GUIDE SUR L'UTILISATION DES CAPTEURS

Tome 3

Réalisé par Thomas MARTINEZ

28 juin 2020

Le but de ce troisième tome est le même que pour le précédent : Apprendre à utiliser les capteurs sur une carte Arduino. Dans ce dernier tome sur les capteurs, nous verrons comment utiliser les capteurs laser, d'humidité et de contact.

Introduction

Dans le précédent tutoriel, nous avons vu comment utiliser deux des cinq capteurs étudiés dans certains cas. Le but est surtout de comprendre comment reccueillir l'information mesurée. Les codes données servent surtout d'exemple et pas de code à suivre. En effet, la partie traitement dépend du projet pour lequel le capteur est utilisé.

En suivant le même schéma que pour le tutoriel précédent, nous allons voir ensemble comment utiliser les trois capteurs restants, ce qu'il faut fournir en entrée et ce que l'on reçoit en sortie.

Table des matières

⋆ ⋆ ⋆

A Le capteur laser

A.1 Le branchement

Pour notre explication, nous utiliserons comme exemple un laser commun et une photorésistance LDR1000. Ces deux composants sont assez communs et abordables.

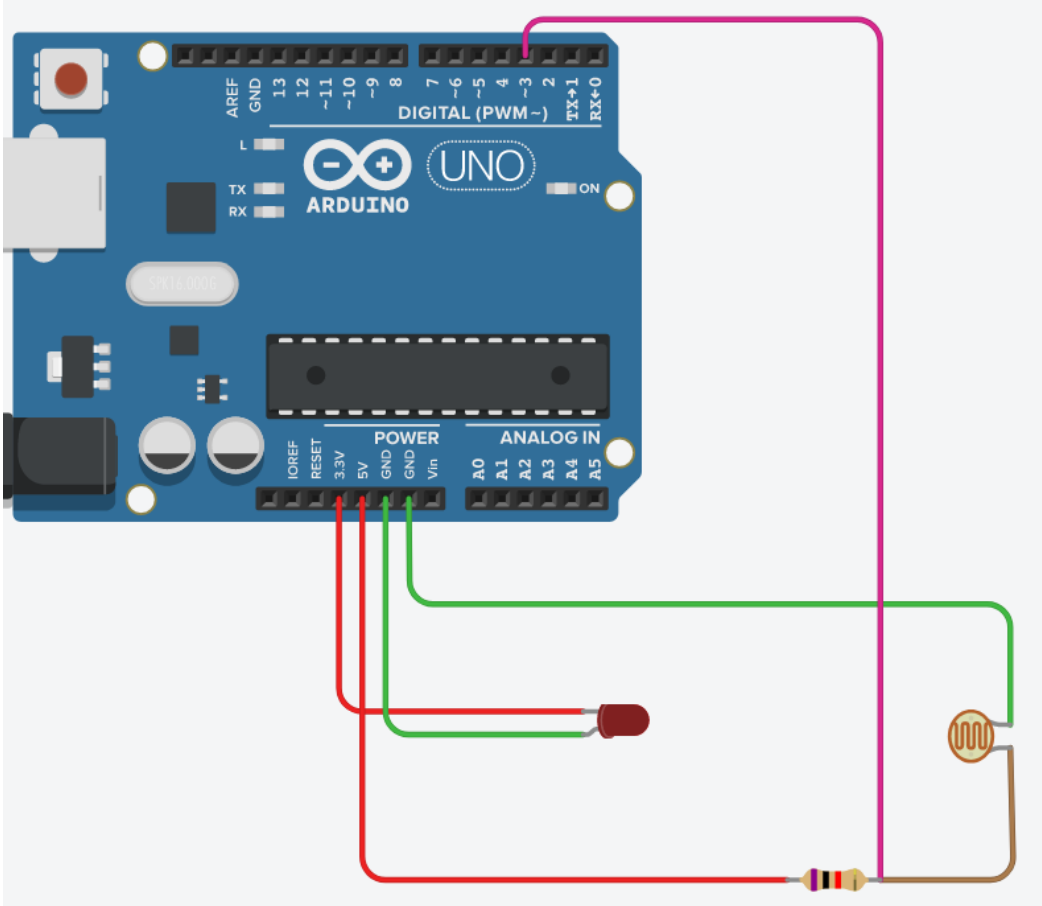

FIGURE 1 – Exemple de branchement

On peut observer que le montage est assez simple :

- Le laser (modélisé ici par une LED) a seulement besoin d'être alimenté. Ici l'alimentation est de 3,3V mais celle-ci dépend de votre laser, on peut donc avoir besoin de 5V.
- La photorésistance est en série avec une résistance afin de pouvoir faire un pont diviseur de tension. La tension aux bornes de la résistance est donc lue sur le pin A3 (qui devra être paramètré en entrée).

Comme il a été stipulé dans le tome 1, le capteur laser est un TOR (Tour Ou Rien). Soit il y a un obstacle devant le laser, soit il n'y en a pas.

Pour rappel, le principe est le suivant : La résistance de la photorésistance évolue selon la luminosité, elle est élevée dans l'obscurité (de l'odre du MΩ) et faible quand elle est exposée à la lumière (quelques centaines de Ω).

Dans ce circuit, j'ai utilisé une résistance de 7kΩ mais la résistance à choisir dépend de l'endoit où le capteur est utilisé. La photorésistance peut être parasitée par l'éclairage ambiant de l'endoit où le système est utilisé, il faudra alors faire des tests afin de connaitre les valeurs que celle-ci peut prendre. La variation de resistance n'est pas immédiate même si elle est rapide, il faut y penser car cela pourrait parasiter nos mesures (si le système mesure deux inversions à cause de la plage de valeurs incertaines).

Le tout est de s'assurer que lorsqu'il n'y a pas d'obstacle, la tension lue aux bornes de la résistance est supérieure à 3V et dans le cas contraire, que la tension lue soit inférieure à 1.5V. En effet, ces deux valeurs sont les tensions seuils pour le basculement de l'état logique lu sur le pin en entrée. En dessous de 1,5V, on lira un état LOW et au dessus de 3V on trouvera l'état HIGH. Entre les deux, l'état que le carte arduino donnera est incertain et selon certains, aléatoire.

On a donc en entrée l'alimentation des deux circuits et en sortie l'état du Pin A3 qui sera soit LOW soit HIGH selon la valeur de la tension aux bornes de la résistance.

À Savoir

Une foi encore, le pin D3 a été choisi arbitrairement, tout autre pin paramètrable de la carte est utilisable.

Rappel : La formule du pont diviseur de tension est :

$$U = V_{alim} \frac{R_{étudié}}{R_{branche}}$$

A.2 Le code

Voici un exemple de code qui prend en compte les deux sens d'inversion si nécessaire.

Codage

```
1  void setup()
2
3    /* Numero de la broche d'entree */
4    const byte READ_PIN = 3; // Broche de lecture
5    int previous = LOW // En supposant que l'on commence avec un etat bas sur A3
         sinon mettre HIGH
6
7   /* Initialise le port serie */
8   Serial.begin(115200);
9
10  /* Initialise la broche */
11  pinMode(READ_PIN, INPUT);
```

Codage

```
1  void loop() {
2
3    reading = digitalRead(READ_PIN);
4
5    // Ici nous allons verifier s'il y a un une inversion d'etat : de 1 a 0 -- ou -- de
         0 a 1, on peut mettre un delay au cas ou on s'attend a detecter les obstacles
         pendant un certain temps pour eviter les soucis impliques par la plage de
         tension incertaines (entre 1,5V et 3V).
6
7    if (reading == HIGH && previous == LOW ) {
8      // Ce que vous voulez
9      previous = HIGH    // On verifie s'il y a eu une inversion puis on
           sauvegarde le dernier etat dans previous pour le reutiliser
10     }
11
12   if (reading == LOW && previous == HIGH){
13     // Ce que vous voulez
14     previous = LOW     // Meme maniere de fonctionner que pour le test
           precedent
15     }
16
17   delay(100);
18 }
```

B Le capteur d'humidité

B.1 Le branchement

Nous allons voir comment utiliser un capteur d'humidité. Pour cela, nous allons nous appuyer sur un capteur DHT22 qui permet de connaître la température et l'humidité entre 0 et 100% avec une précision de 2 à 5%

FIGURE 2 – Capteur DHT22 avec signification des broches

Ici, nous observons que ce système possède trois branches à connecter :

- Le GND et l'alimentation 5V.
- La broche DATA à connecter sur un pin paramétré en tant qu'entrée.

On utilisera deux librairies : DHT-sensor-library et Adafruit_Sensor toutes deux disponibles sur github. Nous allons étudier un montage avec un écran pour pouvoir lire la valeur de l'humidité.

On utilisera donc le branchement suivant comme exemple :

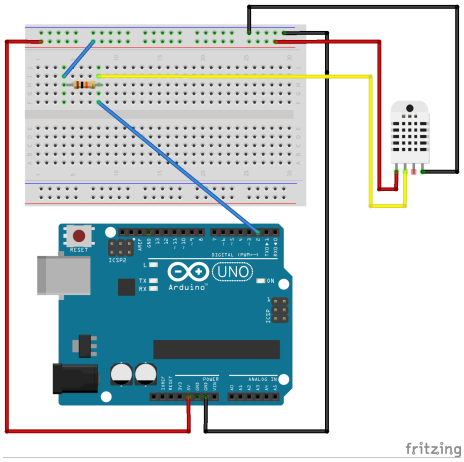

FIGURE 3 – Montage sans écran

Le montage est relativement simple. Il y a les branches d'alimentation et la branche DATA relié ici au pin 2. Lorsque l'on ajoute l'écran il y a plus de câbles mais il n'y a rien de compliqué en plus.

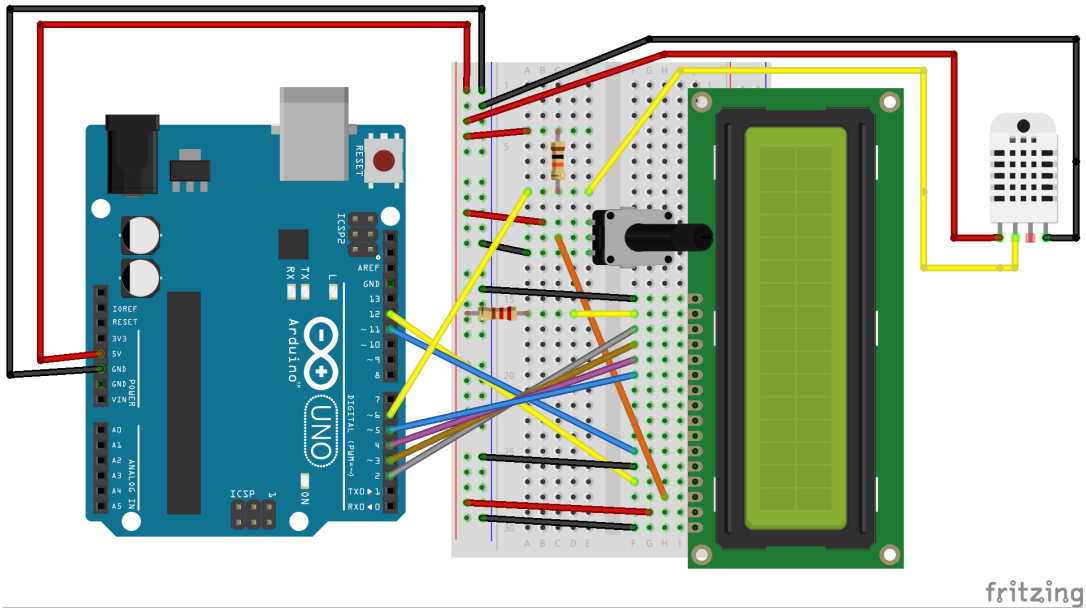

FIGURE 4 – Montage avec écran

B.2 Le code

Le code n'est pas particulièrement compliqué à comprendre. À l'aide de la libraire DHT nous allons obtenir les données mesurées puis il faudra les envoyer sur l'écran.

Le code qui suit est fait pour le montage deux, où le pin de lecture de la DATA est le pin 6.

Codage

```
#include <LiquidCrystal.h>
#include "DHT.h"

#define DHTPIN 6 // Notre pin d'entree

#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2); // Taille de l'ecran LCD

  lcd.print("Initialization");
  lcd.setCursor(0, 1);
  lcd.print("...");
}

void loop() {
  delay(2000); //Il faut attendre avant la mesure selon la datasheet du DHT pour
      qu'il s' initialise

  lcd.clear();
  lcd.setCursor(0, 0);

  float h = dht.readHumidity();
  float t = dht.readTemperature(); //On va ici garder l'information de
      temperature, si besoin

  if (isnan(h) || isnan(t)) { //On verifie s'il y a une erreur
    lcd.print("Failed to read from DHT sensor!");
    return;
  }

  float hic = dht.computeHeatIndex(t, h, false);

  lcd.print("H: "); //Partie Humidite
  lcd.print(h);
  lcd.print(" %");

  lcd.setCursor(0, 1);

  lcd.print(t); // Partie temperature

  lcd.print(" -- ");
  lcd.print(hic);
}
```

C Le capteur de contact

C.1 Le branchement

Pour les capteurs de fin de course (ou interrupteurs de contact), nous avons le choix parmi deux méthodes d'utilisation :

- Brancher le système en série avec un montage électrique plus conséquent (comme l'alimentation d'un moteur). De ce fait, si le contact n'est pas assurer, le circuit est ouvert et rien ne se passe.
- Lire la valeur en sortie du capteur et agir selon cette valeur.

La première méthode peut être retrouvée dans des systèmes munis d'une sécurité (comme des bancs d'essai qui ne tournent que si la porte est fermée) ou alors pour détecter des fin de course (pour un ouvre portail par exemple).

La seconde méthode est permet d'agir même lorsque l'interrupteur n'est pas actionné. On pourra donc retrouver ce système dans des systèmes automatisés (par exemple sur une ligne de production, lorsqu'une unité est terminée, elle va actionnée l'interrupteur qui va actionner un vérin pour la faire dtomber dans une boîte)

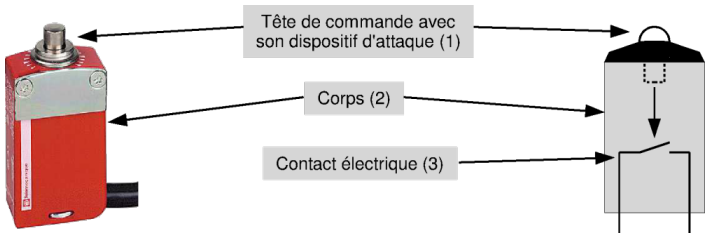

FIGURE 5 – Rappel principe du capteur

Nous allons ici présenter la seconde méthode d'utilisation qui la plus compliquée des deux mais qui reste simple en comparaison avec les autres capteurs vus jusqu'ici.

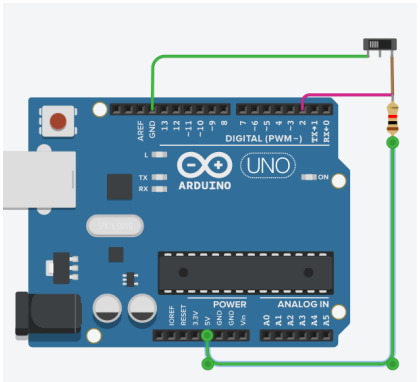

FIGURE 6 – Branchement du capteur

C.2 Code

Pour la partie codage, nous allons tout simplement tester l'état du pin sur lequel on va lire la tension aux bornes de la résistance.

Codage

```
1  void loop() {
2
3    reading = digitalRead(READ_PIN);
4
5    // Ici on suppose avoir parametre le pin lie a la resistance en pin d'entree et
         nomme READ_PIN
6
7    if reading == HIGH {
8      // Ce que vous voulez
9      }
10   else {
11     // Ce que vous voulez
12     }
13
14   delay(400);
15 }
```

D Conclusion

C'est ainsi que se termine ces tutoriels sur les capteurs. Le but était de vous donner les informations nécessaires pour l'utilisation des capteurs depuis le choix à faire jusque dans la programmation. Si vous maîtriser les quelques capteurs et cas particuliers développés jusqu'ici, vous ne devriez avoir aucun mal à prendre en main d'autres systèmes de mesure.

Il ne vous reste plus qu'à vous lancer ! Faites attentions à vous en manipulant et bon courage !