CREPP

Ensemble des Ateliers 2021-2022

Club de Robotique et d'Electronique
Programmable de Ploemeur

7 juin 2022

Table des matières

Section 1

Préambule

1.1 Informations

- Document réalisé en LaTeX par Nicolas Le Guerroué pour le Club de Robotique et d'Electronique Programmable de Ploemeur (CREPP)
- Permission vous est donnée de copier, distribuer et/ou modifier ce document sous quelque forme et de quelque manière que ce soit.
- Version du 7 juin 2022
- Taille de police : 11pt

nicolasleguerroue@gmail.com

- Ce document centralise les supports utilisés et produits par le Club de Robotique et d'Électronique Programmable de Ploemeur dans le cadre des ateliers **Programmation de microcontrôleurs** .
  Un glossaire présente les acronymes et abbreviations utilisés.
- **Dans la mesure du possible, évitez d'imprimer ce document si ce n'est pas nécessaire. Il est optimisé pour une visualisation sur un ordinateur et contient beaucoup d'images.**

Versions

octobre 2021	Fusion des supports d'ateliers
novembre 2021	Ajout de l'atelier sur les servomoteurs
décembre 2021	Ajout de l'atelier sur les moteurs pas-à-pas
janvier 2022	Ajout de l'annexe pour l'installation des bibliothèques ESP8266
février 2022	Ajout de l'atelier pour le serveur Web ESP8266 NodeMCU
mars 2022	Ajout de l'atelier pour la partie Réseaux (Adressage) et de l'atelier Capteurs
mai 2022	Ajout de l'atelier sur les modules de communication

1.2 Conventions

Commandes

Les commandes à saisir sont dans des encadrés similaires :

```
sudo apt-get update
```

Exemple de commande

Parfois, ces encadrés contiendront des instructions qu'il faudra placer dans certains fichiers.

Références

- Les fichiers sont indiqués par le repère FILE fichier
- Les dossiers sont indiqués par le repère DIR dossier
- Les logiciels sont indiqués par le repère LIB logiciel[1]
- Les adresses IP sont indiquées par le repère IP Adresse IP
- Les adresses MAC sont indiquées par le repère MAC Adresse MAC
- Les liens sont indiqués par le repère LINK Lien
- Les broches génériques des composants sont indiquées par le repère PIN Broche et se scindent en deux parties :
  - Les broches d'entrée par IN Broche
  - Les broches de sortie par OUT Broche
-

1. Sont également concernés les paquets Linux et les bibliothèques des langages

Glossaire

ARP *Address Resolution Protocol*
Table qui gère la correspondance entre les adresses IP et MAC.

DHCP *Dynamic Host Configuration Protocol*
Serveur qui gère l'attibution des adresses IP sur le serveur.

HTML *HyperText Markup Language*
Langage basé sur des balises.

HTTP *Hypertext Transfer Protocol*
Protocole de transfert hypertexte.

I2C *Inter-Integrated Circuit (Bus)*
Bus de communication.

IDE *Integrated Developpment Environnment*
Environnment de Développement Intégré.

IP *Internet Protocol*
Adresse d'une machine sur un réseau.

LCD *Liquid Crystal Display*
Ecran à Affichage à Cristaux Liquide.

MOSFET *Metal Oxide Semiconductor Field Effect Transistorl*
Transistor à effet de champ à structure métal-oxyde-semiconducteur.

OLED *Organic LED*
Écran LED organique.

SCL *Serial Clock*
Ligne d'horloge pour le protocole I2C.

SDA *Serial Data*
Ligne de données pour le protocole I2C.

SPI *Serial Peripheral Interface*
Interface Série pour Périphérique.

UART *Universal Asynchronous Receiver / Transmitter*
Transmetteur / Receveur Asynchorne Universel.

Première partie

Les servomoteur

Théorie sur les servo-moteur et applications pratiques avec Arduino

Section 2

Présentation

Les servo-moteurs sont utilisés lorsqu'on souhaite un asservissement en position d'un axe de rotation [1]

2.0.1 Asservissement

Un asservissement est un processus de correction pour maintenir une consigne.
Par exemple, un régulateur de vitesse dans une voiture est un système asservi car la vitesse doit être constante quelle que soit la pente.

2.0.2 Architecture

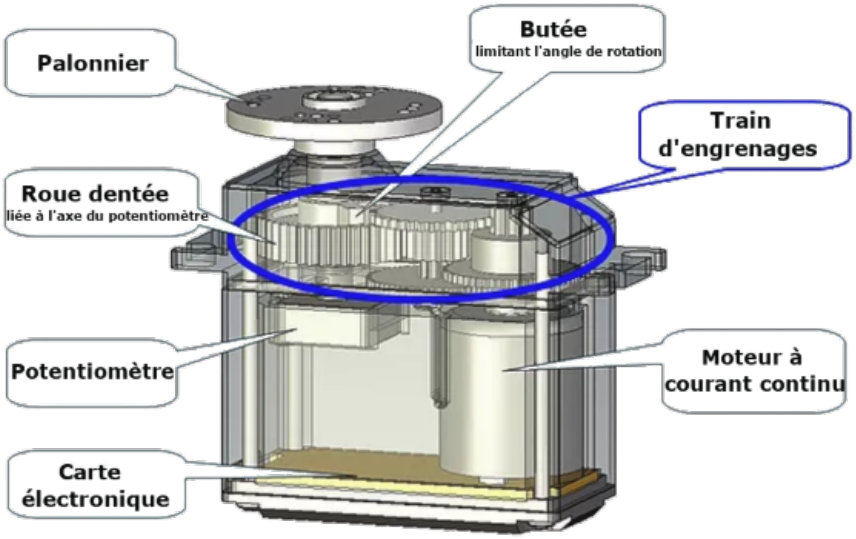


FIGURE 2.1 – Constitution d'un servo-moteur

1. Pulse Width Modulation : Modulation par Largeur d'Impulsion

2.0.3 Domaines d'application

- Modélisme
- Robotique

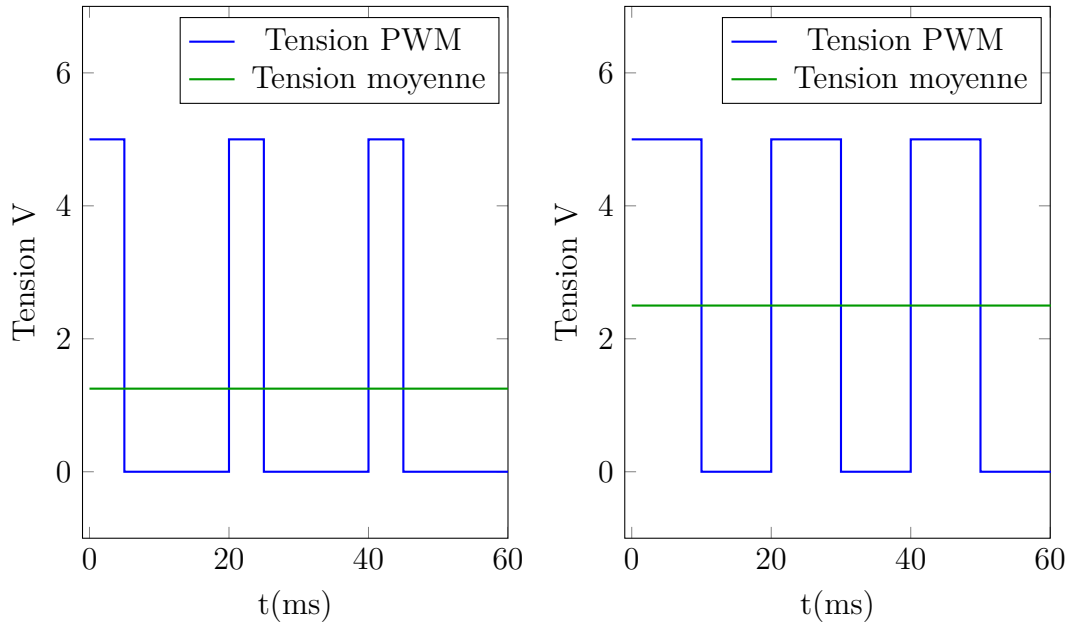
2.0.4 Commande des servo-moteurs

Les servo-moteurs ont besoins d'être contrôlés via un signal PWM.

2.1 Principe de la PWM

La PWM est la création d'un signal numérique dont le temps à l'état haut est variable. On fait varier le rapport cyclique (appelé r) qui est compris entre 0 et 1.

$$r = \frac{T_{on}}{T_{signal}}$$



(a) r=0.25 (b) r=0.5

FIGURE 2.2 – Différents rapports cycliques

2.1.1 Applications de la PWM

En faisant varier la tension de sortie dans le temps rapidement (¿50Hz), on peut simuler une tension analogique. Quelques applications :

- Contrôle de la luminosité d'une LED
- Contrôle de servo-moteurs

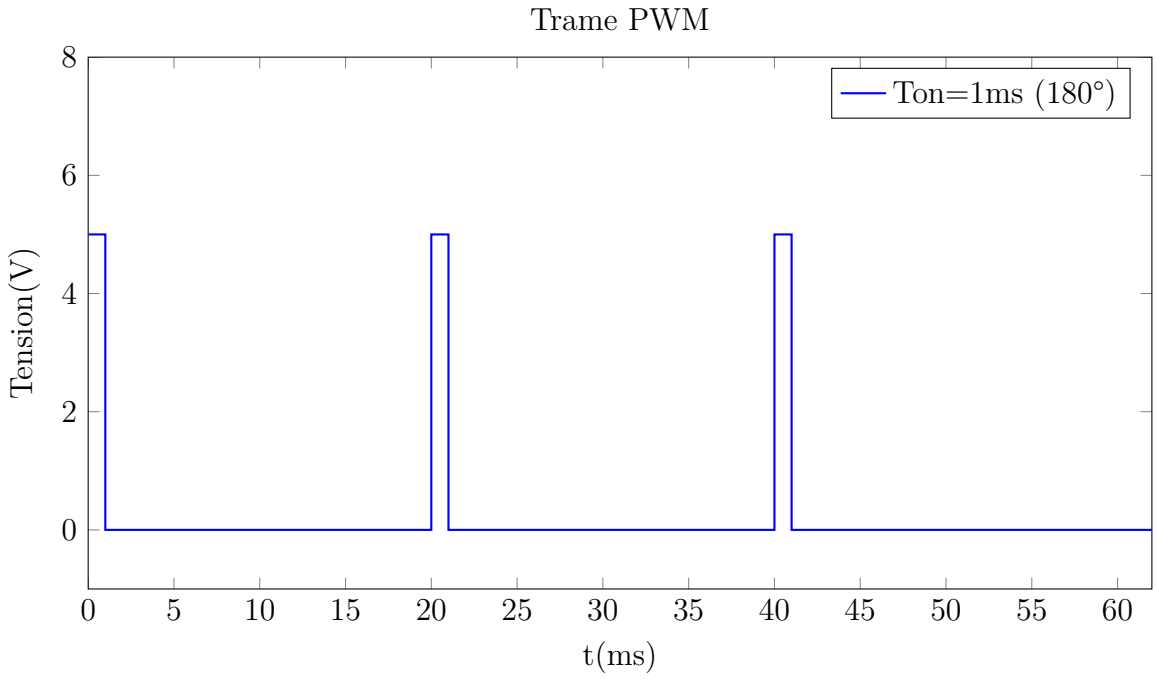
2.1.2 Code Arduino

Voici un code d'exemple pour faire varier la luminosité d'une LED.

```
const int pin_led = 11; //Selection d'une broche PWM

float duty_cyle[11] = {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};//
 Création d'un tableau avec les différents
rapports cycliques

void setup() {

    pinMode(pin_led, OUTPUT);  //Mise en sortie de la broche LED

}//Fin setup

void loop() {

    for(int i=0;i<11;i++)
    {
        int value_r = duty_cyle[i]*255.0; //Conversion d'une valeur entre 0 et
 1 en une valeur entre 0 et 255
        analogWrite(pin_led, value_r); //Change le rapport cyclique pendant 3 s
        delay(600);        //Attend 0.6s
    }


}//Fin loop
```

Variation de la luminosité d'une LED

2.1.3 Trame de commande servomoteurs



2.1.4 Branchement d'un servo-moteur

- Câble noir ou marron : GND
- Câble rouge : +5V
- Câble blanc ou jaune : Signal Arduino (11)

2.1.5 Code Arduino

```
#include <Servo.h>      //Inclusion de la bibliothèque Servo
Servo myservo;  // Création d'un objet Servo
int pos = 0;    //Angle du servomoteur

void setup() {

  myservo.attach(11);  //Choix de la broche du servo moteur

}

void loop() {

  for (pos = 0; pos <= 180; pos += 1) { //Parcours la plage angulaire [0-180]
 degré par degré
```

```
    myservo.write(pos);              //Actualise la position
    delay(15);                       //Attend 15 ms avant l'actualisation

   }//Fin for

   for (pos = 180; pos >= 0; pos -= 1) {     //Parcours la plage angulaire
  [0-180] degré par degré

    myservo.write(pos);              //Actualise la position
    delay(15);                       //Attend 15 ms avant l'actualisation

   }//Fin for
}//Fin loop
```

Variation de la position d'un servo-moteurs

2.2 Caractéristiques

2.2.1 Electriques

- Tension de commande et d'alimentation : 5̃V

2.2.2 Mécaniques

- Couple de sortie (Nm)
- Vitesse de rotation (degré/temps)

Deuxième partie

Les moteurs pas-à-pas

Théorie sur les moteurs pas-à-pas et applications pratiques avec Arduino et ESP8266

Section 3

Présentation

Les moteur pas-à-pas sont utilisés lorsqu'on souhaite un asservissement en position d'un axe de rotation avec une précision inégalée par les servomoteurs.

3.0.1 Constitution

Les moteur pas-à-pas sont constitués de :

- Plusieurs bobines (un pôle forme une paire de bobines)
- Un aimant qui sert d'axe de rotation

3.0.2 Les types de moteur pas-à-pas

- moteur pas-à-pas à phase bipolaire

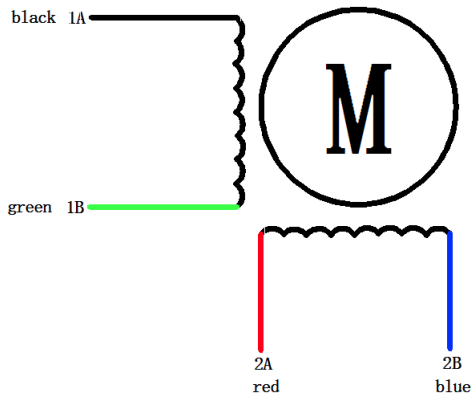


FIGURE 3.1 – moteur pas-à-pas bipolaire

- moteur pas-à-pas à phases unipolaires

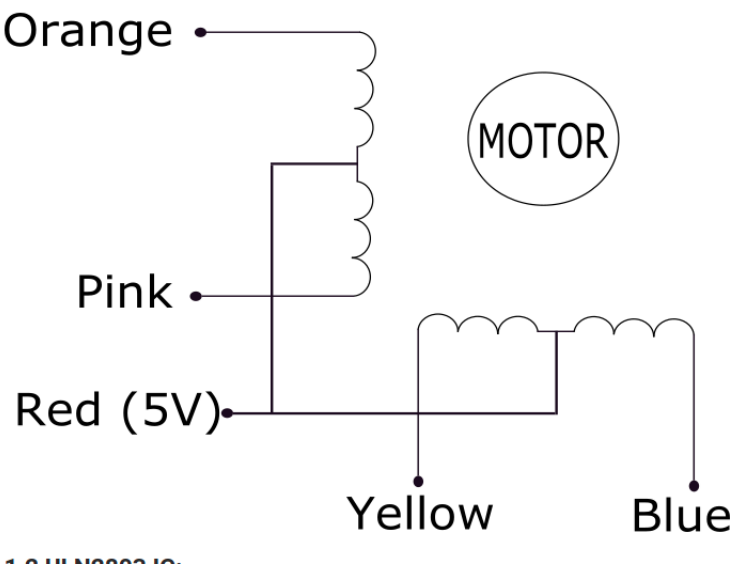


FIGURE 3.2 – moteur pas-à-pas unipolaires

- moteur pas-à-pas à reluctance variable (non abordés ici)

3.0.3 Principe

En faisant varier le champ électromagnétique des différentes phases, on peut faire varier la position angulaire de l'aimant.

Exemples avec phases bipolaires

En alimentant une paire de phases avec une tension positif, l'aimant se place dans l'alignement du champ électromagnétique formé par la paire de phase.
En alimentant la paire de phase avec une tension négative, l'aimant se place dans le sens contraire.

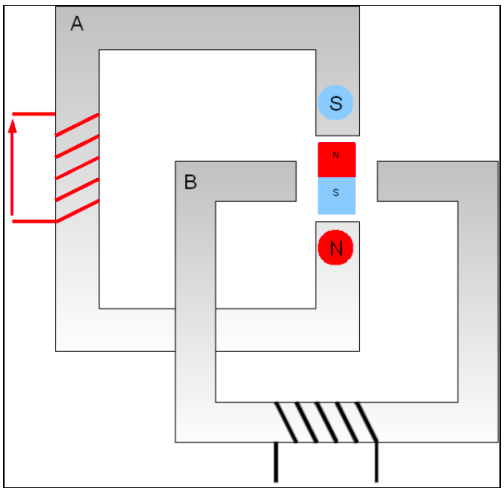


FIGURE 3.3 – Pas 1

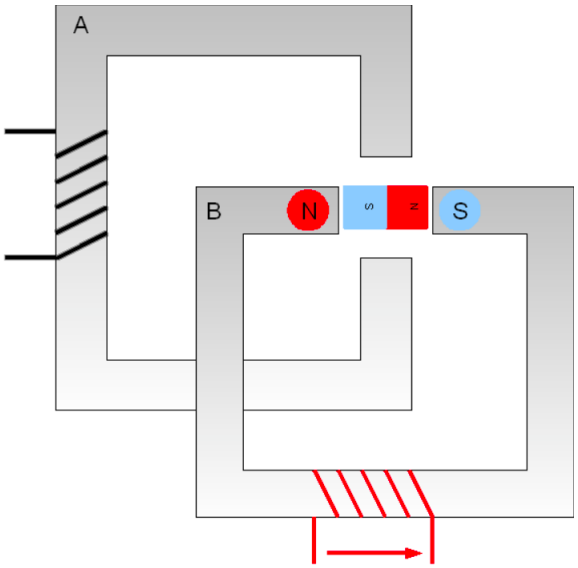


FIGURE 3.4 – Pas 2

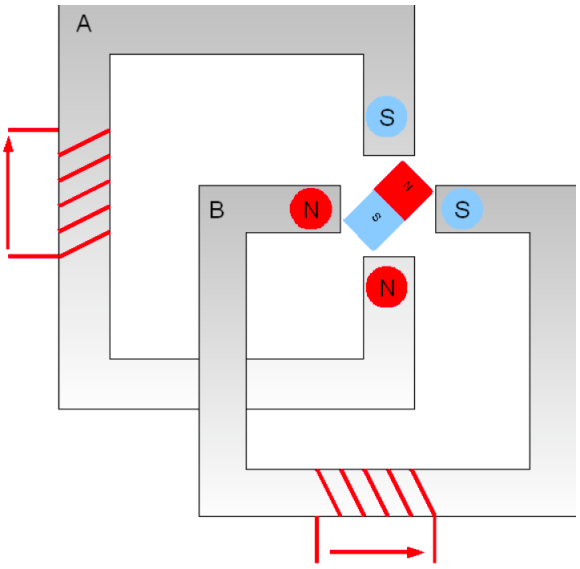


FIGURE 3.5 – Pas 3

Exemples avec phases unipolaires

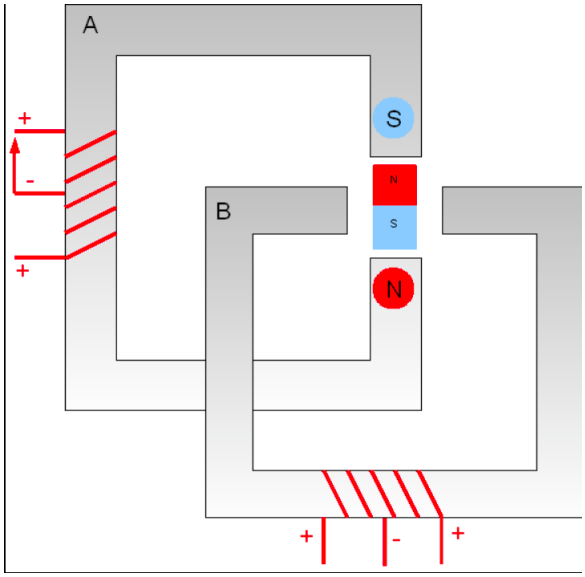


FIGURE 3.6 – Pas 1

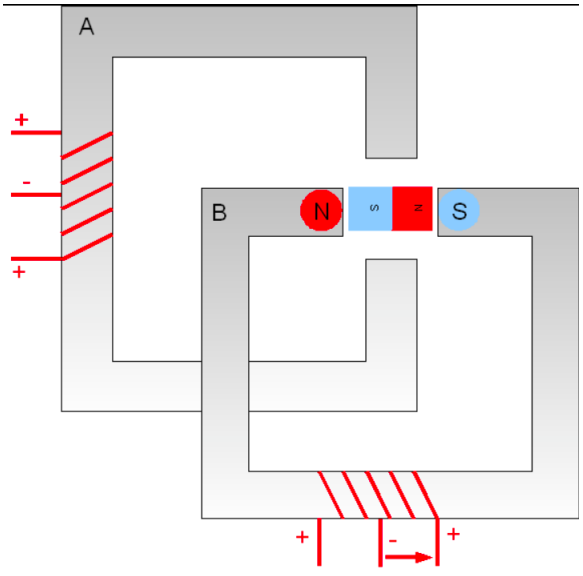


FIGURE 3.7 – Pas 2

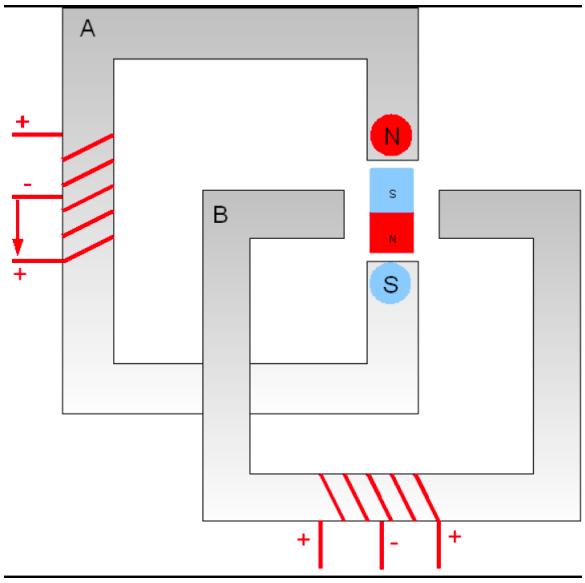


FIGURE 3.8 – Pas 3

Les moteurs possèdent plus de phases car un débattement de 45° est vite limité.

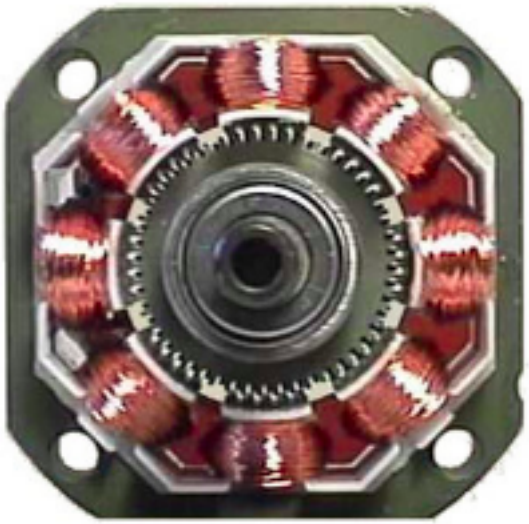


FIGURE 3.9 – Un intérieur de moteur

Les moteur pas-à-pas unipolaires présentent l'avantage de faire circuler un courant positif dans le circuit de commande. Ils sont donc plus simples à mettre en oeuvre mais ils nécessitent plus de bobinage.

Section 4

Commande des moteurs pas-à-pas

4.1 Moteurs unipolaires

Les moteur pas-à-pas unipolaires ont besoins d'être contrôlés via un circuit adaptés, le plus connu est le REF ULN2803

FIGURE 4.1 – Driver ULN2803

Pour les moteurs unipolaires, il faut mettre une des phases à la masse pour faire circuler le courant dans la phase [1]

On constate sur la figure suivante un montage Darlington : deux transistors NPN forme un seul transistor dont le coefficient β est le produit de chacun des coefficients β de chaque transistor.

Cela permet de contrôler des charges avec très peu de courant de commande. [2]

1. Due au cable relié à l'alimentation positive du moteur
2. Se référer à la partie **Circuits de puissance**

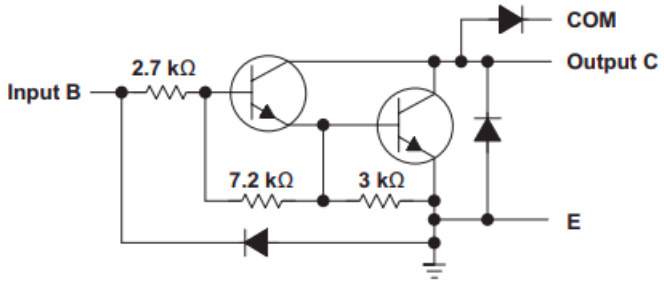


FIGURE 4.2 – Contrôle d'une phase

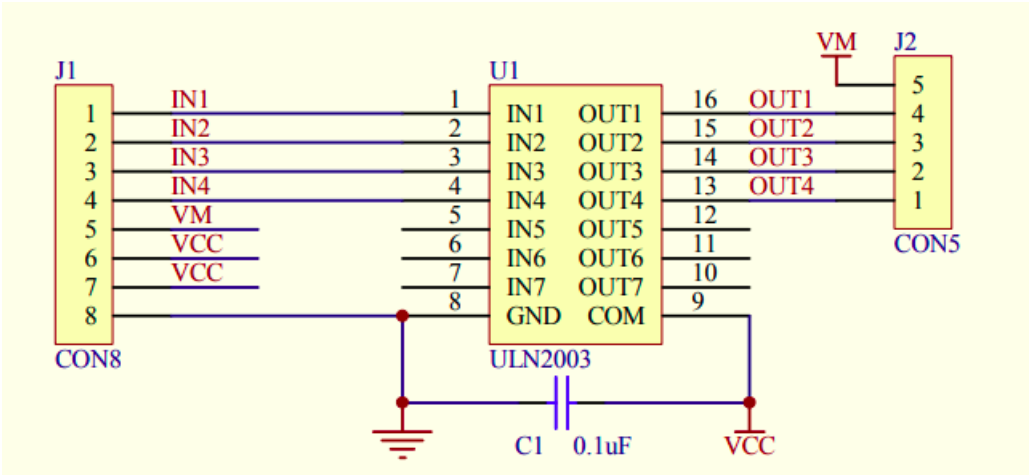


FIGURE 4.3 – Driver de controle

4.2 Moteurs bipolaires

On a vu qu'il fallait inverser la tension de commande au borne des bobines. Pour cela on peut utiliser le montage en pont en H.

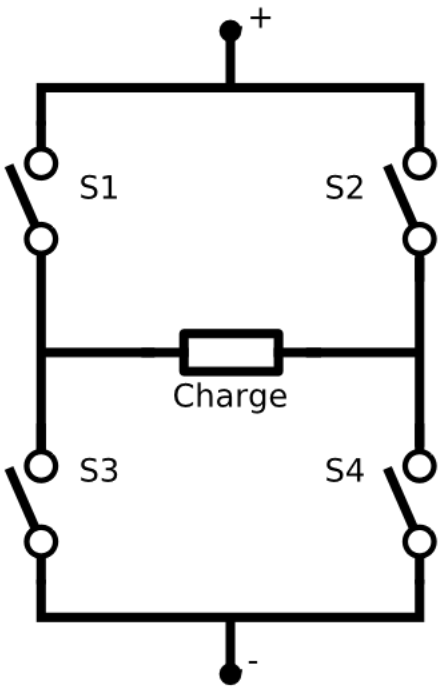


FIGURE 4.4 – Structure du pont en H

- En activant S1 et S4 (fermeture du circuit), la charge est parcourue par un courant allant de gauche à droite
- En activant S2 et S3 (fermeture du circuit), la charge est parcourue par un courant allant de droite à gauche

Et qui dit inversion de courant dit inversion de tension. Notre objectif est atteint, nous pouvons mettre des tensions positives et négatives aux bornes des phases de nos moteurs.

Ce principe est également utilisé pour contrôler les moteurs à courant continu
On peut utiliser le circuit REF L298

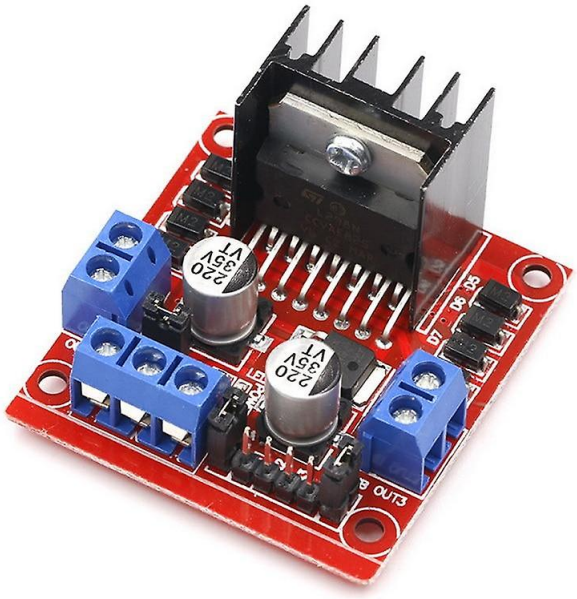


FIGURE 4.5 – Un pont en H intégré

4.2.1 Avantages et inconvénients des moteurs pas-à-pas

- Très grande précision en boucle ouverte [3]
- Couple élevé en bas régime
- Plus lent qu'un servomoteur
- Complexité de mise en oeuvre

4.2.2 Domaines d'application

- Imprimantes
- Machines CNC

4.3 Comment distinguer les différents types de moteurs ?

- 2 fils = moteur à courant continue
- 3 fils = servomoteurs
- 4 fils = moteur pas-à-pas bipolaire
- 5 fils = moteur pas-à-pas unipolaire

3. Contrôle sans asservissement, contrairement aux servomoteurs

Section 5

Exemples

5.1 Mise en pratique avec Arduino

Nous utiliserons un moteur pas-à-pas **28BYJ-48** de type unipolaire.

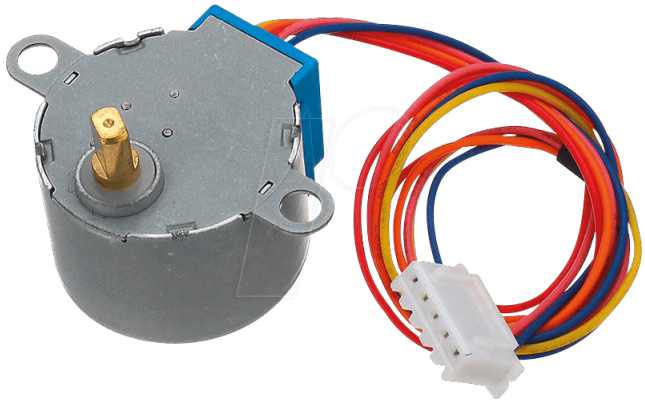


FIGURE 5.1 – Le moteur 28BYJ-48

Les caractéristiques sont les suivantes :

- Nombre de pas : 2048
- Tension d'alimentation : 5V

Pour augmenter le nombre de pas, on ajoute un train d'engrenage.

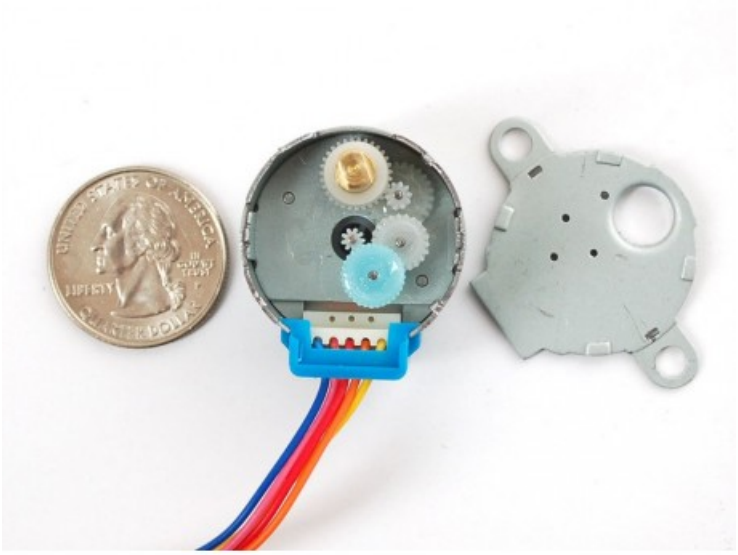


FIGURE 5.2 – Une augmentation du nombre de pas

5.1.1 Liste du matériel

- Carte Arduino Uno
- Driver ULN2803
- Moteur moteur pas-à-pas 28BYJ-48 ou équivalent
- Câbles

5.1.2 Branchements

Nous utiliserons les broches 8, 9, 10 et 11 et l'alimentation 5V du moteur sera fournie par la broche +5V de l'Arduino

- D1 sur IN1
- D2 sur IN3
- D5 sur IN2
- D6 sur IN4
- Vin sur Vcc
- Gnd sur Gnd

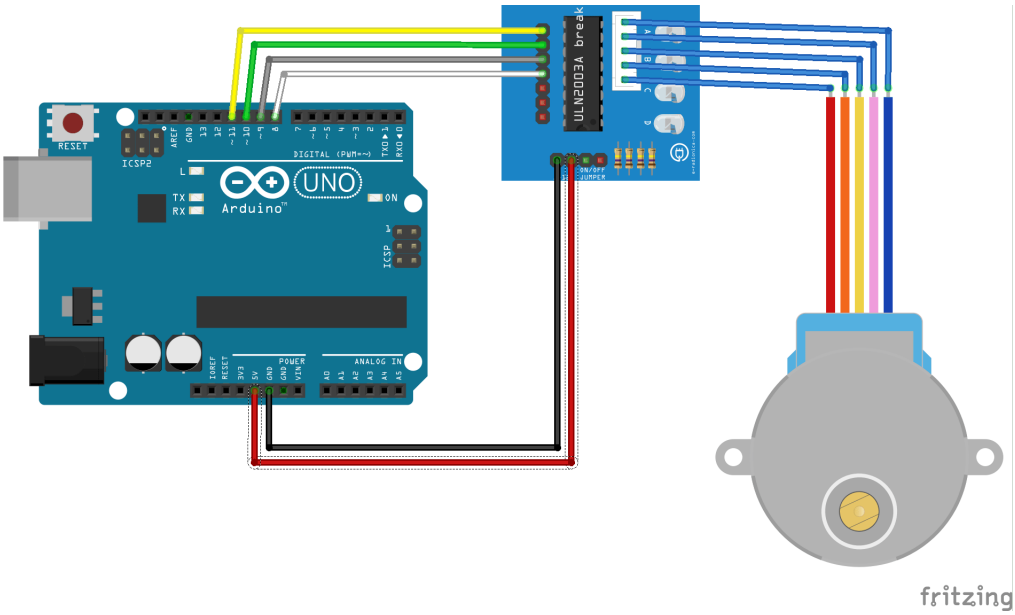


FIGURE 5.3 – Schéma Arduino

5.1.3 Code Arduino

Ce code fait tourner le moteur d'un tour, attend 2 secondes puis fait un tour dans l'autre sens avec un délai de 2s.

```
#include <Stepper.h> //Inclusion de la bibliothèque Stepper

int nbPas = 2048; //Nombre de pas pour le moteur 28BYJ-48

#define IN1 8  //Broche IN1
#define IN2 9  //Broche IN2
#define IN3 10  //Broche IN3
#define IN4 11  //Broche IN4

Stepper moteur(nbPas, IN1, IN3, IN2, IN4); //Création de l'objet moteur

  void setup() {

    moteur.setSpeed(10); //On définit la vitesse à 10 tr/min

  }//Fin setup

  void loop() {

    moteur.step(nbPas);   //On avance de nbPas pas, c'est à dire un tour complet
   (sens horaire)
```

```
    delay(2000);           //pause de 2s
    moteur.step(-nbPas);   //On avance de -nbPas pas, c'est à dire un tour complet
    (sens anti-horaire)
    delay(2000);           //pause de 2s

  }//Fin loop
```

Code minimaliste Arduino

5.2 Mise en pratique avec ESP8266

Nous utiliserons le même moteur pas-à-pas 28BYJ-48

5.2.1 Liste du matériel

- Carte ESP8266 NodeMCU (ESP-12)

FIGURE 5.4 – ESP12 NodeMCU

  Cette carte fait partie de la famille des ESP8266 et se programme directement avec l'Éditeur Arduino. L'installation des bibliothèques pour l'ESP12 est détaillée en annexe du document.

- Driver ULN2803
- Moteur moteur pas-à-pas 28BYJ-48 ou équivalent (moteur pas-à-pas unipolaire)
- Câbles

5.2.2 Branchements

Les numéros des broches sont différents sur les cartes ESP812 (modèle NodeMCU).
Voici les équivalences des broches entre le code et l'emplacement physique.

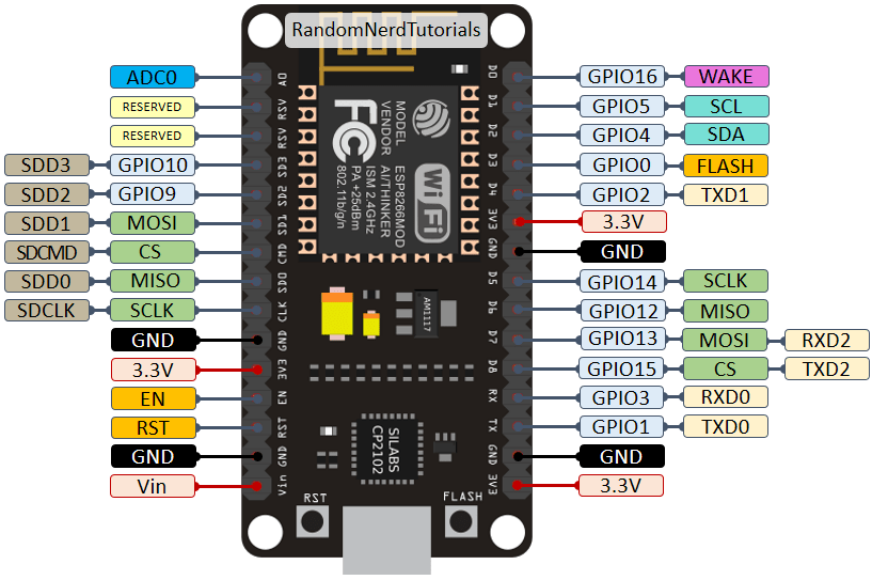


FIGURE 5.5 – Broches ESP12

Nous utiliserons les broches D1, D2, D5 et D6 et l'alimentation 5V du moteur sera fournie par la broche Vin de l'ESP12

- D1 sur IN1
- D2 sur IN3
- D5 sur IN2
- D6 sur IN4
- Vin sur Vcc
- Gnd sur Gnd

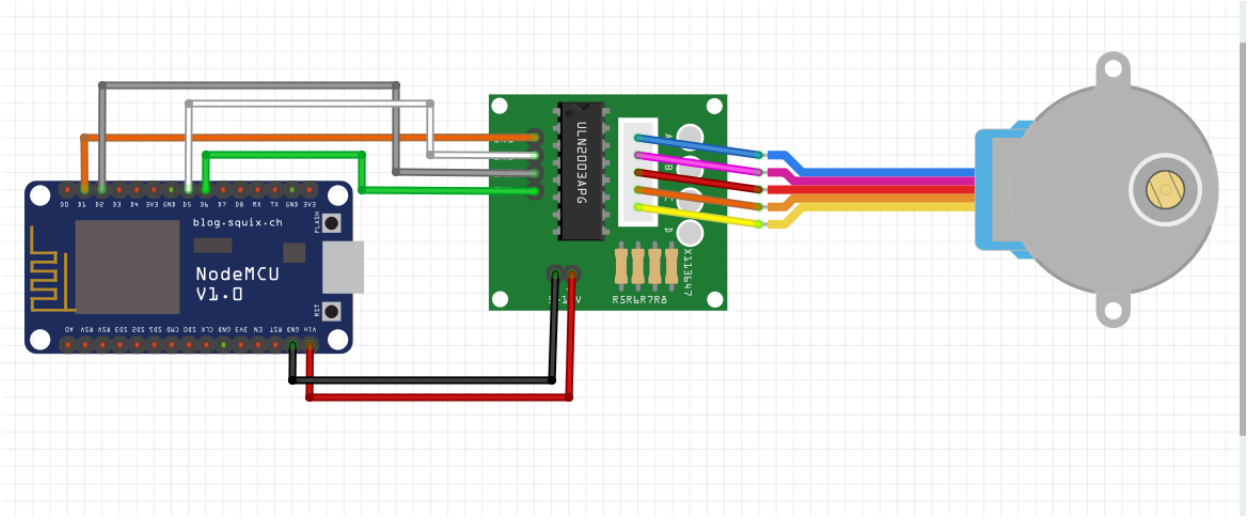


FIGURE 5.6 – Schéma ESP12

5.2.3 Code ESP12

Ce code fait tourner le moteur d'un tour, attend 2 secondes puis fait un tour dans l'autre sens avec un délai de 2s.

```
#include <Stepper.h> //Inclusion de la bibliothèque Stepper

int nbPas = 2048; //Nombre de pas pour le moteur 28BYJ-48

#define IN1 D1  //Broche IN1
#define IN2 D5  //Broche IN2
#define IN3 D2  //Broche IN3
#define IN4 D6  //Broche IN4

Stepper moteur(nbPas, IN1, IN3, IN2, IN4); //Création de l'objet moteur

  void setup() {

    moteur.setSpeed(10); //On définit la vitesse à 10 tr/min

  }//Fin setup

  void loop() {

    moteur.step(nbPas);   //On avance de nbPas pas, c'est à dire un tour
 complet (sens horaire)
    delay(2000);          //pause de 2s
    moteur.step(-nbPas);  //On avance de -nbPas pas, c'est à dire un tour
 complet (sens anti-horaire)
    delay(2000);          //pause de 2s

  }//Fin loop
```

Code minimaliste ESP12

Troisième partie

Les interfaces de puissance

Théorie sur les interfaces de puissance et applications pratiques avec Arduino

Section 6

Introduction

Pour certains projets plus évolués, on souhaite utiliser des composants tels que des moteurs ou des résistances (chauffage, ventilation).
Or, on constate rapidement que le branchement direct de ces éléments sur une carte Arduino va se révéler impossible.

En effet, la carte Arduino est prévu pour délivrer de **faibles courants** et **faibles tensions** .
Nous allons donc créer un circuit où la puissance et la commande sont dissociés.

Section 7

Les transistors bipolaires

7.1 Présentation

Une des moyens pour créer notre circuit de puissance est le transistor bipolaire. Ce composant possède trois broches :

- Le collecteur (C)
- la base (B)
- l'émetteur (E)

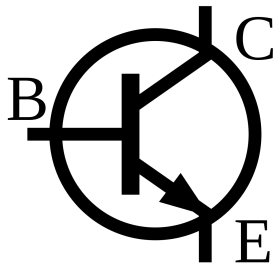


FIGURE 7.1 – La représentation du transistor bipolaire

7.2 Conventions

Afin de simplifier les calculs par la suite, posons les normes suivantes :

- Le courant entrant dans le Collecteur est appelé I_C
- Le courant entrant dans la Base est appelé I_B
- Le courant sortant de l'émetteur est appelé I_E
- La tension entre la Base et l'Émetteur est appelée V_{be}

- La tension entre le Collecteur et l'Émetteur est appelée V_{ce}

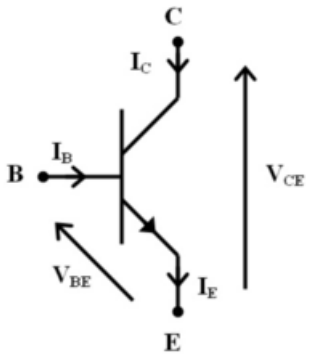


FIGURE 7.2 – Conventions du transistor bipolaire

Les flèches au sein du transistor indiquent le sens de déplacement du courant sur les broches.

7.2.1 Les familles de transistors bipolaires

Les transistors bipolaires sont classés en deux catégories :

- Les transistors NPN [1]
- Les transistors PNP

Le principe de fonctionnement est similaire entre ces deux familles, seul le branchement et le niveau de commande diffère.
Dans ce document, nous utiliserons essentiellement des transistors NPN car ces derniers utilisent des grandeurs positives.

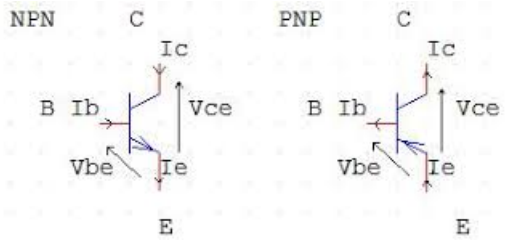


FIGURE 7.3 – Transistors NPN et PNP

7.3 Les paramètres de sélection du transistor

Notre transistor doit dans un premier temps répondre à deux contraintes :

1. Le nom de ces familles provient du type de jonction utilisé en interne. Pour plus de renseignements, consulter les diodes et semi-conducteurs

- La tension admissible sur V_{ce} [2] doit être supérieur à la tension d'alimentation de notre circuit. Concrètement, si notre circuit est alimenté en 48V mais que le transistor ne supporte pas plus de 30V, il va être détruit.
- Le transistor doit supporter un courant plus élevé que le courant maximal transitant dans notre circuit. Pour contrôler un moteur consommant 1 Ampère, je dois donc choisir un transistor pouvant contrôler au moins 2 Ampère.

Pour la suite de la présentation, on supposera que notre transistor a été dimensionné pour répondre à ces deux contraintes.

7.4 Le principe

Ce type de transistor fonctionne comme une vanne pour une canalisation. Il est possible de réguler le débit de la canalisation avec la vanne.

Le transistor bipolaire permet de contrôler un courant important avec un faible courant.

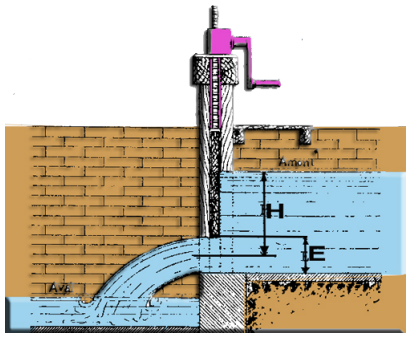


FIGURE 7.4 – Le rôle du transistor

.

Ici, notre transistor joue le rôle de la vanne et permet de bloquer le courant (électrons) ou bien de les laisser passer.

Le courant de l'élément à contrôler (moteur, résistance de puissance) transite entre le collecteur et l'émetteur et le courant de commande passe par la base, comme l'illustre la figure suivante.

2. Cette tension est indiquée dans les documentations techniques

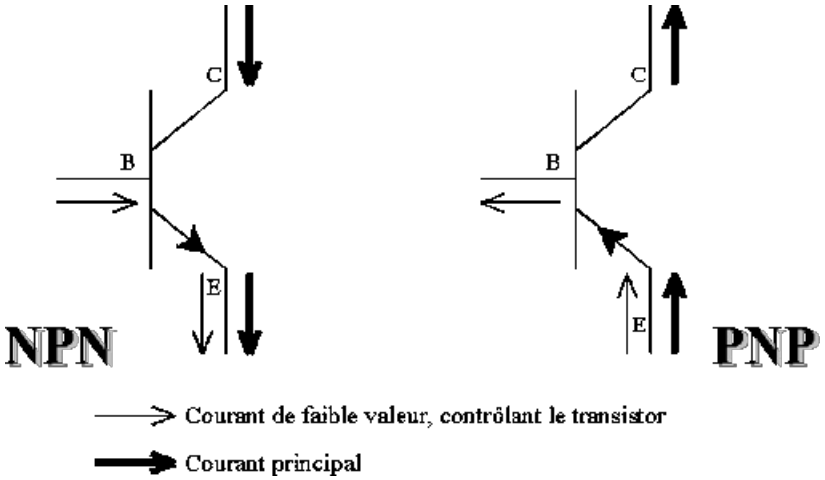


FIGURE 7.5 – Courant de commande et de puissance

La relation fondamentale reliant le courant de puissance et de commande est la suivante :

$$\boxed{I_C = \beta \cdot I_B}$$

Le paramètre β, appelé **gain du transistor**[3] est une caractéristique interne de notre transistor, c'est à dire qu'il dépend du type de transistor que nous choisissons.
Les courants I_C, I_B, I_E sont exprimés dans la même unité (Ampère, milliampères..) pour une formule homogène.

Les transistors de puissance possède des gains de l'ordre de la dizaine alors que les transistors de signal (faibles courants) ont un gain pouvant facilement atteindre 200 ou 300.

Remarque

Plus notre β est faible, plus il va falloir injecter un courant important dans notre base

Question 1. *Et que devient notre broche "**Émetteur** " ?*

*>>> **1.** Notre émetteur est relié à la masse du circuit et permet de le fermer pour que les électrons puissent circuler.*

Le courant circulant dans l'émetteur est simplement la somme des courants entrant dans le transistor.
d'où :

$$\boxed{I_E = I_B + I_C}$$

7.5 Exemple

On souhaite commander l'arrêt et la marche d'un moteur consommant au maximum 0.5A et alimenté avec une tension de 9V.

3. le gain est sans dimension (unité) et est appelé h_{fe} dans les documentations

Nous choisissons un transistor permettant de commuter 1A (sécurité) avec $\beta = 30$

Question 2. *Quel doit-être le courant injecté dans la base ?*

>>> **2.** *On applique la formule précédent et on obtient :*

$$I_B = \frac{I_C}{\beta} = \frac{0.5}{30} = 16mA$$

7.6 Mise en pratique

7.6.1 Branchements

Maintenant que nous connaissons les tensions et courants nécessaires à notre transistor et à notre moteur, nous allons le commander avec une carte Arduino.

Tout d'abord, il convient de placer le moteur entre notre alimentation et le collecteur.

Remarque

Toutes les charges à contrôler avec ce type de transistor se placent entre l'alimentation et le collecteur.

Enfin, il ne nous reste plus qu'à relier une sortie numérique de l'Arduino vers notre base par l'intermédiaire d'une résistance.

La résistance va servir à imposer le courant dans la base de notre transistor .

Nous obtenons donc le schéma suivant.

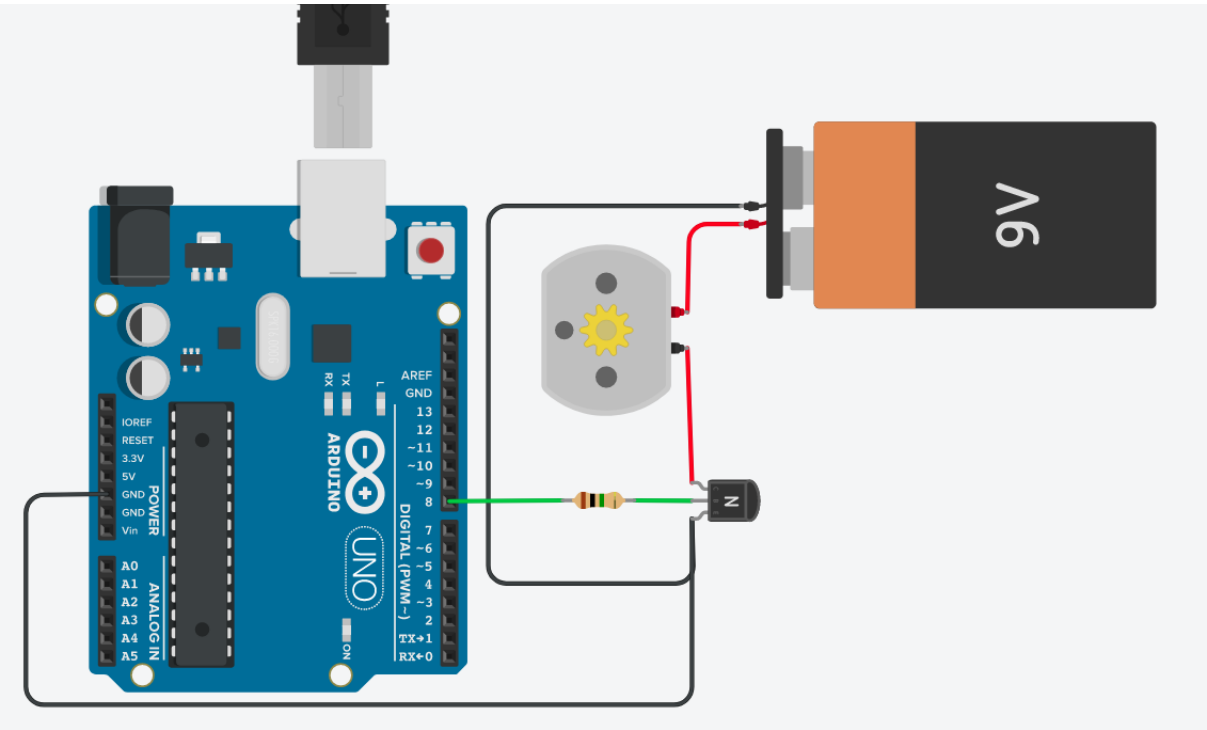


FIGURE 7.6 – Branchement du transistor bipolaire

7.6.2 Dimensionnement de la résistance

On souhaite obtenir un courant de $16mA$ dans notre base et on sait que l'Arduino délivre du $5V$ en sortie.

Nous somme donc tentés de dire que $R_b = \frac{U_{arduino}}{I_B} = \frac{5}{0.016} = 312\Omega$ [4]

Hélas, il y a peu de chance que votre moteur tourne dans les conditions optimales.
Il convient d'avoir à l'esprit que notre β trouvé dans la documentation n'est que théorique et qu'il peut être en réalité inférieur.

Remarque

Une des conventions non officielles admet que pour de la commutation en **Tout ou Rien** [a], on divise la valeur théorique de notre β par 2.
Nous allons donc prendre donc un β valant 15.

a. Le transistor laisse passer tout le courant nécessaire ou rien du tout

On refait donc les calculs.

$$I_B = \frac{I_C}{\beta} = \frac{0.5}{15} = 32mA$$

4. On part de la loi d'Ohm qui dit que $U = R.I$

Une dernière chose : les transistors bipolaires entraînent une chute de tension entre la base et l'émetteur (V_{be}).
Cette chute de tension dépend de la technologie des transistors bipolaires :

- $0.7V$ pour les transistors au silicium
- $0.3V$ pour les transistors au germanium

Dans l'extrême majorité des cas, on utilisera des transistors au silicium.
La tension disponible aux bornes de la résistance est donc de $4.3V$ $(5 - 0.7)$
D'où :

$$\boxed{R_b = \frac{U_{arduino} - V_{be}}{I_b} = \frac{4.3}{0.032} = 134\Omega}$$

7.6.3 Exemple de programme Arduino

Voici un code permettant de faire tourner le moteur périodiquement pendant 5 secondes puis de l'arrêter pendant 5 secondes.

```
#define D8 8     //Broche 8 de l'Arduino

void setup() {

  pinMode(D8, OUTPUT); //Mise en sortie de la broche

}//End setup

void loop() {

  digitalWrite(D8, HIGH);     //Déclencher la rotation du moteur
  delay(5000);                //Délai de 5s
  digitalWrite(D8, LOW);      //Fin de la rotation du moteur
  delay(5000);                //Délai de 5s

}//End loop
```

Code Arduino avec transistors NPN

Section 8

Les transistors MOSFET

MOSFET

8.1 Présentation

Nous avons vu l'utilisation des transistors bipolaires.
Ces derniers sont assez contraignants à mettre en oeuvre car ils sont commandés en courant.

Nous allons utiliser cette fois-ci la technologie des MOSFET [1] car ces derniers ont l'avantage d'être contrôlés en **tension** .

Ce composant possède trois broches :

- Le drain (D)
- la porte (G) [2]
- la source (S)

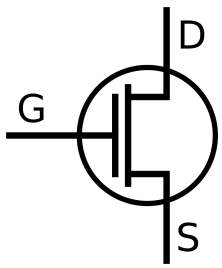


FIGURE 8.1 – La représentation du transistor MOSFET

1. MOSFET : Metal Oxide Semiconductor Field Effect Transistor = Transistor à effet de champ à structure métal-oxyde-semi-conducteur
2. 'G' pour Gate

8.2 Conventions

Afin de simplifier les calculs par la suite, posons également les normes suivantes :

- Le courant entrant dans le Drain est appelé I_D
- Le courant entrant dans la Porte est appelé I_G
- Le courant sortant de la Source est appelé I_S
- La tension entre la Porte et la Source est appelée V_{GS}
- La tension entre le Drain et la Source est appelée V_{DS}

8.2.1 Les familles de transistors MOSFET

Les transistors MOSFET sont classés en deux catégories :

- Les transistors MOSFET à canal N [3]
- Les transistors MOSFET à canal P

Le principe de fonctionnement est similaire entre ces deux familles, seul le branchement et le niveau de commande diffère.
Dans ce document, nous utiliserons essentiellement des transistors MOSFET à canal N car ces derniers utilisent des grandeurs positives.

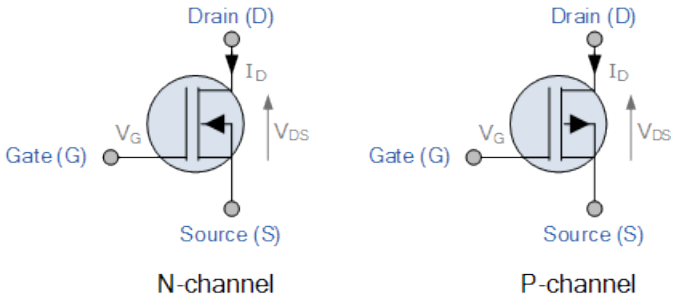


FIGURE 8.2 – Transistors à canal N et P

8.3 Les paramètres de sélection du transistor

Les paramètres de sélection de nos transistors MOSFET sont identiques aux transistors bipolaires, c'est à dire :

- La tension admissible sur V_{DS} du transistor
- Le courant admissible entre le Drain et la Source.

Pour la suite de la présentation, on supposera que notre transistor a été dimensionné pour répondre à ces deux contraintes.

3. Le nom de ces familles provient du type de jonction utilisé en interne. Pour plus de renseignement, consulter les diodes et semi-conducteurs

8.4 Le principe

Ce type de transistor fonctionne comme les transistors bipolaires mais est commandé en tension et non en courant.

Par analogie, le drain joue le rôle du collecteur, la source celui de l'émetteur et la porte celui de la base. Le courant de l'élément à contrôler (moteur, résistance de puissance) transite entre le drain et la source et la tension de commande est aux bornes de la porte.

Les transistors MOSFET deviennent passant [4] lorsque la tension sur la porte dépasse une tension de déclenchement appelée $V_{GS_{th}}$. Cette valeur est généralement comprise entre 2 et 4 Volts. [5]

Lorsque cette tension $V_{GS_{th}}$ est atteinte, notre transistor peut être remplacé d'un point de vue électrique entre le drain et la source par une résistance de très faible valeur, appelée $R_{DS_{on}}$

8.5 Comparaison avec les transistors bipolaires

Par nature, la porte du MOSFET est vue comme un condensateur. Le transistor ne consomme pas de courant, excepté pendant les commutations.
Ainsi, le courant est nul dans la porte pour maintenir le moteur en marche alors que pour un bipolaire, il faut maintenir un courant dans la base.

Les MOSFET sont donc plus économes en énergie que les bipolaires.
De plus, ils peuvent généralement supporter des courants plus importants que les bipolaires.

En revanche, en hautes fréquences, les MOSFET sont moins réactifs du fait de leur capacité en entrée.

8.6 Mise en pratique

Nous souhaitons faire tourner le même moteur que celui utilisé avec notre transistor bipolaire. Nous allons le commander avec une carte Arduino.

8.6.1 Branchements

Tout d'abord, il convient de placer le moteur entre notre alimentation et le drain.

4. Le transistor laisse passer tout le courant autorisé.
5. Lorsque la tension V_{GS} est inférieure à $V_{GS_{th}}$, I_D vaut $K \cdot ((V_{GS} - V_{th}) \cdot V_{DS} - \frac{1}{2}V_{DS}^2)$
Cette relation montre que l'étude en amplification est plus complexe car non linéaire.

Remarque

Toutes les charges à contrôler avec ce type de transistor se placent entre l'alimentation et le drain.

Enfin, il ne nous reste plus qu'à relier une sortie numérique de l'Arduino vers notre porte **sans** résistance.

Nous obtenons donc le schéma suivant.

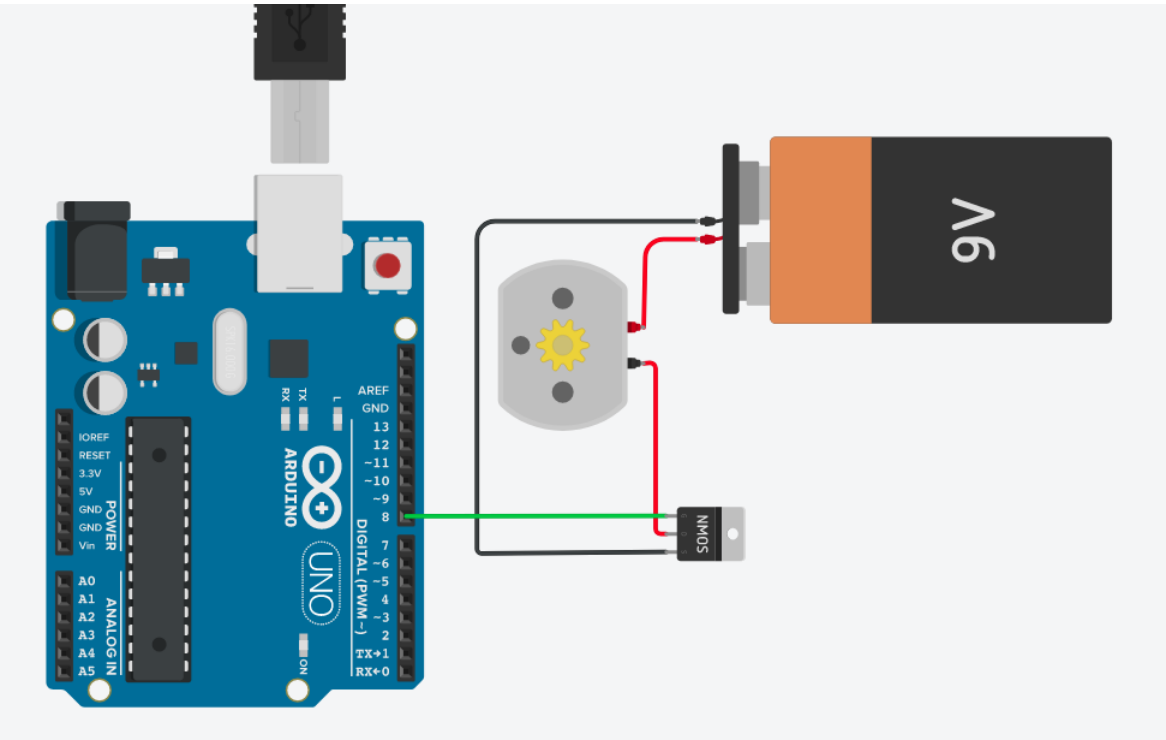


FIGURE 8.3 – Branchement du transistor MOSFET

8.6.2 Exemple de programme Arduino

Voici un code permettant de faire tourner le moteur périodiquement pendant 5 secondes puis de l'arrêter pendant 5 secondes. Il s'agit du même code que pour le transistor bipolaire.

```
#define PIN 11     //GATE du transistor

void setup() {

  pinMode(PIN, OUTPUT); //Mise en sortie de la broche

}//Fin setup

void loop() {
```

```
  digitalWrite(PIN, HIGH);    //Mise en route du transistor
  delay(5000);                //Délai de 5s
  digitalWrite(PIN, LOW);     //Arret du transistor
  delay(5000);                //Délai de 5s

}//Fin loop
```

Code Arduino avec MOSFET

Section 9

Conclusion

9.1 Ce qu'il faut retenir

Nous avons à notre disposition tout un ensemble de technologies pour contrôler la partie puissance.

Les transistors ne sont pas adaptés pour commuter une charge sur secteur (230V), cette partie sera donc réservée aux relais.
En revanche, pour toutes les tensions continues, les transistors sont adaptés et prennent moins de place en encombrement.

9.2 Les fiches techniques

L'intégralité des informations disponibles pour un transistor sont disponibles dans un document complet appelé **Datasheet** .

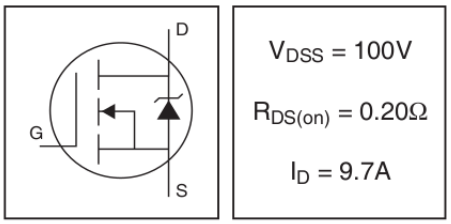
Ce document détaille les broches, les caractéristiques électriques, propose des schémas d'exemples....

Par exemple, voici quelques extraits de la documentation du transistors IRF520 [1] :

1. Transistor de puissance

IRF520NPbF

HEXFET® Power MOSFET



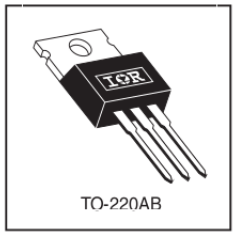


FIGURE 9.1 – Extrait n°1 du IRF520

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	100	—	—	V	V_{GS} = 0V, I_D = 250µA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.11	—	V/°C	Reference to 25°C, I_D = 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	—	0.20	Ω	V_{GS} = 10V, I_D = 5.7A ④
$V_{GS(th)}$	Gate Threshold Voltage	2.0	—	4.0	V	V_{DS} = V_{GS}, I_D = 250µA

FIGURE 9.2 – Extrait n°2 du IRF520

On retrouve sur cette figure la valeur de $R_{DS_{on}}$ et de $V_{GS_{th}}$

Quatrième partie

Les réseaux

Introduction aux réseaux

Section 10

Les réseaux

Un réseau est un ensemble de machines[1] reliées entre elles. Dans notre cas, la liaison sur le réseau se fera en Ethernet ou WiFi.

IP Pour communiquer entre elles, il faut définir des adresses (IP[2]) exactement comme une adresse postale pour transmettre un courrier.

10.0.1 Les adresses MAC

Une adresse MAC (Medium Access Control) est une adresse unique qui désigne la machine[3]. Elle est de la forme MAC XX :XX :XX :XX :XX :XX (six octets usuellement exprimés et hexadécimal) et reste invariante dans le temps.

Contrairement aux adresses IP IP , les adresses MAC ne sont normalement pas attribuées explicitement par configuration ; elles sont au contraire attribuées à la production de la carte réseau par son fabriquant

La table ARP[4] ARP d'une machine sert à associer des adresses IP à des adresses MAC.

10.0.2 Les adresse IP

Une adresse IP est une adresse donnée à une machine qui rejoint le réseau.

A quoi ressemble une adresse IP ?

- Version 4 (Ipv4) : Les adresses sont codées sur 32 bits

IP 192.168.0.1 = BIN 1100000.10101000.0000000.00000001 (En Binaire)

1. Serveurs, PC, Tablettes, téléphones, imprimantes
2. Internet Protocol
3. Plus précisément l'adresse de la carte réseau de la machine
4. Address Resolution Protocol

- version 6 (Ipv6) : les adresses sont codées sur 128 bits
  IP FE80 :0000 :0000 :0000 :020C :76FF :FE21 :1C3B. L'adresse de version 4 (IPv4) est encore actuellement la plus utilisée.

Dans un réseau, chaque machine possède une adresse IP fixée par l'administrateur du réseau. Il est interdit de donner la même adresse à 2 machines différentes sous peine de dysfonctionnement.

Une adresse IPv4 est une suite de 32 bits (4 octets) notée en général a.b.c.d avec a, b, c, et d des entiers « décimal » compris entre 0 et 255. Chaque valeur a, b, c ou d représente dans ce cas une suite de 8 bits.

Exemple 1. *Une machine qui a comme adresse IP 134.214.80.12 (En décimal) :*

- *a vaut 134 soit (1000 0110) en binaire.*
- *b vaut 214 soit (1101 0110) en binaire.*
- *c vaut 80 soit (0101 0000)*
- *d vaut 12 vaut (00001100).*

En binaire, l'adresse IP s'écrit IP *10000110.1101 0110.0101 0000.0000 1100 et puisque le codage se fait sur 8 bits les valeurs seront obligatoirement comprises entre 0 et 255.*

Le net-id et le host-id

Au sein d'un même réseau IP, toutes les adresses IP commencent par la même suite de bits. L'adresse IP d'une machine va en conséquence être composée de 2 parties :

- Net-id : La partie fixe de l'adresse IP
- Host-id : La partie réservée aux machines qui viennent sur le réseau

Masque de réseau IP

Le masque du réseau permet de connaître le nombre de bits du net-id. On appelle N ce nombre. Il s'agit d'une suite de 32 bits composée en binaire de N bits à 1 suivis de 32-N bits à 0.
C'est le masque du réseau qui définit la taille d'un réseau IP : c'est-à-dire la plage d'adresses assignables aux machines du réseau.

Ainsi, pour connaître le net-id, on va faire un ET (& ou .) logique entre le masque et l'adresse IP.

Exemple 2. *Prenons l'adresse IP* IP *192.168.1.0 et le masque* MASK *255.255.255.0.*
Que vaut le net-id ?
Pour rappel :

- *0.0 = 0*
- *0.1 = 0*

- *1.0 = 0*
- *1.1 = 1*

Solution 1. *L'adresse* IP *192.168.1.0* *vaut* BIN *1100000.10101000.0000000.00000001* *et L'adresse* MASK *255.225.255.0* *vaut* BIN *11111111.11111111.11111111.00000000*

01100000.10101000.00000000.00000001
ET 11111111.11111111.11111111.00000000
= 01100000.10101000.00000000.00000000 = BIN *01100000.10101000.00000000.00000000*

L'adresse IP *192.168.0.0* *obtenue représente donc* ***l'adresse du réseau*** .

Les autres bits à 0 sont donc réservés aux périphériques du réseau. Étant sur 8 bits, il y a donc $2^8 - 2$ *adresses disponibles sur ce réseau.*

10.0.3 Les adresses interdites

Il est interdit d'attribuer à une machine d'un réseau l'adresse du réseau et l'adresse de broadcast (diffusion) .

L'adresse de diffusion

Cette adresse permet à une machine d'envoyer une info à toutes les machines d'un réseau. Cette adresse est celle obtenue en mettant tous les bits de l'host-id à 1. Dans notre cas c'est l'adresse IP 192.168.0.255

L'adresse de réseau

Cette adresse est celle obtenue après avoir fait le ET entre l'adresse IP et le masque de sous-réseau. Dans notre cas c'est l'adresse IP 192.168.0.0

10.1 Choix des adresses IP

Sur un réseau local, les adresses sont choisies par un serveur DHCP [5] DHCP .
Les adresses peuvent être :

- Statiques : Un appareil sur le réseau possède la même adresse après connexion puis déconnexion.
- Dynamiques : l'adresse IP varie dans le temps au bout d'une période d'expiration après une déconnexion (bail d'une journée par exemple)

A de rares exceptions, vous n'êtes pas censé attribuer une adresse IP vous-même à une machine.

5. **DHCP** : Dynamic Host Configuration Protocol

10.2 Quelques exemples de type de réseau

Un réseau IP peut avoir une taille très variable :

- Une entreprise moyenne aura un réseau comportant une centaine de machines.
- Un campus universitaire aura un réseau comportant de quelques milliers à quelques dizaines de milliers de machines.
- Le réseau d'une multinationale (un grand fournisseur d'accès par exemple) peut comporter des millions de postes.

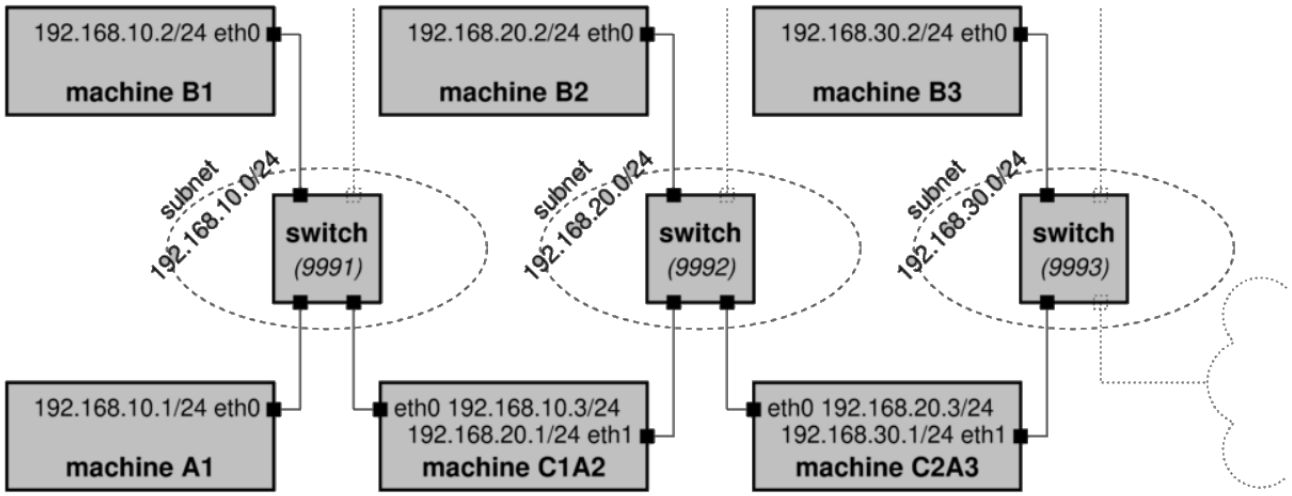


FIGURE 10.1 – Un réseau plus évolué

Il existe 2 types de réseau :

- Les réseaux publics Internet où chaque équipement connecté doit posséder une adresse unique et enregistrée au niveau mondial.
- Les réseaux privés, dans ce cas le choix des adresses est libre et ne doivent être uniques que dans ce réseau.

Si un réseau privé doit être inter-connecté avec le réseau Internet, il faudra alors utiliser des adresses privées qui ne puissent correspondre à des adresses publiques utilisées sur Internet. Des plages d'adresses réservées à usage privé existent et elles ne sont donc pas acheminées par les routeurs Internet, ce qui supprime tout risque de conflit (cf. document annexe).

10.3 Récupération des adresse IP

Voici 2 commandes pour récupérer l'adresse IP et le masque de sous-réseau.

Pour Windows

Sous un terminal Windows [6]

```
ipconfig
```

Récupération des informations IP sous Windows

```
C:\Users\Admin>ipconfig

Configuration IP de Windows


Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6. . . . . . . . . . . . . .: 2a01:e0a:227:a340:a4f7:9a72:aa0a:347f
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:2d8f:1bca:f420:b777
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:3d02:1319:b97a:cbdb
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:4448:9d1:fe45:3916
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:907f:4342:42d7:8451
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:99ff:f749:3636:af2e
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:ac70:9311:f199:596e
   Adresse IPv6 temporaire . . . . . . . .: 2a01:e0a:227:a340:ddc1:2e12:5b54:a4ed
   Adresse IPv6 de liaison locale. . . . .: fe80::a4f7:9a72:aa0a:347f%7
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.1.41
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . : fe80::72fc:8fff:fe47:689c%7
                                            192.168.1.254
```

FIGURE 10.2 – La commande ipconfig

Pour Linux

Sous un terminal Linux, on peut utiliser la commande LIB ifconfig [7]

```
ifconfig
```

Récupération des informations IP sous Linux

6. Saisir **cmd** dans le gestionnaire de programme
7. Si non installée sous Linux, saisir CMD sudo apt-get install net-tools

```
nico@nico-ThinkPad-L580:~$ ifconfig
enp0s31f6: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether e8:6a:64:7d:be:44  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xe1200000-e1220000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 2228  bytes 337363 (337.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2228  bytes 337363 (337.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp5s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.49  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 2a01:e0a:227:a340:2080:e8d3:7ee3:2b7e  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::c48e:e202:5ade:c5fc  prefixlen 64  scopeid 0x20<link>
        inet6 2a01:e0a:227:a340:d757:63b3:a442:734e  prefixlen 64  scopeid 0x0<global>
        ether fc:77:74:1e:76:ed  txqueuelen 1000  (Ethernet)
        RX packets 42721  bytes 38163746 (38.1 MB)
        RX errors 0  dropped 6  overruns 0  frame 0
        TX packets 24604  bytes 6842107 (6.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

FIGURE 10.3 – La commande ipconfig

Cinquième partie

Mise en place d'un serveur ESP12

Serveur Web avec ESP12

Section 11

Un serveur Web avec ESP12

L'objectif de ce chapitre est de créer un serveur Web pour contrôler la led interne de l'ESP12, une carte basée sur les ESP8266 (Broche D4)

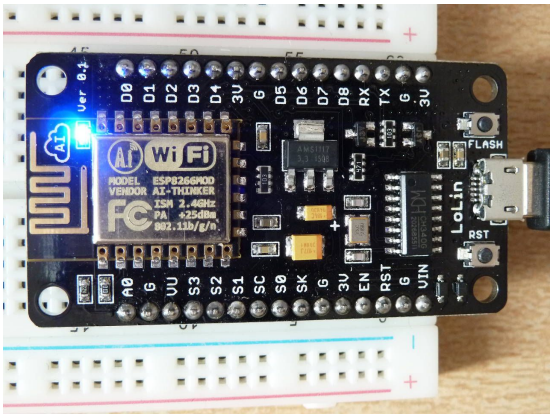


FIGURE 11.1 – La led interne de l'ESP

11.1 Architecture du mini-projet

Pour communiquer entre le client (utilisateur) et l'ESP12, nous utiliserons un routeur qui servira de passerelle.

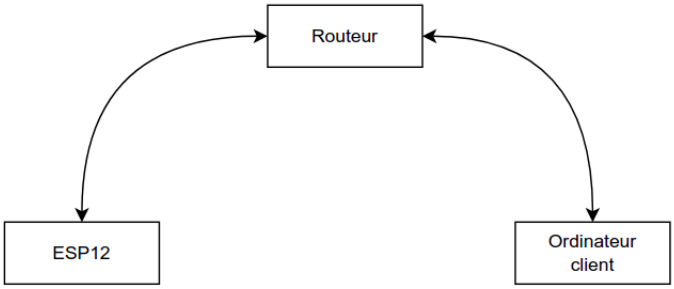


FIGURE 11.2 – Architecture du projet

L'ESP12 se connecte dans un premier temps au routeur. Une fois connecté, tout client sur le même réseau peut se connecter à l'ESP12 en saisissant l'adresse de l'ESP12 dans le navigateur.

11.2 Base des requêtes

Dès que nous allons sur une page Web, nous faisons une requête, c'est à dire que l'on va demander l'affichage d'une page Web à un serveur distant.

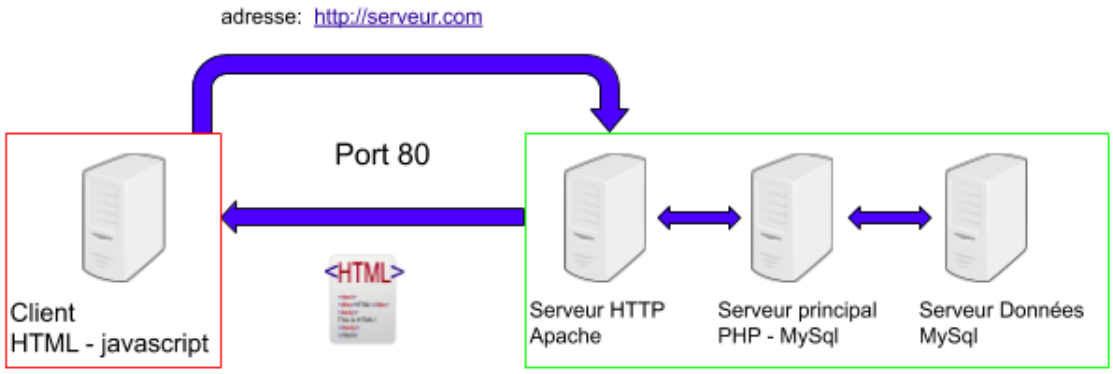


FIGURE 11.3 – Architecture client-serveur

Afin de récupérer une page sur le serveur, nous avons besoin de connaitre 3 élements :

- L'adresse IP IP ou l'adresse du serveur : Ici ce sera l'adresse IP de l'ESP12
- L'emplacement de la page sur le serveur : L'emplacement '/' désigne la racine du serveur
- Le port de communication entre le client et le serveur : port 80 par défaut pour le protocole HTTP [1] HTTP

11.2.1 Une petite explication sur les ports

Pour recevoir et transmettre des données à d'autres ordinateurs, un ordinateur (ou serveur) a besoin de ports. Cependant, les ports physiques tel que le port Eternet communiquent avec beaucoup de services.

Ainsi, des ports virtuels ont été créés.

Chaque port virtuel est codé sur 16 bits et permet de faire communiquer un service ou un logiciel. [2]. Il y a donc potentiellement 65536 ports disponibles. Certains numéros de port sont réservés à certains services
Les ports de 0 à 1023 sont déjà réservés à des services particuliers et le port par défaut pour les serveurs Web est le 80.

1. le protocole HTTPS utilise le port 443
2. Par exemple, le service SSH communique sur le port 22

11.2.2 Les types de requêtes

Lorsque nous nous connectons à un serveur Web, nous faisons principalement deux types de requêtes [3]

- Requêtes GET

  Les requêtes GET sont des requêtes avec des éléments passés via l'URl de la page. Elles sont donc visibles via la barre d'adresse :

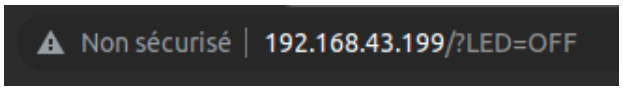


FIGURE 11.4 – L'adresse avec la requête GET

  Chaque élément est séparé avec le symbole LBL & et la liste des arguments commencent avec le symbole LBL ? Comme nous avons un seul élément passé via l'URl, nous n'avons pas le symbole LBL &

  Ici, nous avons un argument ARG LED avec la valeur VAL OFF

- Requêtes POST

  Ces requêtes ne passent pas les arguments via l'adresse URl. Cette section ne sera pas abordée dans le cadre de l'atelier.

11.3 Connexion au routeur

La carte ESP12 a besoin du nom du routeur ainsi que de son mot de passe. Dans le programme FILE serveur_Web_simple.ino, il faut préciser le nom et le mot de passe sur les lignes suivantes :

```
//Par exemple
const char* ssid     = "Creafab_invite";
const char* password = "MonTraficEstJournalise";
//Il faut mettre le nom du routeur chez soi et le mot de passe associé
```

Identifiant et mot de passe

11.4 Lancement du programme

Pour sélectionner la carte ESP12, veuillez vous reporter à l'annexe **UTILISATION DE L'ESP12 SOUS ARDUINO** .

3. Les webSockets ne seront pas abordées

Remarque

Il faudra activer la liaison série de la carte. Pour cela, lors du choix de la carte dans le logiciel Arduino, dans la section **Outils** > **Types de cartes** > **ESP12 NodeMCU** , il faut bien vérifier que la case **Serial** est activée

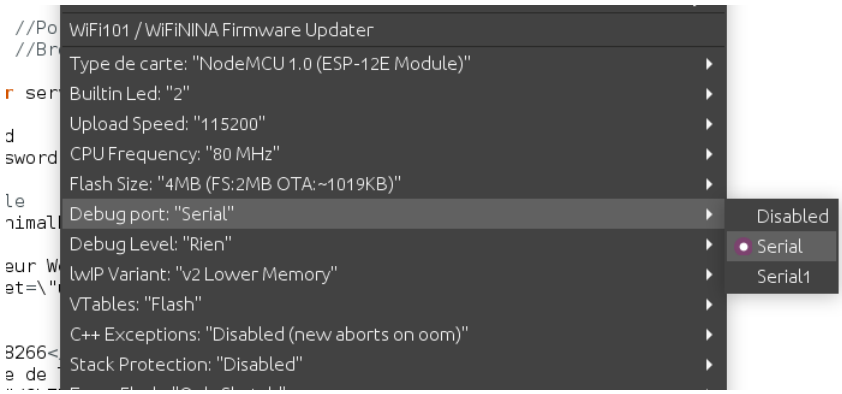


FIGURE 11.5 – Sélection du mode Série

Une fois le programme téléversé, il vous faudra récupérer l'adresse IP de l'ESP12.

Pour cela, une fois que le code est téléversé, veuillez ouvrir la fenêtre du moniteur série, l'adresse IP de l'ESP va apparaître au bout de quelques secondes.

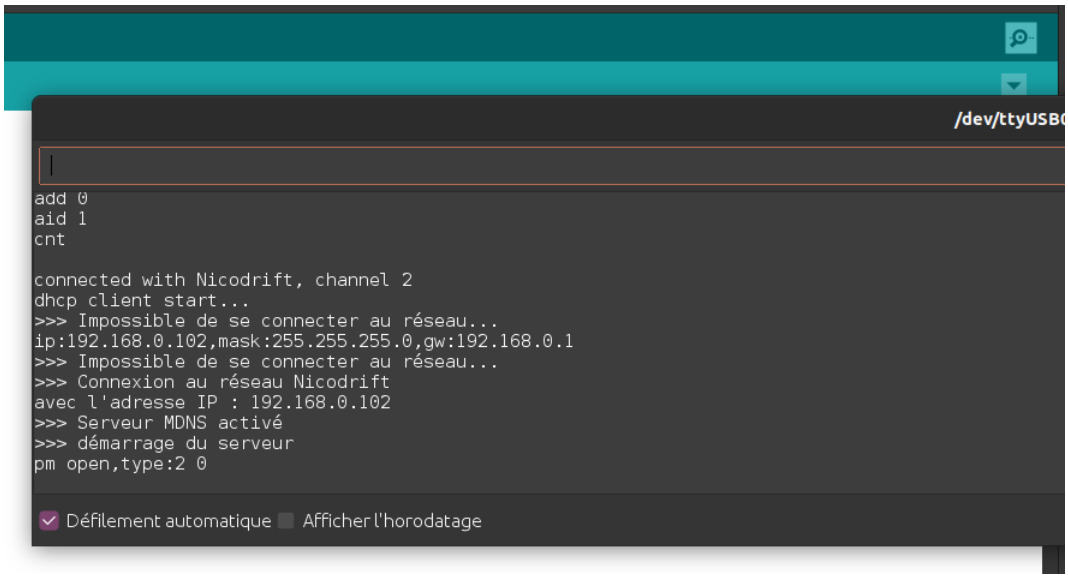


FIGURE 11.6 – Affichage de l'adresse IP

Si aucune données n'apparaît, faite un reset de la carte ESP12 en appuyant sur la touche RST de la carte.

Il ne vous reste plus qu'à rentrer l'adresse IP obtenue dans un navigateur internet.

En l'occurrence, l'adresse IP dans ce cas là est 192.168.0.102

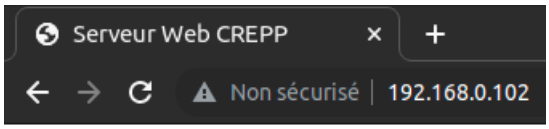


FIGURE 11.7 – Connexion au serveur

Le résultat doit être le suivant

FIGURE 11.8 – Résultat

11.5 Explication du programme

Une explication du langage HTML est disponible en annexe (section HTML)

11.5.1 En tête du code

Après avoir importé les bibliothèques liées à l'ESP12, nous définissons un objet **ESP8266WebServer** qui attend comme argument le port du serveur, c'est à dire le 80.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

#define PORT 80 //Port par défaut
#define LED D4  //Broche de la LED

ESP8266WebServer server(PORT);
```

Importation des bibliothèques

On précise ensuite le mot de passe et le nom du routeur de communication.

```
const char* ssid     = "Nom-reseau-Internet";
const char* password = "Mot-de-passe-Routeur";
```

Identifiant et mot de passe du routeur

Ensuite, nous allons définir et créer notre page HTML.

Deux versions de pages Web seront proposées :

- Une version minimale sans aucune mise en forme ajoutée.

```
const String minimalPageContent = "<html>\
<head>\
    <title>Serveur Web CREPP</title>\
    <meta charset=\"utf-8\"/> \
    </head>\
    <body>\
    <h1>Led ESP8266</h1><br>\
        <h3>Contrôle de la LED sur la broche D4</h3><br>\
          <a href=\"/?LED=ON\"><button >Allumer</button></a>\
          <a href=\"/?LED=OFF\"><button >Eteindre</button></a>\
      </body>\
    </html>";
```

Page minimale

- Une version plus élaborée en utilisant la bibliothèque LIB Bootstrap qui permet de faire de jolies mises en forme très facilement.

```
const String fullPageContent = "<html>\
<head>\
  <title>Serveur Web CREPP</title>\
  <meta charset=\"utf-8\"/> \
  <link rel=\"stylesheet\" href=\"https://stackpath.bootstrapcdn
.com/bootstrap/4.3.1/css/bootstrap.min.css\" integrity=\"sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T\"
crossorigin=\"anonymous\">\
</head>\
<body style=\"margin-left:5%;\">\
  <h1>Led ESP8266</h1><br>\
  <h3>Contrôle de la LED sur la broche <span class=\"badge badge
-secondary\">D4</span></h3><br>\
    <a href=\"/?LED=ON\"><button class=\"btn btn-success\">
Allumer</button></a>\
    <a href=\"/?LED=OFF\"><button class=\"btn btn-danger\">
Eteindre</button></a>\
</body>\
```

```
            </html>";
```

Page plus élaborée

11.5.2 Fonction setup

Lançons nous dans le code de la fonction **setup**
On définit la led en sortie et on se connecte au routeur.

```
void setup() {

    pinMode(LED, OUTPUT);       //LED en sortie
    digitalWrite(LED, LOW);     //LED éteinte
    Serial.begin(115200);       //Communication à 115200 bits/s
    WiFi.begin(ssid, password); //Connexion
    Serial.println("");         //Retour à la ligne
```

Initialisation

Puis on vérifie que nous sommes bien connecté et on affiche les informations de connexion.

```
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.println(">>> Impossible de se connecter au réseau...");
    }//Fin while

    Serial.print(">>> Connexion au réseau ");
    Serial.println(ssid);
    Serial.print("avec l'adresse IP : ");
    Serial.println(WiFi.localIP());
```

Vérification de la connexion

Enfin on vérifie que le MDSN [4] est activé (optionnel)

```
    if (MDNS.begin("esp8266")) {   //Multicast DNS
    Serial.println(">>> Serveur MDNS activé");
    }
}
```

Vérification de la connexion

On définit (toujours dans le setup) les pages accessibles par le client ainsi que la fonction appelée lors de la requête. Il s'agit de l'emplacement '/' et sa fonction associées est la fonction **mainPage**

4. Multicast DNS, un serveur de résolution de nom de domaine

```
    server.on("/", mainPage);           //Affichage de la page principale si requ
   ête sur '/' -> saisir IP dans le navigateur
```

Redirection sur la page principale

On redirige également l'utilisateur sur une page dédiée si l'adresse demandée n'existe pas.

```
    server.onNotFound(notFoundPage);    //Affichage de la page d'erreur si
   adresse non valide
```

Redirection en cas d'erreur

Enfin, on initialise le serveur.

```
    server.begin();                     //Initialisation du serveur
    Serial.println(">>> démarrage du serveur");
```

Initialisation du serveur

11.5.3 Fonction loop

Le code à l'intérieur de la fonction **loop** se contente de gérer de manière transparente le comportement du serveur.

```
void loop()
{
    server.handleClient(); //Gestion des clients sur le serveur
}//Fin loop
```

Fonction principale

11.6 Code complet

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

#define PORT 80 //Port par défaut
#define LED D4  //Broche de la LED

ESP8266WebServer server(PORT);

const char* ssid     = "Nom-reseau-Internet";
const char* password = "Mot-de-passe-Routeur";

//Page principale
const String minimalPageContent = "<html>\
<head>\
```

```
    <title>Serveur Web CREPP</title>\
    <meta charset=\"utf-8\"/> \
    </head>\
    <body>\
    <h1>Led ESP8266</h1><br>\
        <h3>Contrôle de la LED sur la broche D4</h3><br>\
          <a href=\"/?LED=ON\"><button >Allumer</button></a>\
          <a href=\"/?LED=OFF\"><button >Eteindre</button></a>\
      </body>\
    </html>";

    const String fullPageContent = "<html>\
      <head>\
        <title>Serveur Web CREPP</title>\
        <meta charset=\"utf-8\"/> \
        <link rel=\"stylesheet\" href=\"https://stackpath.bootstrapcdn.com/
   bootstrap/4.3.1/css/bootstrap.min.css\" integrity=\"sha384-
   ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T\"
   crossorigin=\"anonymous\">\
      </head>\
      <body style=\"margin-left:5%;\">\
        <h1>Led ESP8266</h1><br>\
        <h3>Contrôle de la LED sur la broche <span class=\"badge badge-secondary
   \">D4</span></h3><br>\
          <a href=\"/?LED=ON\"><button class=\"btn btn-success\">Allumer</button
   ></a>\
          <a href=\"/?LED=OFF\"><button class=\"btn btn-danger\">Eteindre</button
   ></a>\
      </body>\
    </html>";


void setup() {

    pinMode(LED, OUTPUT);       //LED en sortie
    digitalWrite(LED, LOW);     //LED éteinte
    Serial.begin(115200);       //Communication à 115200 bits/s
    WiFi.begin(ssid, password); //Connexion
    Serial.println("");         //Retour à la ligne


    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.println(">>> Impossible de se connecter au réseau...");
```

```
    }

    Serial.print(">>> Connexion au réseau ");
    Serial.println(ssid);
    Serial.print("avec l'adresse IP : ");
    Serial.println(WiFi.localIP());

    if (MDNS.begin("esp8266")) {   //Multicast DNS
        Serial.println(">>> Serveur MDNS activé");
    }

    server.on("/", mainPage);           //Affichage de la page principale si requ
   ête sur '/' -> saisir IP dans le navigateur
    server.onNotFound(notFoundPage);    //Affichage de la page d'erreur si
   adresse non valide

    server.begin();                     //Initialisation du serveur
    Serial.println(">>> démarrage du serveur");

}//Fin setup

void loop()
{

    server.handleClient(); //Gestion des clients sur le serveur

}//Fin loop


void mainPage() //Page principale
{

    if(server.arg("LED")=="ON") //Lecture de l'argument 'LED'
    {
         digitalWrite(LED, LOW); //On allume la led
    }//Fin if
    else if(server.arg("LED")=="OFF")
    {
        digitalWrite(LED, HIGH);   //On eteint la led
    }//Fin else if
    else {
     //nothing
    }
    server.send(200, "text/html", fullPageContent); //On envoie la page
   principale
```

```
}//FIn mainPage


void notFoundPage()  //Gestion si mauvaise URL
{
    server.send(404, "text/plain", "Page introuvable !\n\n");

}//Fin notFoundPage
```

Code complet

Sixième partie

Les capteurs et périphériques

Les capteurs et périphériques

Section 12

Principes et théorie

12.1 Objectifs

Ce chapitre a pour but de faire un petit tour d'horizon des différents capteurs et de leur mise en place.
Une mise en exemple sera faite avec le code de base pour réaliser un serveur Web.

12.2 Les types de capteurs

Il existe une multitude de capteurs :

- Capteur de distance
- Capteur de température
- Capteur de présence
- Capteur de pression-humidité
- Capteur de position (potentiomètre, joystick, fin de course)
- Capteur de particules fines
- Capteur d'accélération

12.3 Les modes de transmission

Voici les différentes façon de communiquer.
Pour nos exemples, nous nous baserons sur deux périphériques qui communiquent entre eux.

- Simplex : la communication est unidirectionnel , c'est à dire que le périphérique A envoie des informations au périphérique B mais le B ne peut pas envoyer au A.

FIGURE 12.1 – L'analogie du mode Simplex

- Half-duplex : la communication se fait dans les deux sens mais avec un décalage dans le temps. Si le périphérique A communique, le B ne peut pas envoyer des informations en meme temps que le A.

FIGURE 12.2 – L'analogie du mode Half-Duplex

- Full-duplex : les périphériques peuvent communiquer en même temps, comme dans une conversation téléphonique.

FIGURE 12.3 – L'analogie du mode Full-Duplex

12.4 Les protocoles de communication

Pour communiquer, de nombreux protocoles existent mais voici les principaux.

12.4.1 Le bus I2C

Le bus I2C I2C est un bus informatique qui a émergé dans les années 80 pendant la «guerre des standards» lancée par les acteurs du monde électronique.
Conçu par Philips pour les applications de domotique et d'électronique domestique, il permet de relier facilement un microcontrôleur et différents circuits récepteurs tels que des capteurs de pression, température....

C'est un bus série synchrone bidirectionnel half-duplex avec 2 broches utilisées pour communiquer :

- SDA : Serial Data (ligne de données) SDA
- SCL : Serial Clock (ligne d'horloge) SCL

Une masse est commune aux périphériques.

Les échanges ont toujours lieu entre un seul maître et un (ou tous les) esclave(s), toujours à l'initiative du maître [1] et pour éviter les conflits électriques les broches SDA et SCL sont de type **Collecteur Ouvert** . Il faut donc ajouter des résistances de tirage mais ces dernières sont généralement intégrées.

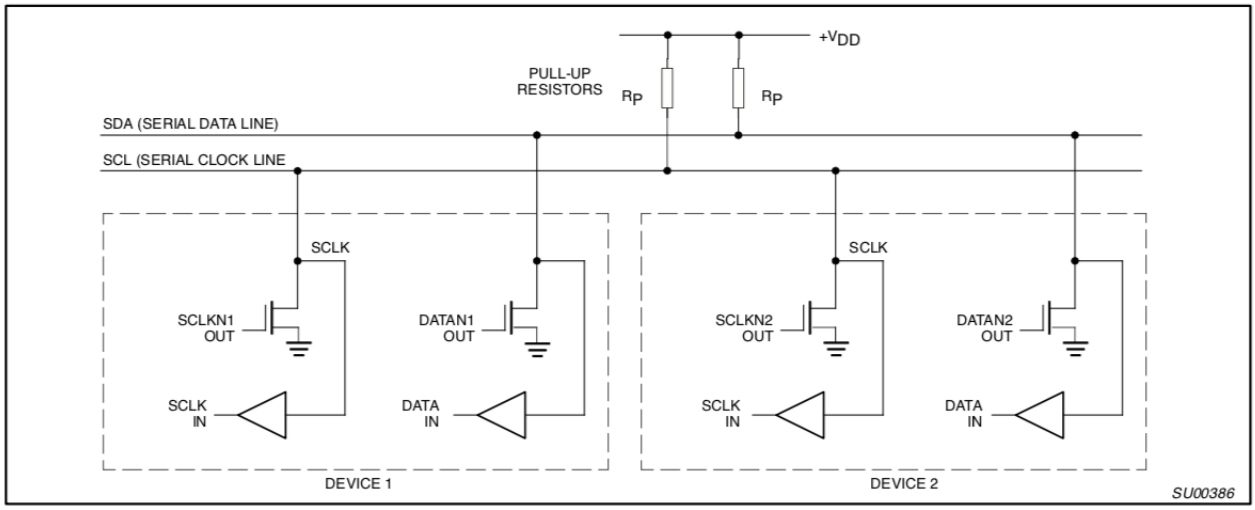


Figure 4. Connection of I²C-bus devices to the I²C-bus

FIGURE 12.4 – Les résistances de rappel du bus I2C

Il existe d'innombrables périphériques exploitant ce bus, il est même implémentable par logiciel dans n'importe quel microcontrôleur.

A chaque composant est attribué une adresse physique qui permettra les échanges.
Cette adresse est codée sur 7 bits, ce qui fait que le bus I2C peut supporter en théorie 127

1. Jamais de maître à maître ou d'esclave à esclave, cependant, rien n'empêche un composant de passer du statut de maître à esclave et réciproquement

périphériques [2].
Par exemple, on pourra trouver sur un même bus I2C :

- 1 écran OLED (adresse 0x3C) OLED
- 1 écran LCD (0x27) LCD
- 1 capteur de pression BME180 (0x35)

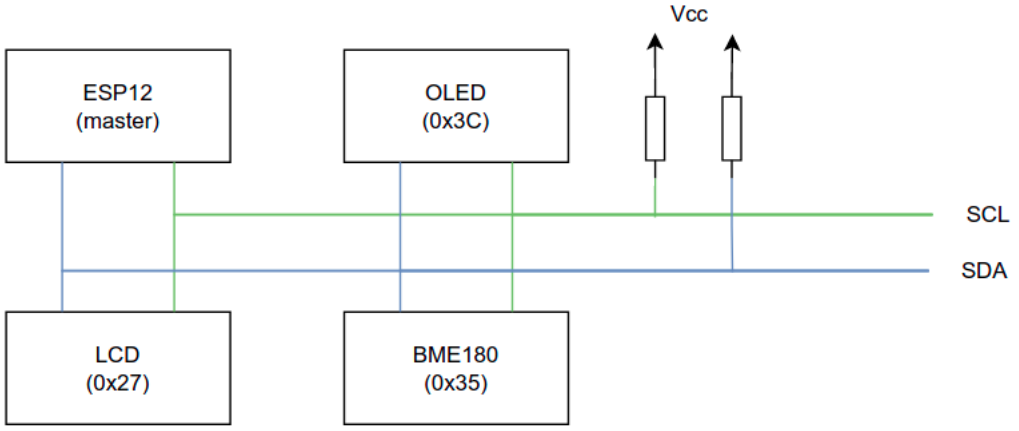


FIGURE 12.5 – Un réseau de capteurs

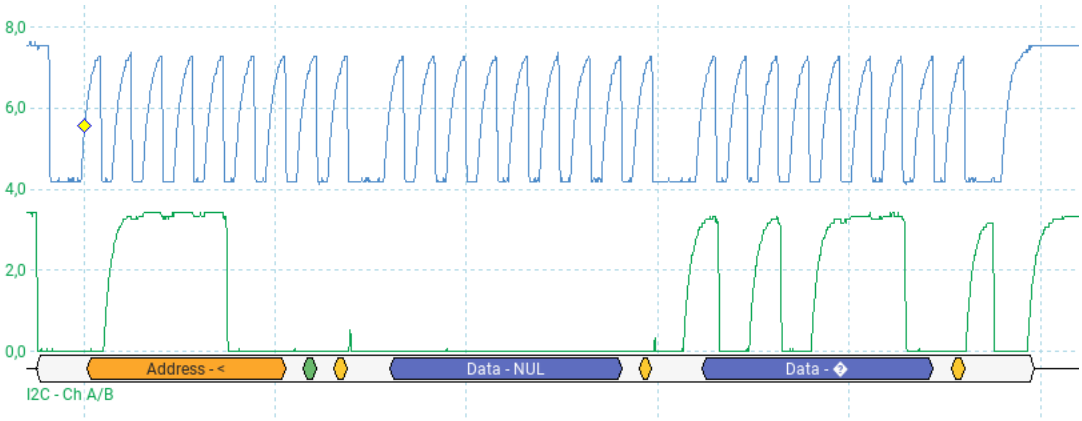


FIGURE 12.6 – Une capture de trame I2C

12.4.2 Les changements d'adresses

Lorsqu'on souhaite connecter plusieurs périphériques ayant la même adresse (par exemple 2 capteurs de température), il est possible pour certains périphériques de mettre certaines broches à un certain niveau logique pour définir l'adresse.

2. en réalité moins car il faut tenir compte de la capacité de ligne

12.4.3 Le bus SPI

Le bus SPI[3] SPI est full-duplex et développé par Motorola dans le milieu des années 80. La liaison est de type maitre-esclave ou le maitre sélectionne l'esclave avec qui il veut communiquer avec une broche PIN SS.

Le bus comporte 4 broches :

- SCLK : Serial Clock, Horloge (généré par le maître)
- MOSI : Master Output Slave Input (généré par le maître)
- MISO : Master Input Slave Output (généré par l'esclave)
- SS : Slave Select, Actif à l'état bas (généré par le maître)

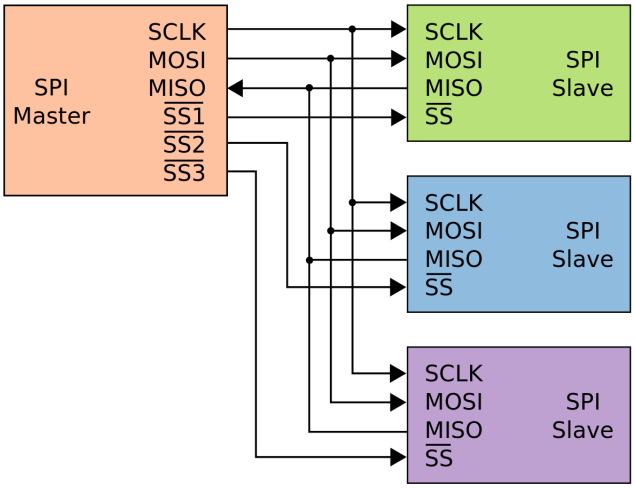


FIGURE 12.7 – Un bus SPI

Protocole

- Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer par l'utilisation du signal SS
- L'esclave répond aux requêtes du maître

12.4.4 La liaison UART

UART

Il s'agit d'une liaison série avec deux broches :

- RX

3. SPI : Serial Peripheral Interface

- TX

Il permet uniquement de faire communiquer plus de deux appareils entre eux, contrairement aux bus I2C ou SPI, on ne peut pas relier plusieurs périphériques.

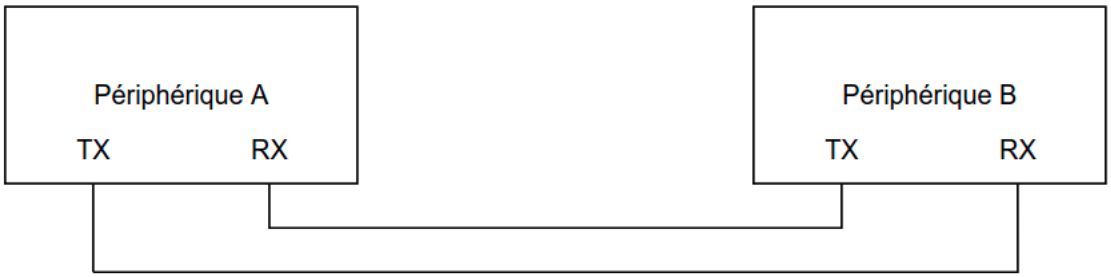


FIGURE 12.8 – Une communication UART

Protocole

- Un bit de start toujours à 0 pour synchroniser la communication
- Un champ de données de 7 à 8 bits
- Un bit de parité (paire ou impaire)
- Un bit de stop

FIGURE 12.9 – Le protocole UART

Au repos, la ligne est au niveau logique HAUT.

Vitesse de communication

La liaison étant asynchrone, il faut que les périphériques communiquent à la même vitesse. Cette dernière est normalisée et représente le nombre de bit par seconde (baud [4])

- 1 200 bauds
- 2 400 bauds
- 4 800 bauds
- 9 600 bauds
- 19 200 bauds

4. 1 baud représente 1 symbole par seconde.

- 38 400 bauds
- 57 600 bauds
- 115 200 bauds

Une trame en exemple

On constate bien que le niveau au repos est au niveau HAUT.

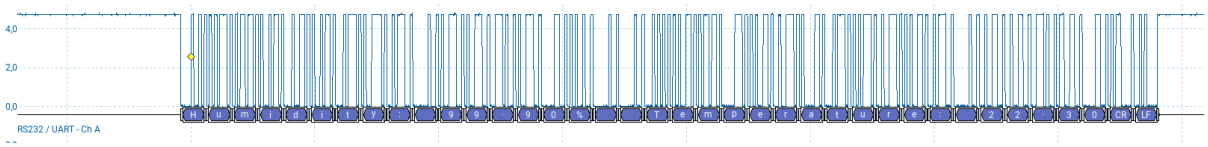


FIGURE 12.10 – Une capture de trame UART à 9600 bauds

12.4.5 D'autres exemple

- Le bus CAN [5] est un bus série half-duplex couramment utilisé dans l'industrie automobile et avionique. La transmission suit le principe de transmission en paire différentielle et possèdent donc deux lignes CAN L (CAN LOW) et CAN H (CAN HIGH). Tous les équipements, appelés noeuds, souhaitant communiquer via le bus sont connectés et peuvent échanger des informations.
- Protocole One-Wire qui utilise un seul câble pour communiquer.
- Protocole MODBus [6]

12.5 Les capteurs de distance

Différentes technologies sont utilisées pour mesurer une distance, cependant elles possèdent leurs avantages et inconvénients.

- Infrarouge
  - Bon marché
  - Assez précis
  - Non-linéaires
  - Sensibilité à la lumière ambiante
  - Dépend du coefficient de réflexion lumineuse de la surface en face du capteur
- Laser

5. Controller Area Network
6. Utilisé dans les automates industriels

- Très précis
- Longue distance
- Prix

- Ultra-sonore
  - Prix
  - Ne dépend pas de la couleur de la surface en face du capteur
  - Précision parfois arbitraire

12.5.1 Les capteurs infrarouges

Ce capteur envoie une tension qui dépend de la distance de l'objet.

FIGURE 12.11 – Un capteur de distance infrarouge

Cependant, cette tension n'est pas proportionnelle à la distance [7]

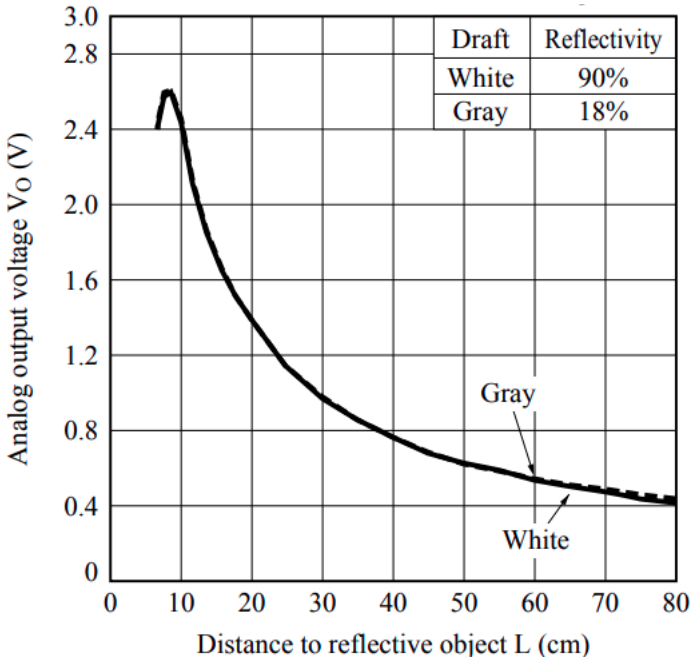


FIGURE 12.12 – La tension de sortie en fonction de la distance

7. Extrait de la datasheet du capteur

12.5.2 Les capteurs ultrasons

Principe

Le principe de ce capteur repose sur le temps de propagation d'une onde sonore dans l'air. En connaissant le temps d'une aller-retour et la vitesse de propagation, on peut déterminer la distance de l'objet.

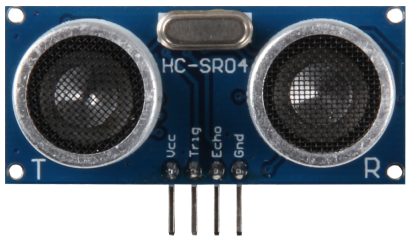


FIGURE 12.13 – Un capteur de distance HCSR-04

Séquence de la mesure

- On envoie une impulsion de 10µs sur la broche PIN TRIGGER du capteur.
- Le capteur envoie une dizaine d'impulsions ultrasonores à 40 kHz
- Les ondes se propagent et rebondissent sur l'obstacle
- Le capteur renvoie le temps de propagation avec la broche PIN ECHO en mettant la sortie à l'état haut durant le temps de l'aller-retour.

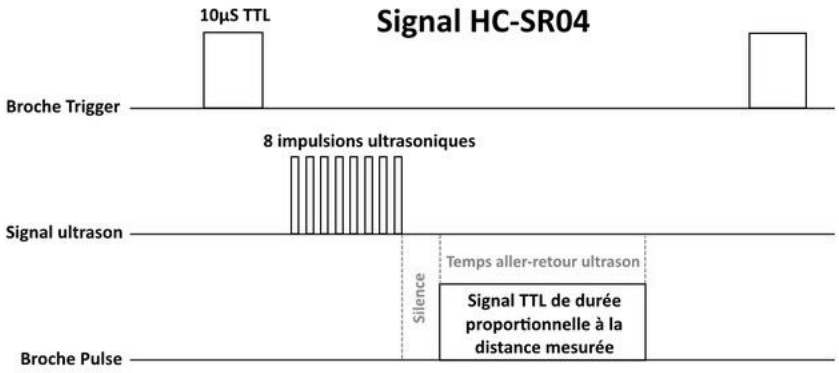


FIGURE 12.14 – L'algorithme de la mesure

Voici le relevé de la broche PIN TRIGGER et PIN ECHO

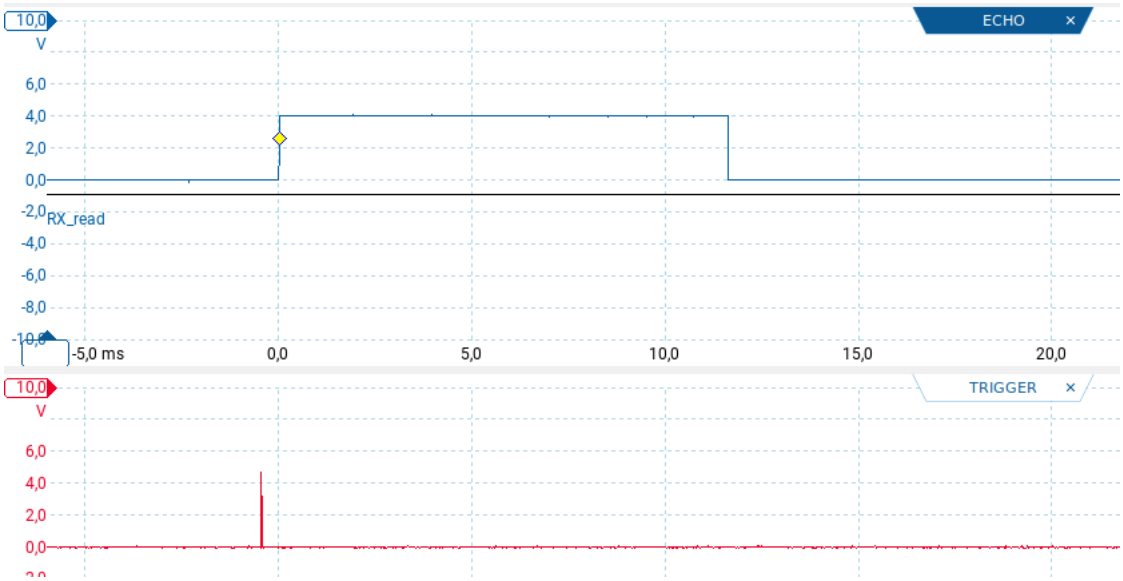


FIGURE 12.15 – Les broches TRIGGER et ECHO

12.6 Les écrans OLED

Les écrans OLED[8] sont des afficheurs graphiques compacts avec une résolution de 128×64 pixels ou 64x32 pixels.
Leur résolution plus élevée que des écrans LCD permet de faire des petits dessins.

Ces écrans utilisent le protocole I2C pour communiquer.

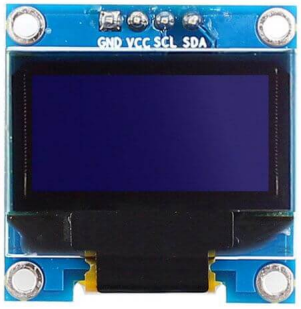


FIGURE 12.16 – Un écran OLED

8. Organic Light Emitting Diode : DEL organique

12.7 Les capteurs de température

12.7.1 Les capteurs numériques

Les capteurs de température DHTXX [9] sont des capteurs de température et d'humidité fonctionnant entre 3.3 et 5V.

Protocole

Le protocole de communication se fait sur un seul câble. [10]

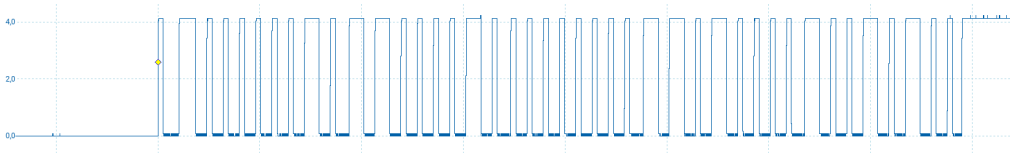


FIGURE 12.17 – Une trame du capteur DHT22

Branchements

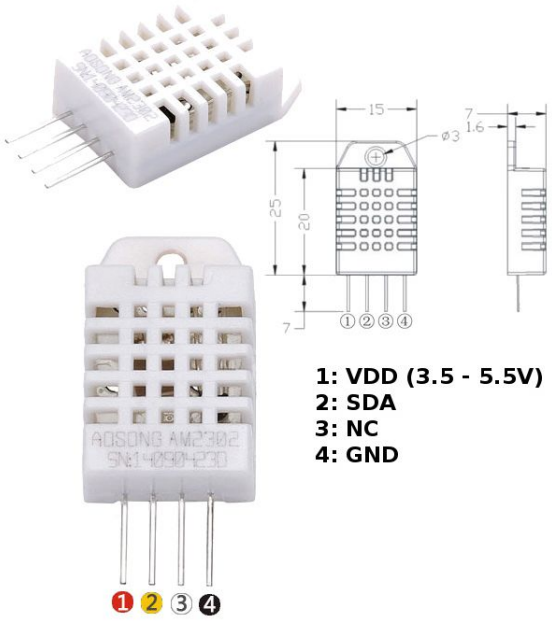


FIGURE 12.18 – Le capteur DHT22

9. DHT11 ou DHT22
10. Protocole de type "One-Wire"

12.7.2 Les capteurs analogiques

Enfin, certains capteurs transmettent leurs données via une tension analogique.
Pour certains capteurs, il suffire le lire une tension pour obtenir indirectement la grandeur physique.

Par exemple, le capteur LM35 sort une tension de 10 mV/°C.

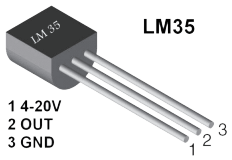


FIGURE 12.19 – Le capteur de température LM35

Si le capteur sort une tension de 210 mV, cela veut dire qu'il fait 21°C.

12.8 Les capteurs PIR

FIGURE 12.20 – Un capteur PIR

12.8.1 Principe

Les capteurs PIR (Passive-Infra-Red) détectent les rayonnements infrarouges émis par un objet.
Puisque tout objet émet un rayonnement infrarouge, le capteur PIR est muni de deux cellules sensibles aux infrarouges qui vont détecter ces rayons infrarouges réfléchit ou émit par l'objet.

Lorsqu'il n'y a pas de mouvement, le niveau d'infrarouge reçu est le même pour les deux cellules. Lors du passage d'un objet, l'émission de ces rayons va être modifiée sur une cellule puis sur l'autre ce qui va permettre de détecter le mouvement.

Le cache blanc, qui couvre et protège généralement le capteur, est une lentille de Fresnel avec plusieurs facettes qui permet de concentrer le rayonnement infrarouge sur les cellules.

12.8.2 Utilisation

Ces capteurs possèdent une broche de sortie qui est mise à l'état HAUT pendent une certaine durée [11] lorsqu'il y a détection d'un mouvement.

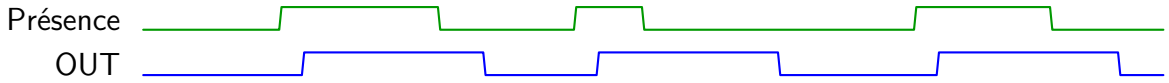


FIGURE 12.21 – Diagramme temporel du capteur

12.8.3 Applications

- Allumage d'une lumière à la détection d'un mouvement
- Activation d'une alarme lors de l'intrusion d'une personne

12.9 Les relais électromagnétiques

12.9.1 Principe

Les relais sont des interrupteurs commandés électriquement. Une bobine alimentée sous faible tension (5 à 24 V) génère un champ électromagnétique qui fait déplacer une membrane qui va ouvrir ou fermer le circuit.

Le relai offre une isolation galvanique entre le circuit de commande et de puissance, c'est à dire qu'il n'y a aucune liaison conductrice entre ces deux circuits.

Un relai est caractérisé par :

- La tension de commande : 5 à 24V
- Le courant de coupure : Ex 30A
- La tension maximale dans le circuit de puissance : Ex 230V
- Sa position au repos [12] : Normalement Ouvert (NO) ou Normalement Fermé (NF)
- Sa durée de vie : les relais sont garantis pour un nombre de commutation, par exemple 1 million.

11. Cette durée est réglable avec le potentiomètre sur le capteur
12. Position en absence de tension sur la commande

12.9.2 Symbole

Le relai se symbolise de la façon suivante :

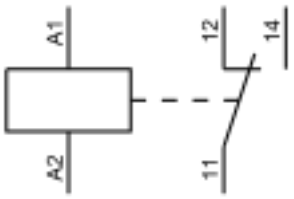


FIGURE 12.22 – Le symbole du relai

On distingue clairement la partie de commande (rectangle) et le circuit qui s'ouvre ou se ferme pour laisser passer le courant.

12.9.3 La diode de roue libre

La bobine de commande du relai nécessite un certain courant [13] qu'une broche de microcontrôleur ne peut pas fournir. Pour cela on utilise un composant qui fera l'interface entre le microcontrôleur et le relai : le transistor.

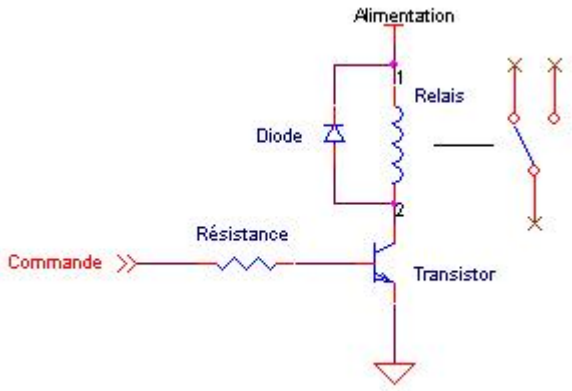


FIGURE 12.23 – Utilisation d'un relai

Cependant, lors de la fermeture du relai [14], le courant est brutalement coupé, or, les bobines s'opposent aux variations de courant.
Cela engendre une surtension qui va se répercuter sur le transistor.

Cette surtension vaut :

$$U = L \cdot \frac{dI}{dt}$$

13. De l'ordre de la centaine de mA
14. Mise à 0 de la broche de commande de transistor

Avec :

- U la tension en Volt aux bornes de la bobine
- L la valeur en Henry de l'inductance [15]
- I la variation de courant en Ampère dans la bobine

On met donc une diode en parallèle de la bobine pour que l'énergie accumulée dans la bobine passe dans la diode.
Cette diode est appelé **diode de roue libre** .

En exemple, une simulation sans diode est faite avec LTSpice :

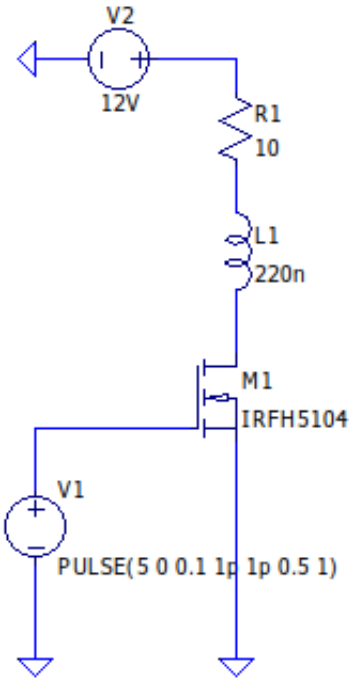


FIGURE 12.24 – Une simulation LTSpice

On active le transistor pendant 100 ms puis on le désactive et on observe la tension aux bornes du transistor.

15. Bobine

FIGURE 12.25 – Une surtension sur le transistor

On constate un pic à 24V, c'est à dire le double de l'alimentation 12V.

Maintenant, faisons la même simulation avec une diode de roue libre.
Pour ces diodes, on privilégie des diodes Schottky à commutation rapide.

FIGURE 12.26 – Une surtension plus faible

La surtension ne vaut plus que quelques dizaines de mV.

12.9.4 En pratique

Pour utiliser des relais avec des microcontrôleurs, on utilise le plus souvent des relais qui intègrent une interface de contrôle.

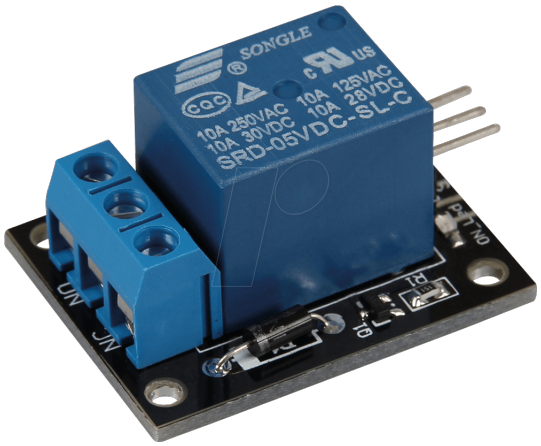


FIGURE 12.27 – Un relai avec une interface de contrôle

Il suffit généralement d'alimenter le relai en 5V constant et une broche active le relai si elle passe au niveau logique HAUT.

Section 13

Mise en pratique

Les codes suivants seront utilisés avec le serveur Web mis en place avec l'ESP12.
Cela permettra de réaliser une interface plus élaborée avec des capteurs et actionneurs.

13.1 Utilisation du DHT11

13.1.1 Objectif

L'intégration du code permettant de lire la température permettra d'obtenir l'interface suivante, à savoir un graphique pour visualiser la température et l'humidité en temps réel.

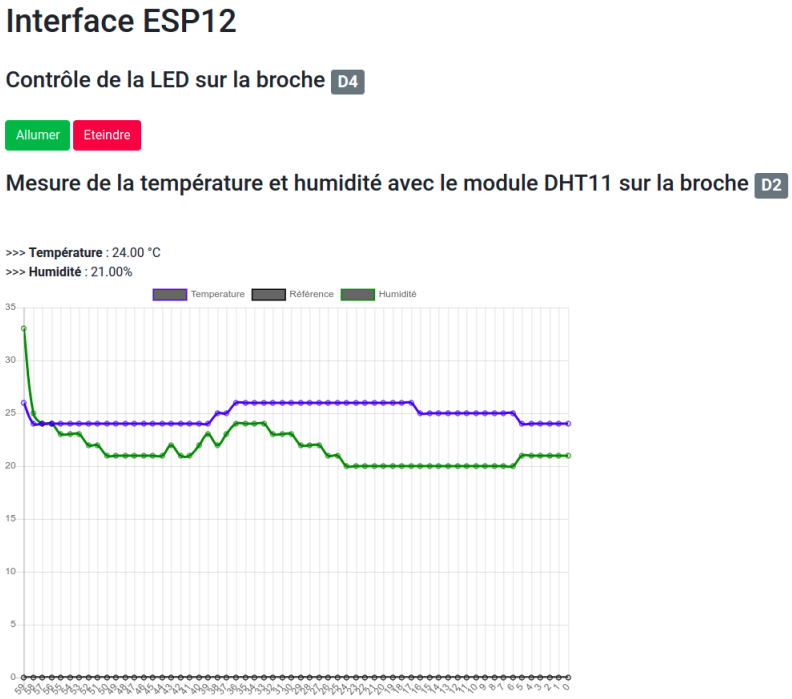


FIGURE 13.1 – Le rendu de l'interface

13.1.2 Branchements

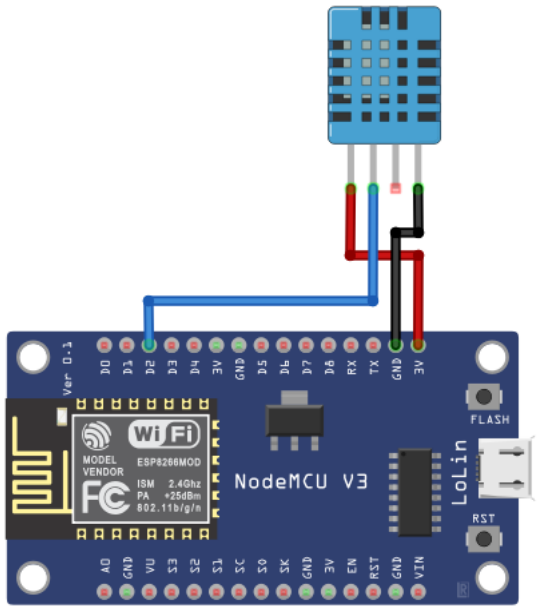


FIGURE 13.2 – Le branchement du module DHT

13.1.3 Programme de test

Avant de tester le code complet du DHT11/22 avec le serveur Web , on va essayer le module DHT sans le serveur. Pour cela, on va téléverser le programme suivant [1] :

```
#include "DHT.h"
#define DHTPIN D2     //Broche du capteur

#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22
//#define DHTTYPE DHT21   // DHT 21

DHT dht(DHTPIN, DHTTYPE);

void setup() {

  Serial.begin(115200);
  dht.begin();          //Initialisation du capteur
}

void loop() {
```

1. Fichier FILE temperature_humidite.ino

```
    float h = dht.readHumidity();        //Récupère la température
    float t = dht.readTemperature();     //récupère l'humidité

    Serial.print("Humidite: ");
    Serial.print(h);
    Serial.print("\%  Temperature: ");
    Serial.print(t);
    Serial.println("C ");

    delay(2000);

}
```

Programme DHT11/22 minimaliste

Si vous obtenez le résultat suivant en lançant la console série, c'est que le capteur est fonctionnel.

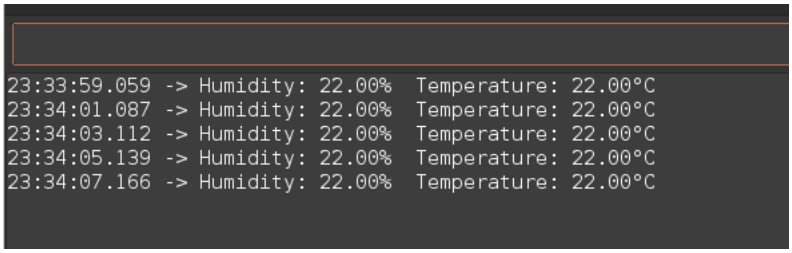


FIGURE 13.3 – Le capteur DHT fonctionnel

Il ne vous reste plus qu'à lancer le programme FILE Serveur_Web_DHT11_Graphe.ino.

13.1.4 Explications

Pour gérer les températures et les valeurs d'humidité dans le temps, un tableau 'circulaire' est utilisé dans le programme.
Il consiste à remplir au fur et à mesure un tableau et quand celui-ci est plein, on décale les valeurs pour ajouter la dernière.

Prenons un exemple avec un tableau de 5 éléments auquel on cherche à ajouter le cycle suivant : 21,22,21,22,23,24,25,26

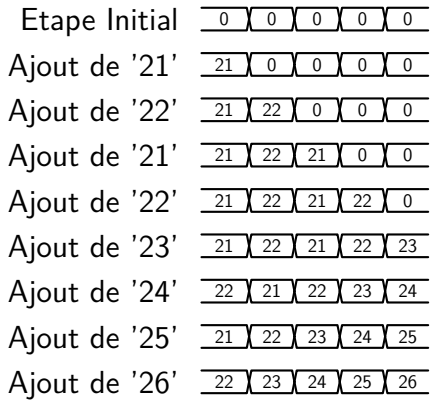


FIGURE 13.4 – Les tableaux 'circulaires'

Ce mouvement cyclique est géré par la fonction FUNC updateRings dans le fichier LIB circularRings.h
La fonction principale pour gérer le graphique est la fonction FUNC getString :

```
temperature = dht.readTemperature();
humidity  = dht.readHumidity();
```

Lecture de la température et de l'humidité

Une fois ces deux données lues, on actualise les tableaux circulaires contenant les températures, les valeurs d'humidité et les références du graphique [2].

```
updateRings(&current_index, NB_DATA_TEMP, temperature, humidity); //Mise à
jour des tableaux
```

Actualisation des tableaux

On créer ensuite les chaines de caractères pour générer le graphique :

```
String dataTemperatures = concatenateArray(temperatures, current_index);
String dataHumidities = concatenateArray(humidities, current_index);
String dataReferences = concatenateArray(references, current_index); //
Tableau contenant toutes les valeurs à 0, pour afficher la référence sur le
graphique

String dataTime ="[";
for (int i=current_index;i>0;i--) {
  dataTime += "'"+String(i)+"',";
}
dataTime += "]";
```

Génération du graphique

Une fois toutes les données, on génère la page dans son intégralité :

2. Tableau contenant que des '0'

```
const String fullPageContent = "<html>\
<head>\
  <title>Serveur Web CREPP</title>\
  <meta charset=\"utf-8\"/> \
  <meta http-equiv=\"refresh\" content=\""+String(REFRESH_PAGE_DELAY)+"\">\
  <link rel=\"stylesheet\" href=\"https://stackpath.bootstrapcdn.com/bootstrap
   /4.3.1/css/bootstrap.min.css\" integrity=\"sha384-ggOyR0iXCbMQv3Xipma34MD+dH
   /1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T\" crossorigin=\"anonymous\">\
  <script src=\"https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.
   js\"></script>  \
  </head>\
<body style=\"margin-left:5\%;\">\
  <h1>Interface ESP12</h1><br>\
  <h3>Contrôle de la LED sur la broche <span class=\"badge badge-secondary\">D4</
   span></h3><br>\
    <a href=\"/?LED=ON\"><button class=\"btn btn-success\">Allumer</button></a>\
    <a href=\"/?LED=OFF\"><button class=\"btn btn-danger\">Eteindre</button></a><
   br><br>\
  <h3>Mesure de la température et humidité avec le module DHT11 sur la broche <
   span class=\"badge badge-secondary\">D2</span></h3><br>\
    <br>\
    >>> <b>Temperature</b> : "+String(temperature)+" C<br>\
    >>> <b>Humidite</b> : "+String(humidity)+"%\
    <div style='max-width:40\%;'><canvas id=\"myChart\" width=\"600\" height
   =\"450\"></canvas></div> \
      <script> \
var ctx = document.getElementById('myChart'); \
var temperatures = "+dataTemperatures+";\
var references = "+dataReferences+";\
var humidities = "+dataHumidities+";\
var time = "+dataTime+";\
var myChart = new Chart(ctx, {type: 'line', data: { labels: time, datasets: [{
   label: 'Temperature', data: temperatures, borderColor: 'blue',backgroundColor
   : '', fill: 0}, { label: 'Référence', data: references, borderColor: 'black',
   backgroundColor: '', fill: 1}, { label: 'Humidité', data: humidities,
   borderColor: 'green',backgroundColor: '', fill: 0}]}})</script>\
</body>\
</html>";
```

Génération de la page

13.2 Utilisation du HC-SR04

13.2.1 Branchements

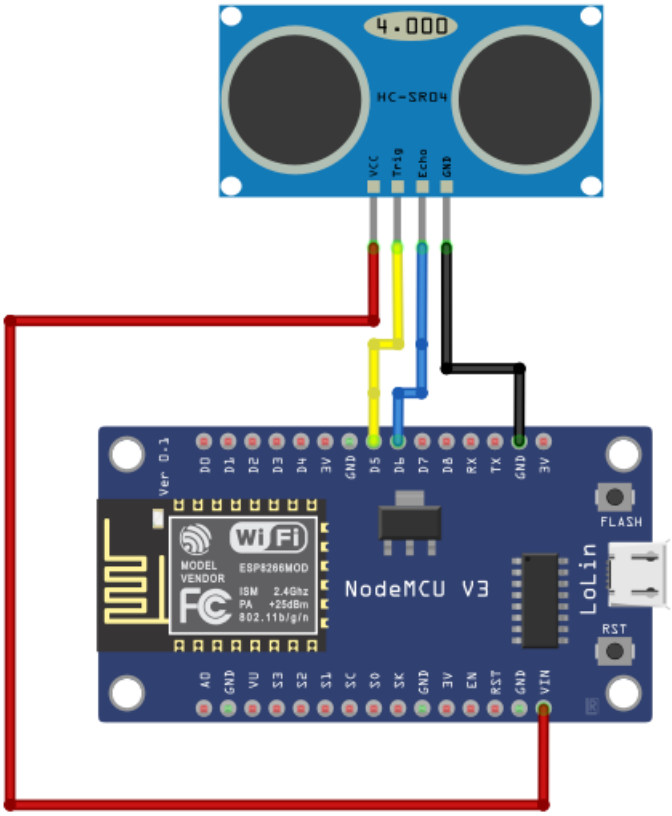


FIGURE 13.5 – Branchement du capteur

13.2.2 Code complet

```
#define TRIGGER_PIN D5  //Broche Trigger
#define ECHO_PIN D6     //Broche Echo

#define SOUND_VELOCITY 0.034 //Vitesse en cm/us

float distance = 0.0;

void setup() {

  Serial.begin(9600);         //Communication à 9600 bauds
  pinMode(TRIGGER_PIN, OUTPUT); //Broche Trigger en sortie
  pinMode(ECHO_PIN, INPUT);     //Broche Echo en entrée
```

```
  digitalWrite(TRIGGER_PIN, LOW);

}//End setup

void loop() {

  digitalWrite(TRIGGER_PIN, HIGH); //Envoie une impulsion de 10us
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  int duration = pulseIn(ECHO_PIN, HIGH); //Récupère le temps à l'état Haut de la
    broche ECHO

  distance = duration * SOUND_VELOCITY/2; //Distance en cm, v=d/t

  Serial.print("Distance (cm) = ");
  Serial.println(distance);

  delay(1000);
}
```

Code complet pour le capteur HCSR-04

13.3 Utilisation d'un capteur PIR

13.3.1 Branchements

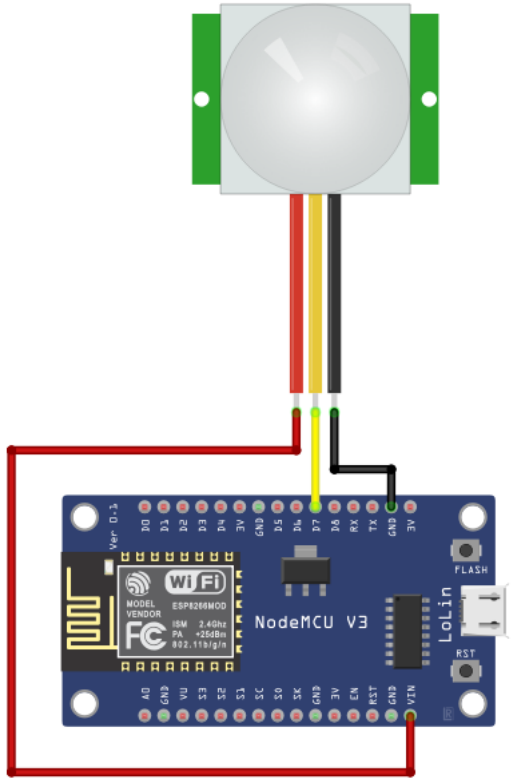


FIGURE 13.6 – Branchement du capteur PIR

13.3.2 Code complet

```
#define LED D2    //Broche de la LED
#define OUT D7     //Broche du capteur PIR

int status = LOW;     //Statut du mouvement : LOW : pas de mouvement détecté
bool outValue = 0;    //Valeur du capteur
long beginTime = 0;   //instant du déclenchement lors de la ldétection d'un
   mouvement

void setup()
{

  pinMode(LED, OUTPUT); //LED en sortie
  pinMode(OUT, INPUT);  //Broche du capteur en entrée
```

```
  Serial.begin(9600);   //Vitesse de communication à 9600 bauds

}//End setup

void loop(){

  outValue = digitalRead(OUT);          //Lire létat du capteur

  if (outValue == HIGH)                 //Détection d'un mouvement
  {
    digitalWrite(LED, HIGH);            //Allume la LED

    if (status == LOW)
    {
      Serial.println(">>> Mouvement ");
      status = HIGH;                    //Mise à jour du statut du capteur
      beginTime = millis();             //Sauvegarde du temps

    }//End if

    }
    else                                //Aucun mouvement
    {

      digitalWrite(LED, LOW);           //Eteint la LED

      if(status == HIGH)                //Fin de détection
      {
        Serial.print(">>> Fin du mouvement");
        status = LOW;                                 //Mise à jour du statut du
   capteur
        unsigned long duree = millis() - beginTime; //Calcul de la durée du
   mouvement
        Serial.print(">>> Duree = ");
        Serial.print(duree);
        Serial.println(" ms");

      }//End if
    }//End else
}//End loop
```

Code complet

13.4 Utilisation d'un écran OLED

13.4.1 Récupération de l'adresse I2C

Pour tous les périphériques I2C, il convient de connaître l'adresse du périphérique. Or parfois on ne s'en rappel plus. Il existe un code qui permet de scanner toutes les adresses possibles et d'afficher l'adresse du composant qui est relié au bus I2C.

Voici le code, disponible dans les exemples de la classe Wire [3] :

```
void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.print(address,HEX);
      Serial.println("  !");
      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
```

3. Classe qui gère le protocole I2C

```
  }
  else {
    Serial.println("done\n");
  }
  delay(5000);

}
```

Un scanner I2C

13.4.2 Code complet

```
#include "SSD1306Ascii.h"
#include "SSD1306AsciiAvrI2c.h"

#define I2C_ADDRESS 0x3C

SSD1306AsciiAvrI2c oled;

void setup() {

  //Init size
  oled.begin(&Adafruit128x64, I2C_ADDRESS);
  oled.setFont(Adafruit5x7);
  oled.clear();
  oled.set2X();


  oled.println("CREPP");
  oled.set1X();
  oled.println("Club de");
  oled.println("Robotique et");
  oled.println("d'Electronique");
  oled.println("Programmable");

}
void loop() {

}
```

Code complet

13.4.3 Branchements

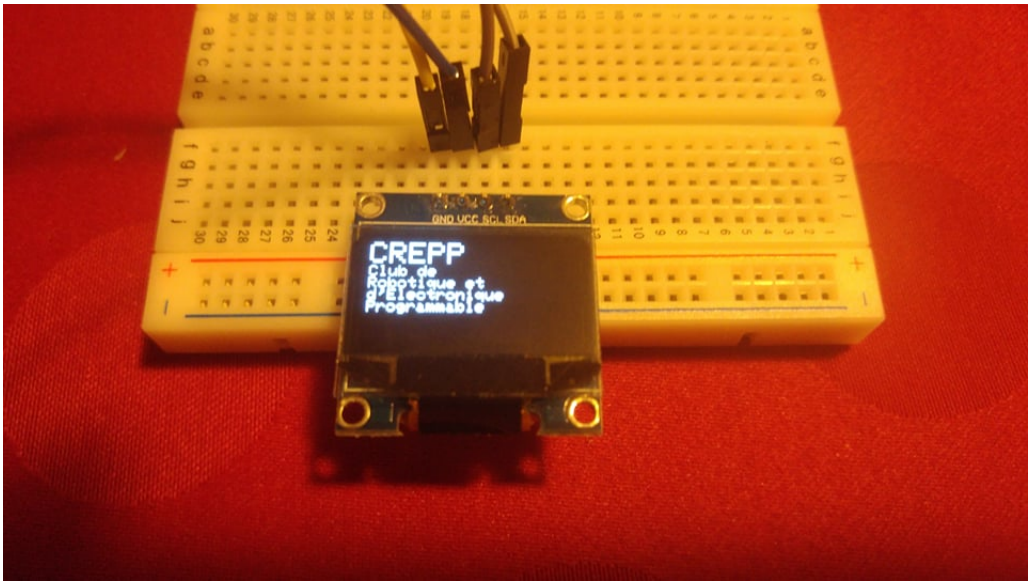


FIGURE 13.7 – L'écran OLED fonctionnel

Section 14

Les broches d'interruptions

Dans certains cas, il est souhaitable de récupérer la valeur d'une broche à tout moment du programme, même quand celui ci est occupé dans une tâche et même dans une fonction de temporisation [1].

Pour remédier à ce problème, on peut utiliser les **broches d'interruption** qui permettent de récupérer la main sur l'ensemble du programme lorsque'un évènement survient sur une broche.

Concrètement, lorsque un évènement e survient sur la broche b, la fonction f est appelée, quelque soit l'état du programme principal.

Prenons le cas d'un bouton qui doit changer l'état d'une LED à n'importe quel moment du programme.

```
  int ledPin = 13;    //Led interne
  int BOUTON = 2;  //Bouton relié à la broche 2 avec une résistance de charge

  volatile int state = LOW;  //Etat courant de la LED

  void setup() {

    Serial.begin(9600);//Vitesse de communication à 9600 bit/s

    pinMode(ledPin, OUTPUT);                //Led mise en sortie
    pinMode(BOUTON, INPUT_PULLUP);    //Bouton mis en entrée

    attachInterrupt(digitalPinToInterrupt(BOUTON), onEvent, CHANGE);  //Appel
de la fonction onEvent à chaque changement de front du bouton
    Serial.println("Init");
  }
```

1. Voir delay(), delayMicroseconds()

```
void loop() {

 delay(5000); //Pause du programme principal

}

void onEvent() {

  state = !state; //Inverse l'état de la LED

  if(state){
    Serial.println("ON");
  }else{
    Serial.println("OFF");
  }
   digitalWrite(ledPin, state); //Met à jour l'état de la LED
}
```

Code d'exemple

Ici, quelque soit l'action effectuée dans la fonction loop, dès qu'un front montant est détecté sur la broche BOUTON (2), la fonction onEvent() sera exécutée et changera l'état de la LED à chaque front

14.0.1 Mode d'interruption

Il existe différents modes pour les broches :

- RISING : front montant
- FALLING : Front descendant
- CHANGE : Front montant et descendant

14.0.2 Chronogrammes d'interruption

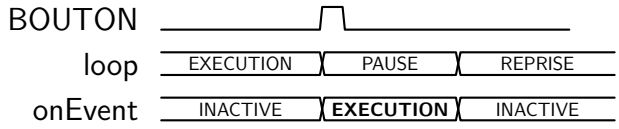


FIGURE 14.1 – Exemple avec mode RISING

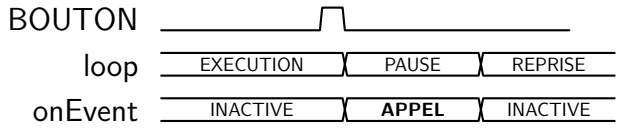


FIGURE 14.2 – Exemple avec mode FALLING

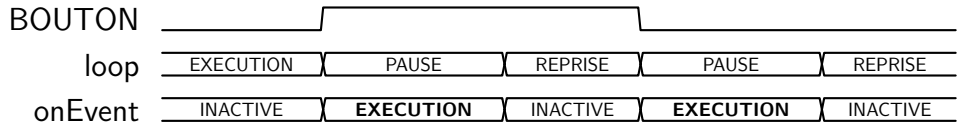


FIGURE 14.3 – Exemple avec mode CHANGE

Par exemple, on souhaite réagir prioritairement à un capteur PIR quel que soit l'état du programme.

14.0.3 Exemple avec les capteur PIR

Les capteurs PIR (Passive-Infra-Red) détectent les rayonnements infrarouges émis par un objet.
Puisque tout objet émet un rayonnement infrarouge, le capteur PIR est muni de deux cellules sensibles aux infrarouges qui vont détecter ces rayons infrarouges réfléchit ou émit par l'objet.

Lorsqu'il n'y a pas de mouvement, le niveau d'infrarouge reçu est le même pour les deux cellules. Lors du passage d'un objet, l'émission de ces rayons va être modifiée sur une cellule puis sur l'autre ce qui va permettre de détecter le mouvement.

Le cache blanc, qui couvre et protège généralement le capteur, est une lentille de Fresnel avec plusieurs facettes qui permet de concentrer le rayonnement infrarouge sur les cellules.

14.0.4 Utilisation

Ces capteurs possèdent une broche de sortie qui est mise à l'état HAUT pendent une certaine durée[2] lorsqu'il y a détection d'un mouvement.

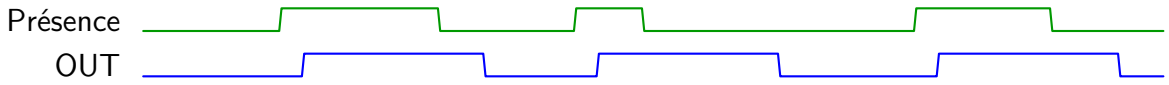


FIGURE 14.4 – Diagramme temporel du capteur

14.0.5 Code sans interruption

Voici un code d'exemple pour réagir dès qu'une présence est détectée.

```
#define LED 13    //Broche de la LED
#define OUT 2     //Broche du capteur PIR

void setup()
```

2. Cette durée est réglable avec le potentiomètre sur le capteur

```
{

  pinMode(LED, OUTPUT); //LED en sortie
  pinMode(OUT, INPUT);  //Broche du capteur en entrée
  Serial.begin(9600);   //Vitesse de communication à 9600 bauds

}//End setup

void loop(){

  outValue = digitalRead(OUT);          //Lire létat du capteur

    if (outValue == HIGH)                 //Détection d'un mouvement
    {
        Serial.println("Detection");
    }//End else
}//End loop
```

Code sans interruption

Maintenant, supposons que la carte exécute un programme qui prend plusieurs secondes. Comment faire ?

On peut utiliser une interruption externe

```
    #define LED 13    //Broche de la LED
    #define OUT 2     //Broche du capteur PIR

    void setup() {

      Serial.begin(9600);//Vitesse de communication à 9600 bit/s

      pinMode(LED, OUTPUT); //LED en sortie
      pinMode(OUT, INPUT);  //Broche du capteur en entrée

      attachInterrupt(digitalPinToInterrupt(OUT), presence, RISING);  //Appel de
   la fonction presence à chaque changement de front du bouton
      Serial.println("Init");
    }

    void loop() {

     delay(5000); //Pause du programme principal

    }
```

```
void presence() {

  Serial.println("Presence !");
}
```

Code avec interruption

Septième partie

Les modules de communication

Les modules de communication sans fil

Section 15

Principes et théorie

15.1 Objectifs

Ce chapitre a pour but de faire un petit tour d'horizon des différents modules de communication et les technologies associées.

15.2 Les différents modules

Il existe une multitude de modules :

- Modules Infrarouge
- Modules Radio basses fréquence (433MHz)
- Modules Radio hautes fréquences (Wifi, Bluetooth) à 2.4 GHz

Les deux derniers modules se déclinent en une multitude de modules :

- Modules Lora
- Modules Xbee
- Modules HC-04 ou HC-05 [1]
- Modules Crius

15.3 Les modules Bluetooth

La plupart des modules Bluetooth communiquent en liaison série (broche RX et TX) et se configurent avec les commandes AT.

1. Les modules HC-05 sont configurables en mode maître ou esclave et les modules HC-06 en mode esclave uniquement.

15.3.1 Configuration

Il s'agit d'un jeu d'instruction pour gérer les paramètres des modules comme l'identifiant, le nom, la vitesse de communication, etc.

Ces commandes étaient utilisée à l'origine pour les modem Hayes Smartmodem 300 et sont donc également appelées **commandes Hayes** .

Chaque commande est envoyée sous la forme d'une ligne de texte encodée en ASCII, débutant par le mot **AT** et se terminant par le caractère ŝeul (code ASCII 13).
Le module retourne une réponse sous la forme d'une ou plusieurs lignes selon la commande envoyée, chaque ligne se terminant par les caractères ŝuivi de
(codes ASCII 13 et 10).

15.3.2 Branchements

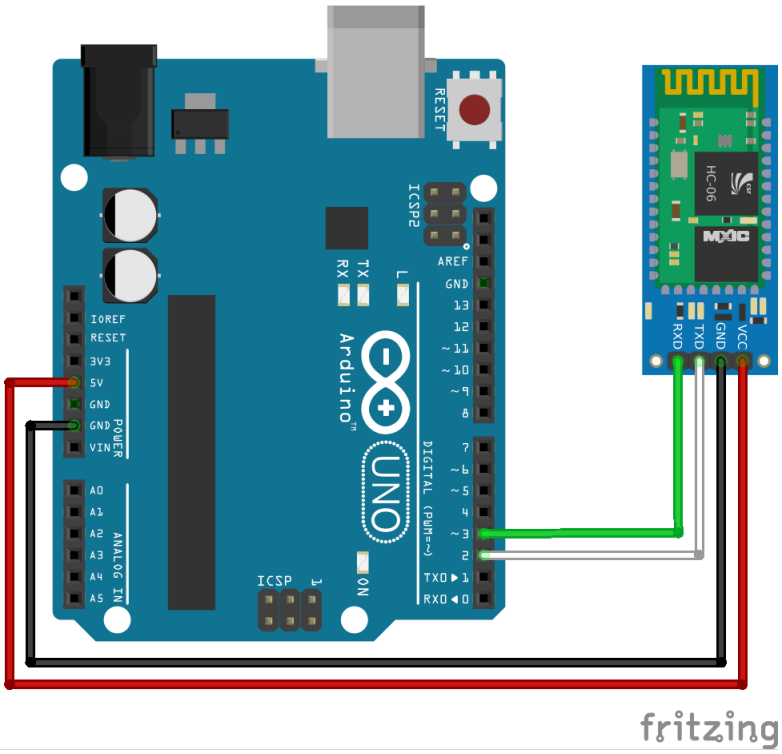


FIGURE 15.1 – Branchements du module Bluetooth

Pour l'exemple, nous nous baserons sur un module HC-06 ou HC-05.

il faut relier le « +5V » du module au 5 Volts de la carte Arduino et la masse du module à celle de la carte.
Ensuite, nous allons relier la broche TX du module à la broche 12 de la carte et la broche RX à la broche 10.

15.3.3 Code Arduino

Point-clé

Afin de lire les données du module, nous allons ”émuler” une voie série, en l'occurrence les broches 3 et 2.
C'est-à-dire que nous allons déclarer que ces broches recevront et enverront des données.

Pour cela, il faut utiliser la bibliothèque **SoftwareSerial** . Dans le logiciel Arduino, allez dans **croquis** puis **inclure une bibliothèque** et sélectionnez **SotfwareSerial** .

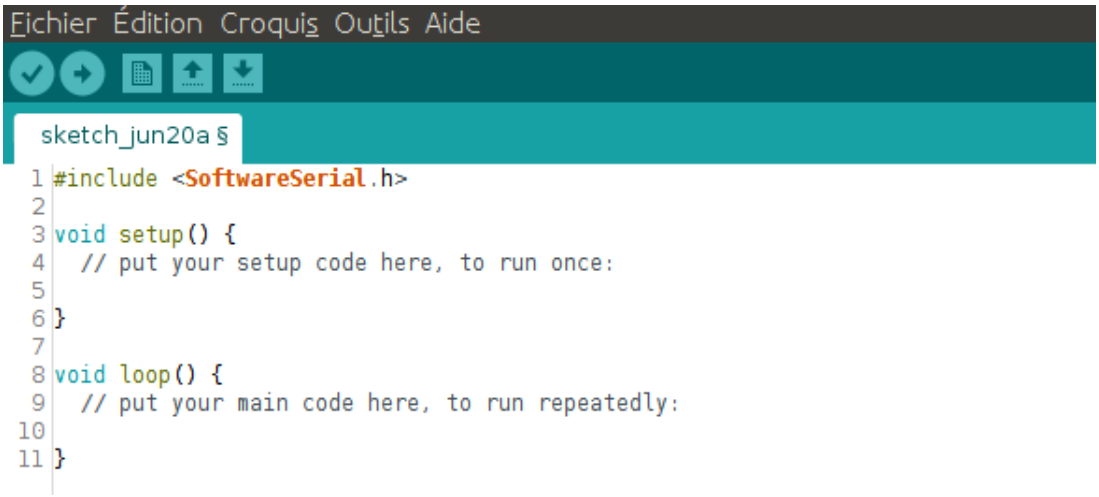


FIGURE 15.2 – Inclusion de la bibliothèque SoftwareSerial

Maintenant, nous allons définir les broches du module : :

```
const int RX = 3; //RX du module
const int TX = 2;
```

Définitions des broches RX et TX

Après ceci, il faut déclarer un objet **SoftwareSerial** qui prendra en argument respectivement les broches TX et Rx, un peu comme lors de la déclaration d'un écran LCD (« LiquidCrystal lcd (RS,E,D4,D5,D6,D7) ; »).

On obtient donc :

```
#include <SoftwareSerial.h>

const int RX = 3; //RX du module
const int TX = 2;

SoftwareSerial device(RX,TX);
```

Définition de l'objet SoftwareSerial

Bien entendu, ”device” peut être remplacé par ce que vous voulez.
Ensuite, on déclare que la communication carte-module peut débuter avec ”device.begin(9600) ;”
Où 115200 correspond à la vitesse de transmission en bauds (comme ”Serial.begin(9600) ;”) ;

```
SoftwareSerial device(RX,TX);
void setup() {
    device.begin(9600);
}
```

Vitesse de communication

Remarque importante

Par la suite, en cas d'erreurs de transmission Bluetooth (pas de données...), il conviendra de vérifier le branchement des broches RX et TX (essayer de les intervertir) et d'éventuellement changer la vitesse de communication car certains modules communiquent à 115200 bauds !

Pour lire les données du module, ce sont les mêmes fonctions que pour le port série :
En effet :

Pour le port série :

- Serial,begin(115200) ;
- Serial.available() ;
- Serial.read() ;
- Serial.print() ;
- Serial.println() ;

Pour le module Bluetooth :

- crius.begin(115200) ;
- device.available() ;
- device.read() ;
- device.print() ;
- device.println() ;

Tant que des données (caractères) sont disponibles, nous allons les assembler en une chaîne de caractère (concaténation).
Ensuite, avec un **if** , nous allons voir si cette chaîne en question correspond par exemple à "a".

Il faut donc définir un caractère x et une chaîne de caractère.
Donc dans le programme Arduino, avant la fonction **setup** , on rajoute :

```
#include <SoftwareSerial.h>

const int RX = 3; //RX du module
const int TX = 2;

char c;
String message;
```

Définition des structures du message

La boucle while va permettre de lire les données : Dans la boucle **loop** : on écrit :

```
void loop() {

   while() {

   }//Fin while

}//Fin void loop
```

Boucle de lecture partielle

Maintenant que la boucle va attendre des données, il suffit de les lire et de les transformer en chaîne de caractère.
Pour cela :

- on lit le premier caractère c
- on définit que la chaîne message = message + c

On obtient :

```
void loop() {

   while(device.available()>0) {

       c = device.read();
       message = message + c;
   }//Fin while

}//Fin void loop
```

Boucle de lecture complète

La structure conditionnelle est très simple :
Après la boucle « while », mettez :

```
void loop() {
```

```
    while(device.available()>0) {

        c = device.read();
        message = message + c;
    }//Fin while

    if(message=="c") {

        Serial.println("C);
    }//Fin if message=="c"

}//Fin void loop
```

Structure conditionnelle

15.4 Les modules radio

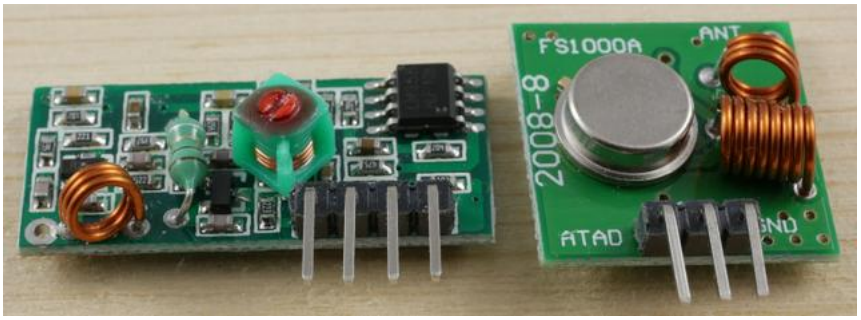


FIGURE 15.3 – Un module receveur et émetteur

15.4.1 Choix de l'antenne

La plupart du temps, on utilise une antenne **quart d'onde** ou **demi-onde** .
La fréquence vaut :

$$\lambda = \frac{c}{\nu}$$

avec λ la longueur d'onde en mètre, c la vitesse de l'onde en $m \cdot s^{-1}$ et ν la fréquence de l'onde en Hz

D'ou :

$$\lambda = \frac{3 \cdot 10^8}{433 \cdot 10^6} = 0.69\ m$$

Or on prend une antenne valant le quart du résultat, c'est à dire 17 cm.

15.4.2 Branchements

Module émetteur

- IN VCC sur le OUT +5V de l'Arduino
- IN GND sur le OUT GND de l'Arduino
- IN Data sur la broche OUT D12 de l'Arduino

Module receveur

- IN VCC sur le OUT +5V de l'Arduino
- IN GND sur le OUT GND de l'Arduino
- IN Data sur la broche OUT D11 de l'Arduino

15.5 Les modules Xbee

Ce module communique via une liaison série.
L'un des inconvénient de ce module est l'espacement des broches de 2mm et non de 2.54 mm. Il faut donc utiliser un shield adapté.

La fréquence de communication est cette fois ci de 2.4GHz.

Section 16

Introduction au projet MySensors

16.1 Présentation

Ce chapitre a pour but d'introduire la mise en place d'une passerelle et d'une sonde MySensors dans un projet de domotique

16.1.1 Organigramme

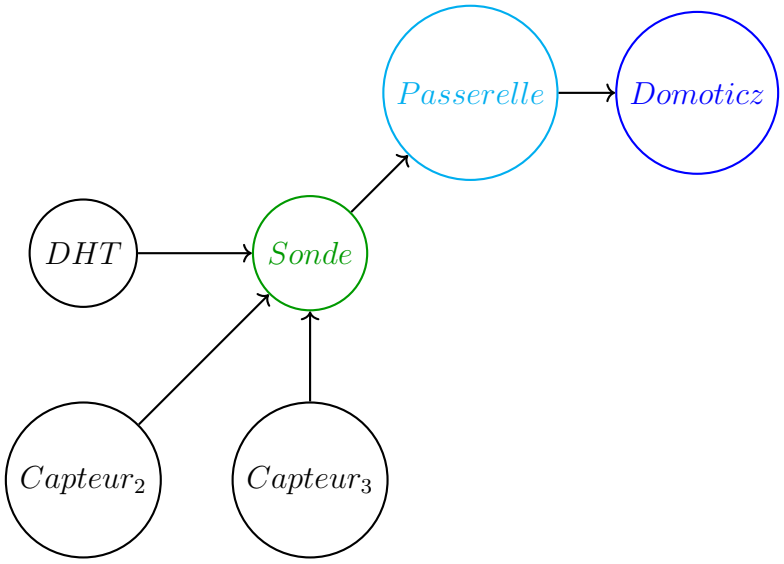


FIGURE 16.1 – Les différents composants du projet

16.1.2 Principe

Les capteurs vont être analysés par la sonde MySensors.
Cette dernière enverra à distance les informations vers la passerelle qui se chargera d'envoyer les informations au serveur Domoticz via une liaison USB.

Une sonde représente un endroit physique, un lieu de mesure.
Si vous souhaitez par la suite faire d'autres relevés dans un endroit différent, il suffira d'ajouter une sonde et de garder la passerelle.
Chaque sonde est caractérisée par un identifiant de noeud (NODE_ID) et chaque capteur possède un identifiant enfant sur la sonde qui lui est rattachée (CHILD_ID)

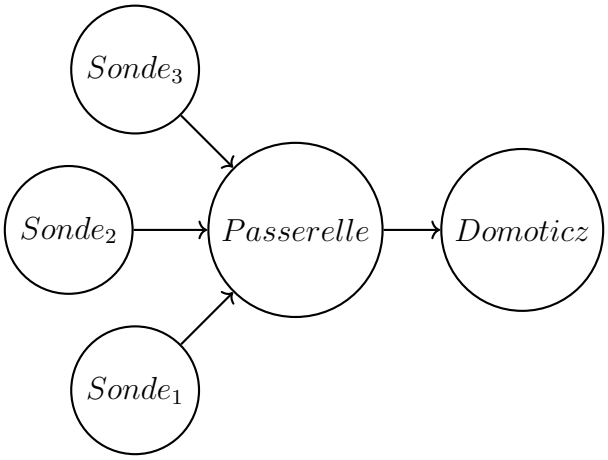
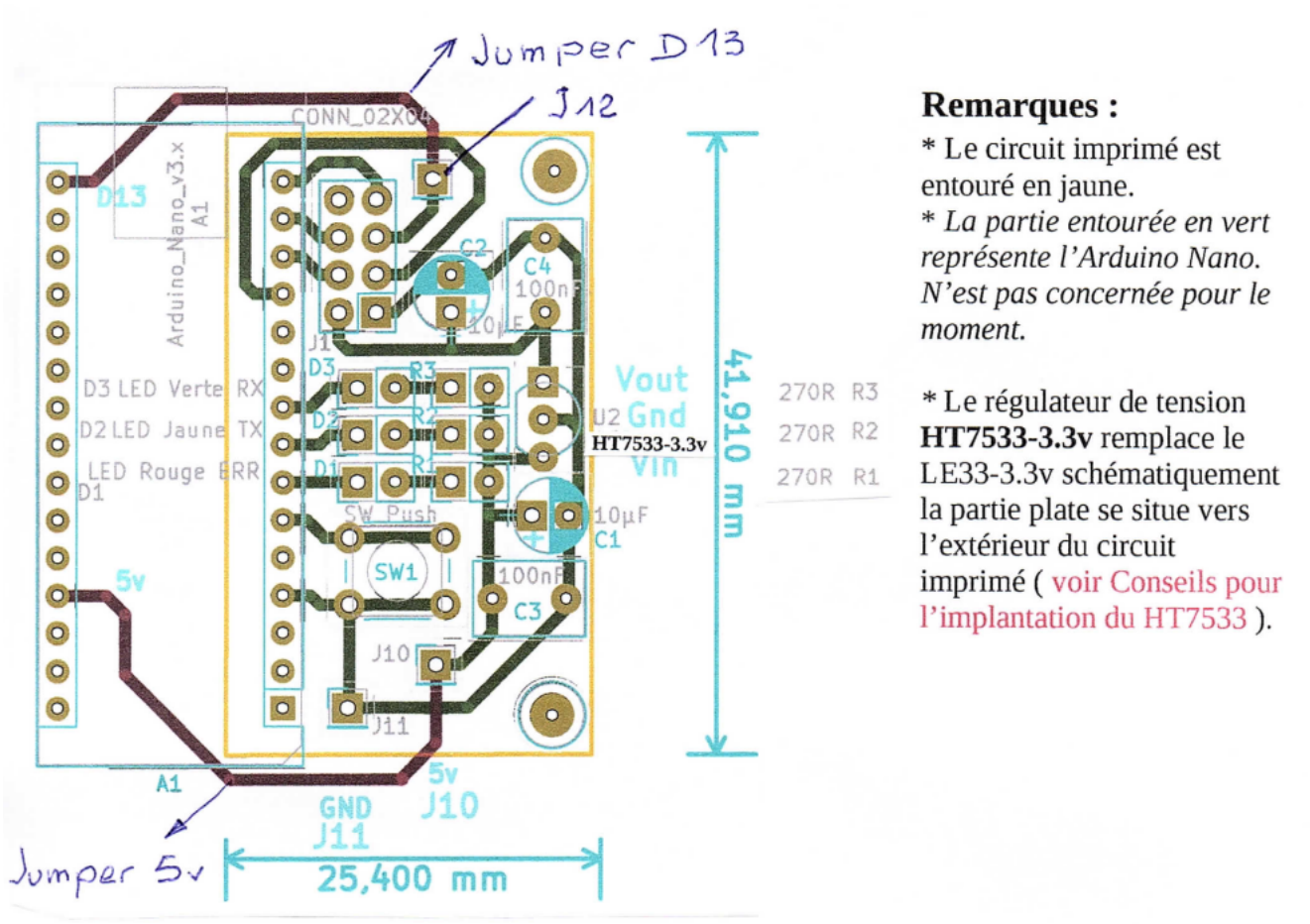


FIGURE 16.2 – Une extension possible

16.2 Présentation de la passerelle



Remarques :

* Le circuit imprimé est entouré en jaune.
* *La partie entourée en vert représente l'Arduino Nano. N'est pas concernée pour le moment.*

* Le régulateur de tension **HT7533-3.3v** remplace le LE33-3.3v schématiquement la partie plate se situe vers l'extérieur du circuit imprimé (voir Conseils pour l'implantation du HT7533).

FIGURE 16.3 – Circuit imprimé vu de dessus, coté composants

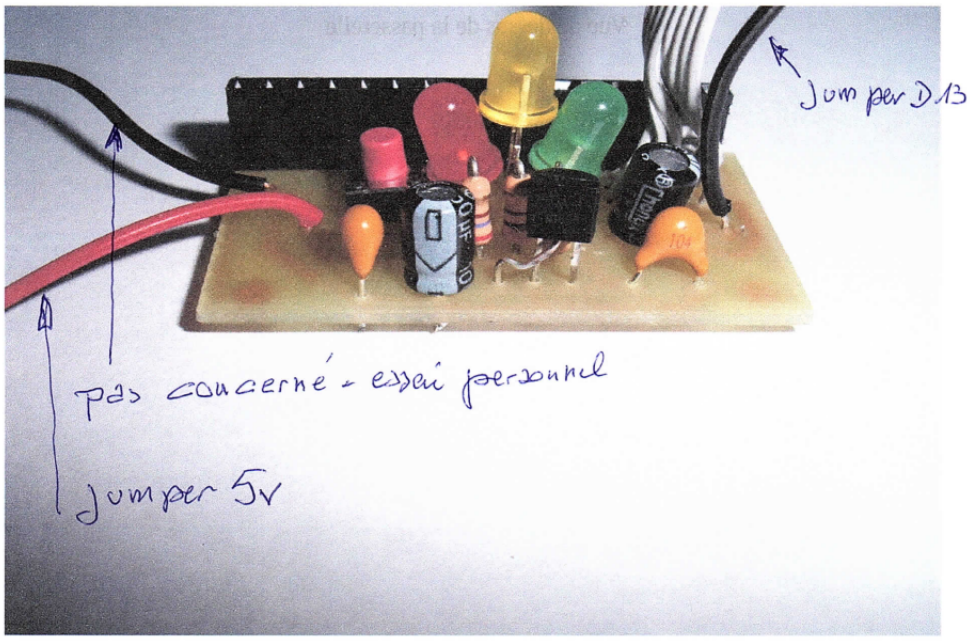


FIGURE 16.4 – Vue de coté

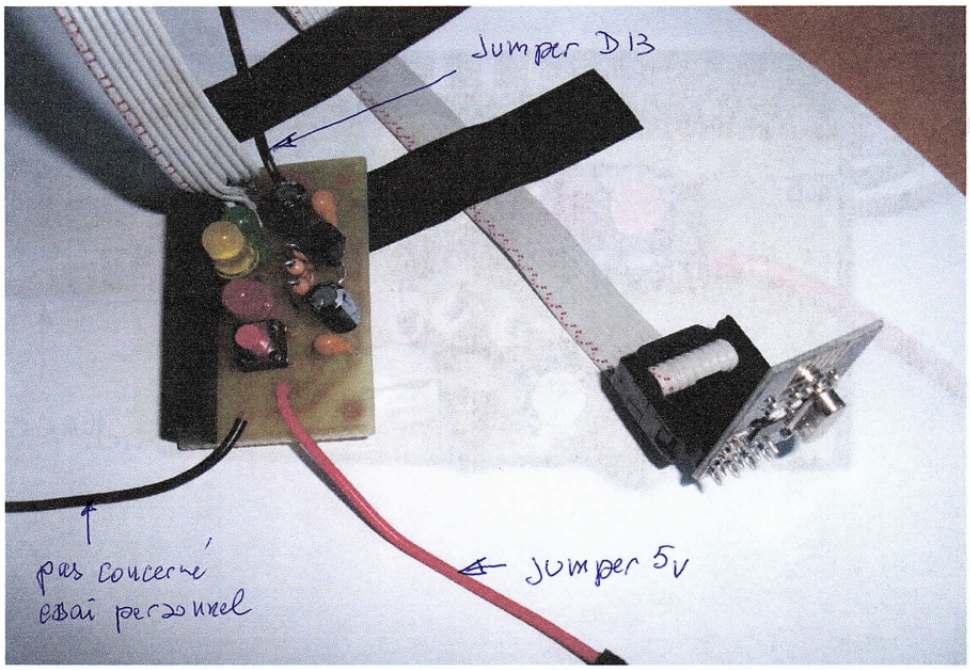


FIGURE 16.5 – Vue de la passerelle et de l'émetteur

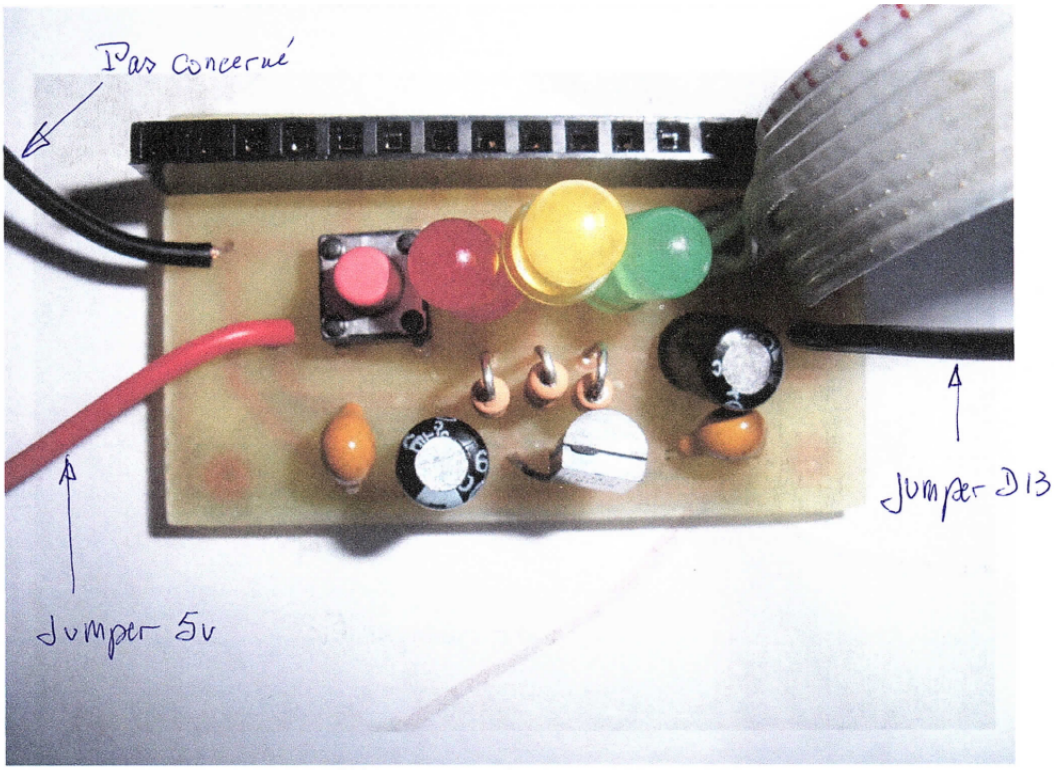


FIGURE 16.6 – Vue de dessus

16.3 Présentation de la sonde

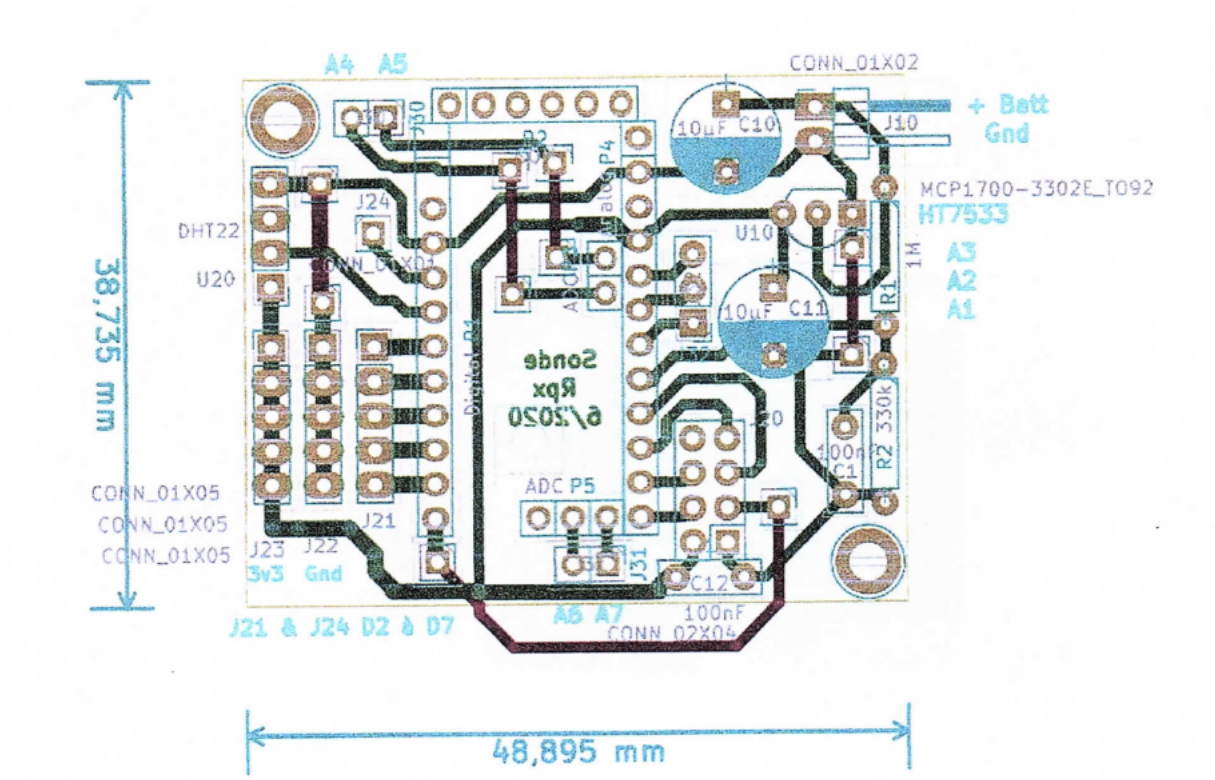


FIGURE 16.7 – Circuit imprimé vu de dessus, coté composants

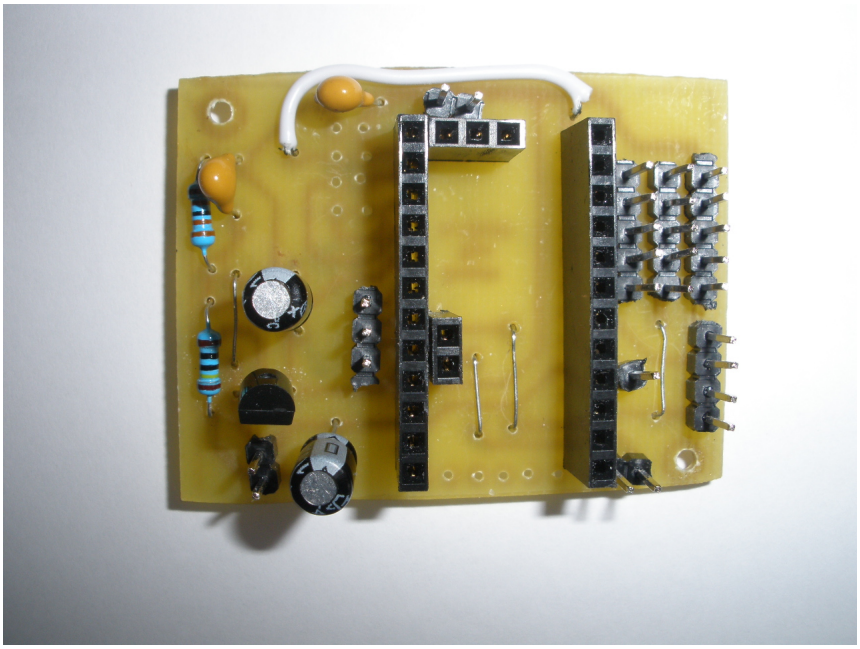


FIGURE 16.8 – Sonde vue de dessus sans la nappe

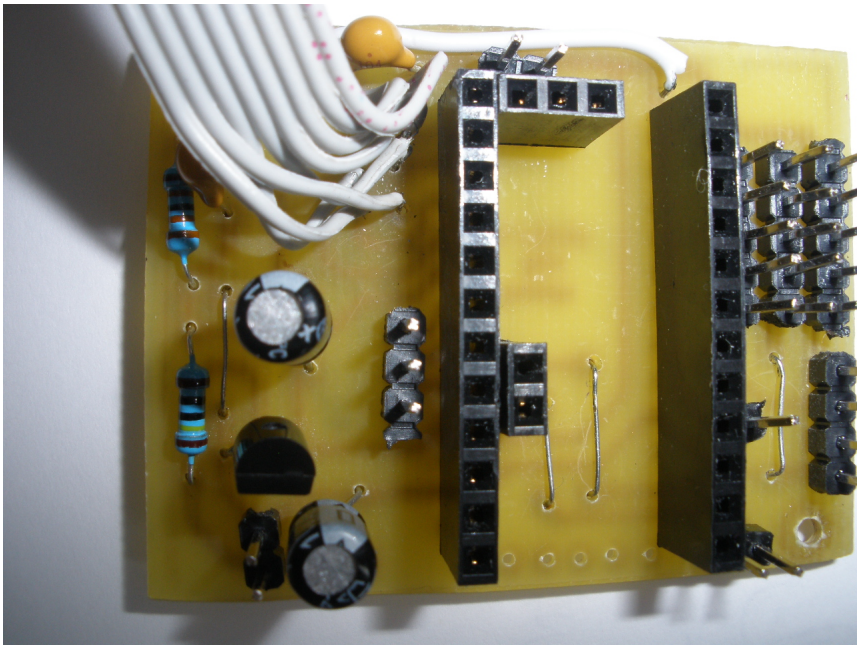


FIGURE 16.9 – Sonde vue de dessus avec la nappe

Section 17

Configuration de Domoticz

Une fois que la passerelle est fonctionnelle, nous allons configurer Domoticz pour que la plateforme reçoive les données en provenance de la passerelle.

17.1 Ajout de la passerelle

Tout d'abord, allez dans la section **Configuration ¿ Matériel**

FIGURE 17.1 – Emplacement du matériel

Ensuite, saisissez les informations suivantes :

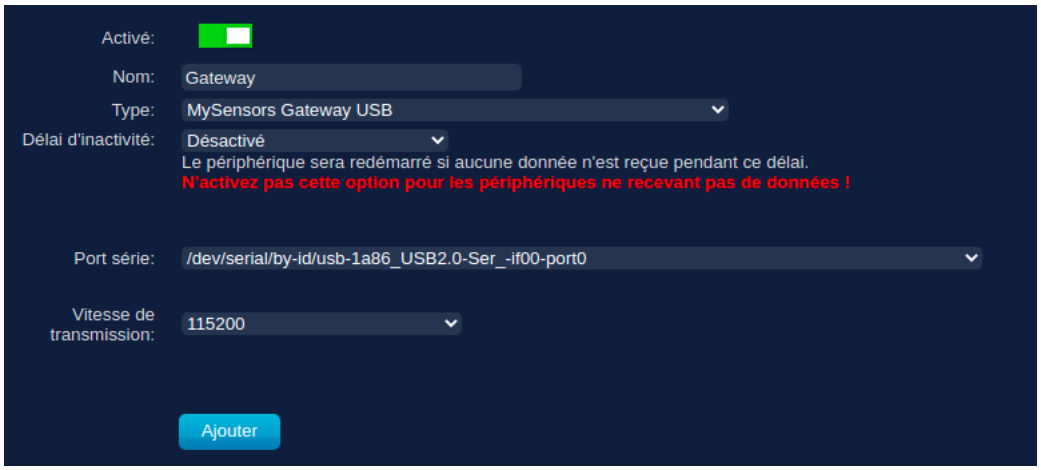


FIGURE 17.2 – Paramétrage de la passerelle

Le port série sélectionné sera celui où est raccordé la passerelle en liaison USB. Il ne faut pas prendre les noms simplifiés des ports USB (*COM_XXX*) mais le nom le plus complet.
Pour plus de simplicité, veuillez déconnectez tous les autres périphériques du Raspberry-Pi
.

S'il s'agit d'un autre capteur (température...) il suffit de parcourir la liste pour le trouver ?

17.2 Recherche des capteurs

Visualisons les données en provenance de la sonde en allant dans
Configuration ¿ Matériel
.

L'ensemble de vos dispositif apparaît. En cas de liste trop longue, saisissez **Gateway** dans la barre de recherche.

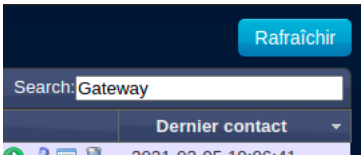


FIGURE 17.3 – Recherche de la passerelle

Remarque

Si le dispositif n'apparaît pas immédiatement, patientez quelques instants.

FIGURE 17.4 – La passerelle est détectée

Pour visualisez les valeurs des capteurs, il faut sélectionner la passerelle avec l'ID de la sonde (ici, 30).

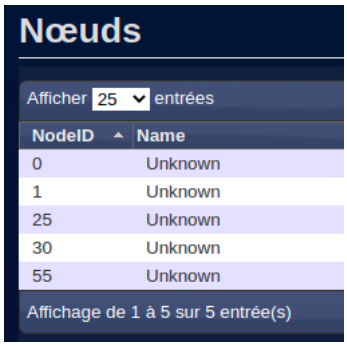


FIGURE 17.5 – Sélection de la passerelle

En cliquant dessus, on voit que la partie **Enfants** est mise à jour et contient les 3 capteurs avec les ID définis dans le programme de la sonde (31,32 et 33 en ce qui me concerne)

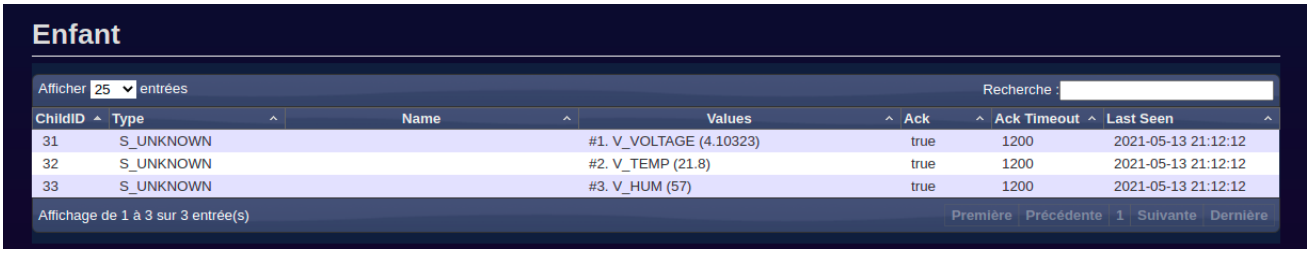


FIGURE 17.6 – Visualisation des enfants

17.3 Visualisation des données

Maintenant que nous savons que la sonde envoi les bonnes données, nous allons ajouter les capteurs dans les dispositifs. Pour cela, allez dans **Configuration ¿ Dispositifs** , les 3 capteurs de la sonde (Tension batterie, humidité et température) apparaissent dans la liste.
Si vous ne les trouvez pas, vous pouver nettoyer la page des capteurs en sélectionnant les capteurs non-utilisés et en le mettant à la poubelle.

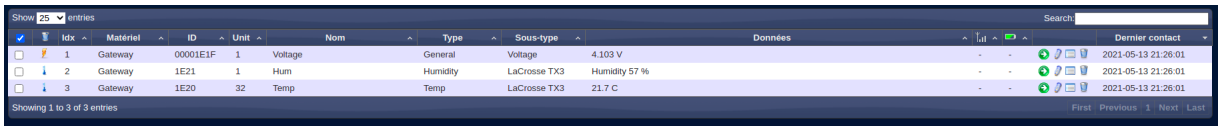


FIGURE 17.7 – Visualisation des capteurs

Les capteurs apparaissent sous les 3 noms suivants :

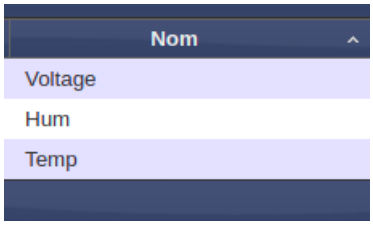


FIGURE 17.8 – Nom des capteurs

Pour ajouter un dispositif, il suffit de cliquer sur la flèche verte et de choisir le nom du dispositif.

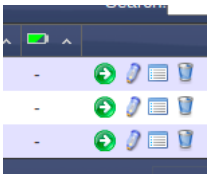


FIGURE 17.9 – Ajout des dispositifs

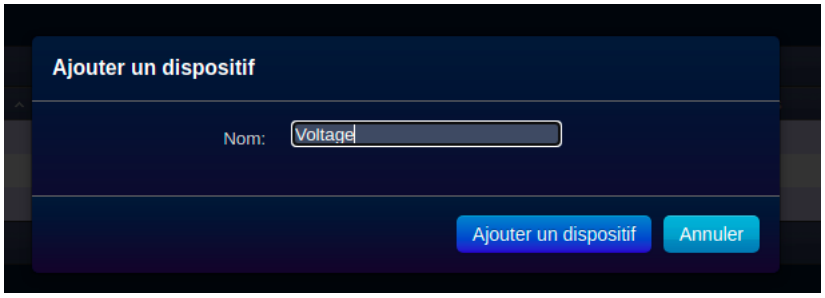


FIGURE 17.10 – Ajout des dispositifs - Sélection du nom

Il suffit de cliquer dans le menu **Mesures**

FIGURE 17.11 – Mesures

Et apparaît la tension de la batterie.

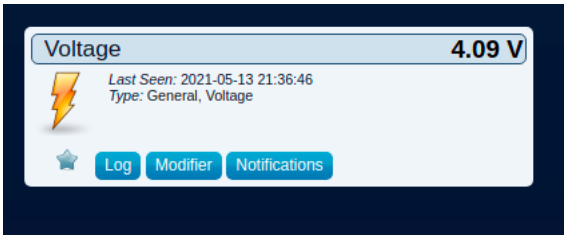


FIGURE 17.12 – tension de la batterie

On procède de même pour l'humidité et la température, les dispositifs seront mis dans l'onglet **Température** .

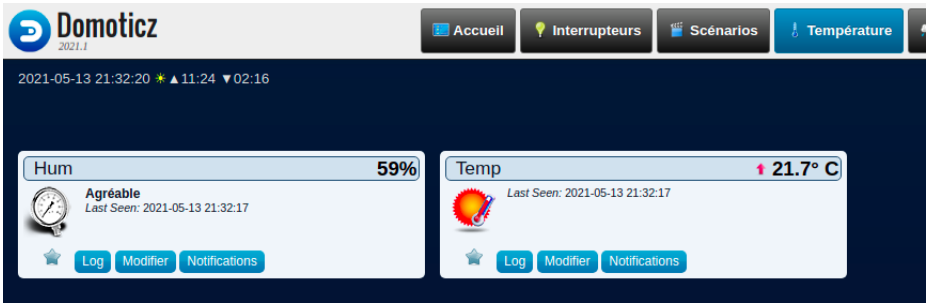


FIGURE 17.13 – Mesures de l'humidité et de la température

Pour visualiser les données, il suffit de cliquer sur le bouton **logs**

17.4 Conclusion

Nous avons vu une utilisation de Domoticz via les sondes MySensors mais énormément de dispositifs domotiques existent sur Domoticz.

Huitième partie

Annexes

Annexe A

Utilisation de l'ESP12 sous Arduino

FIGURE A.1 – ESP12 NodeMCU

A.1 Installation des bibliothèques et cartes ESP8266

La carte ESP12 NodeMCU est prévue pour être programmée directement via l'IDE [1] Arduino. Cette carte fait partie de la grande famille des ESP8266.
Pour installer les bibliothèques et cartes sur le logiciel Arduino, il faut réaliser les étapes suivantes :

- Ouvrir les préférences du logiciel Arduino dans KEY Fichiers - Préférences

1. IDE : Environnement de Développement Intégré

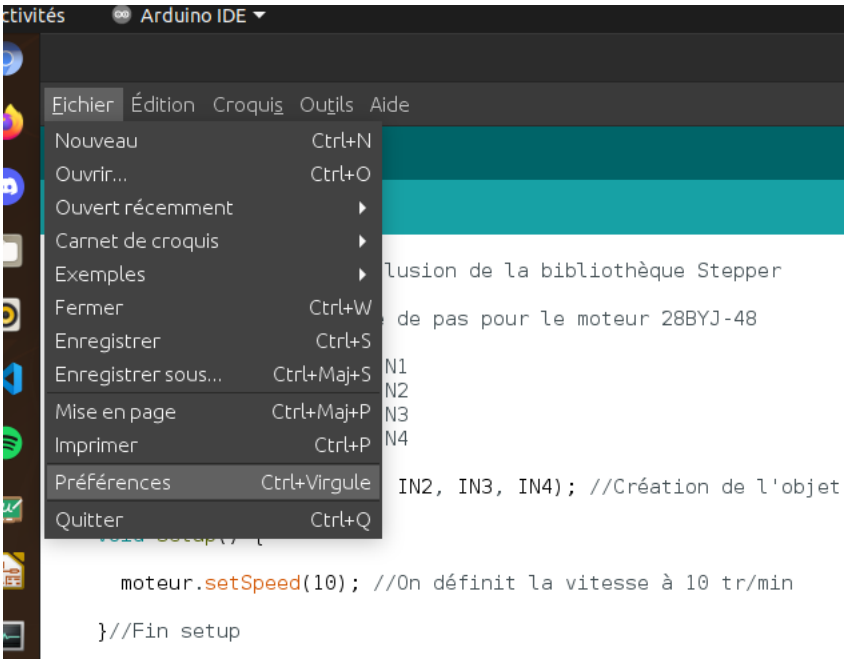


FIGURE A.2 – Préférences Arduino

- Dans le champ **URL de gestionnaire de cartes supplémentaires** , mettre le lien suivant : LINK http://arduino.esp8266.com/stable/package_esp8266com_index.json

Avertissement

Veuillez vérifier l'URL après le copier-coller car les underscores (”tirets du 8”) peuvent disparaître.

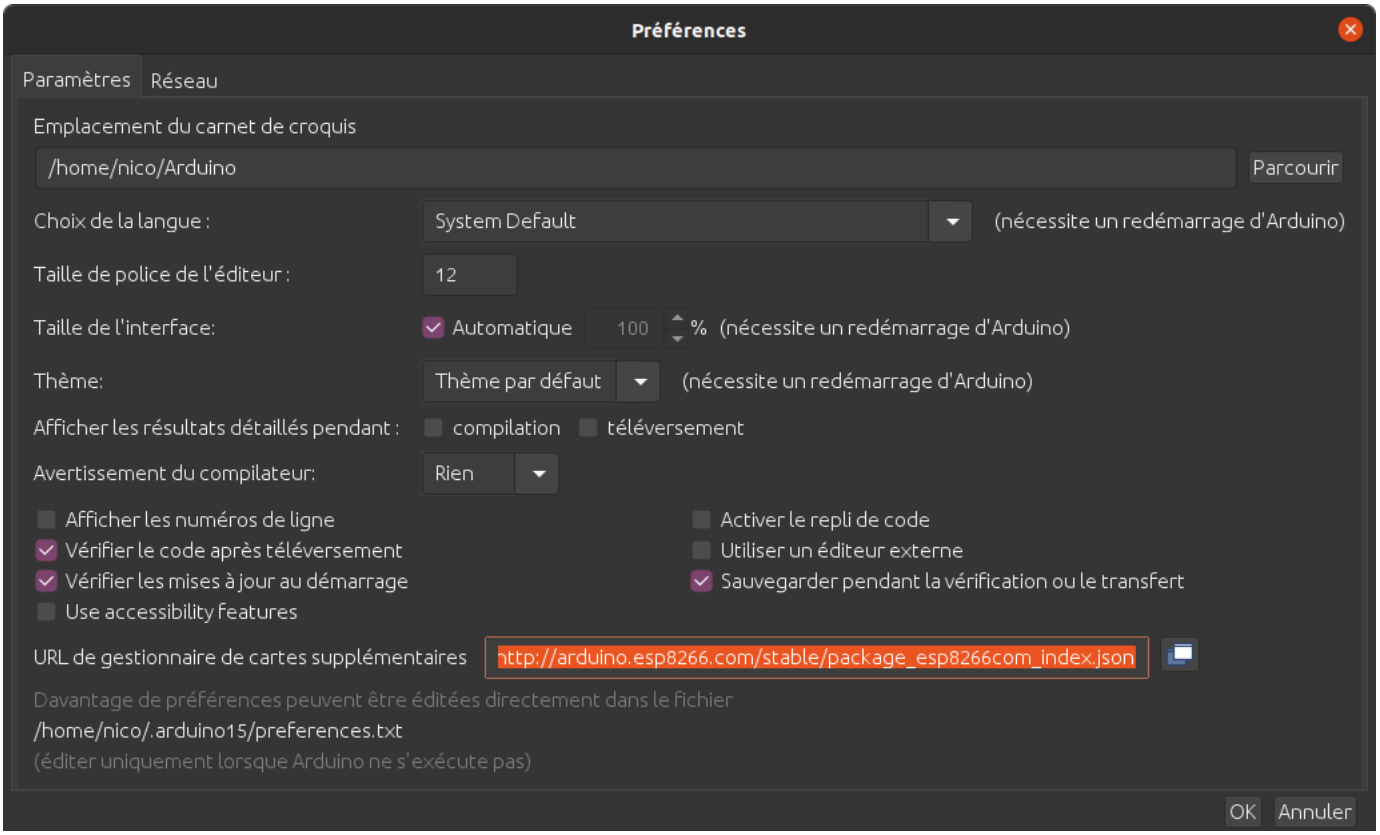


FIGURE A.3 – Lien pour les cartes ESP8266

Puis faire KEY OK

- Fermer le logiciel Arduino
- Lancer le logiciel Arduino
- Allez dans KEY Outils - Type de carte - Gestionnaire de carte

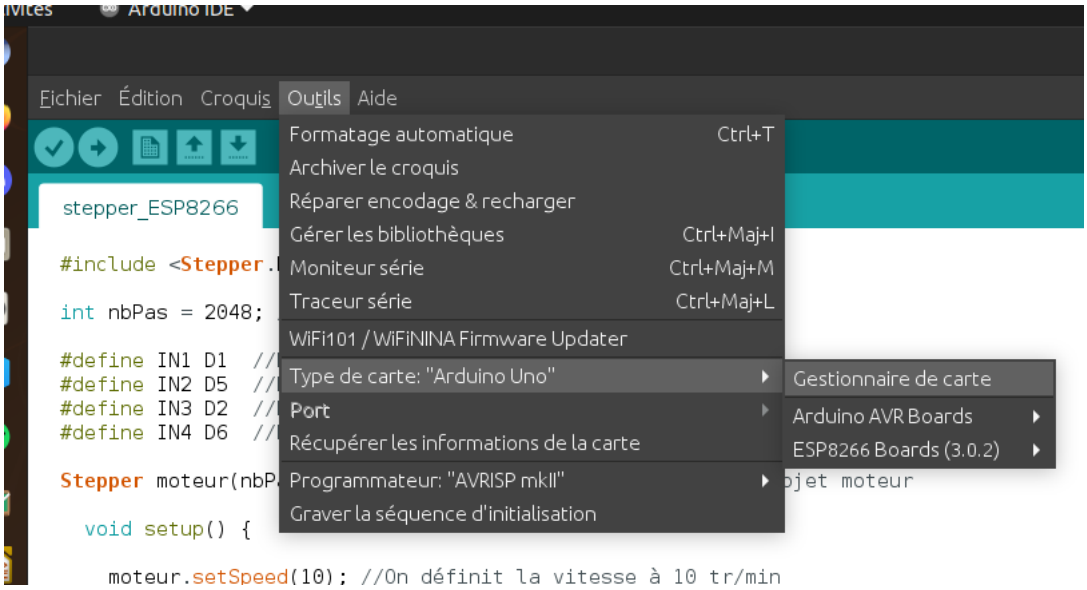


FIGURE A.4 – Gestionnaire des cartes ESP8266

et faire une recherche avec le mot clé KEY esp8266

FIGURE A.5 – Installation des bibliothèques ESP8266

Il ne vous reste plus qu'à cliquer sur KEY Installer et redémarrer le logiciel Arduino.

A.2 Recherche des cartes ESP8266

Lors de la programmation d'une carte ESP8266 NodeMCU, il faudra donc aller dans KEY Outils - Type de carte - ESP8266 Boards NodeMCU X.X (ESP12 Module)

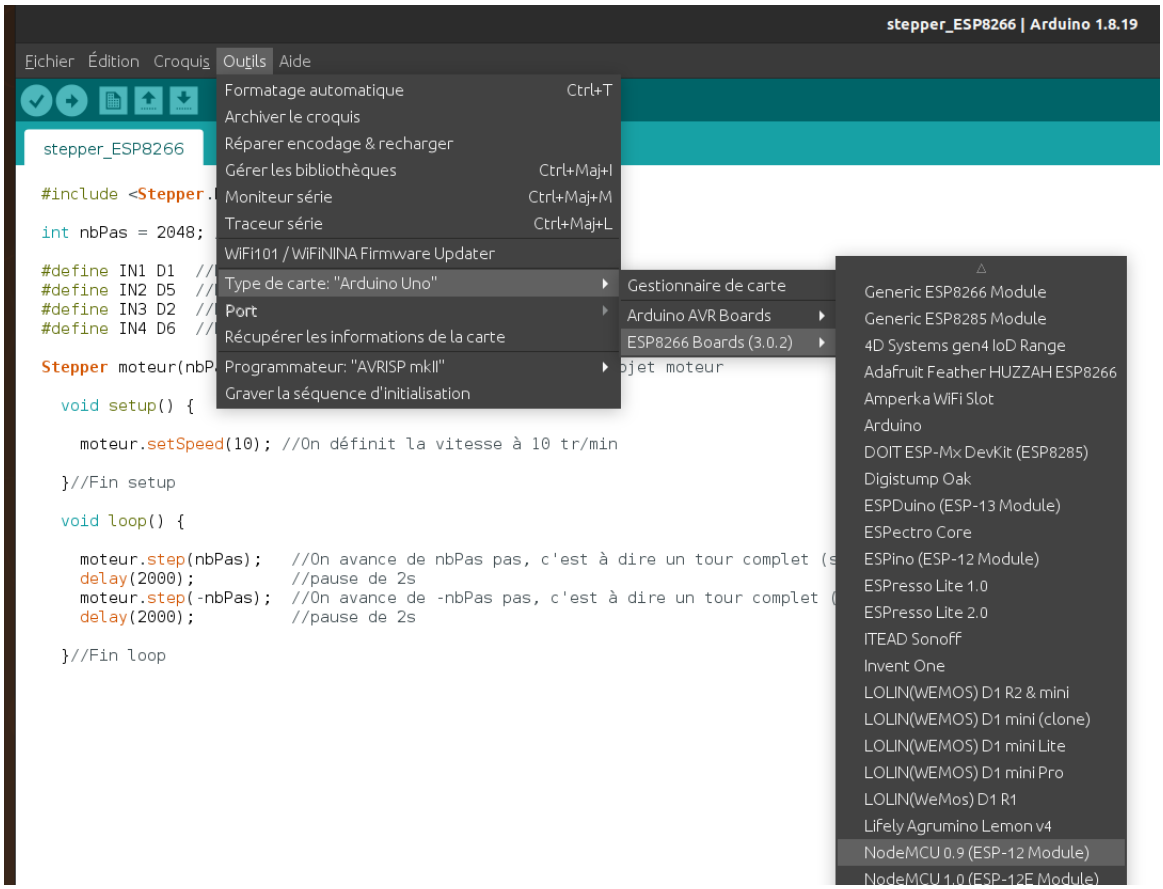


FIGURE A.6 – Sélection de la carte ESP12

Afin de tester le bon fonctionnement, nous vous invitons à tester le programme **Blink** disponible dans les exemples.

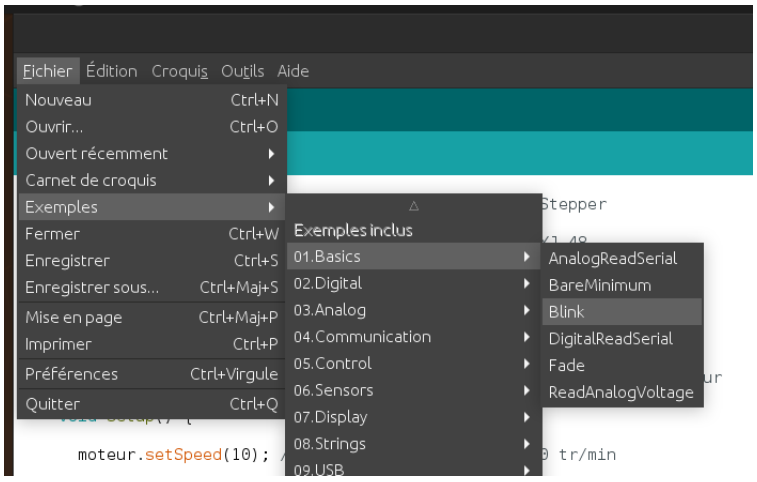


FIGURE A.7 – Emplacement de l'exemple Blink

La led bleue de l'ESP12 devrait clignoter si l'installation s'est correctement effectuée.

A.3 Recherche des cartes Arduino

Pour la programmation des cartes Arduino, il suffira de sélectionner
KEY Outils - Type de carte - Arduino AVR Boards - Carte X en fonction du modèle de votre carte.

Annexe B

Langage HTML

Le HTML [1] HTML est un langage contenant des balises, c'est à dire des marqueurs spécifiques pour organiser la page Web.

Il existe deux types principaux de balises :

- Les balises en paire (Par exemple $< h1 >< /h1 >$)
- Les balises orphelines (Par exemple $< img >$)

Une balise commence par un chevron ouvrant et se termine par un chevron fermant. Toutes les balises fermantes (pour les balises en paire) sont de la forme $< /nom_balise >$

Toute page HTML commencer par la balise html

Remarque

Pour mettre du code en commentaire, c'est à dire ne pas le visualiser dans la page Web de rendu, il faut mettre le code entre CMT $< --!$ et CMT $-- >$

```
<html>
    <!-- Ceci est mon début de page HTML -->
</html>
```

Première balise HTML

B.1 La forme de la page

La page Web est scindée en 2 entitées :

- L'en-tête (header), marqué avec la balise CMT $< head >< /head >$
- Le corps (body), marqué avec la balise CMT $< body >< /body >$

1. HyperText Markup Language

```
<html>
    <head>
        <!-- Ceci est un header -->
    </head>

    <body>
        <!--! Ceci est un body -->
    </body>
</html>
```

Page minimaliste HTML

B.2 L'en-tête

L'en-tête va contenir les informations et les paramètres de la page, notamment :

- Le titre de la page
- Les importations des bibliothèques (feuilles de style)
- Les icônes
- L'encodage de la page (UTF-8)

```
<head>
        <--! Titre en haut de la page -->
        <title>Titre de la page</title>

        <--! Ajout d'une feuille de style -->
        <link rel="stylesheet" href="style.css" type="text/css">

        <--! Ajout d'une icone -->
        <link rel="icon" href="icone.ico">

        <--! Encodage UTF-8 -->
        <meta charset="utf-8">

</head>
```

L'en-tête

B.3 Le corps

Le corps va contenir l'ensemble des informations affichées sur la page

- Les titres, sous-titre, sous-sous-titre
- Les paragraphes
- Les images
- Les liens
- Les sections de code
- ...

B.3.1 Ajout des titres

Un titre en HTML est vue avec un niveau hiérarchique. Un titre de page est au niveau 1, un sous-titre avec un niveau 2, etc...

Pour mettre un titre, il faut donc écrire CMT $< h1 > Titre < /h1 >$, un sous-titre c'est CMT $< h2 > Sous - titre < /h2 >$

B.3.2 Ajout des images

Pour ajouter une image, il faut connaître son emplacement dans le système (ordinateur) ou bien sur Internet (url).

Il s'agit de la balise orpheline CMT $< img >$

```
<--! Image locale -->
<img src="monImage.png" alt="Impossible de charger l'image">

<--! Image Internet -->
<img url="www.crepp.oreg/image.png" alt="Impossible de charger l'image">
```

Ajout d'une image

B.3.3 Ajout des liens

Pour ajouter une image, il faut connaître son emplacement dans le système (ordinateur) ou bien sur Internet (url).

Il s'agit de la balise orpheline CMT $< img >$

```
<--! Lien -->
<a href="monChemin" > Texte du lien</a>
```

Ajout d'un lien

Les liens permettent de pointer sur d'autres pages, que ce soit sur le serveur courant ou bien un autre.

Annexe C

Installation de Domoticz

C.1 Installation de Domoticz sur Linux

Veuillez ouvrir un terminal puis saisir les commandes suivantes :

```
sudo apt-get -y install cmake make gcc g++ libssl-dev git libcurl4-openssl-dev
   libusb-dev python3-dev curl zlib1g-dev zlib1g
```

Installation de domoticz

Puis lancez le script d'installation avec la commande suivante

```
  sudo curl -L https://install.domoticz.com | bash
```

Installation de domoticz

Le terminal devrait afficher un contenu similaire :

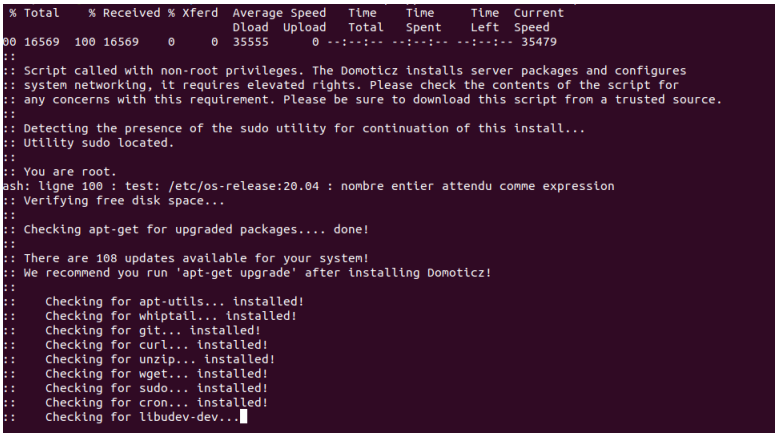


FIGURE C.1 – Vérification des bibliothèques

Ensuite, une interface utilisateur se lance dans le terminal :

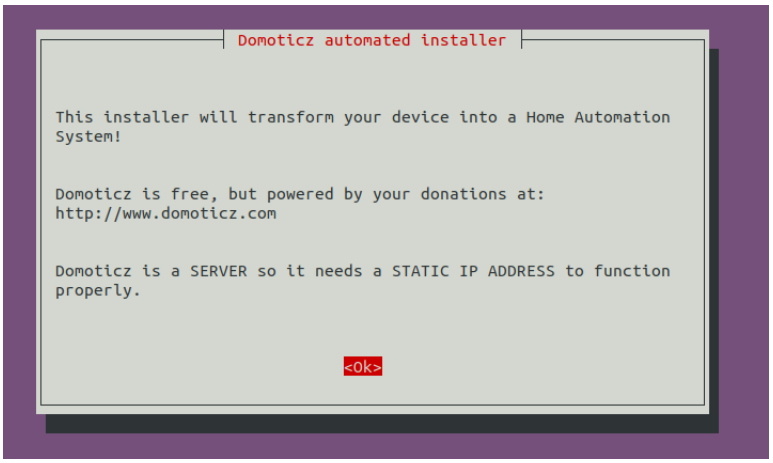


FIGURE C.2 – Présentation de Domoticz

Il faut saisir la touche KEY ENTREE pour afficher la fenêtre suivante. Une deuxième fenêtre apparaît. Veuillez sélectionner le service HTTP (par défaut) puis KEY ENTREE

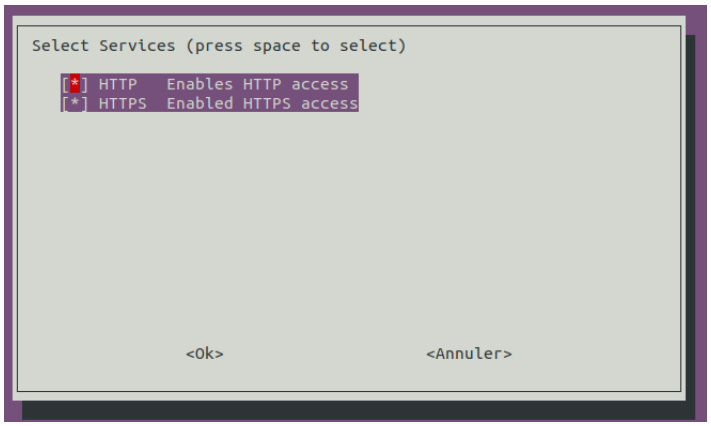


FIGURE C.3 – Choix du protocole par défaut

Nous allons ensuite choisir le port 8080 pour communiquer sur le réseau (par défaut : 8080)

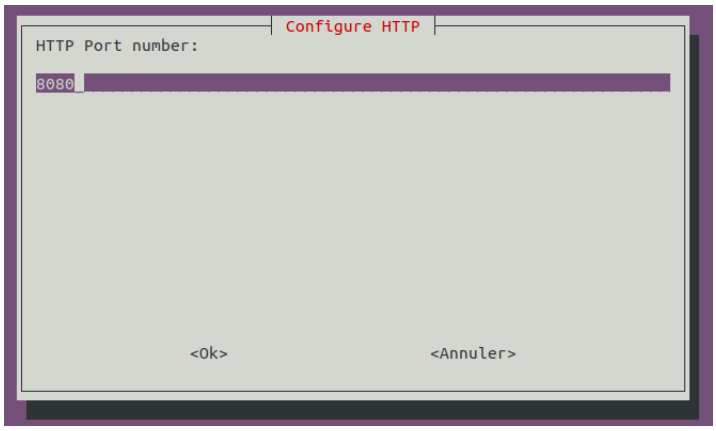


FIGURE C.4 – Choix du port

Nous utiliserons le port 443 (HTTPS) pour un protocole plus sécurisé.

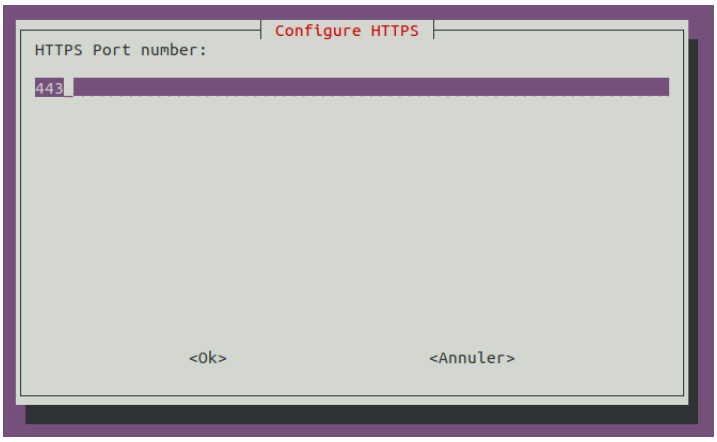


FIGURE C.5 – Protocole HTTPS

Il ne vous reste plus qu'à choisir l'emplacement du logiciel Domoticz.
Par défaut, Domoticz le place dans vos documents personnels (/home/nom_utilisateur)

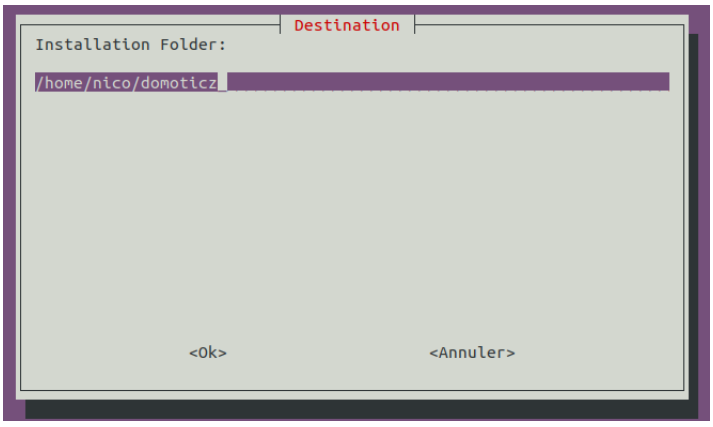


FIGURE C.6 – Emplacement des fichiers Domoticz

Il ne vous reste plus qu'à valider l'installation :

FIGURE C.7 – Validation de l'installation

Puis dans votre navigateur internet, saisir :

```
localhost:8080
```

Lancement de Domoticz

Annexe D

Installations de bibliothèques

Lors de nouveaux projets, certaines bibliothèques peuvent être manquantes.
On s'en aperçoit quand on clique sur le bouton de vérification (bouton tout à gauche) du code :

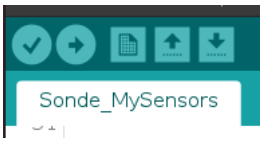


FIGURE D.1 – Bouton de vérification

.

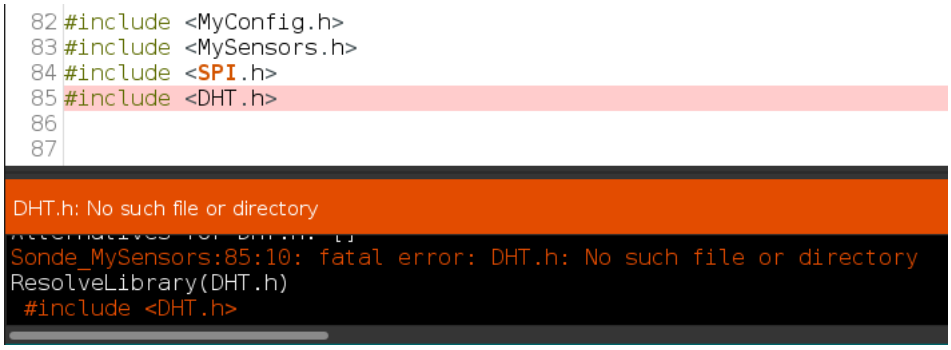


FIGURE D.2 – La bibliothèque DHT manquante

Une erreur de ce type nous indique que la bibliothèque LIB DHT est manquante.

Il existe deux façons d'installer des bibliothèques Arduino.

D.1 Ajout via le gestionnaire de bibliothèques

Tout d'abord, veuillez vous rendre dans le menu **Outils - Gérer les bibliothèques** .

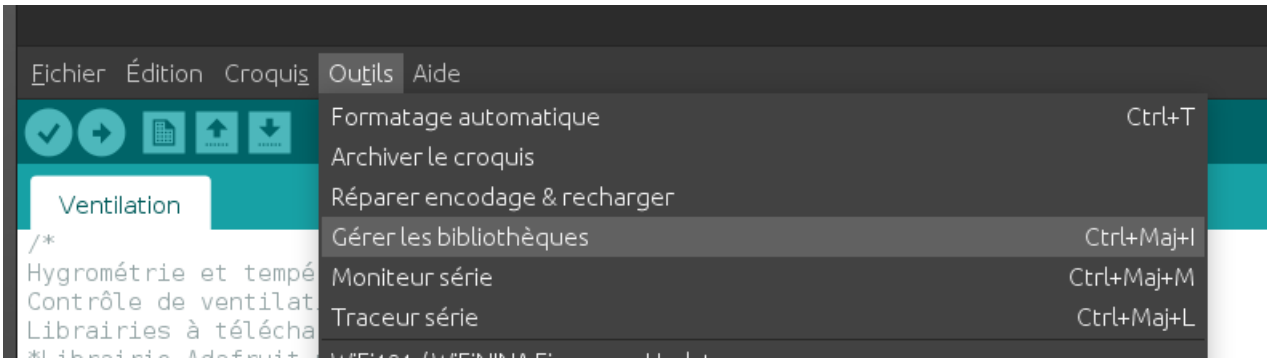


FIGURE D.3 – Le gestionnaire de bibliothèques

Vous tombez sur une interface similaire :

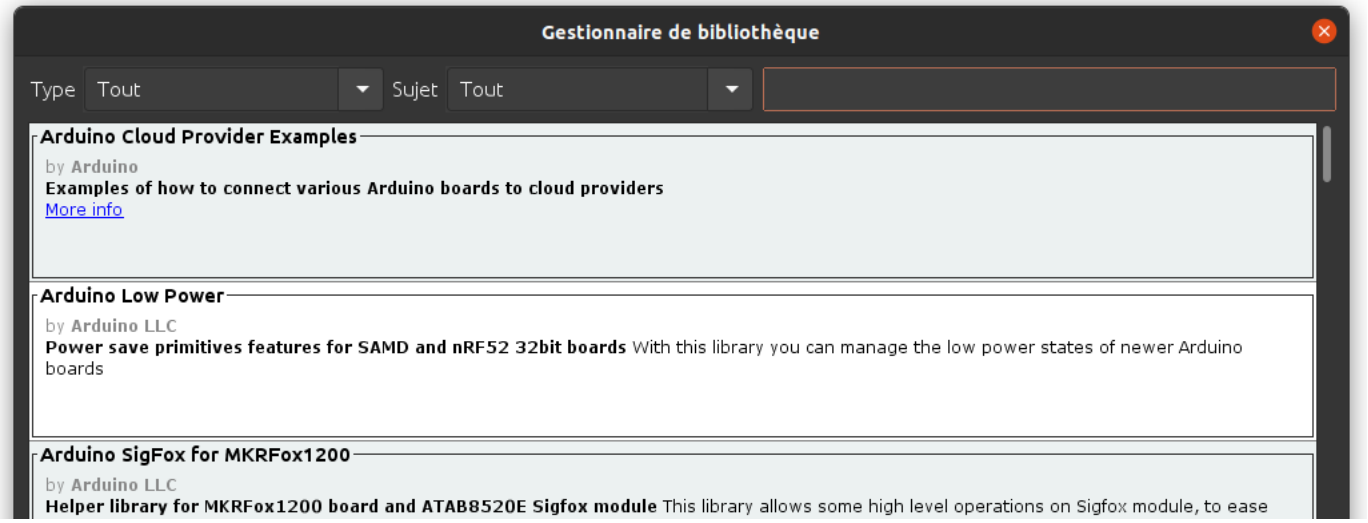


FIGURE D.4 – Les bibliothèques existantes

Cette page affiche toutes les bibliothèques disponibles via le gestionnaire de bibliothèques. Dans la barre de saisie en haut à droite, il faut indiquer le nom de la bibliothèque désirée. Prenons par exemple la bibliothèque DHT :

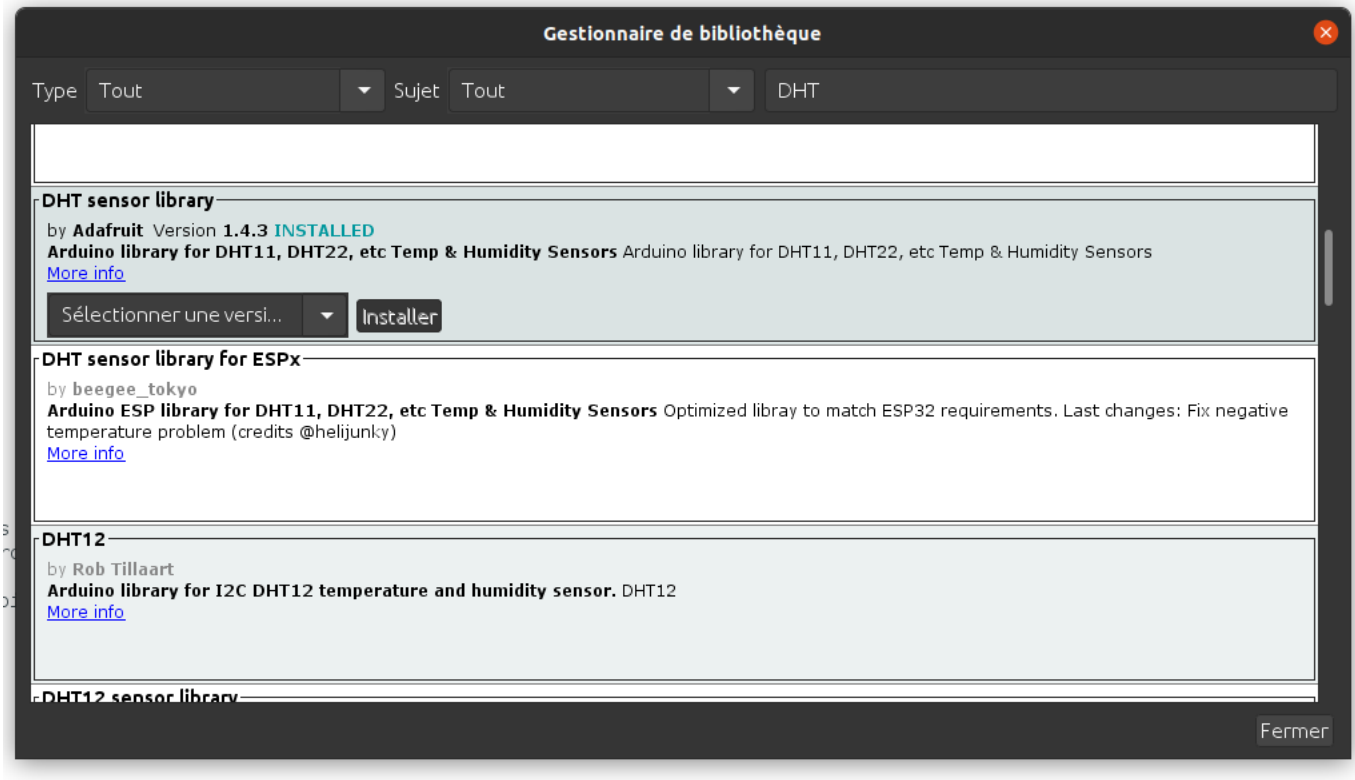


FIGURE D.5 – Ajouter une bibliothèque DHT

Il faut parcourir la liste et trouver la bibliothèque LIB DHT sensor library de chez **Adafruit** . Il ne reste plus qu'à sélectionner la version puis cliquer sur KEY Installer.

Pour que les changements soit pris en compte, il faut redémarrer l'IDE IDE Arduino.

D.2 Ajout via un fichier ZIP

Cette méthode est un peu plus longue mais parfois, pour certaines bibliothèques non gérées par le gestionnaire de bibliothèques, nous n'avons pas le choix.

En premier lieu, il faut trouver la bibliothèque sur Internet. Par exemple, la bibliothèque LIB DHT de chez **Adafruit** est disponible à l'adresse suivante : https://github.com/adafruit/DHT-sensor-library

Il ne reste plus qu'à cliquer sur KEY Code - Download ZIP et le dossier compressé va se placer dans vos téléchargements.

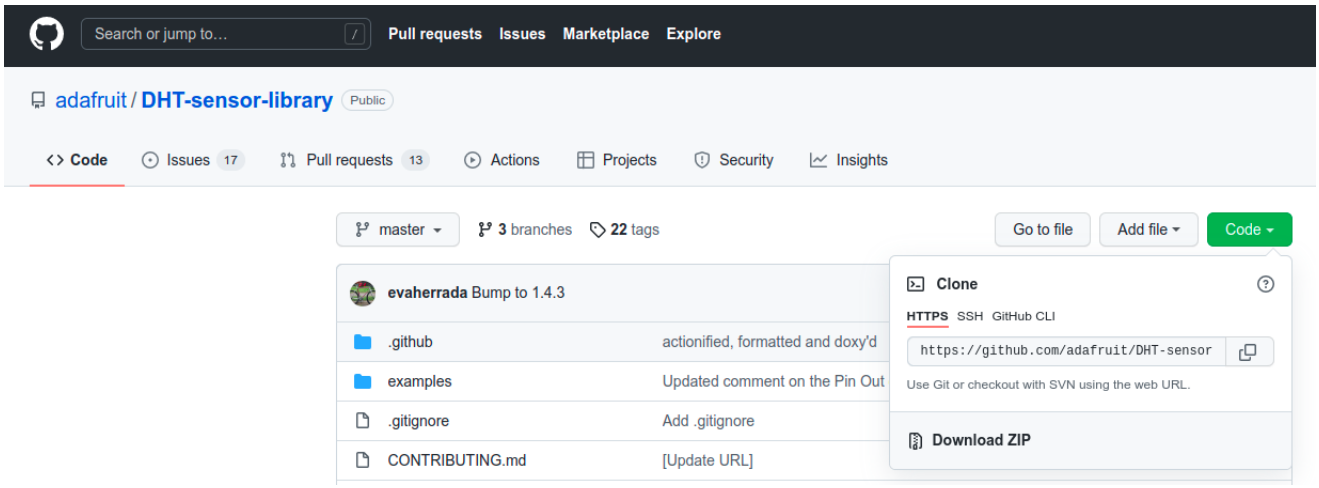


FIGURE D.6 – Téléchargement de la bibliothèque DHT

Enfin, pour installer la bibliothèque, il suffit d'aller dans **Croquis - Inclure une bibliothèque - Ajouter la bibliothèque .ZIP**

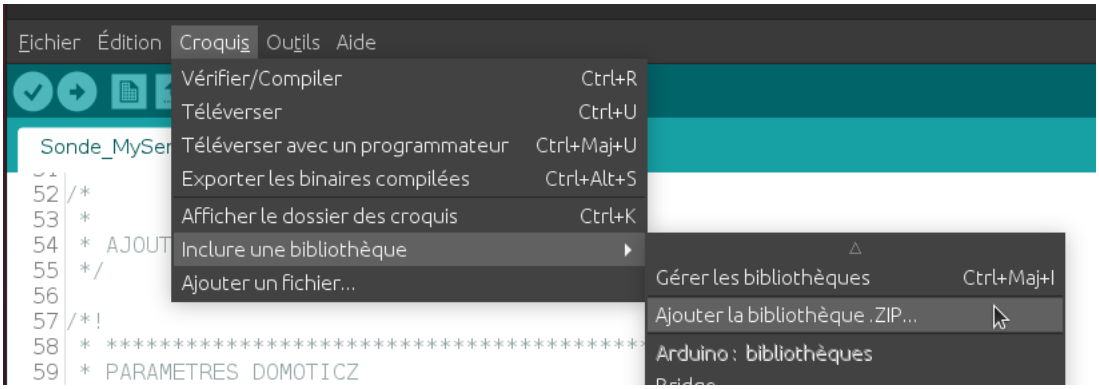


FIGURE D.7 – Ajout d'une bibliothèque

Il ne reste qu'à trouver le fichier **DHT_sensor-master.zip** et à faire **OK**

Annexe E

Utilisation des ressources du CREPP

E.1 Présentation

Ce document a pour objectif d'expliquer le fonctionnement de Git pour les ateliers du CREPP. Un répertoire Git a été créé pour centraliser les supports des ateliers ainsi que les codes et projets produits depuis la création du CREPP en 2012.
Il est donc en perpétuelle amélioration.

E.2 Localisation

Le répertoire est disponible à l'adresse suivante : LINK https://github.com/CREPP-PLOEMEUR ou bien en passant sur le site du CREPP [1], dans la section **Ressources GIT**

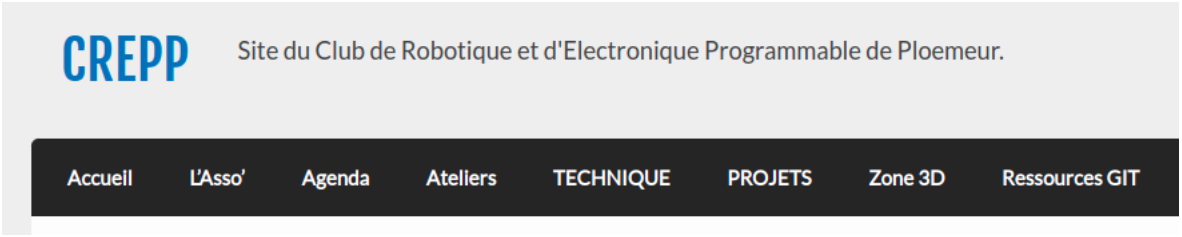


FIGURE E.1 – Accès aux ressources GIT

E.3 Menu principal

Une fois le lien cliqué, vous tombez directement sur cette interface

1. LINK crepp.org

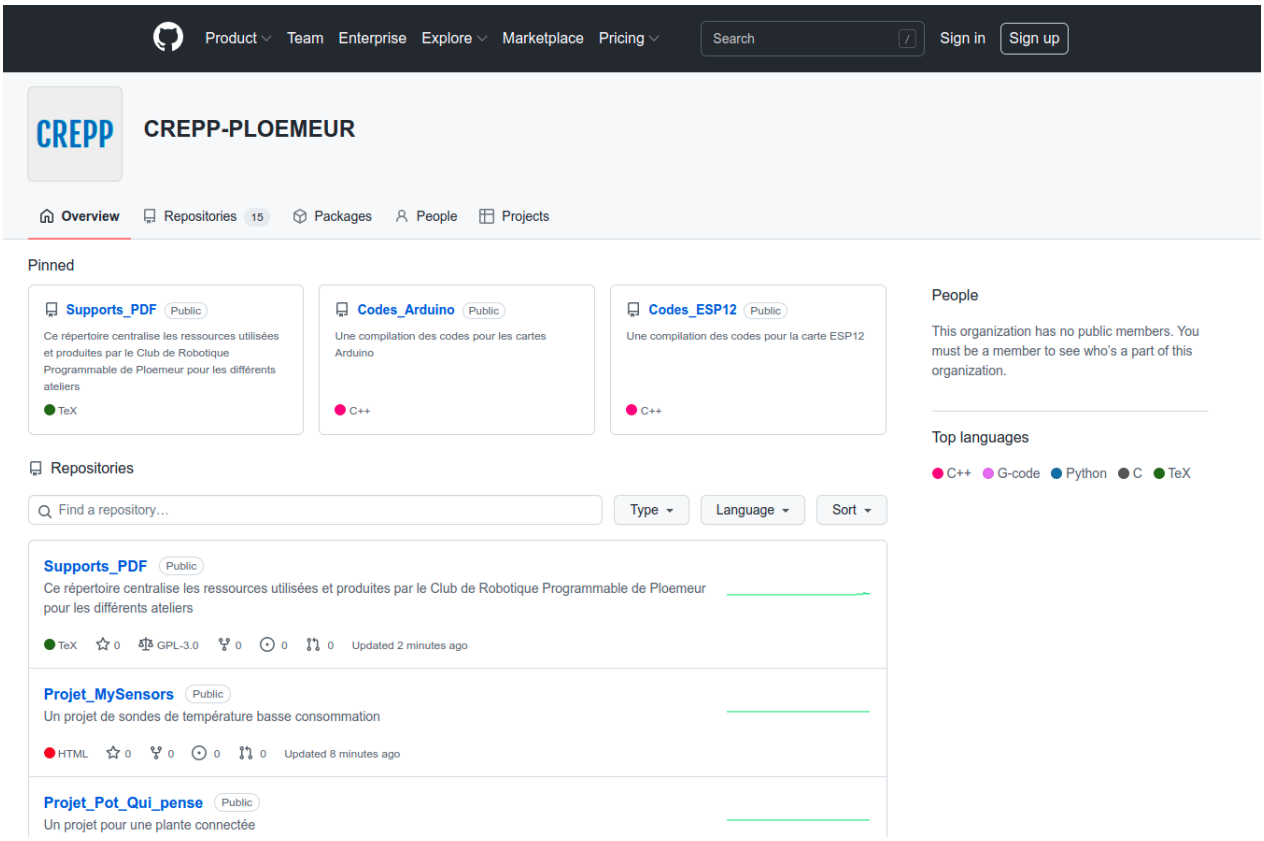


FIGURE E.2 – La page principale du répertoire Git

Les répertoires facilement accessibles sont :

- Le répertoire **Supports_PDF** regroupe les supports PDf utilisés pour les Ateliers Arduino et Microcontrôleurs.
- Le répertoire **Codes_Arduino** est une compilation des codes Arduino utilisés lors des ateliers.
- Le répertoire **Codes_ESP12** est une compilation des codes ESP12 des ateliers.

Sous ces trois répertoires, vous avez accès à l'ensemble des répertoires du CREPP. Pour afficher tous les répertoires, vous pouvez cliquer sur **View all repositories** en bas de la page.

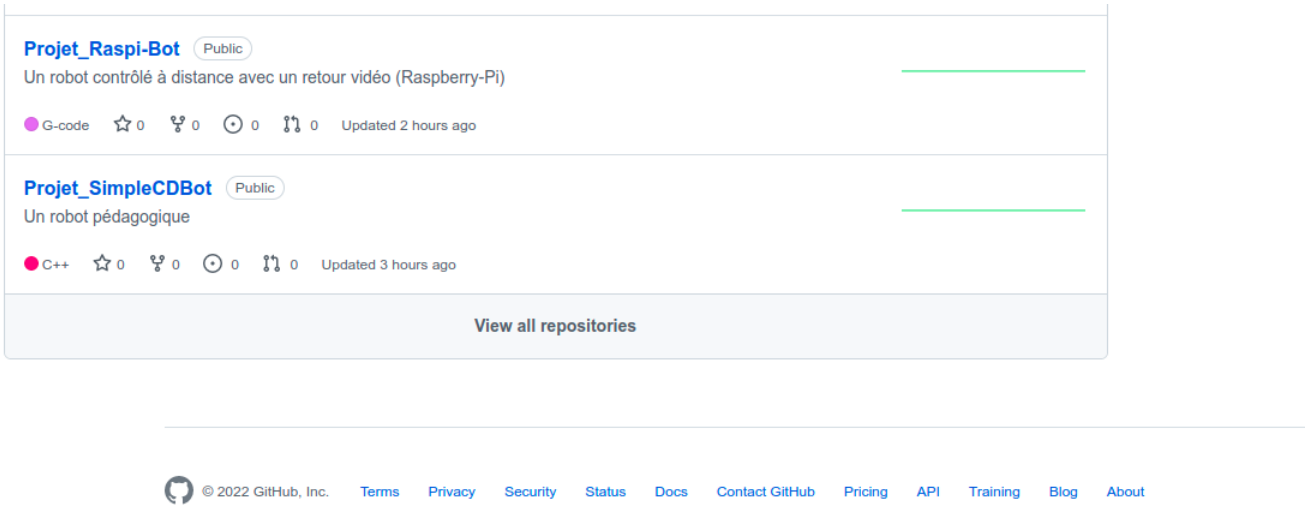


FIGURE E.3 – Afficher l'ensemble des répertoires

Il est possible de classer les répertoires en cliquant sur **Sort** et de les classer en fonction de :

- Les dates de modifications
- Les noms

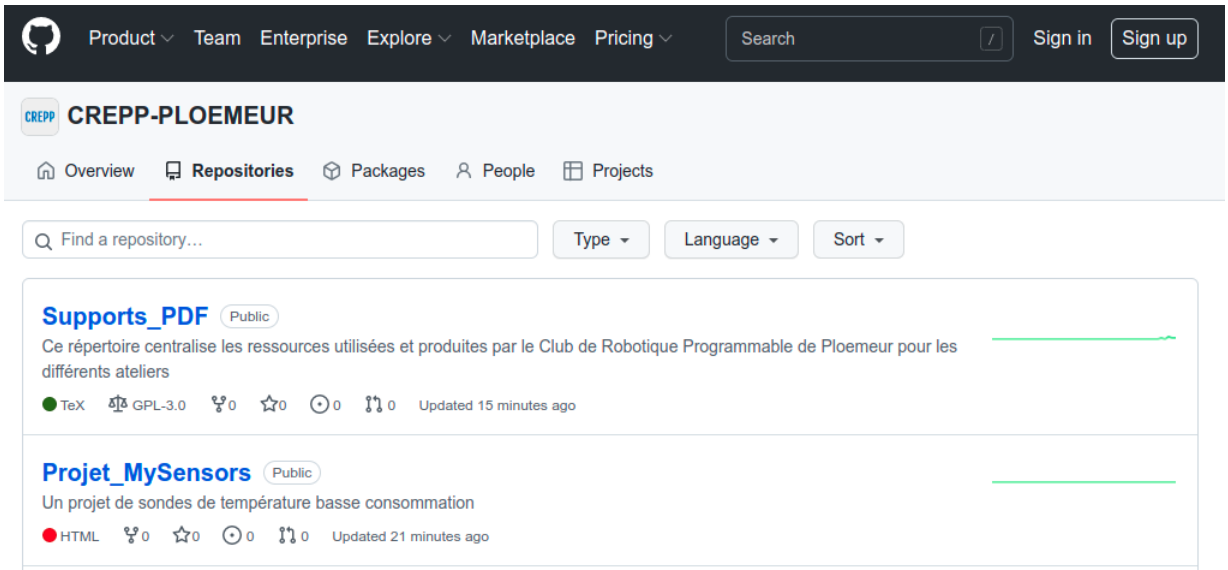


FIGURE E.4 – Trier les répertoires

Actuellement, voici les répertoires :

- codes_Arduino
- Codes_ESP12

- Codes_Pico
- Projets_Ateliers_Jeunes
- Projet_Capteur_pollution_atmospherique
- Projet_Cocci-Bot
- Projet_Crepp-Rap
- Projet_Fauteuil_roulant
- Projet_MySensors
- Projet_Pot_Qui_pense
- Projet_Raspi-Bot
- Projet_SimpleCDBot
- Projet_Ventilateur
- Supports_PDF

E.4 Exploration d'un répertoire

E.4.1 L'arborescence

En cliquant sur un répertoire, l'arborescence de ce dernier apparaît avec le premier rang des dossiers et les fichiers sur le même niveau.
En cliquant sur les noms des dossiers, on peut parcourir l'arborescence du répertoire complet.

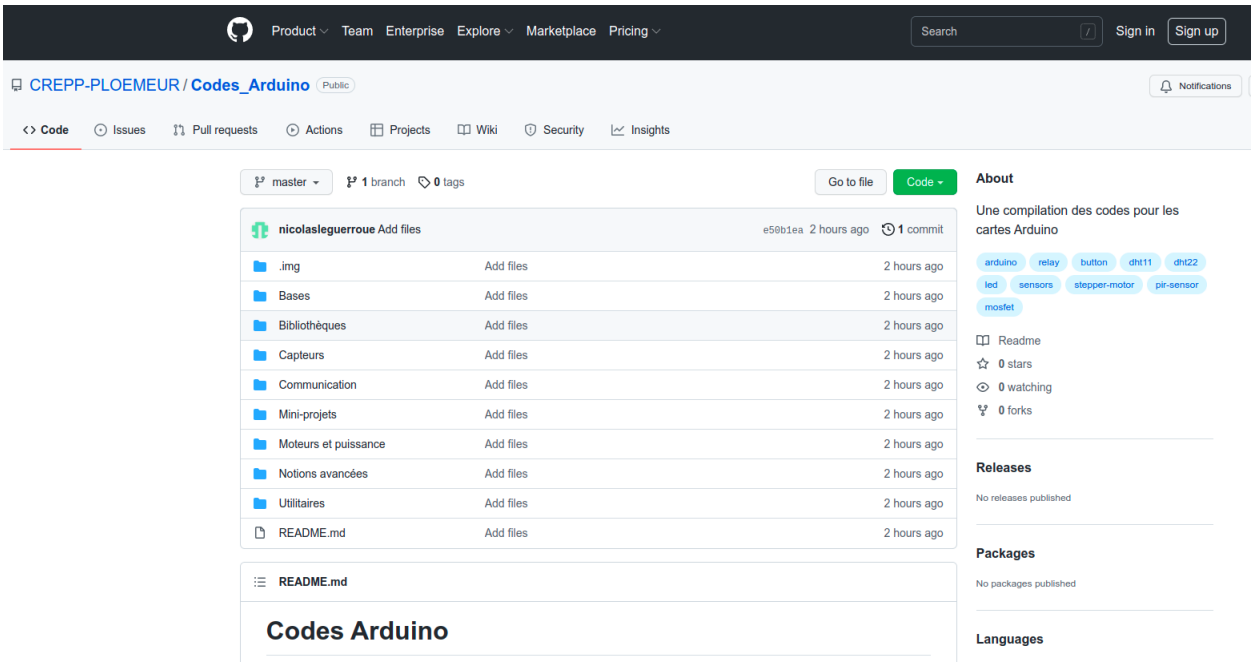


FIGURE E.5 – L'arborescence du répertoire

Une description du répertoire est disponible avec un fichier README.md. (ici le répertoire Codes_Arduino)

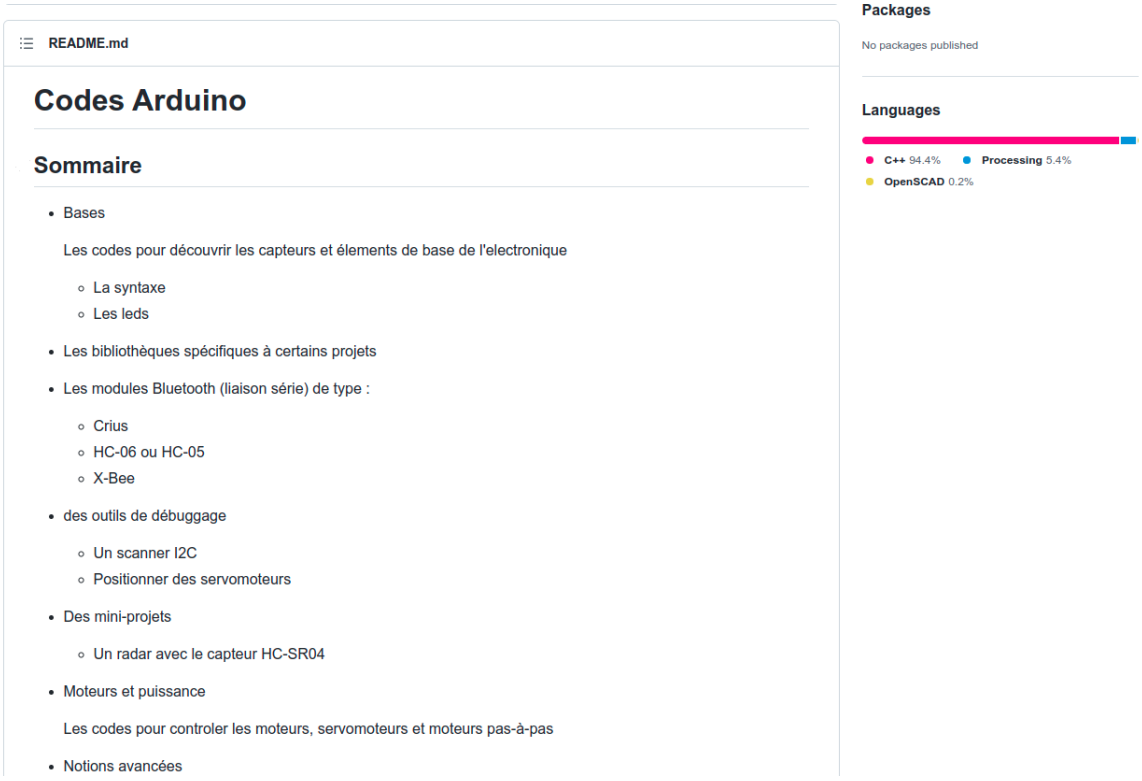


FIGURE E.6 – Une description du répertoire

Les principaux langages utilisés dans le répertoires sont indiqués à droite de la section **README** .

E.4.2 Récupération d'un fichier

Un fichier contenant du code

Pour récupérer le contenu d'un fichier particulier, il faut parcourir l'arborescence pour le trouver.
Une fois le fichier localisé, il faut cliquer sur le fichier afin de voir son contenu :

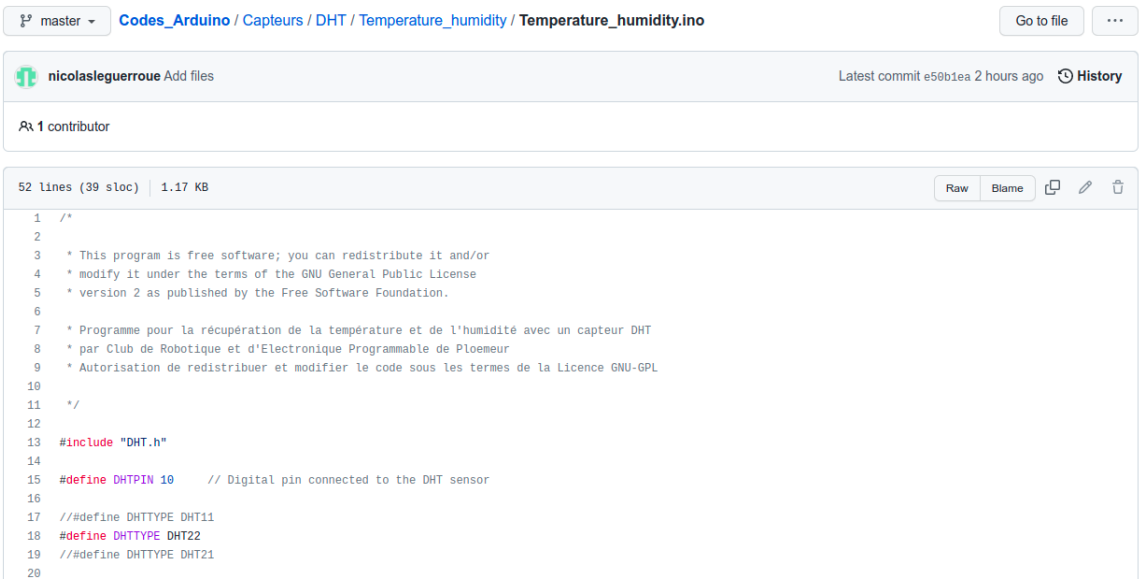


FIGURE E.7 – Un contenu de fichier

Enfin, il ne reste plus qu'à cliquer sur **Copy raw contents** pour copier tout le texte du fichier dans le presse-papier.

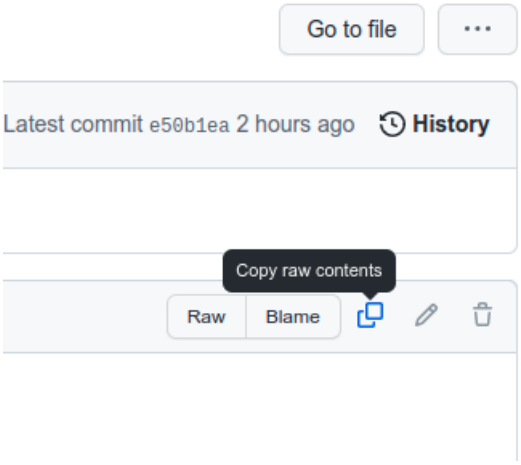


FIGURE E.8 – Copier le contenu d'un fichier

Un fichier PDF

Pour récupérer le fichier PDF, après l'avoir localisé, il suffit de cliquer sur le bouton **Download** .

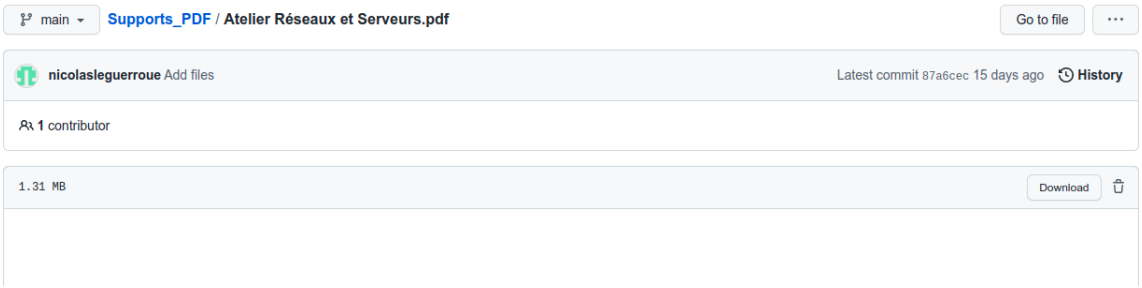


FIGURE E.9 – Téléchargement d'un fichier PDF

Remarque

Cette méthode est contraignante quand nous sommes ammenés à manipuler plusieurs fichiers au sein d'un même répertoire.
La méthode suivante va vous expliquer comment télécharger directement tout un répertoire pour travailler par la suite en local.

E.5 Téléchargement d'un répertoire

Pour télécharger un répertoire dans son intégralité, il faut tout d'abord se placer à la racine de celui-ci en cliquant sur le nom du répertoire (Codes_Arduino) :

FIGURE E.10 – Déplacement à la racine du répertoire

Ensuite, il faut cliquer sur le bouton vert 

pour dérouler un petit menu puis cliquer sur **Download ZIP**

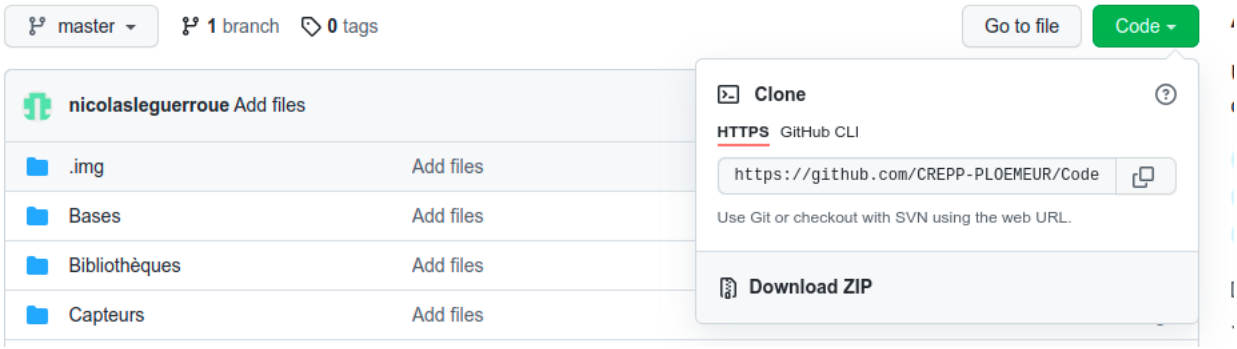


FIGURE E.11 – Téléchargement du répertoire

Le répertoire va se télécharger au format ZIP dans vos téléchargements avec le suffixe **-master** . Par exemple, le répertoire **Codes_Arduino** sera téléchargé sous le nom **Codes_Arduino-master.zip** .

Il ne vous reste plus qu'à extraire le fichier pour explorer le répertoire.

Liste des figures

Index