ohm resistor. The resistor protects the LED from overvoltage since the LED operates on a maximum of 3V, and the 5V from the Arduino port will damage it over time.

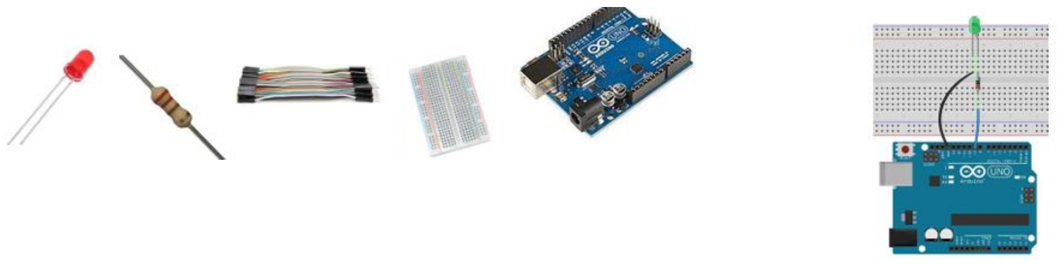


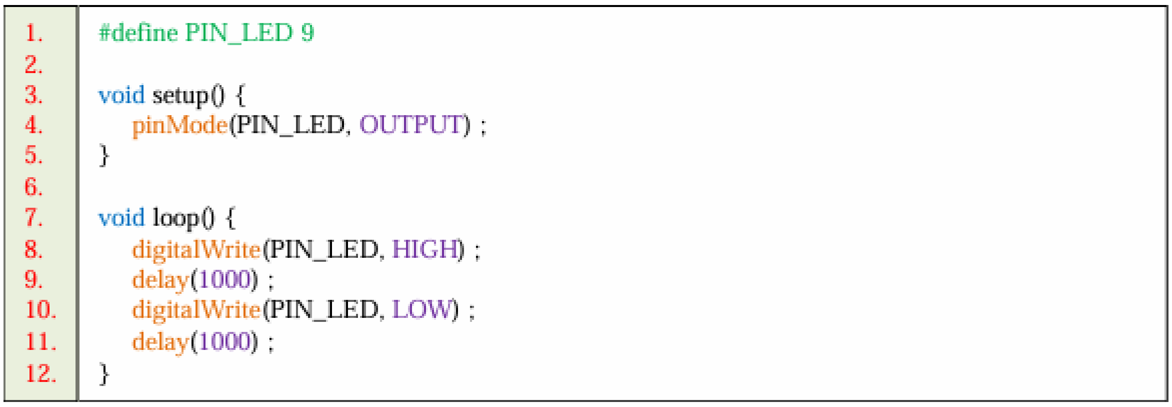
Figure 4: Materials and circuit diagram on Arduino

We start by looking at the graphical interface of the editor where we write the source code. Next, we create a simple circuit with an LED (small bulb) and control it automatically.

After making the connections, we proceed to program the Arduino board using the Arduino IDE software. Once the software is started, we create a new project and name it "controle_LED". Of course, the port where the LED is connected must be configured as an output, as we need a signal from the Arduino to the LED. Specifically, either a current flows to light the LED, or no current flows and the LED turns off.

Therefore, the first thing to do is to configure pin number 9 as an output using the pinMode function in OUTPUT mode. Then, in the loop function, we turn the LED on and off with the digitalWrite function, toggling between HIGH and LOW to start and stop the current. It is advisable to add a small delay between the two states of the LED to see the result clearly, as without a delay, the transition between the two states might not be visible. Once the code is error-free, we compile it and upload it to the Arduino board. As a result, we get a constantly blinking LED.

A very practical tip to avoid errors due to port numbers, especially when interacting with multiple ports, is to define the port numbers in constants (line 1, Figure 5) and use names instead of numbers (lines 4, 8, and 10, Figure 5).



```
#define PIN_LED 9

void setup() {
    pinMode(PIN_LED, OUTPUT) ;
}

void loop() {
    digitalWrite(PIN_LED, HIGH) ;
    delay(1000) ;
    digitalWrite(PIN_LED, LOW) ;
    delay(1000) ;
}
```

Figure 5: Source code for blinking an LED with Arduino

2. **Application: Lab Session 1: Temperature measurement**

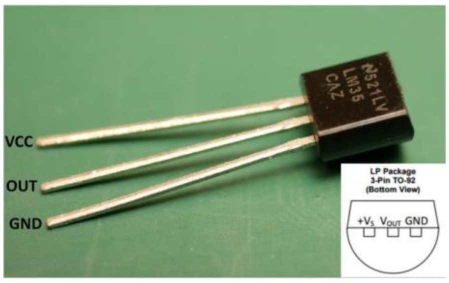
• **Sensor LM35 and DHT11,** their operating principles (reference: datasheet)

• Demonstration setup

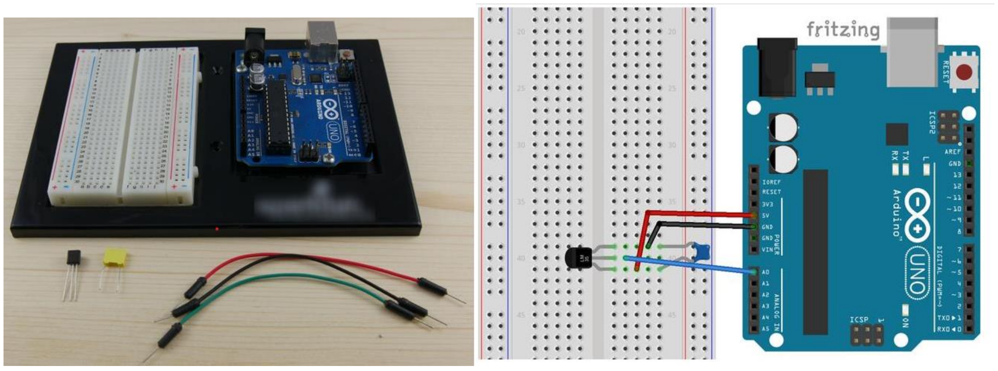
• Demonstration code

• Result

TP1: Using an LM35 sensor,



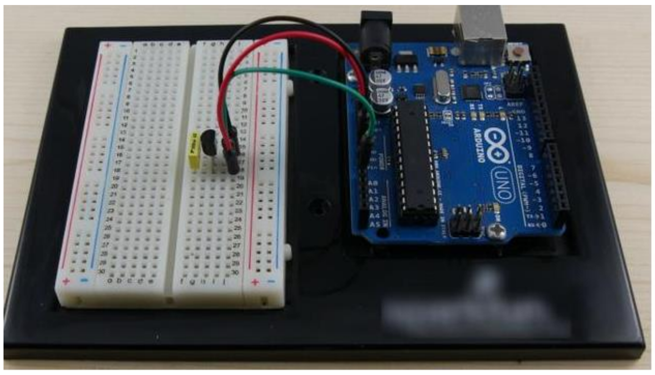
Pinout of the LM32 sensor



Wiring the LM32 module to the Arduino Uno

To start our setup, we will connect the VCC pin of the sensor to the 5V power supply of the Arduino board using a wire. Next, we will do the same with the GND pin of the sensor, connecting it to the GND pin of the Arduino board.

To do things properly (because yes, we like to do things properly), we will connect a 100nF capacitor (a decoupling capacitor in technical terms) between the VCC and GND pins of the sensor. The capacitor should be placed as close to the sensor as possible to be effective.



Assembly

We then complete the circuit by connecting the sensor output to the A0 pin of the Arduino board with a wire.

Demonstration code Now that we have our setup, let's move on to the code! The purpose of our code will be to:

1. Read the voltage on the A0 pin.
2. Convert the measured value into a temperature (for display).
3. Send the value to the PC (for display).
4. Repeat from step 1.

In the **loop()** function, we will do three things:

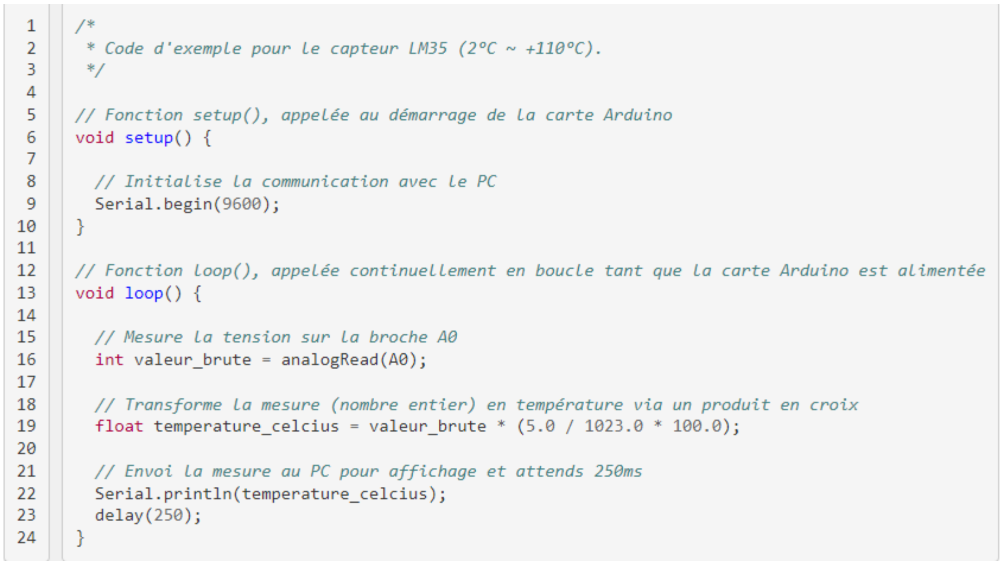
1. Measure the voltage on the A0 pin using **analogRead()**.
2. Convert the measurement result into a float value by doing a simple cross-multiplication. Reminder: 5V = 5000mV = 1023 from **analogRead()**, 10mV = 1°C, therefore, temperature = measured_value * (5.0 / 1023.0 * 100.0).
3. Send the value to the PC and wait a few milliseconds to give time to read what is happening on the PC side.

N.B. We use **value * (5.0 / 1023.0 * 100.0)** in the cross-multiplication calculation because during program compilation, the type of the values in an operation determines the type of the result. If we use **value * (5 / 1023 * 100)**, 5, 1023, and 100 are integers, and the result is an integer, which is not our goal; we want a calculation with float numbers. So, we use 5.0, 1023.0, and 100.0 to force a calculation with float numbers.

N.B. We multiply by 100 in the calculation because in 5 volts (= 5000mV) there are 100 times 10mV (= 1°C).

Conversion to degrees Fahrenheit for those interested: The formula is: fahrenheit = celsius * 1.8 + 32.

The complete code with comments:



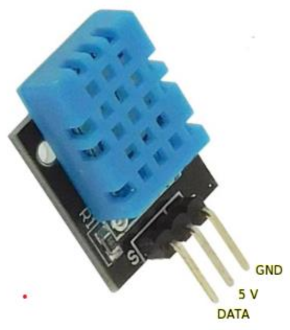
```
/*
 * Code d'exemple pour le capteur LM35 (2°C ~ +110°C).
 */

// Fonction setup(), appelée au démarrage de la carte Arduino
void setup() {

  // Initialise la communication avec le PC
  Serial.begin(9600);
}

// Fonction loop(), appelée continuellement en boucle tant que la carte Arduino est alimentée
void loop() {

  // Mesure la tension sur la broche A0
  int valeur_brute = analogRead(A0);

  // Transforme la mesure (nombre entier) en température via un produit en croix
  float temperature_celcius = valeur_brute * (5.0 / 1023.0 * 100.0);

  // Envoi la mesure au PC pour affichage et attends 250ms
  Serial.println(temperature_celcius);
  delay(250);
}
```

Result: After uploading the program to the Arduino board, open the serial monitor (under the tools tab) and select the correct communication speed (usually 9600 baud). You should see the temperature from the sensor displayed in real-time.

If your setup is correct, you should see the values in the serial monitor change when you pinch or blow on the sensor.

Application 2: Lab Session 2: Measure temperature and humidity with the DHT11 sensor

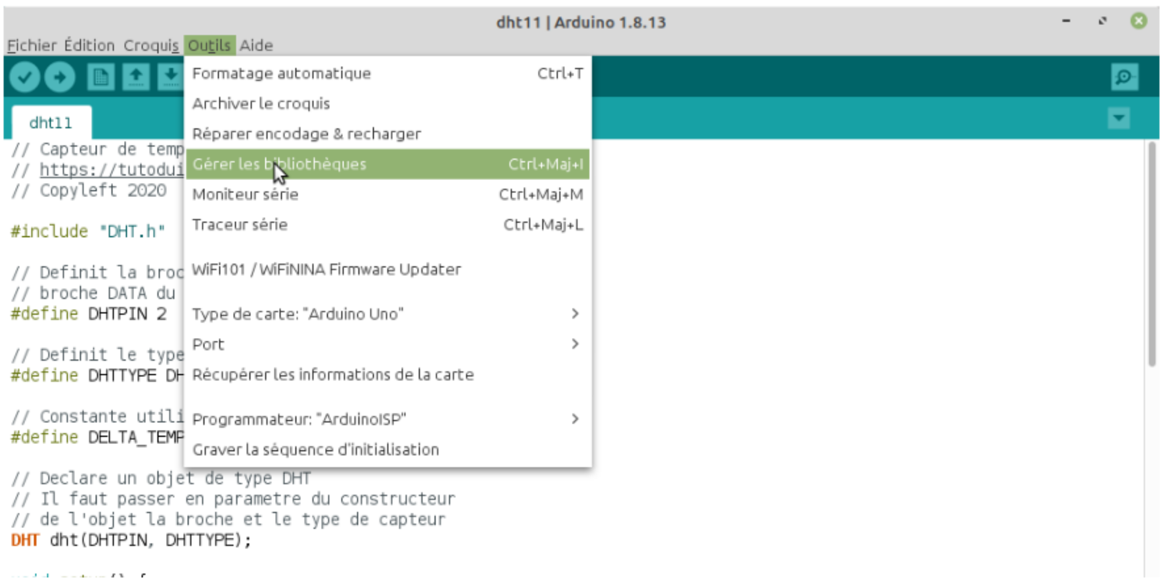
The DHT11 sensor has 4 pins, but it is often sold on a support board that has 3 pins. It communicates with the Arduino very simply through one of its digital inputs. The other 2 pins are for its 5V power supply and ground (GND).



Pinout of the sensor DHT11 Module

The setup

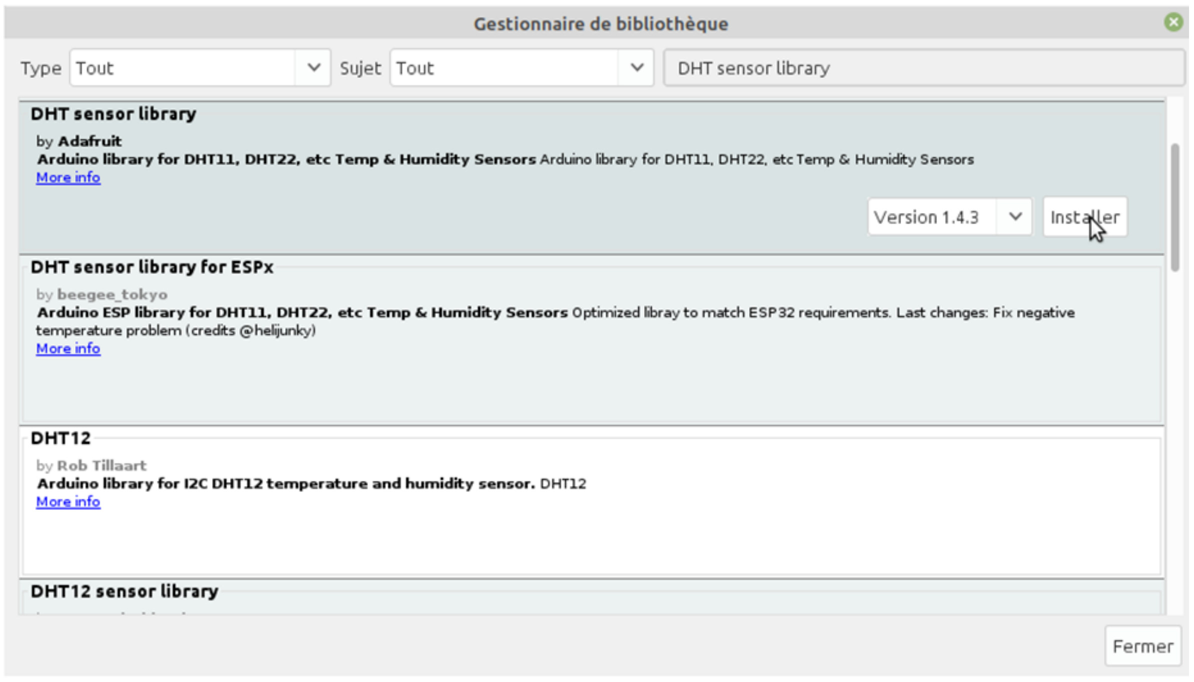
The setup is very simple; just connect the 5V of the Arduino Uno to the 5V pin, the ground (GND) of the Arduino Uno to the GND pin, and the DATA pin of the sensor to digital pin 2 of the Arduino Uno, for example.


dht11 | Arduino 1.8.13
Fichier Édition Croquis Outils Aide
dht11
Formatage automatique Ctrl+T
Archiver le croquis
Réparer encodage & recharger
Gérer les bibliothèques Ctrl+Maj+I
Moniteur série Ctrl+Maj+M
Traceur série Ctrl+Maj+L
WiFi101 / WiFiNINA Firmware Updater
Type de carte: "Arduino Uno"
Port
Récupérer les informations de la carte
Programmateur: "ArduinoISP"
Graver la séquence d'initialisation

```
// Capteur de temp
// https://tutodui
// Copyleft 2020

#include "DHT.h"

// Definit la broc
// broche DATA du
#define DHTPIN 2

// Definit le type
#define DHTTYPE DH

// Constante utili
#define DELTA_TEMP

// Declare un objet de type DHT
// Il faut passer en parametre du constructeur
// de l'objet la broche et le type de capteur
DHT dht(DHTPIN, DHTTYPE);
```

Opening the library management menu

Then, simply search for and add the "DHT sensor library" from Adafruit.



DHT sensor library
by **Adafruit**
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors
More info

DHT sensor library for ESPx
by **beegee_tokyo**
Arduino ESP library for DHT11, DHT22, etc Temp & Humidity Sensors Optimized libray to match ESP32 requirements. Last changes: Fix negative temperature problem (credits @helijunky)
More info

DHT12
by **Rob Tillaart**
Arduino library for I2C DHT12 temperature and humidity sensor. DHT12
More info

DHT12 sensor library

Adding the "DHT sensor library" The sketch (program) for the Arduino is very simple thanks to this "DHT sensor library".

```
// Capteur de temperature et d'humidite DHT11
// https://tutoduino.fr/
// Copyleft 2020
#include "DHT.h"
// Definit la broche de l'Arduino sur laquelle la
// broche DATA du capteur est reliee
#define DHTPIN 2
// Definit le type de capteur utilise
#define DHTTYPE DHT11
// Declare un objet de type DHT
// Il faut passer en parametre du constructeur
// de l'objet la broche et le type de capteur
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);

  // Initialise la capteur DHT11
  dht.begin();
}
void loop() {
  // Recupere la temperature et l'humidite du capteur et l'affiche
  // sur le moniteur serie
  Serial.println("Temperature = " + String(dht.readTemperature())+" °C");
  Serial.println("Humidite = " + String(dht.readHumidity())+" %");
  // Attend 10 secondes avant de reboucler
  delay(10000);
}
```

The temperature and humidity are displayed on the serial monitor of the Arduino.

3. Using an IR Receiver and a Remote Control with Arduino

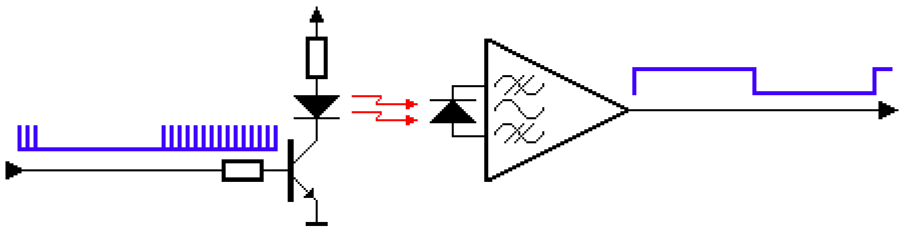
Infrared (IR) remote controls are commonly used to control various devices such as TVs, air conditioners, DVD players, etc. We will use an IR receiver with Arduino to detect the infrared signals from the remote control and perform actions based on the buttons pressed.

Required materials :

- Arduino Uno
- Infrared (IR) receiver
- Infrared remote control (compatible with the IR receiver)
- Connection cables

An infrared (IR) remote control uses infrared signals to communicate with a receiver. When you press a button on your remote control, it emits a coded infrared signal that is picked up by the IR receiver. The IR receiver then decodes the signal and sends the corresponding information to the connected device, such as a TV or DVD player, which executes the corresponding action.

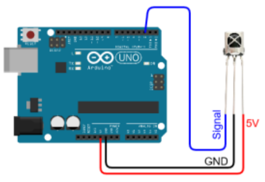
IR Receiver and Remote Control The IR receiver consists of an infrared sensor that detects the infrared signals emitted by the remote control. It then converts these signals into electrical signals that the Arduino can understand and process. The IR receiver sends the infrared signal data to the Arduino, which analyzes them and performs actions based on the buttons pressed.



Signal sent and detected by the IR emitter (left) and the receiver (right)

Connecting the IR Receiver to the Arduino The first step is to connect the IR receiver to your Arduino. Here is the connection diagram: For the Arduino UNO, we choose digital port 2 (as it can be used to manage interruptions).

```
Récepteur IR  ->  Arduino
  GND        ->   GND
  VCC        ->   5V
  OUT        ->   Broche numérique 2
```

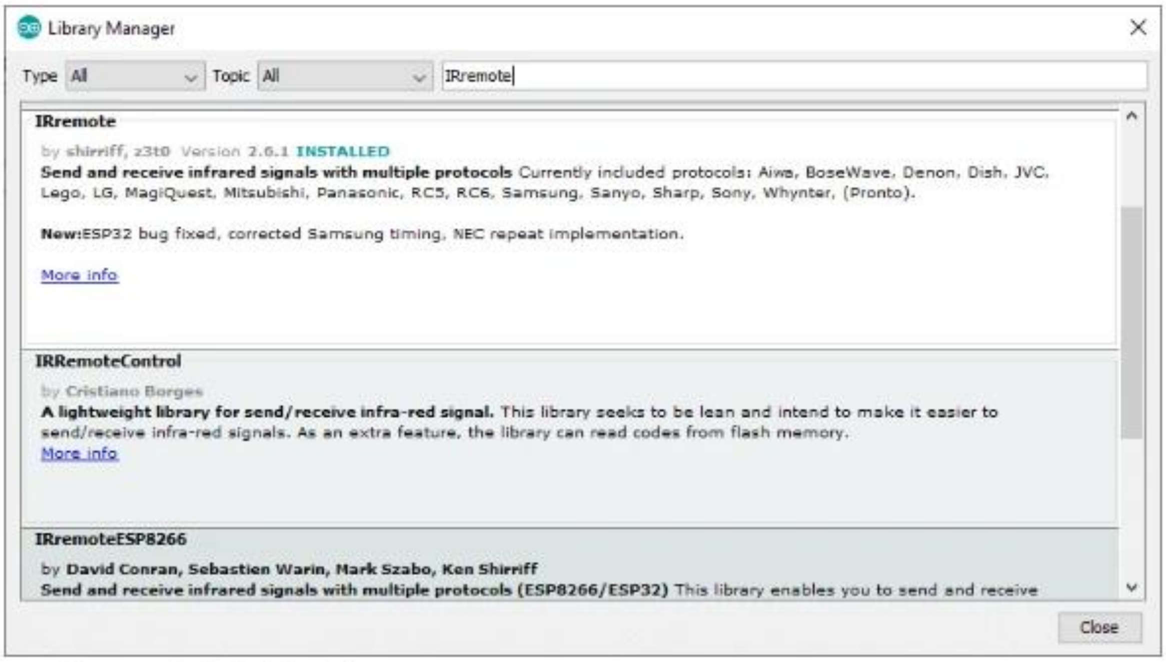


Wiring Diagram

Make sure to connect the pins correctly. Once the connections are made, you can move on to the next step.

Installing the IR Remote Library To communicate with the IR receiver, we need to install the IRremote library. Here's how to do it:

1. Open the Arduino IDE on your computer.
2. Go to "Sketch" > "Include Library" > "Manage Libraries".
3. Search for "IRremote" in the search bar.
4. Select the "IRremote" library by Shirriff.
5. Click "Install" to install the library.



IRremote

by shirriff, z3t0 Version 2.6.1 INSTALLED
Send and receive infrared signals with multiple protocols Currently included protocols: Aiwa, BoseWave, Denon, Dish, JVC, Lego, LG, MagiQuest, Mitsubishi, Panasonic, RC5, RC6, Samsung, Sanyo, Sharp, Sony, Whynter, (Pronto).

New:ESP32 bug fixed, corrected Samsung timing, NEC repeat implementation.

More info

IRRemoteControl

by Cristiano Borges
A lightweight library for send/receive infra-red signal. This library seeks to be lean and intend to make it easier to send/receive infra-red signals. As an extra feature, the library can read codes from flash memory.
More info

IRremoteESP8266

by David Conran, Sebastien Warin, Mark Szabo, Ken Shirriff
Send and receive infrared signals with multiple protocols (ESP8266/ESP32) This library enables you to send and receive

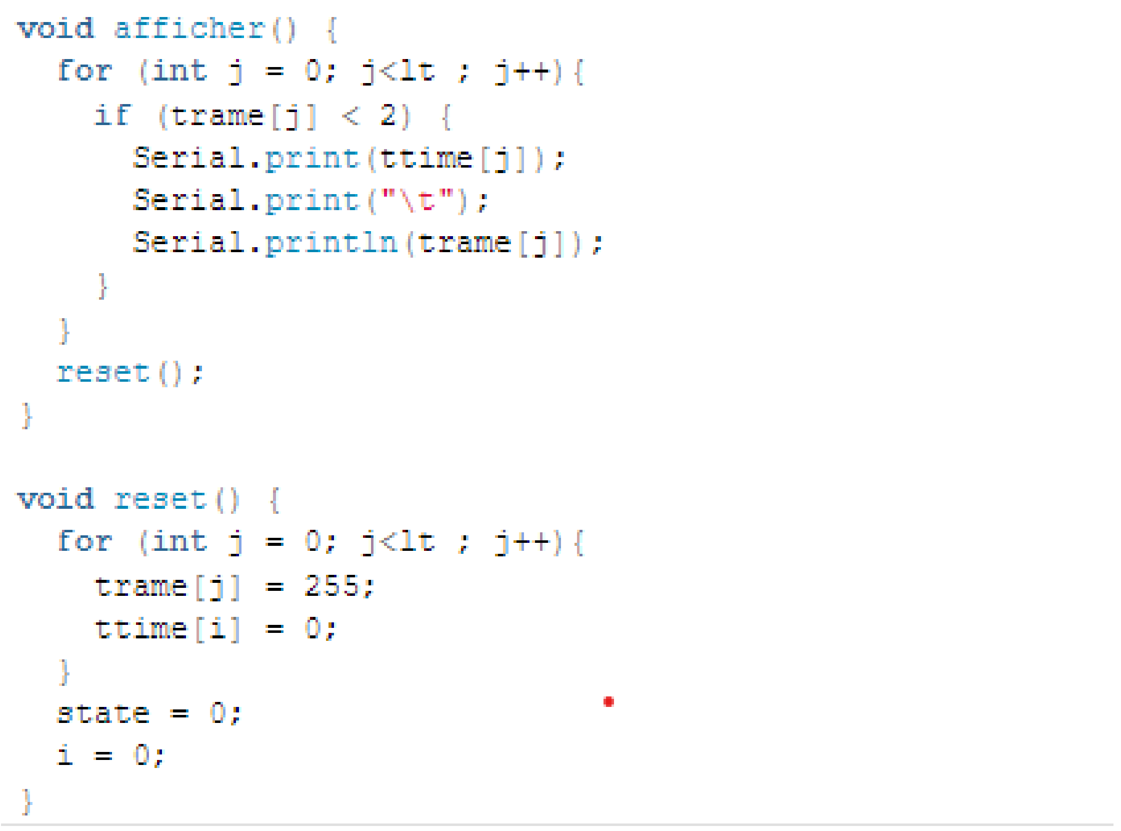
Installation of the IRremote Library

Arduino Code Configuration

Now, we will write an Arduino code to detect infrared signals from the remote control. Here is an example of the code.

```
const byte interruptPin = 2;
const byte lt = 200;
volatile byte state = 0;
volatile byte trame[lt];
volatile unsigned long ttime[lt];
volatile byte i = 0;
volatile unsigned long startt;
const unsigned long tlimite = 150000;
volatile unsigned long t;

void setup() {
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), front, CHANGE);
  Serial.begin(115200);
  reset();
}
```

```
void front() {
  t = micros();
  if (i == 0) {
    startt = t;
  }
  if (t-startt > tlimite) {
    afficher();
  }
  else {
    trame[i] = state;
    ttime[i] = t-startt;
    i++;
    state = 1-state;
    trame[i] = state;
    ttime[i] = ttime[i-1];
    i++;

    if (i >= lt) {
      afficher();
    }
  }
}
```



```
void afficher() {
  for (int j = 0; j<lt ; j++){
    if (trame[j] < 2) {
      Serial.print(ttime[j]);
      Serial.print("\t");
      Serial.println(trame[j]);
    }
  }
  reset();
}

void reset() {
  for (int j = 0; j<lt ; j++){
    trame[j] = 255;
    ttime[i] = 0;
  }
  state = 0;
  i = 0;
}
```

Acquisition procedure

After uploading: the program to the microcontroller:

• Open the serial monitor of the Arduino IDE,

• Press a button on the remote control (facing the receiver),

• Copy the data from the serial terminal (Ctrl+A then Ctrl+C), WARNING: make sure to copy ALL the data of the frame

• Open a spreadsheet software (MS Excel, LibreOffice Calc, ...),

• Paste the data into a spreadsheet (Ctrl+V),

• Insert an XY graph

• Remove unnecessary values (especially at the end of the acquisition),

• Print the curve or copy it into the digital document for annotation. Decoding ¬ Consult the documentation on the NEC

Protocol. Required work

1. Using the information provided in the receiver's datasheet, give the carrier frequency of the IR signal.
2. Measure the duration of the start sequence on the curve acquired with the Arduino. Compare it with the one specified in the datasheet.

3. Using the documentation on the NEC protocol, decode the frame (annotate the logical values directly on the printed or copied curve).
4. Deduce the address of the remote control (in hexadecimal format).