



Expérimenter

Chaine d'information

Communication et information

ARDUINO Système microprogrammé



| | | |
|-----------|------------------------------------|-----------|
| 1. | CONTEXTE | 5 |
| 1.1. | Description | 5 |
| 1.2. | Asservissement | 5 |
| 1.3. | Axe de découverte | 6 |
| 1.4. | Les matériels expérimentaux | 6 |
| 1.5. | La démarche | 7 |
| 1.6. | L'objectif | 8 |
| 2. | LES CAPTEURS | 9 |
| 2.1. | Le capteur de tension | 9 |
| 2.1.1. | DFR Analog Voltage Divider V2 | 9 |
| 2.1.2. | Principe de fonctionnement | 9 |
| 2.1.3. | Schéma théorique du capteur | 9 |
| 2.2. | Le capteur de courant | 10 |
| 2.2.1. | DFR Analog 20A CurrentSensor | 10 |
| 2.2.2. | Principe de l'ACS 712 | 10 |
| 2.2.3. | Schéma de câblage | 10 |
| 2.3. | Les capteurs de fin de course | 11 |
| 2.3.1. | DFR : Digital Push Button (Yellow) | 11 |
| 2.3.2. | Schéma théorique | 11 |
| 2.3.3. | Schéma de câblage | 11 |
| 2.4. | Le capteur de rotation | 12 |
| 2.4.1. | Encodeur Simple Cytron | 12 |
| 2.4.2. | Principe de fonctionnement | 12 |
| 2.4.3. | Schéma de câblage | 13 |
| 3. | LA COMMANDE DE PUISSANCE | 13 |
| 3.1. | Le driver moteur | 13 |
| 3.1.1. | Driver OSEPP MTD 01 | 13 |
| 3.1.2. | Schéma d'un pont en H | 15 |
| 3.1.3. | Schéma de câblage du driver | 15 |
| 4. | L'IHM | 16 |
| 4.1. | Le potentiomètre de consigne | 16 |
| 4.1.1. | Le schéma de câblage | 16 |

| | |
|--|-----------|
| 4.2. L'afficheur | 16 |
| 4.2.1. Afficheur I2C LCD2004 | 16 |
| 4.2.2. Principe du protocole I2C | 17 |
| 4.2.3. Fonctionnement d'un écran LCD (Liquid Cristal Display) | 18 |
| 4.2.4. Le Pixel | 18 |
| 4.2.5. Exemple de la technologie LCD Couleur | 18 |
| 4.2.6. Bibliothèque | 19 |
| 5. STRUCTURE ET COMMUNICATION DES SYSTEMES MICRO PROGRAMMES | 19 |
| 5.1. Structure d'un système micro programmé | 19 |
| 5.2. Système ARDUINO UNO | 19 |
| 5.3. Communication dans les systèmes micro programmé | 20 |
| 5.3.1. Communication série | 20 |
| 5.3.2. Communication parallèle | 20 |
| 5.3.3. Les ports de communication ARDUINO | 20 |
| 5.3.4. Langage de programmation ARDUINO | 21 |
| 6. L'IDE D'ARDUINO | 21 |
| 6.1. Vérifications système | 22 |
| 6.2. La programmation ARDUINO | 22 |
| 6.2.1. La structure d'un programme | 22 |
| 6.2.2. Le code | 22 |
| 6.3. Ecrire un programme | 23 |
| 6.3.1. Exemple 1 | 23 |
| 6.3.2. Exemple 2 | 25 |
| 6.3.3. Exemple 3 | 25 |
| 7. LA PROGRAMMATION | 27 |
| 7.1. Algorithme | 27 |
| 7.2. Le quantum | 28 |
| 7.2.1. Définition | 28 |
| 7.2.2. Exemple d'un capteur de tension ± 30 V | 29 |
| 7.2.3. Les capteurs utilisés | 30 |
| 7.3. Les codes ARDUINO | 30 |
| 7.3.1. Capteurs analogiques | 30 |
| 7.3.2. Résultat sur le Moniteur Série | 31 |
| 7.4. Potentiomètre / Moniteur série | 31 |
| 7.4.1. Le code / Format d'affichage des valeurs | 31 |
| 7.4.2. Moniteur série avec la fonction DEC | 31 |
| 7.4.3. Moniteur série avec la fonction BIN | 32 |
| 7.4.4. Moniteur série avec la fonction OCT | 32 |
| 7.4.5. Moniteur série avec la fonction HEX | 32 |

| | |
|--|-----------|
| 7.5. Potentiomètre / Fonction « map » | 32 |
| 7.5.1. Le code | 32 |
| 7.5.2. Moniteur série avec la fonction map | 33 |
| 7.6. Potentiomètre / Centrer la consigne | 34 |
| 7.6.1. Chronogramme | 34 |
| 7.6.2. Code | 34 |
| 7.6.3. Affichage moniteur série | 35 |
| 7.7. Capteur de tension 0 25 Vcc | 35 |
| 7.7.1. Code | 35 |
| 7.7.2. Affichage moniteur série | 36 |
| 7.8. Capteur de courant | 36 |
| 7.8.1. Code | 36 |
| 7.8.2. Affichage moniteur série | 37 |
| 7.9. Les capteurs TOR de fin de course | 37 |
| 7.9.1. Digital Push Button : Code | 37 |
| 7.9.2. Schéma de câblage d'un capteur TOR (Entrée numérique) | 38 |
| 7.9.3. Le code | 38 |
| 7.9.4. L'affichage sur le moniteur série | 39 |
| 7.10. Driver moteur | 39 |
| 7.10.1. Chronogramme | 39 |
| 7.10.2. Code de test | 40 |
| 7.10.3. Code avec consigne de vitesse | 41 |
| 7.11. Afficheur I2C | 42 |
| 7.11.1. Le code : | 42 |
| 7.12. Le codeur | 43 |
| 7.12.1. Algorithme | 43 |
| 7.12.2. Principe | 43 |
| 7.12.3. Le code | 44 |
| 7.13. Les variables | 45 |
| 7.13.1. char : caractère. | 45 |
| 7.13.2. int : entier standard. | 45 |
| 7.13.3. short : entier court. | 45 |
| 7.13.4. long : entier long | 45 |
| 7.13.5. unsigned | 46 |
| 7.13.6. volatile | 46 |
| 8. LE BANC DE TRAVELLING | 47 |
| 8.1. Diagramme de séquence | 47 |
| 8.2. Diagramme d'état | 47 |
| 8.3. Algorigrammes | 48 |
| 8.3.1. Déplacer | 48 |
| 8.3.2. Afficher la consigne de vitesse | 49 |

| | | |
|--------|---------------------------------|----|
| 8.3.3. | Régulation de vitesse | 50 |
| 8.3.4. | Afficher les grandeurs mesurées | 51 |
| 8.3.5. | Marche avant et marche arrière | 52 |

Les échanges et communications d'informations

Analyser le besoin, l'organisation matérielle et fonctionnelle d'un produit par une démarche d'ingénierie système

Caractériser les échanges d'informations

Instrumenter tout ou partie d'un produit en vue de mesurer les performances

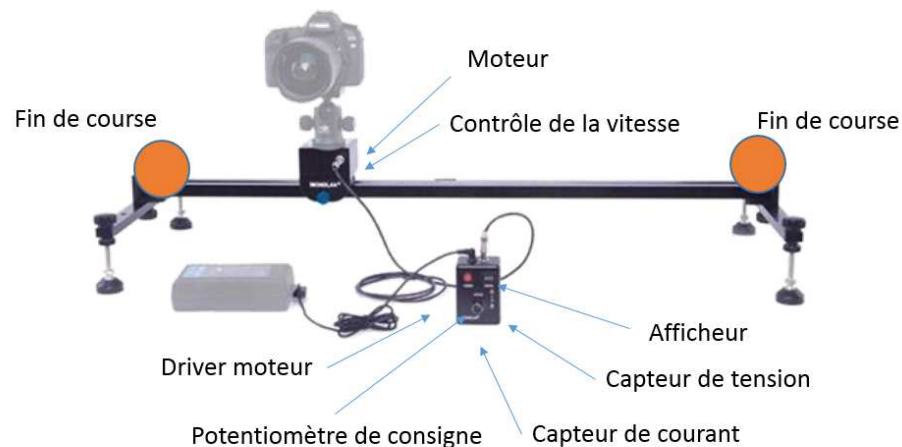
Relever les grandeurs caractéristiques d'un protocole de communication

1. Contexte

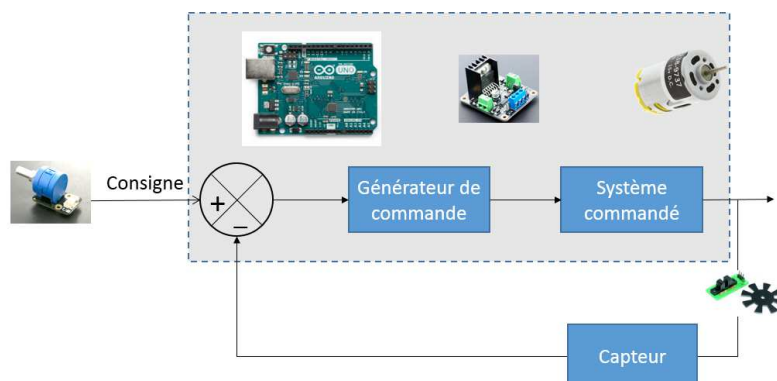
1.1.Description

Dans le cadre du développement **d'un asservissement** d'un banc de travelling, il est nécessaire d'effectuer des recherches sur :

- Le pilotage en vitesse d'un moteur à courant continu
- Le contrôle de vitesse du moteur
- Une commande de vitesse par potentiomètre
- Un affichage des caractéristiques du moteur (U(V), I(A), S(tr/min), Consigne de vitesse(%))
- Des fins de course

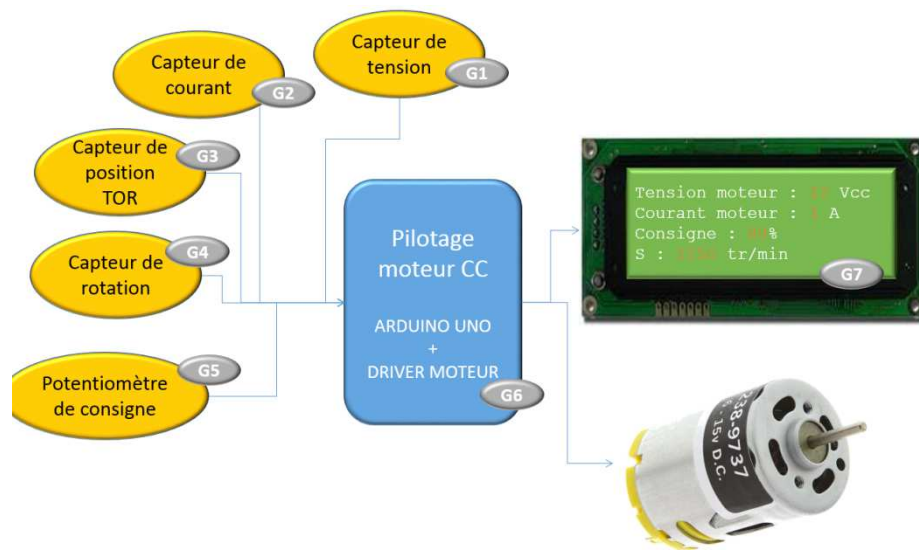


1.2.Asservissement



Le potentiomètre de consigne impose une commande de vitesse au moteur par l'intermédiaire du microcontrôleur et du driver. Le capteur de rotation (encodeur optique) mesure la vitesse effective du moteur. Le programme devra assurer une comparaison ces deux valeur afin d'assurer une vitesse constante quelle que soit la charge transportée.



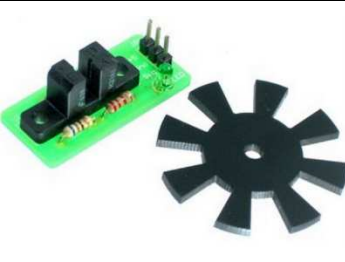
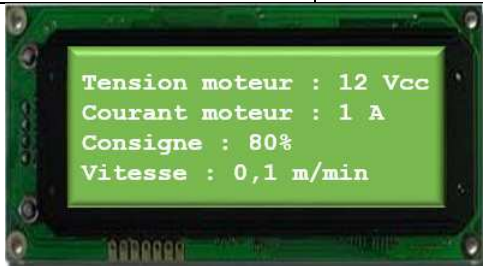

1.3.Axe de découverte



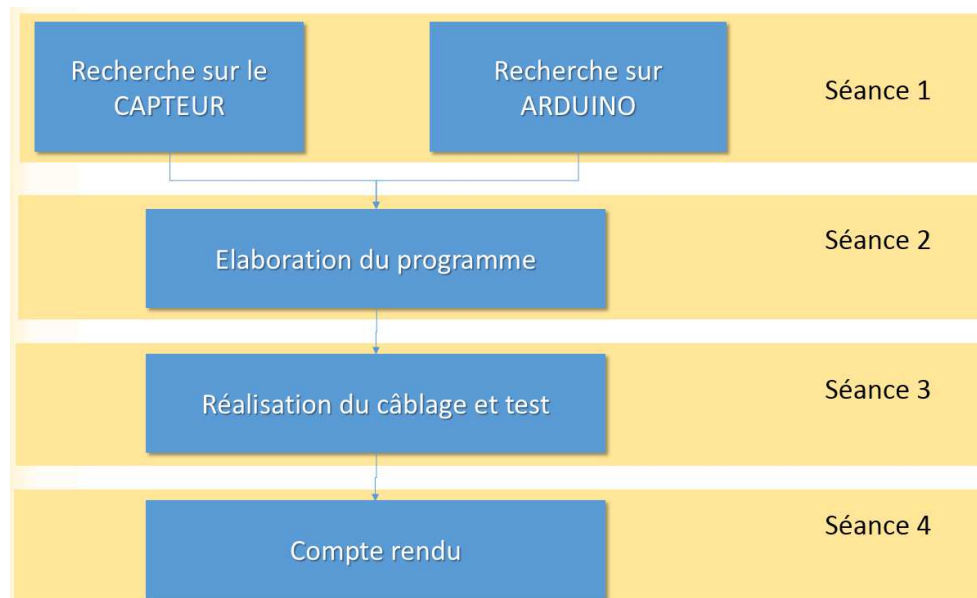
1.4.Les matériels expérimentaux

- Carte Arduino UNO
- Ordinateur avec IDE Arduino
- Un potentiomètre de consigne
- Un capteur de courant à effet hall
- Un capteur de tension
- Deux boutons poussoir (TOR)
- Un codeur incrémental (capteur de rotation)
- Un driver moteur
- Un afficheur

| | | |
|---|--|---|
|  |  |  |
| Gravity: Analog 20A CurrentSensor SKU:SEN0214 https://www.dfrobot.com/product-1570.html | Gravity:Digital Push Button (Yellow) SKU:DFR0029-Y https://www.dfrobot.com/product-73.html | Gravity:Analog Rotation PotentiometerSensor SKU:DFR0058 https://www.dfrobot.com/product-86.html |

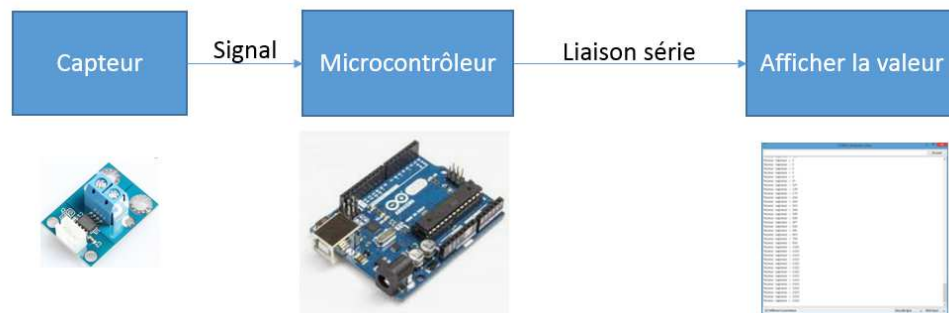
| | | |
|---|---|--|
|  |  |  |
| Gravity: Analog Voltage Divider V2 SKU:DFR0051 https://www.dfrobot.com/product-90.html | MDV 2x2A DC Motor Controller (L298N) SKU:DRI0002 https://www.dfrobot.com/product-66.html | Kit Encodeur Simple Cytron RB-Cyt-39 https://www.robotshop.com/eu/fr/kit-encodeur-simple-cytron.html |
|  |  | |
| Blindage Afficheur LCD 2004 3.3V avec Rétroéclairage Code produit : RB-Suf-63 https://www.robotshop.com/eu/fr/blindage-afficheur-lcd-2004-33v-avec-retroéclairage.html | Microcontrôleur Arduino Uno REV3 SMD Code de Produit : RB-Ard-112 https://www.robotshop.com/eu/fr/microcontrôleur-arduino-uno-rev3-smd.html | |

1.5.La démarche



1.6.L'objectif

Chaque groupe doit développer un programme qui permet d'afficher sur le moniteur série de l'IDE d'Arduino les variations de grandeurs mesurées par le capteur. Dans le cadre du développement, les matériels proposés sont associés à des pages internet qui décrivent leur usage.



2. Les capteurs

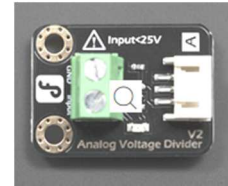
2.1. Le capteur de tension

Dans un but purement pédagogique, la tension aux bornes du moteur sera affichée. La valeur de la mesure sera comparée à la vitesse de rotation.

2.1.1. DFR Analog Voltage Divider V2

Ce capteur renvoie une valeur analogique (tension) comprise entre 0 et 5 Vcc. La tension maximale mesurable est de 25 Vcc.

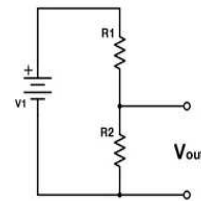
- Interface: Analogique
- Tension d'entrée (DC): Maximum 25Vcc, Minimum 0.0245V
- Détecte la tension de 0 Vcc de 25 Vcc
- Dimensions: 22x30mm



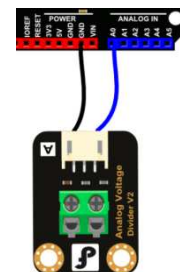
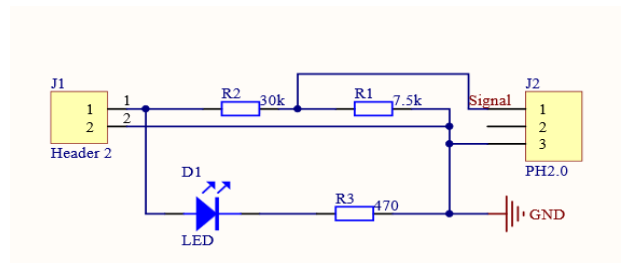
2.1.2. Principe de fonctionnement

Ce capteur est basé sur un pont diviseur de tension.

$$\frac{V_{OUT}}{V_{IN}} = \frac{R_2}{(R_1 + R_2)}$$



2.1.3. Schéma théorique du capteur



Nous retrouvons le pont diviseur de tension et la DEL qui nous informe de l'état du capteur TOR.

$$\begin{aligned} R_1 &= 30k\Omega \\ R_2 &= 7,5k\Omega \\ \frac{V_{OUT}}{V_{IN}} &= \frac{R_2}{(R_1 + R_2)} \\ \frac{V_{OUT}}{V_{IN}} &= \frac{7,5}{(30 + 7,5)} = \frac{1}{5} \end{aligned}$$

Remarque :

Ce capteur mesure des en tension de 0 à 25 Vcc, dans notre cas, la tension sera de ± 12 Vcc. Il faudra choisir un capteur capable de mesurer une tension en \pm .

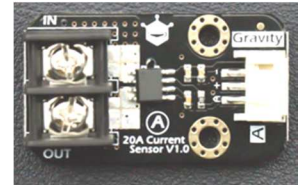
2.2. Le capteur de courant

La valeur du courant est un élément essentiel dans la chaîne de mesure. En effet, lors de l'utilisation du produit, ce sera la seule image du couple fourni par le moteur. Nous intégrerons dans la programmation finale une intensité maximum à ne pas dépasser afin de sécuriser le banc de travelling en cas de blocage.

2.2.1. DFR Analog 20A CurrentSensor

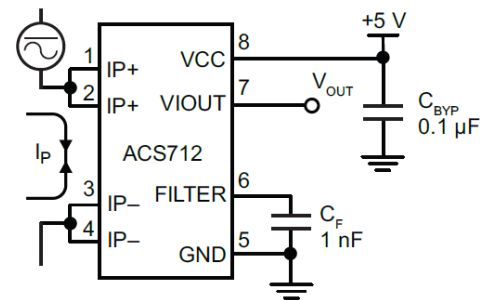
Ce capteur de courant est basé sur le composant ACS 712. C'est un capteur de courant à effet hall.

- Tension d'alimentation : 5 Vcc
- Courant mesuré: 0 ~ $\pm 20A$ DC, 0 ~ 17A (RMS) AC
- Tension du circuit mesuré: 220V AC, 311V DC
- Erreur relative: $\pm 3\%$
- Courant consommé maximum : 13 mA
- Dimensions : 39x22x17mm
- Masse: 18g

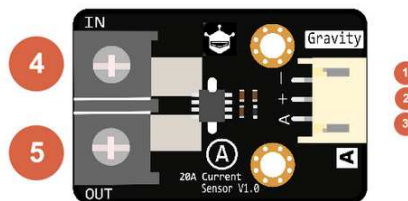


2.2.2. Principe de l'ACS 712

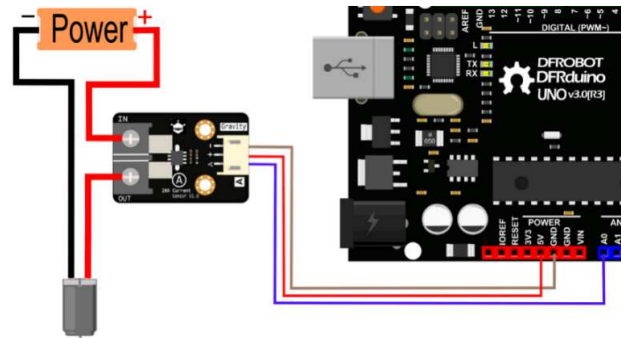
L'ACS712 émet un signal analogique V_{OUT} qui varie linéairement avec le courant alternatif ou continu unidirectionnel ou bidirectionnel échantillonné I_p dans la plage spécifiée. Il peut être nécessaire de mettre en place des condensateurs de filtrage C_F pour éviter le bruit qui dénature le signal.



2.2.3. Schéma de câblage



| Num | Label | Description |
|-----|-------|--------------------------|
| 1 | - | GND |
| 2 | + | 5V Input |
| 3 | A | Signal Output |
| 4 | IN | Measuring current Input |
| 5 | OUT | Measuring current Output |



Pour évaluer ce capteur il faut l'insérer dans un circuit de puissance, l'alimentation d'un MCC par exemple.

2.3. Les capteurs de fin de course

Ces capteurs permettront de stopper le travelling. Seul l'utilisateur pourra commander le démarrage dans le sens inverse. Cette commande de démarrage devra être assurée par un capteur TOR sur l'IHM.

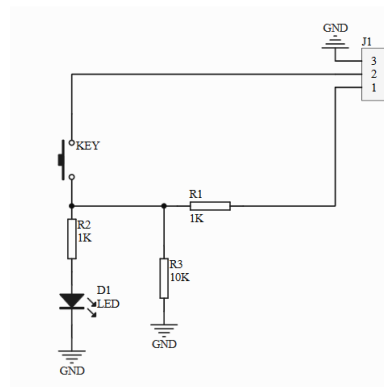
2.3.1. DFR : Digital Push Button (Yellow)

Pour les essais nous utiliserons des capteurs DFR équipé d'une DEL

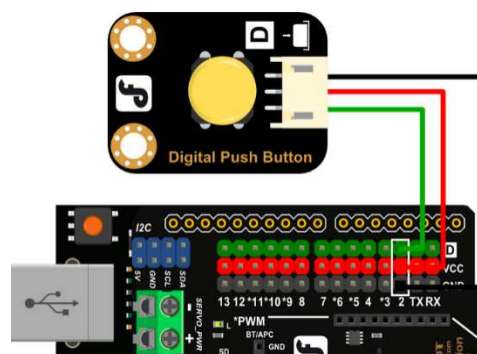
- Tension: 3.3V to 5V
- Interface: **Digitale**
- Dimensions : 22x30mm



2.3.2. Schéma théorique



2.3.3. Schéma de câblage

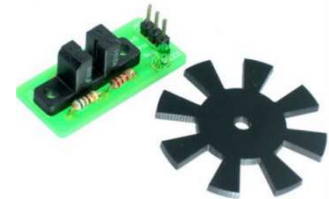


2.4. Le capteur de rotation

Dans un but purement pédagogique, la valeur de la vitesse sera affichée. La valeur nous permettra de corréler les résultats de vitesse avec la tension aux bornes du moteur et la consigne donnée par le potentiomètre afin d'asservir le mouvement

2.4.1. Encodeur Simple Cytron¹

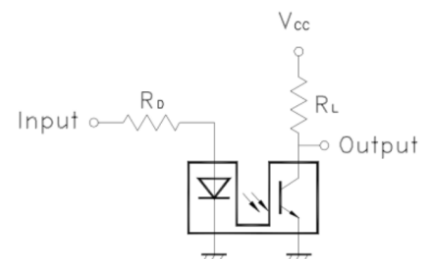
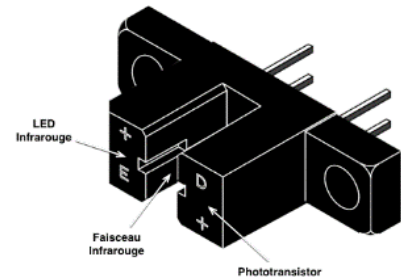
- Carte de circuit imprimé équipée d'un capteur à fourche et d'un connecteur
- Interface simple à 3 broches
- DEL verte intégrée servant d'indicateur
- Courant consommé : 40 mA
- Jusqu'à 1 KHz (1 000 pulsations/s)



2.4.2. Principe de fonctionnement

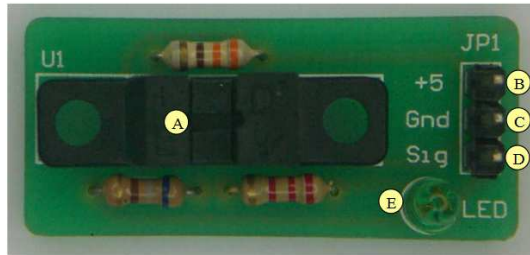
Ce **kit encodeur rotatif simple de Cytron** est équipé avec un disque rainuré (8 emplacements) et une carte capteur à simple interface. L'encodeur rotatif est *un capteur* ou *un transducteur* utilisé pour convertir les données de mouvement rotatif en une série de pulsations électriques lisibles par le contrôleur. Le capteur à fourche est un couple Led et Phototransistor qui délivre une tension lorsque le signal est passant. Le disque rainuré fourni a un diamètre extérieur de 35 mm avec 8 emplacements *fournissant 16 transitions*.

Les modules codeurs rotatifs peuvent être utilisés pour lire la position et la vitesse du moteur. Il peut être connecté à tout microcontrôleur au travers d'une embase à 3 broches. Le capteur de faisceau optique détecte les rainures manquantes du disque rainuré, et génère un train d'impulsions. Ces modules nécessitent du + 5 V et la masse pour être alimentés, et fournissent une sortie de 0 à 5 V. Ils fournissent une sortie à + 5 V lorsque leur faisceau est bloqué, et une sortie à 0 V quand leur faisceau est débloqué. Votre microcontrôleur peut simplement lire les trains d'impulsions à 0-5-0 V.

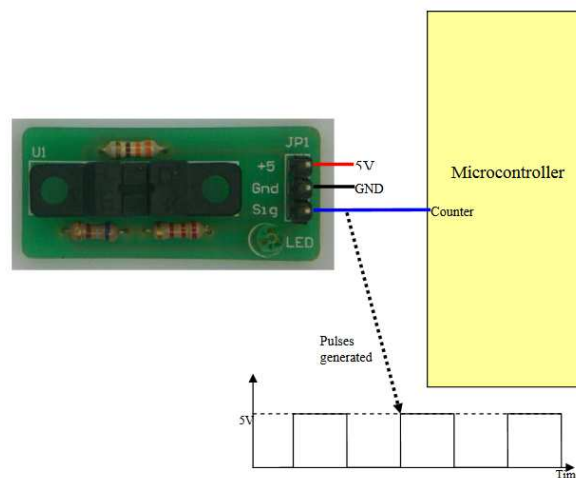


¹<http://www.robotshop.com/media/files/pdf/users-manual-re08a.pdf>

2.4.3. Schéma de câblage



| Label | Function | Label | Function |
|-------|---------------------------|-------|----------------------------|
| A | Optical Sensor | D | Signal output/pulse output |
| B | +5 input supply | E | Indicator LED |
| C | Ground/Negative of supply | | |



3. La commande de puissance

3.1. Le driver moteur

Le driver permet la commande de la puissance de notre système. C'est l'*interface* entre la partie commande et la partie puissance. La carte Arduino UNO est capable de piloter un moteur mais il faut prendre garde au courant consommé. Le driver utilisé permet de piloter un moteur qui consomme un courant maxi de 2 A. Le sens de rotation et la vitesse du moteur sur 8 bits (de 0 à 255) sont programmables.

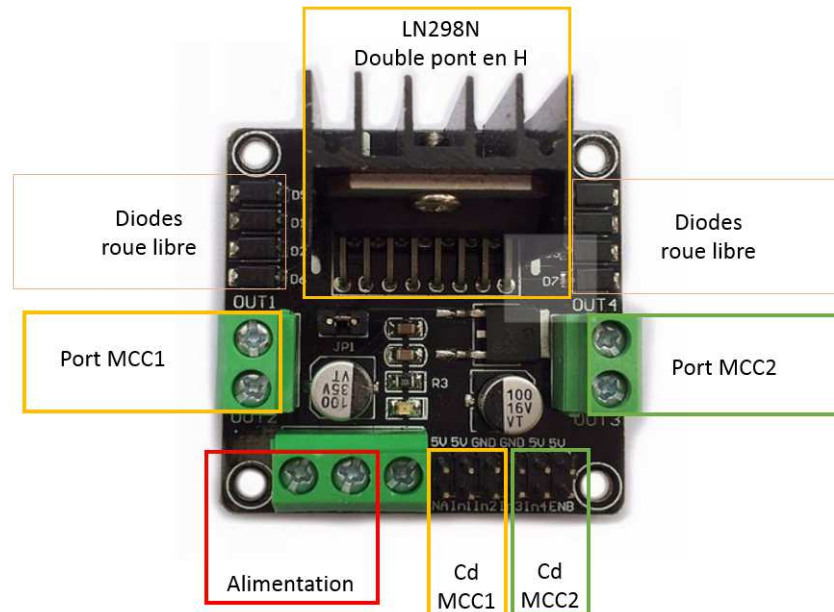
Rappel :

Les sorties numériques de Arduino UNO peuvent supporter un courant maxi de 40 mA, le courant total ne pouvant excéder 200 mA.

3.1.1. Driver OSEPP MTD 01

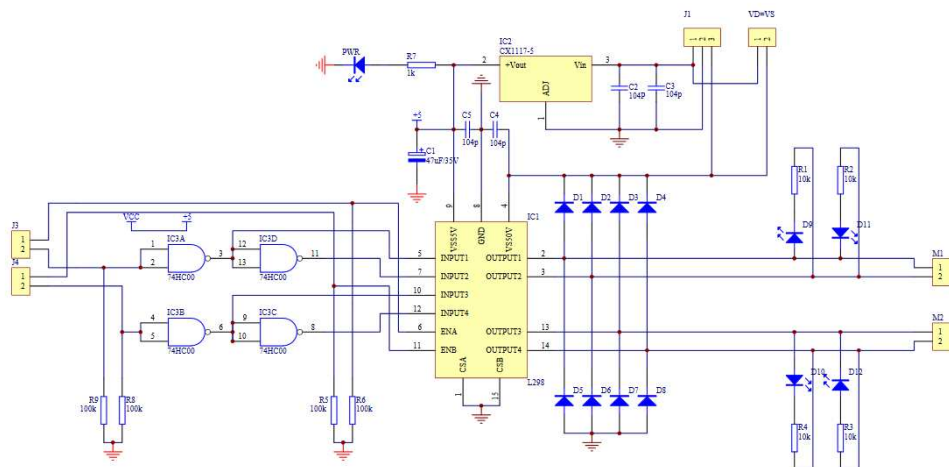
Il s'agit d'un contrôleur à double pont en H ($4,8 < U < 46V$, $I_{max} 2A$). Il peut supporter un courant de 2 A. Il est basé sur le composant L298N qui permet la commande d'un moteur pas à pas quatre fils ou deux moteurs CC en vitesse et en sens de rotation.

C'est une commande MLI (PWM) pilotée par un microprocesseur et un double pont en H. Le dispositif MIL (Modulation de largeur d'impulsion) est une alimentation intermittente à haute fréquence du moteur (environ 20 kHz). Un pont en H piloté par des transistors de puissance permet l'usage du MCC dans les deux sens. Il est équipé de diodes (roue libre) afin de dissiper les courants induits lors de l'arrêt de l'alimentation du MCC.

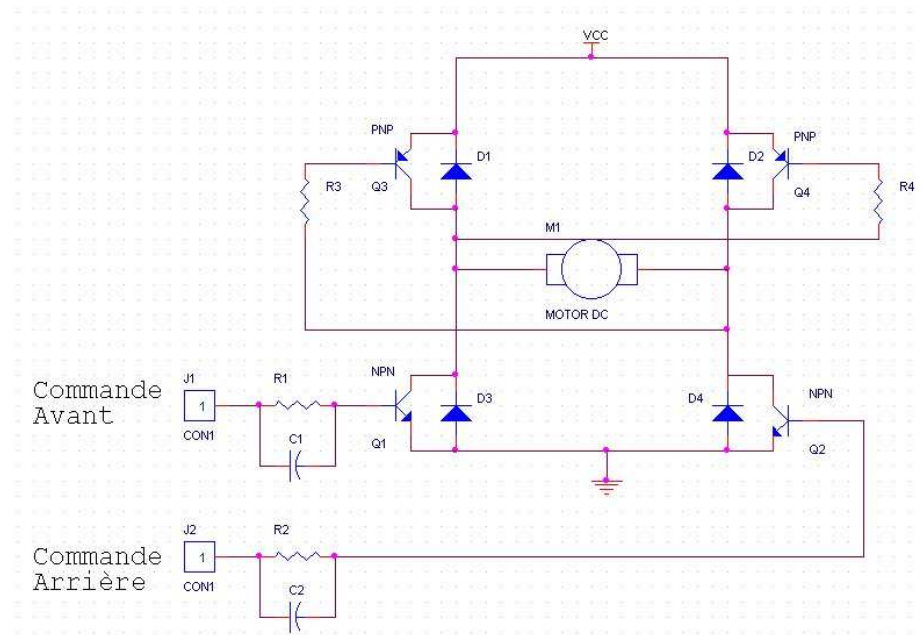


Remarques : Cd MCC1

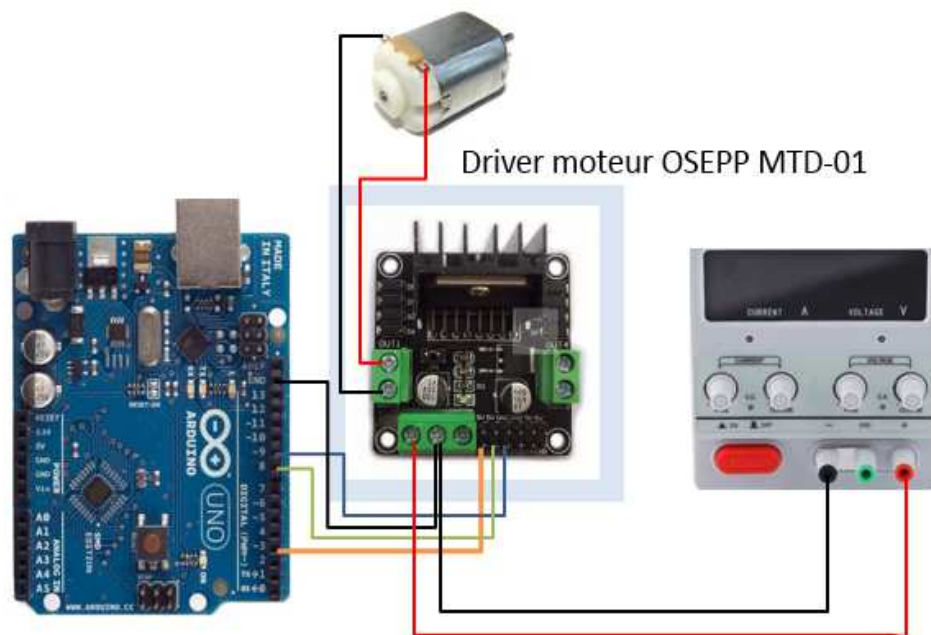
- ENA : Commande de la vitesse du moteur codée sur 8 bits (0 à 255)
- IN1 : commande de rotation en marche avant
- IN2 : commande de rotation en marche arrière



3.1.2. Schéma d'un pont en H



3.1.3. Schéma de câblage du driver

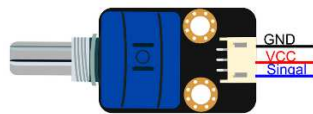


4. L'IHM

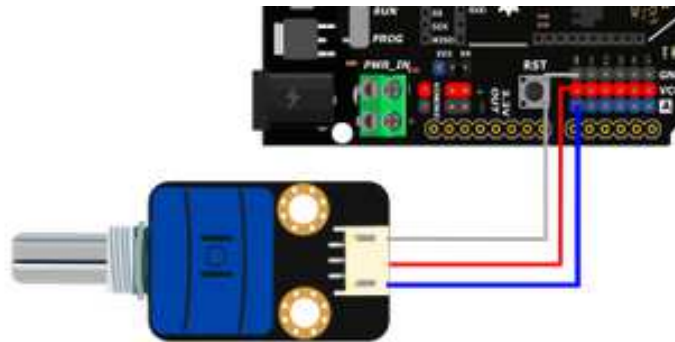
4.1. Le potentiomètre de consigne

Dans un but purement pédagogique, la consigne donnée avec le potentiomètre sera affichée. La valeur nous permettra de corréler les résultats de vitesse avec la tension aux bornes du moteur et la vitesse de rotation effective. Le potentiomètre utilisé est un potentiomètre 10 tours.

4.1.1. Le schéma de câblage



Noir : 0 V
Rouge : + 5 VCC
Bleu : Signal de sortie



4.2. L'afficheur

L'afficheur devra permettre d'informer l'utilisateur :

- La tension aux bornes du moteur en Volts
- Le courant consommé en Ampères
- La consigne en %
- La vitesse de rotation en tr/min

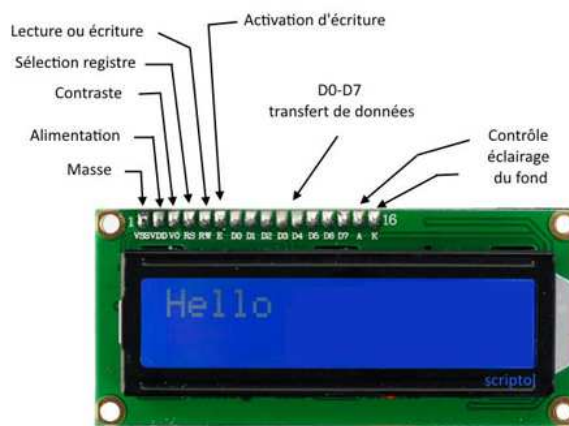
4.2.1. Afficheur I2C LCD2004



Les écrans LCD standards nécessitent plusieurs E / S sur le contrôleur. Cela restreint les autres fonctions du contrôleur. L'écran LCD2004 est muni d'un bus I2C². Il s'agit d'un bus série (Bus série, 8 bits, bidirectionnel) haute performance qui n'a que deux lignes de signal bidirectionnelles, la ligne de données série (SDA) et la ligne d'horloge série (SCL). Il est basé sur le principe « maître esclave ».

Ce protocole permet la communication entre des composants électronique très divers grâce à **seulement trois fils** :

- Signal de donnée : SDA
- Signal d'horloge : SCL
- Signal de référence électrique : masse



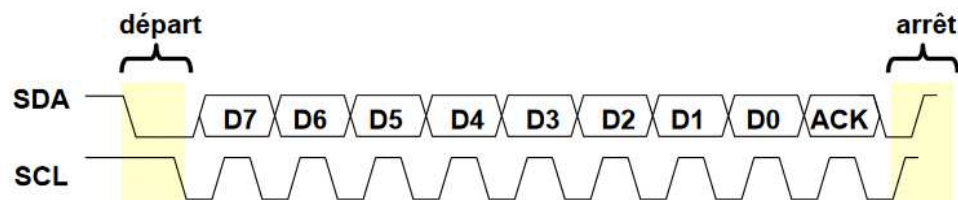
Ecran standard



Ecran I2C

4.2.2. Principe du protocole I2C

1. Le maître transmet le bit de poids fort D7 sur SDA
2. Il valide la donnée en appliquant un niveau '1' sur SCL
3. Lorsque SCL retombe à '0', il poursuit avec D6, jusqu'à ce que l'octet complet soit transmis
4. Il envoie le bit ACK à '1' en scrutant l'état réel de SDA
5. L'esclave doit imposer un niveau '0' pour signaler que la transmission s'est déroulée correctement
6. Le maître voit le '0' (collecteur ouvert) et peut passer à la suite.....



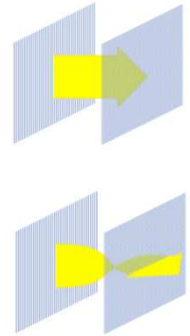
² I2C : Inter Integrated Circuit. Développé au début des années 80 par Philips Semi-conducteur pour permettre de relier facilement à un microprocesseur les différents circuits d'un téléviseur moderne.

4.2.3. Fonctionnement d'un écran LCD (Liquid Cristal Display)

Les cristaux liquides sont des matériaux à la fois des solides et des liquides. Ils s'orientent en fonction de la tension. Ce composant n'émet aucune lumière mais se comporte comme un « interrupteur optique ». C'est pour cette raison que les écrans disposent d'un rétro-éclairage.

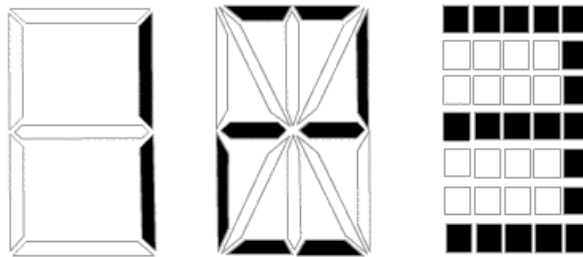
Combiné à une source de lumière, une première plaque striée agit comme un filtre, ne laissant passer que les composantes de la lumière dont l'oscillation est parallèle aux rainures.

En l'absence de tension électrique, la lumière est bloquée par la seconde plaque, agissant comme un filtre polarisant perpendiculaire.

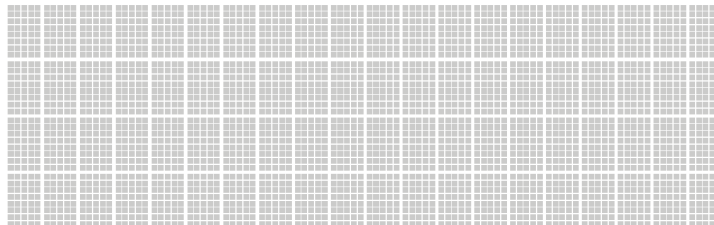


4.2.4. Le Pixel

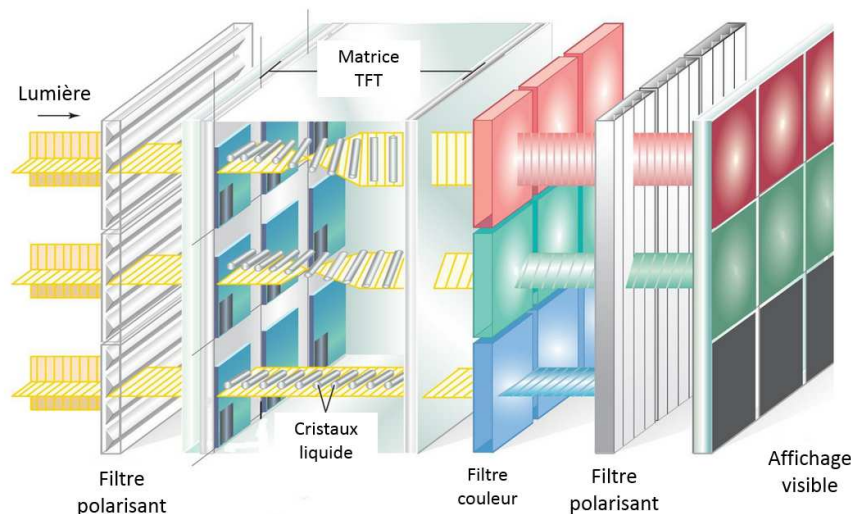
Un affichage est composé de pixel ou de segments, c'est l'élément unitaire de l'affichage.



L'afficheur utilisé possède 4 lignes de 20 caractères composés d'une matrice de 8 x 5 pixels.



4.2.5. Exemple de la technologie LCD Couleur



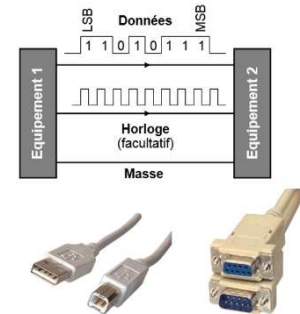
5.3. Communication dans les systèmes micro programmé

5.3.1. Communication série

Dans une transmission **série** les bits sont envoyés **les uns derrière les autres** sur un unique support de transmission.

L'horloge peut être câblée ou non entre l'émetteur et le récepteur. Dans ce cas, la transmission est dite **synchrone** et **asynchrone** dans le cas contraire.

Le débit est faible mais les distances couvertes sont importantes.

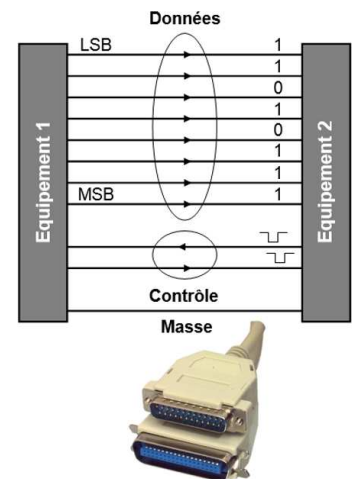


5.3.2. Communication parallèle

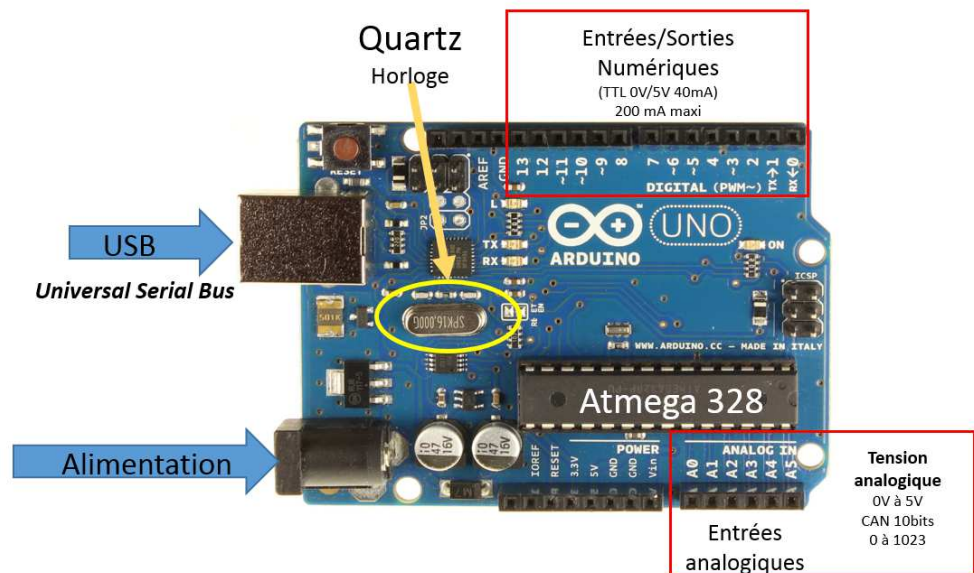
Une transmission **parallèle** (octet par octet généralement) a pour effet d'accroître le **débit**.

Les bits de la donnée sont transmis **simultanément**.

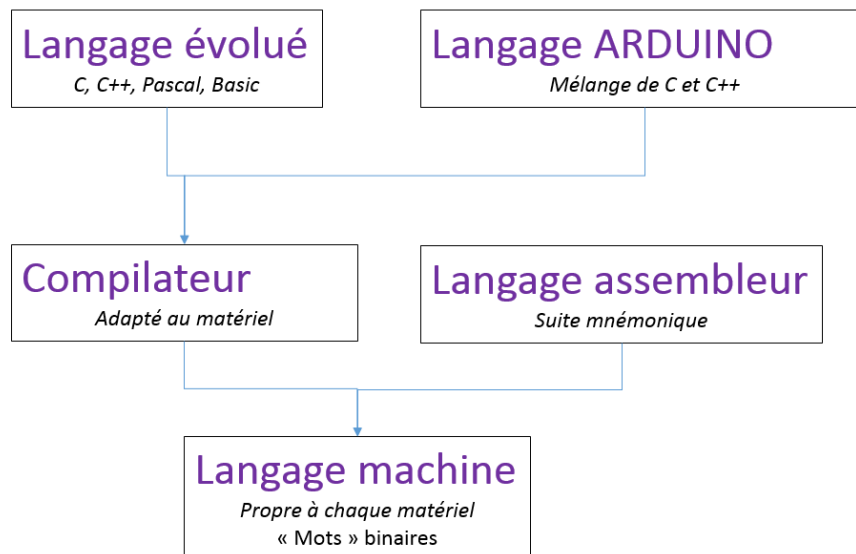
Les équipements à relier comportent autant de fils de données que de bits à transmettre, un ou plusieurs fils de **contrôle** cadencent la transmission.



5.3.3. Les ports de communication ARDUINO



5.3.4. Langage de programmation ARDUINO

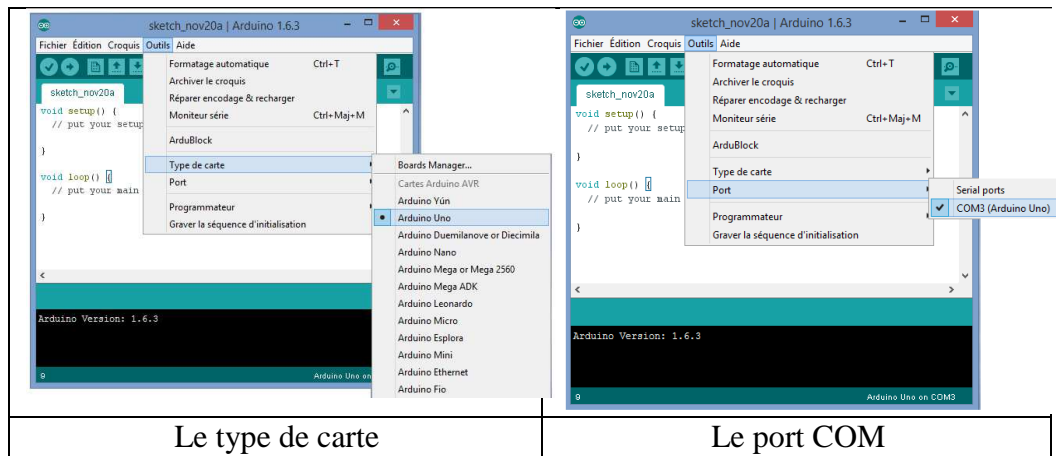


6. L'IDE³ d'ARDUINO



³ Integrated Development Environment : Environnement de développement intégré

6.1. Vérifications système



6.2. La programmation ARDUINO

6.2.1. La structure d'un programme

- Un **premier bloc** permettra d'affecter des variables aux entrées et aux sorties. Cette solution améliore la lecture d'un programme et permet de faciliter les modifications.
- Un **deuxième bloc** qui sera la fonction « `void setup() { }` ». Elle permet d'écrire les fonctions d'initialisation du système et d'affecter les entrées et les sorties.
- Un **troisième bloc** qui sera la fonction « `void loop() { }` ». Elle permet d'écrire le programme désiré qui sera une suite de fonction.

```
const int BP=3;
const int LED=5;
int EBP;

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(BP, INPUT);
  EBP=HIGH;
}

void loop()
{
  EBP=digitalRead(BP);
  if (EBP==LOW) { digitalWrite(LED, HIGH); }
  else { digitalWrite(LED, LOW); }
}
```

6.2.2. Le code

| Fonctions | Codes |
|---|---|
| Effectuer un commentaire | // |
| Délimiter une zone de commentaire | /* Début du commentaire Fin du commentaire */ |
| Terminer un ordre, une fonction | ; |
| Déclarer une constante entière | const int |
| Nommer « Sortie » le port numérique 3 en constante entière | const int Sortie = 3 ; |

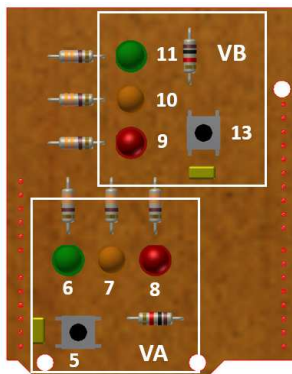
| | |
|---|--|
| Déclarer une variable interne entière nommée « EBP » | <code>int EBP ;</code> |
| Déclarer le port numérique 4 en sortie | <code>pinMode (4, OUTPUT⁴) ;</code> |
| Déclarer la variable « Sortie » en entrée | <code>pinMode (Sortie, INPUT⁵) ;</code> |
| Lire l'état de la variable « BP » | <code>digitalRead (BP) ;</code> |
| Piloter une variable « Moteur » en sortie | <code>digitalWrite (Moteur, HIGH⁶) ;</code> |
| Piloter une variable « Led_Rouge » en sortie | <code>digitalWrite (Led_Rouge, LOW⁷) ;</code> |
| Attendre un temps de 2 secondes | <code>delay (2000⁸) ;</code> |

6.3.Ecrire un programme

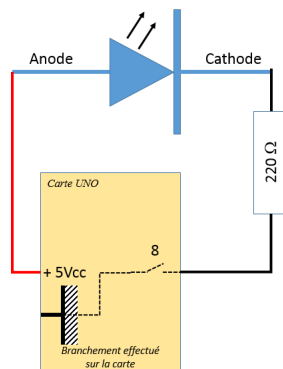
6.3.1. Exemple 1

Allumer une LED Rouge sur le port numérique 8 durant 1 seconde et l'éteindre durant 1 seconde.

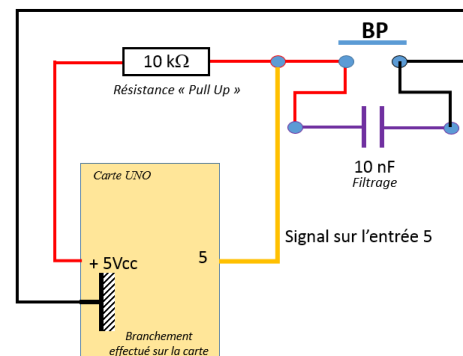
Pour débiter, imaginons un programme qui permet d'allumer une LED durant 1 seconde et de l'éteindre durant 1 seconde. Nous nous appuyons sur la maquette proposée.



La carte pour les essais.



Pour allumer la LED, il faut un 0Vcc sur la broche 8, donc un niveau BAS (LOW).



Le montage du bouton poussoir est effectué en série avec une résistance Pull up⁹ et un condensateur¹⁰.

⁴ Sortie

⁵ Entrée

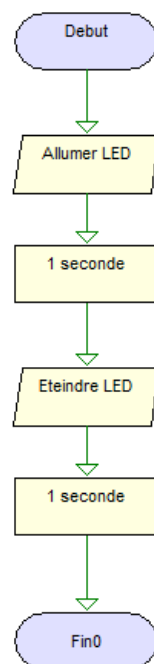
⁶ Haut : correspond à l'état 1 logique donc au +5Vcc sur la platine ARDUINO

⁷ Bas : correspond à l'état 0 logique donc au 0Vcc sur la platine ARDUINO

⁸ Le temps est donné en milli seconde.

⁹ « Résistance de tirage » : elle permet de générer un +5Vcc et un 0Vcc net.

¹⁰ Il permet le filtrage des rebonds mécaniques qui se transforment en parasites.



Déclaration de variable :

Déclarer une variable constante entière définissant la LED_R sur la broche8.

Dans la fonction setup :

Définir la variable LED_R comme une sortie « **OUTPUT** »

Dans la fonction loop

Activer la LED_R pendant 1 seconde

Désactiver la LED_R pendant 1 secondes

Code :

```

//Déclaration de la variable sur le port 8
const int led_rouge=8;

void setup() {
  //déclaration de la led_rouge en sortie
  pinMode(led_rouge,OUTPUT);}

void loop() {
  //Ecriture led_rouge éteinte
  digitalWrite(led_rouge,HIGH);
  //Durée de 1 seconde
  delay(1000);
  //Ecriture led_rouge allumée
  digitalWrite(led_rouge,LOW);
  //Durée de 2 secondes
  delay(1000);}
  
```

6.3.2. Exemple 2

Allumer une LED Rouge (branchée sur le port numérique 8) LED_R par l'appui sur un bouton poussoir BP (branché sur le port numérique 5). La LED s'éteint lorsque l'appui sur le bouton poussoir cesse.

Code nouveau : **if (condition) { ...ordres..... }**
 else { ...ordres... }

Déclaration de variable :

Déclarer une variable constante entière définissant la **LED_R** sur la broche **8**.

Déclarer une variable constante entière définissant la **BP** sur la broche **5**.

Déclarer une variable entière définissant l'état *électrique* du bouton poussoir **EBP**.

Dans la fonction setup :

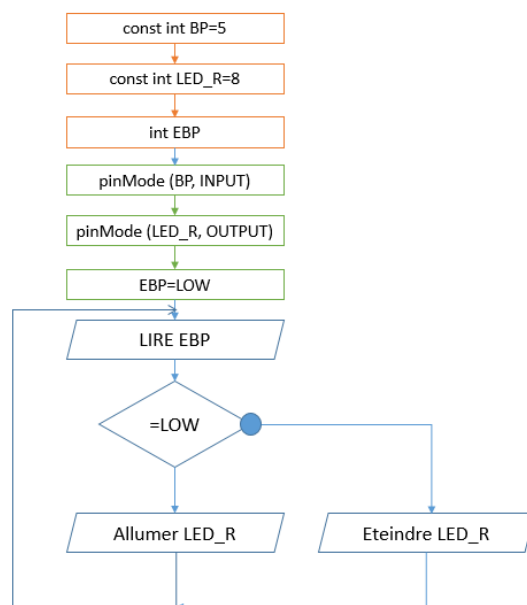
Définir la variable LED_R comme une sortie « **OUTPUT** »

Définir la variable BP comme une entrée « **INPUT** »

Dans la fonction loop

Si l'appui sur BP est effectif, activer la LED_R

Sinon désactiver la LED_R



Le code :

```

//Déclaration des variables
const int BP=5;
const int LED_R=8;
int EBP;

void setup() {
  pinMode(BP, INPUT);
  pinMode(LED_R, OUTPUT);
  EBP=LOW;
}

void loop() {
  EBP=digitalRead(BP);
  if (EBP==LOW) {digitalWrite(LED_R, LOW); }
  else {digitalWrite(LED_R, HIGH); }
}
  
```

6.3.3. Exemple 3

Allumer une LED Rouge (branchée sur le port numérique 8) LED par l'appui sur un bouton poussoir BP (branché sur le port numérique 5). La LED s'éteint lors d'un nouvel appui sur le bouton poussoir.

Code nouveau : **Compteur** BPC
 Incréméntation BPC ++
Modulo % Le modulo est une opération qui permet d'obtenir le reste d'une division.¹¹

Déclaration de variable :

Déclarer une variable constante entière définissant la **LED** sur la broche **8**.

Déclarer une variable constante entière définissant la **BP** sur la broche **5**.

Déclarer une variable entière définissant l'état *électrique* du bouton poussoir **BPE=0**.

Déclarer une variable entière définissant un compteur d'appui sur le bouton poussoir **BPC=0**.

Déclarer une variable entière définissant le dernier état *électrique* du bouton poussoir **DBP=0**.

Dans la fonction setup :

Définir la variable LED_R comme une sortie « **OUTPUT** »

Définir la variable BP comme une entrée « **INPUT** »

Dans la fonction loop

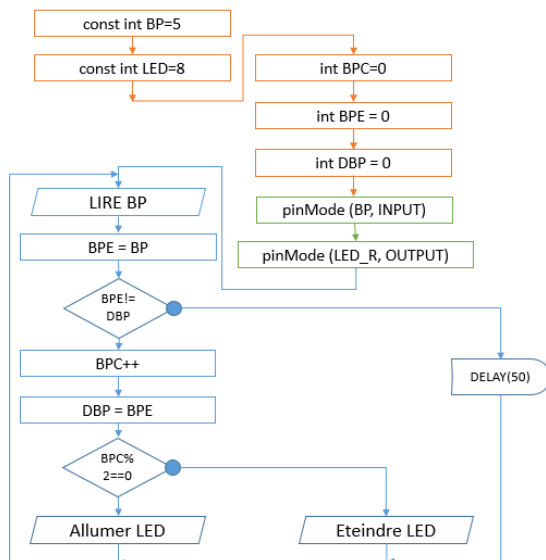
Affecter la valeur de BP à BPE

Si BPE est différent de DBP, et Si BDP est à l'état Haut, incrémenter BPC
 Sinon activer une temporisation de 50 ms.

Déclarer DBP=BPE

Si BPC est « pair », activer la LED

Sinon désactiver la LED.



Le code :

```
// Déclaration des variables
const int BP=5;
const int LED=8;
int BPC=0;
int BPE= 0;
int DBP = 0;

void setup() {
  pinMode(BP, INPUT);
  pinMode(LED, OUTPUT);
}

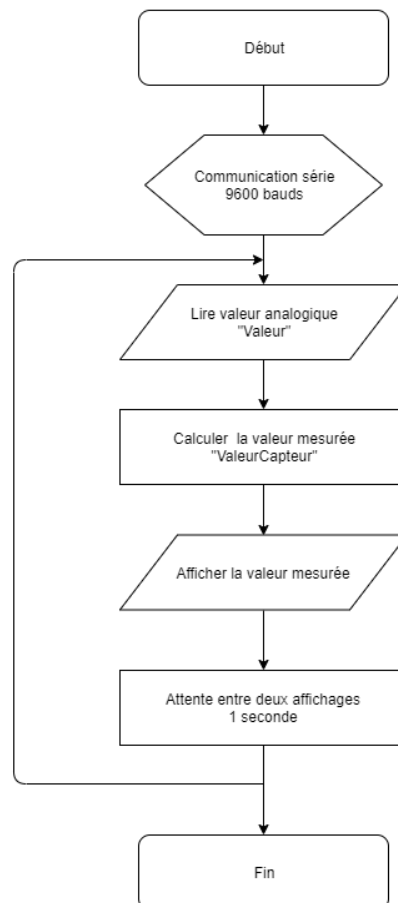
void loop() {
  BPE = digitalRead(BP);
  if (BPE != DBP) {if (DBP == HIGH){BPC++;}
                  else {}
                  delay(50);}

  DBP = BPE;
  if (BPC % 2 == 0){digitalWrite(LED, HIGH);}
  else {digitalWrite(LED, LOW);}
}
```

¹¹ Dans notre cas, nous effectuons un test de parité.

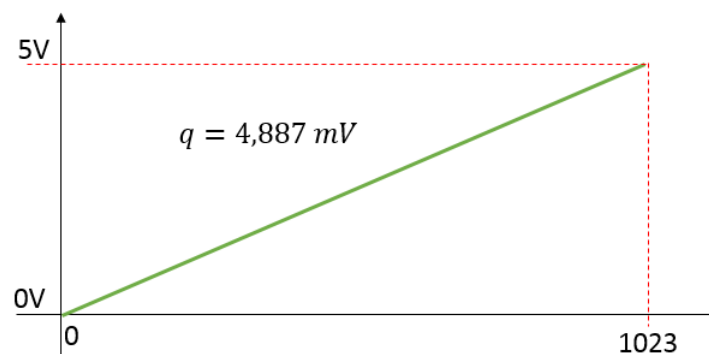
7. La programmation

7.1.Algorigramme



Les capteurs que nous utilisons sont analogiques. Ils seront branchés sur les ports analogiques de la carte UNO Arduino (A0, A1, A2, A3, A4, A5).

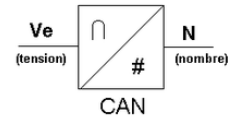
La valeur renvoyée par le capteur est une tension comprise entre 0 Vcc et 5 Vcc. Cette valeur est ensuite convertie par un CAN 10 bits (0 à 1023).



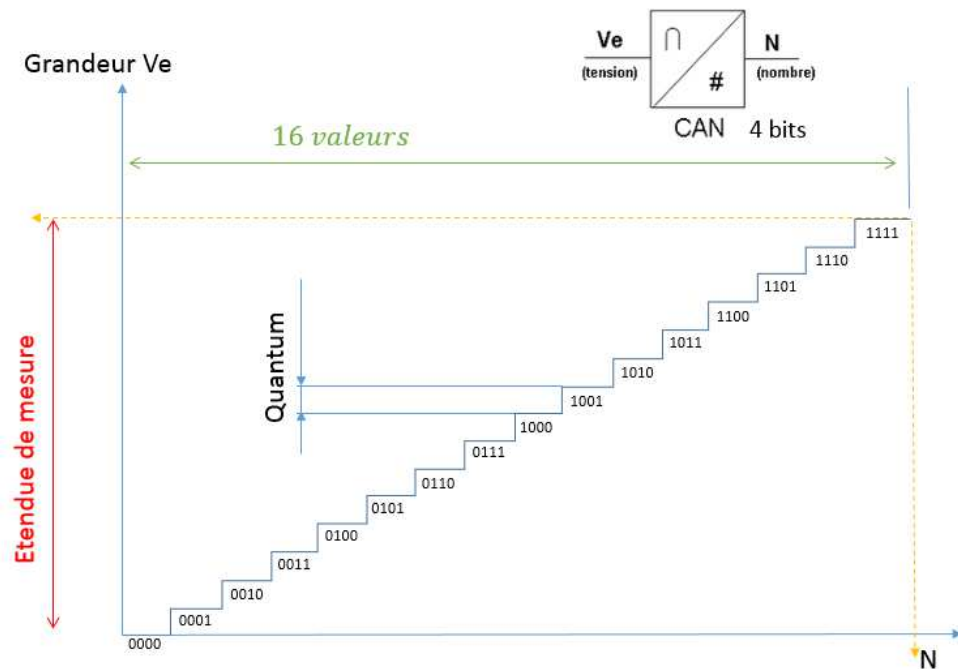
7.2. Le quantum

7.2.1. Définition

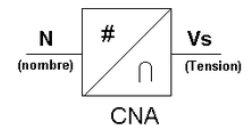
Un CAN (Convertisseur Analogique Numérique) est un dispositif qui transforme une information analogique en une information numérique. La résolution est donnée par la valeur du **quantum** « q ». La résolution est la plus petite variation en entrée correspond à un changement de code en sortie.



$$q = \frac{\text{Valeurmaxi}}{2^n - 1}$$

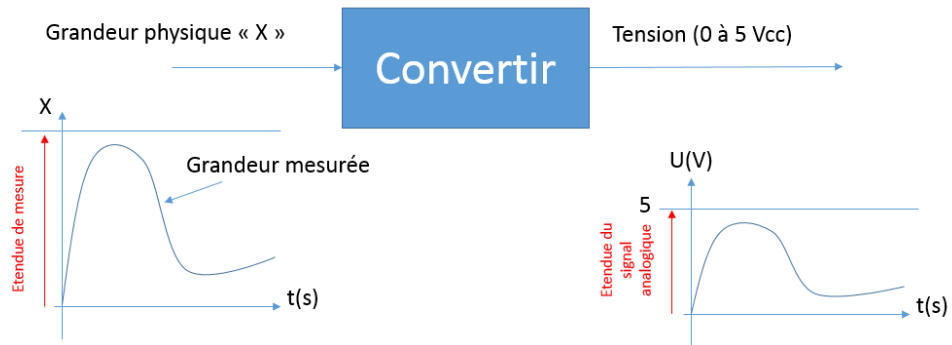


Un CNA (Convertisseur Numérique Analogique) est un dispositif qui transforme une information numérique en un signal analogique. L'information numérique se présente sous la forme d'un mot de n bits (8, 10, 12, 14, 16 bits en général). La conversion consiste à attribuer au nombre binaire une valeur analogique variant, en général de 0 (LSB *Last Significant Bit*) à une valeur maximale (valeur pleine échelle *full scale*). Bien évidemment, le nombre binaire maximum correspond à la valeur pleine échelle. Le LSB correspond à une valeur analogique appelée le **quantum** qui définit la plus petite variation analogique possible.



Un CNA est défini par sa résolution n (par exemple 12 bits) ; connaissant la sortie pleine échelle (10V par exemple) on peut alors calculer le quantum :

$$q = \frac{\text{Valeur maxi}}{2^n - 1}$$



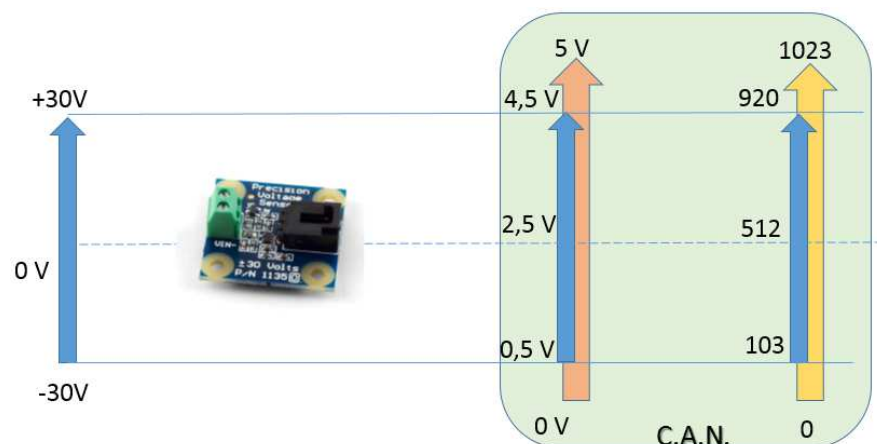
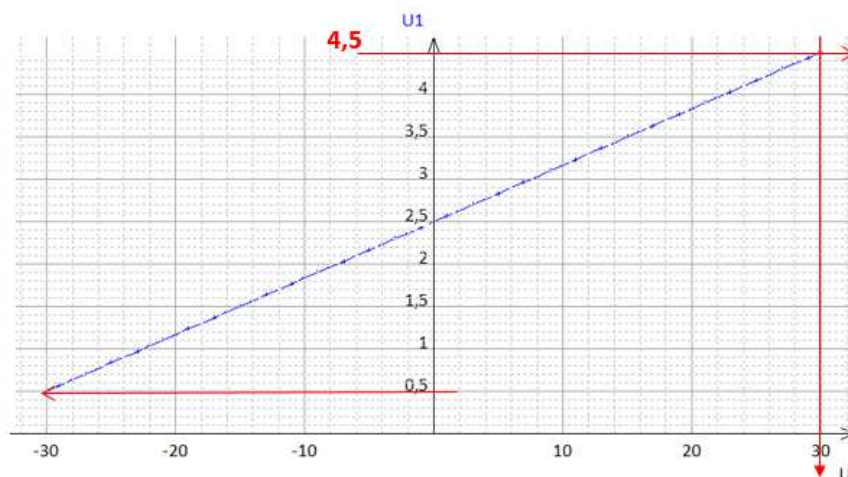
L'étendue de mesure du capteur doit être supérieure à l'étendue de la grandeur mesurée (mesurande).

L'étendue du signal issue du capteur doit être en correspondance avec le signal attendu par l'unité de traitement, dans notre cas de 0 à 5 VCC.

7.2.2. Exemple d'un capteur de tension ± 30 V¹²

La sensibilité du capteur est de 68,1 mV/V ($\pm 2\%$).

Le signal de sortie est compris entre 0,5 Vcc (-30 V) et 4,5 Vcc (+30 V).



¹²<https://www.phidgets.com/?&prodid=108>

7.2.3. Les capteurs utilisés

| | | sensibilité |
|--------------------|-----------|-------------|
| Capteur de tension | 0-30 V | 29,3 mV |
| Capteur de courant | $\pm 20A$ | 100 mV |
| Potentiomètre | 10 tours | 3,52° |

7.3. Les codes ARDUINO

7.3.1. Capteurs analogiques

Le code ci-dessous s'adapte aux différents capteurs dont les sorties sont analogiques (0 à 5 Vcc).

```
float ValeurCapteur;           // Déclaration de la variable ValeurCapteur
                                // correspondant à la valeur mesurée

void setup() {
  Serial.begin(9600);           // Ouverture de la liaison série
}

void loop() {
  int Valeur= analogRead(A0);    //Lecture de la tension nommée
                                // « Valeur »sur le port analogique A0 qui
                                // sera convertie par le CAN 10 bits

  ValeurCapteur=(Valeur*quantum); // Convertit la valeur issue du CAN en
                                // valeur mesurée et nommée
                                // « ValeurCapteur ».

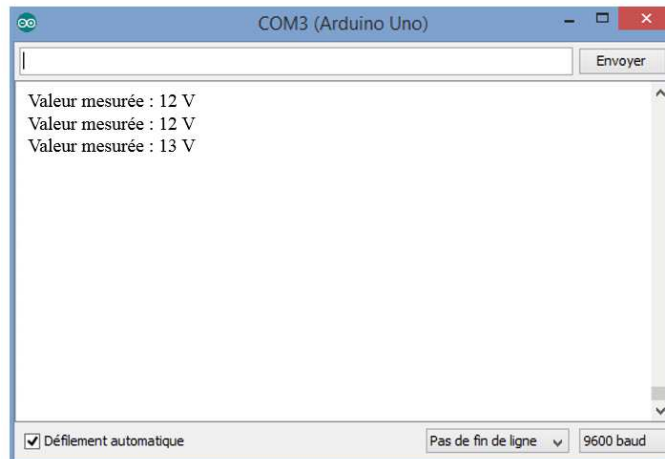
  Serial.print("Valeur mesurée : "); //Affiche sur le moniteur série la chaine de
                                // caractères « Valeur mesurée : »

  Serial.print(ValeurCapteur);     //Affiche sur le moniteur série la chaine de
                                // caractères correspondant à
                                // « ValeurCapteur »

  Serial.println(" unité");        //Affiche sur le moniteur série la chaine de
                                // caractère « unité » et renvoie à la ligne
                                // suivante

  delay(1000);                    //Attente de 1 seconde avant de poursuivre
}
```

7.3.2. Résultat sur le Moniteur Série



7.4. Potentiomètre / Moniteur série

7.4.1. Le code / Format d'affichage des valeurs

```

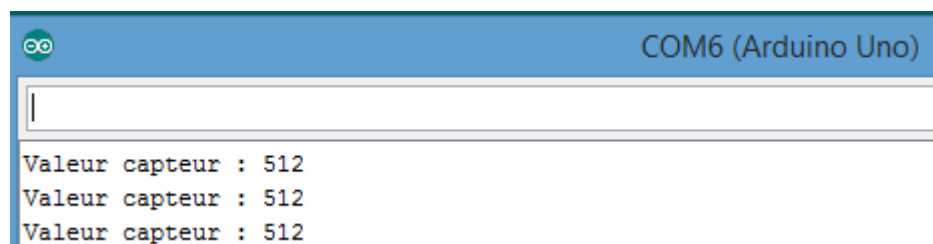
/*
AC-2019-Potentiometre-DEC
Lit la tension sur la broche analogique 0
et affiche la valeur issue du CAN dans le moniteur série.
DEC Decimal
BIN Binaire
OCT Octal
HEX Hexadecimal
*/

void setup() {
  Serial.begin(9600);
}

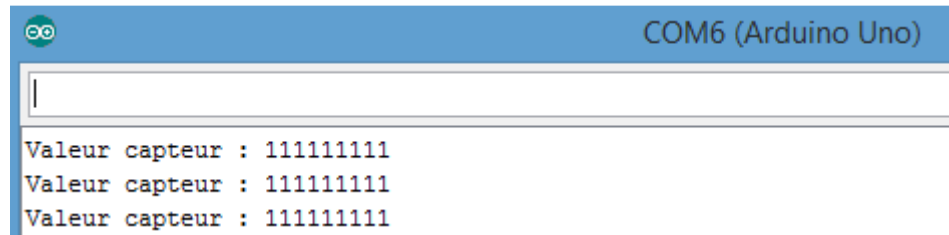
void loop() {
  intValeurCapteur = analogRead(A0);
  Serial.print("Valeur capteur : ");
  Serial.println(ValeurCapteur,DEC);// essais avec BIN, OCT, HEX
  delay(1000);
}

```

7.4.2. Moniteur série avec la fonction DEC



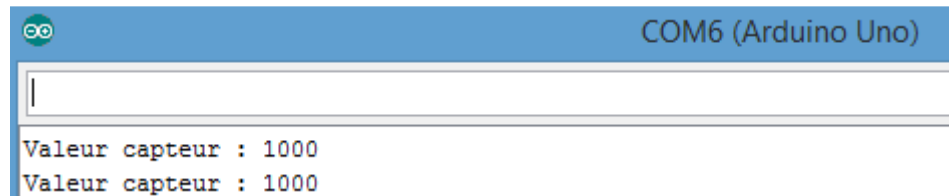
7.4.3. Moniteur série avec la fonction BIN



COM6 (Arduino Uno)

```
Valeur capteur : 11111111
Valeur capteur : 11111111
Valeur capteur : 11111111
```

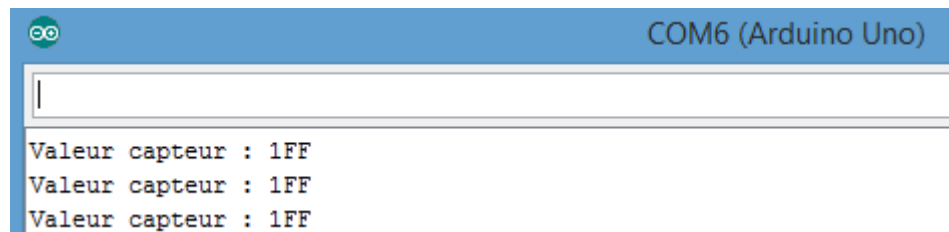
7.4.4. Moniteur série avec la fonction OCT



COM6 (Arduino Uno)

```
Valeur capteur : 1000
Valeur capteur : 1000
```

7.4.5. Moniteur série avec la fonction HEX



COM6 (Arduino Uno)

```
Valeur capteur : 1FF
Valeur capteur : 1FF
Valeur capteur : 1FF
```

7.5. Potentiomètre / Fonction « map »

La fonction « map » permet une conversion rapide de la valeur lue sur un port analogique, convertie par le CAN 10 bits en valeur correspondant à la grandeur physique d'entrée. Dans notre cas, nous utilisons un potentiomètre variant de 0 ohms à 9770 ohms (10 k Ω).

Une mesure a permis de constater la valeur maxi de 9770 Ω pour une valeur de 1000 renvoyée par le CAN. Dans la fonction « map » l'étendue sera donc de 0 à 1000.

Dans ce code, nous proposons l'affichage de la valeur de la résistance associée à l'angle de rotation du potentiomètre (0 à 235°) et à un affichage en pourcentage.

L'affichage du moniteur série est réalisé sous forme de tableau en utilisant la fonction de tabulation « '\t' ».

7.5.1. Le code

```
/*
AC-2019-Potentiometre-Fonction map
Lit la tension sur la broche analogique 0
et affiche la valeur issue du CAN dans le moniteur série.
utilise la fonction map qui permet une conversion rapide
des valeurs issues du CAN
```

Exemple avec un potentiomètre de 0 à 9995 ohms

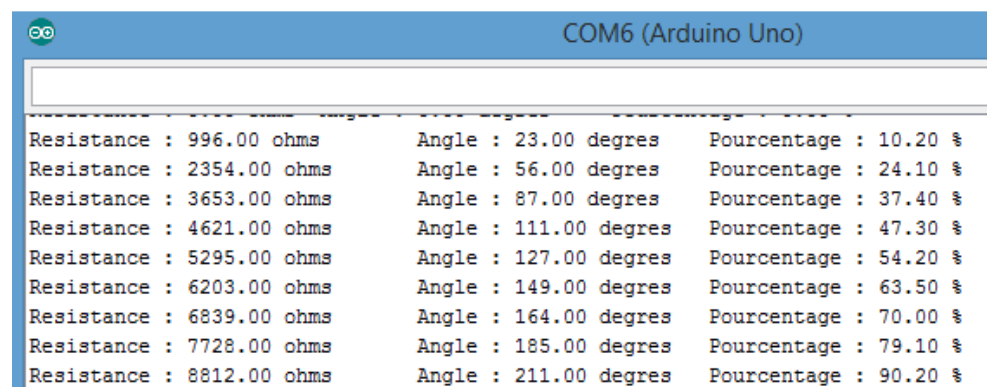
La valeur est donnée en ohms, en degrés, en pourcentage

*/

```
float resistance;
float angle;
float pourcentage;
void setup() {
  Serial.begin(9600);
}

void loop() {
  intValeurCapteur = analogRead(A0);
  resistance=map(ValeurCapteur,0,1000,0,9770);           //fonction map
  angle=map(ValeurCapteur,0,1000,0,235);                 //fonction map
  pourcentage=map(ValeurCapteur,0,1000,0,10000);          //fonction map
  Serial.print("Resistance : ");
  Serial.print(resistance);
  Serial.print(" ohms");
  Serial.print("\t");                                     //tabulation
  Serial.print("Angle : ");
  Serial.print(angle);
  Serial.print(" degrees");
  Serial.print("\t");                                     //tabulation
  Serial.print("Pourcentage : ");
  Serial.print(pourcentage/100);
  Serial.println(" %");
  delay(1000);
}
```

7.5.2. Moniteur série avec la fonction map

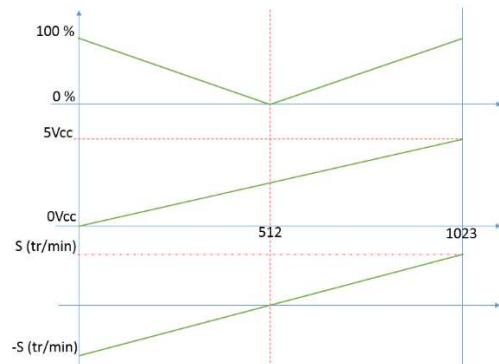
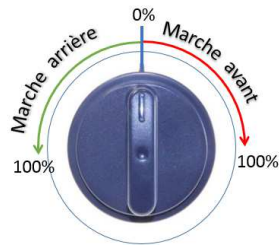


| Resistance | Angle | Pourcentage |
|---------------------------|------------------------|-----------------------|
| Resistance : 996.00 ohms | Angle : 23.00 degrees | Pourcentage : 10.20 % |
| Resistance : 2354.00 ohms | Angle : 56.00 degrees | Pourcentage : 24.10 % |
| Resistance : 3653.00 ohms | Angle : 87.00 degrees | Pourcentage : 37.40 % |
| Resistance : 4621.00 ohms | Angle : 111.00 degrees | Pourcentage : 47.30 % |
| Resistance : 5295.00 ohms | Angle : 127.00 degrees | Pourcentage : 54.20 % |
| Resistance : 6203.00 ohms | Angle : 149.00 degrees | Pourcentage : 63.50 % |
| Resistance : 6839.00 ohms | Angle : 164.00 degrees | Pourcentage : 70.00 % |
| Resistance : 7728.00 ohms | Angle : 185.00 degrees | Pourcentage : 79.10 % |
| Resistance : 8812.00 ohms | Angle : 211.00 degrees | Pourcentage : 90.20 % |

7.6. Potentiomètre / Centrer la consigne

7.6.1. Chronogramme

Pour certaines applications, il peut être nécessaire de centrer la consigne



7.6.2. Code

```

/*
AC-2019-Potentiometre-Consigne centrée
Lit la tension sur la broche analogique 0
et affiche le résultat dans le moniteur série.
Permet de centrer une consigne et de renvoyer
une commande comprise entre 0 et 255.
*/

float pourcentage;
byte consigne; // pour stocker un chiffre compris entre 0 et 255

void setup() {
  Serial.begin(9600);
}

void loop() {
  int ValeurCapteur = analogRead(A0);
  pourcentage=abs((ValeurCapteur*0.1955)-100); // 200/1023=0.1955
  consigne=pourcentage/100*255;
  Serial.print(ValeurCapteur);
  Serial.print("\t");
  Serial.print("Pourcentage : ");
  Serial.print(pourcentage);
  Serial.print("% ");
  Serial.print("\t");
  Serial.print("Cd Vitesse:");
  Serial.println(consigne);

  delay(2000);
}

```

7.6.3. Affichage moniteur série

| COM5 (Arduino Uno) | | |
|--------------------|-----------------------|----------------|
| 1023 | Pourcentage : 100.00% | Cd Vitesse:254 |
| 1023 | Pourcentage : 100.00% | Cd Vitesse:254 |
| 1023 | Pourcentage : 100.00% | Cd Vitesse:254 |
| 970 | Pourcentage : 89.63% | Cd Vitesse:228 |
| 886 | Pourcentage : 73.21% | Cd Vitesse:186 |
| 787 | Pourcentage : 53.86% | Cd Vitesse:137 |
| 686 | Pourcentage : 34.11% | Cd Vitesse:86 |
| 582 | Pourcentage : 13.78% | Cd Vitesse:35 |
| 484 | Pourcentage : 5.38% | Cd Vitesse:13 |
| 392 | Pourcentage : 23.36% | Cd Vitesse:59 |
| 312 | Pourcentage : 39.00% | Cd Vitesse:99 |
| 218 | Pourcentage : 57.38% | Cd Vitesse:146 |
| 123 | Pourcentage : 75.95% | Cd Vitesse:193 |
| 55 | Pourcentage : 89.25% | Cd Vitesse:227 |
| 0 | Pourcentage : 100.00% | Cd Vitesse:255 |
| 0 | Pourcentage : 100.00% | Cd Vitesse:255 |

7.7. Capteur de tension 0 25 Vcc

7.7.1. Code

```

/*
AC-2019-Capteur de tension de 0 cc à 25 Vcc
Pont diviseur de tension 1/5
Lit la tension sur la broche analogique A0
et affiche le résultat dans le moniteur série.
*/

float tension;

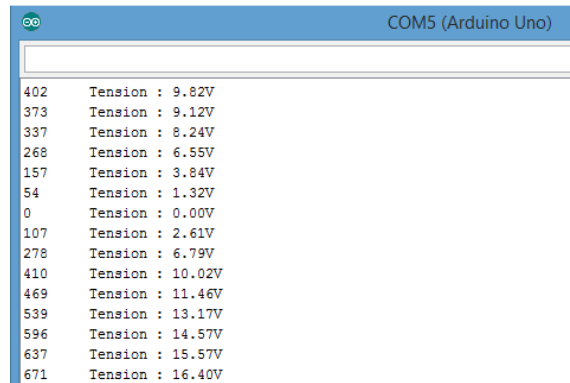
void setup() {
  Serial.begin(9600);
}

void loop() {
  int ValeurCapteur = analogRead(A0);
  tension=(ValeurCapteur/40.92);           // 1023/25=40.92
  Serial.print(ValeurCapteur,DEC);
  Serial.print("\t");
  Serial.print("Tension : ");
  Serial.print(tension);
  Serial.println("V");

  delay(2000);
}

```


7.7.2. Affichage moniteur série



COM5 (Arduino Uno)

| | |
|-----|------------------|
| 402 | Tension : 9.82V |
| 373 | Tension : 9.12V |
| 337 | Tension : 8.24V |
| 268 | Tension : 6.55V |
| 157 | Tension : 3.84V |
| 54 | Tension : 1.32V |
| 0 | Tension : 0.00V |
| 107 | Tension : 2.61V |
| 278 | Tension : 6.79V |
| 410 | Tension : 10.02V |
| 469 | Tension : 11.46V |
| 539 | Tension : 13.17V |
| 596 | Tension : 14.57V |
| 637 | Tension : 15.57V |
| 671 | Tension : 16.40V |

7.8. Capteur de courant

7.8.1. Code

```

/*
AC-2019-Capteur de courant 20A ACS712
Lit l'intensité sur la broche analogique A2
et affiche le résultat dans le moniteur série.
La sensibilité du capteur est de 100 mV/A
L'étendue de tension du signal est 5 Vcc
C'est un capteur en ±20A, le signal est
centré sur 512
*/

float intensite;

void setup() {
  Serial.begin(9600);
}

void loop() {
  int ValeurCapteur = analogRead(A2);
  intensite=((ValeurCapteur-512)*0.05);           // 5/100=0.05
  Serial.print(ValeurCapteur);
  Serial.print("\t");
  Serial.print("Intensite : ");
  Serial.print(intensite);
  Serial.println("A");

  delay(1000);
}

```

7.8.2. Affichage moniteur série

```

COM5 (Arduino Uno)
529 Intensite : 0.85A
529 Intensite : 0.85A
537 Intensite : 1.25A
531 Intensite : 0.95A
535 Intensite : 1.15A
534 Intensite : 1.10A
532 Intensite : 1.00A
534 Intensite : 1.10A
532 Intensite : 1.00A
532 Intensite : 1.00A
531 Intensite : 0.95A
511 Intensite : -0.05A
512 Intensite : 0.00A

```

7.9. Les capteurs TOR de fin de course

7.9.1. Digital Push Button : Code

```

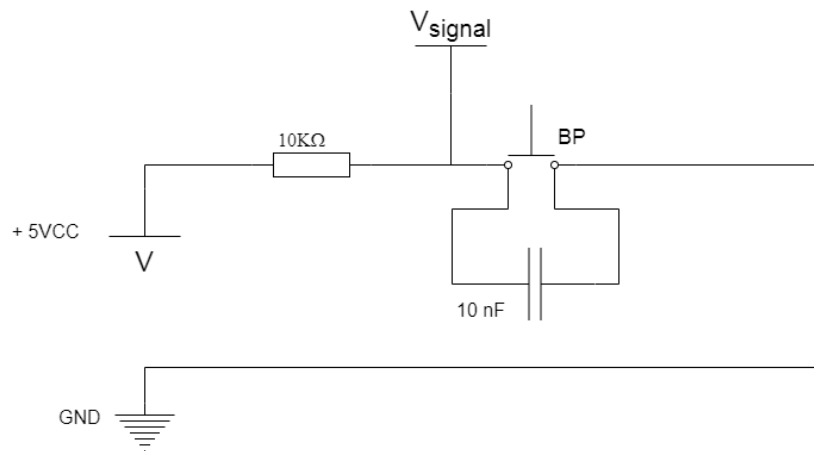
/*
L'appui sur BPG provoque l'arrêt à droite
*/
int BPG = 2;          // BPG sur port numérique 2
int BPD=3 ;           // BPD sur port numérique 3

void setup() {
  Serial.begin(9600) ;
  pinMode(BPG, INPUT); // BPG est une entrée
  pinMode(BPD, INPUT); // BPD est une entrée
}

void loop(){
  int EBPG = digitalRead(BPG); // Lecture de la valeur de BPG 1 ou 0
  if (EBPG == HIGH) {
    Serial.println(« Marche avant »);
  } else {
    Serial.println(« Stop »);
  }
  int EBPD = digitalRead(BPD); // Lecture de la valeur de BPD 1 ou 0
  if (EBPD == HIGH) {
    Serial.println(« Marche arrière »);
  } else {
    Serial.println(« Stop »);
  }
}

```

7.9.2. Schéma de câblage d'un capteur TOR (Entrée numérique)



Une entrée numérique ne peut prendre que deux états, HAUT (HIGH) ou BAS (LOW). L'état haut correspond à une tension de +5V sur la broche, tandis que l'état bas est une tension de 0V. Dans notre exemple, nous utilisons un simple bouton :

- un bouton poussoir
- une résistance de 10k de pull-up
- un condensateur anti-rebond de 10nF

7.9.3. Le code

```

/* Ce programme permet de tester l'application
de deux capteurs de fin course
Un bouton pour simuler l'arrivée à Droite BPD
Un bouton pour simuler l'arrivée à Gauche BPG
Affichage sur le moniteur série
*/
const int BPD=2; // Bouton Droit
const int BPG=3; // Bouton Gauche
int EBPD=0;
int EBPG=0;

//Initialisation

void setup() {
  Serial.begin(9600);
  pinMode(BPD,INPUT);
  pinMode(BPG,INPUT);
  EBPD=LOW;
  EBPG=LOW;
}
//Programme

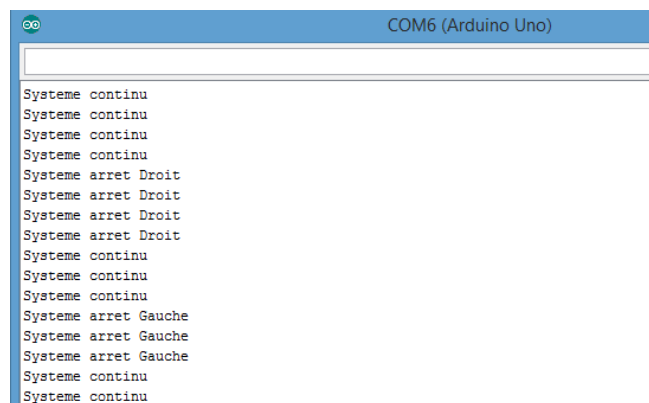
```

```

void loop() {
  EBPD=digitalRead(BPD);
  EBPG=digitalRead(BPG);
  if(EBPD==HIGH&&EBPG==LOW){
    Serial.println("Systeme arret Gauche");
  }
  if(EBPD==HIGH&&EBPG==HIGH){
    Serial.println("Systeme continu");
  }
  if(EBPD==LOW&&EBPG==HIGH){
    Serial.println("Systeme arret Droit");
  }
  delay(1000);
}

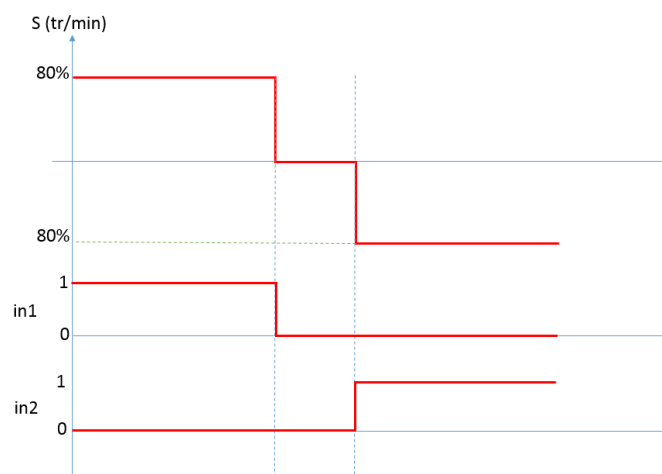
```

7.9.4. L'affichage sur le moniteur série



7.10. Driver moteur

7.10.1. Chronogramme



7.10.2. Code de test

```

/*
AC-2019-Driver OSEPP- DC-003-L298N bidirectionnel.
Code Arduino pour contrôler un moteur continu à partir d'un module L298N et
de le faire tourner dans un sens pendant 5 secondes, puis l'arrête pendant 2
secondes, et le fait repartir dans l'autre sens pendant 5 secondes; le tout à une
vitesse de 200 sur 255. Le driver commande la vitesse sur 8 bits
*/

// Déclaration des variables et des constantes
int enA = 10;
// crée une variable de type "int", nommée "enA" et attachée à la Broche 10, qui
permet de gérer la vitesse du moteur.
int in1 = 9;
// variable de type "int", nommée "in1" et attachée à la Broche 9, qui permet de
gérer le sens de rotation.
int in2 = 8;
// variable de type "int", nommée "in2" et attachée à la Broche 8, qui permet de
gérer l'autre sens de rotation.

void setup() {
  pinMode(enA, OUTPUT);
// indique que la broche de la variable "enA" donc ici la PIN 10, est une sortie.
  pinMode(in1, OUTPUT);
// indique que la broche de la variable "in1" donc ici la PIN 9, est une sortie.
  pinMode(in2, OUTPUT);
// indique que la broche de la variable "in2" donc ici la PIN 8, est une sortie.
}

void loop() {
  digitalWrite(in1,HIGH);
// active la broche in1 (donc la PIN 9) ce qui fait donc tourner le moteur dans le
sens de rotation de in1.
  digitalWrite(in2,LOW);
// désactive la broche in2
  analogWrite(enA,200)
// vitesse du moteur, ici 200 sur un maximum de 255
  delay(5000);
// le moteur tourne dans le sens in1 pendant 5 secondes.

  digitalWrite(in1,LOW);
// désactive la broche in1
  digitalWrite(in2,LOW);
// désactive la broche in2

```

```

    delay(2000);
    // le moteur ne tourne pas pendant 2 secondes.

    digitalWrite(in1,LOW);
    // désactive la broche in1
    digitalWrite(in2,HIGH);
    // active la broche in2
    analogWrite(enA,200);
    // la vitesse du moteur, ici 200 sur un maximum de 255
    delay(500) ;
    // le moteur tourne dans le sens in2 pendant 5 secondes.
}

```

7.10.3. Code avec consigne de vitesse

```

/*
AC-2019-Driver OSEPP- DC-003-L298N bidirectionnel.
Le potentiomètre sur le port A0 conditionne la vitesse.
*/

int ENA = 10;           // Cd vitesse
int IN1 = 9;            // Sens 1
int IN2 = 8;            // Sens 2
byte vitesse;

void setup() {
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  int ValeurCapteur = analogRead(A0);
  vitesse=(ValeurCapteur*255/1023);
  // Le potentiomètre donne la consigne sur 8 bits
  analogWrite(ENA,vitesse);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);

}

```

7.11. Afficheur I2C

7.11.1. Le code :

```

const int Consigne = A0;
const int analogOutPin = 9;
int ValeurConsigne = 0;
int SortieValeur = 0;
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 20, 4);

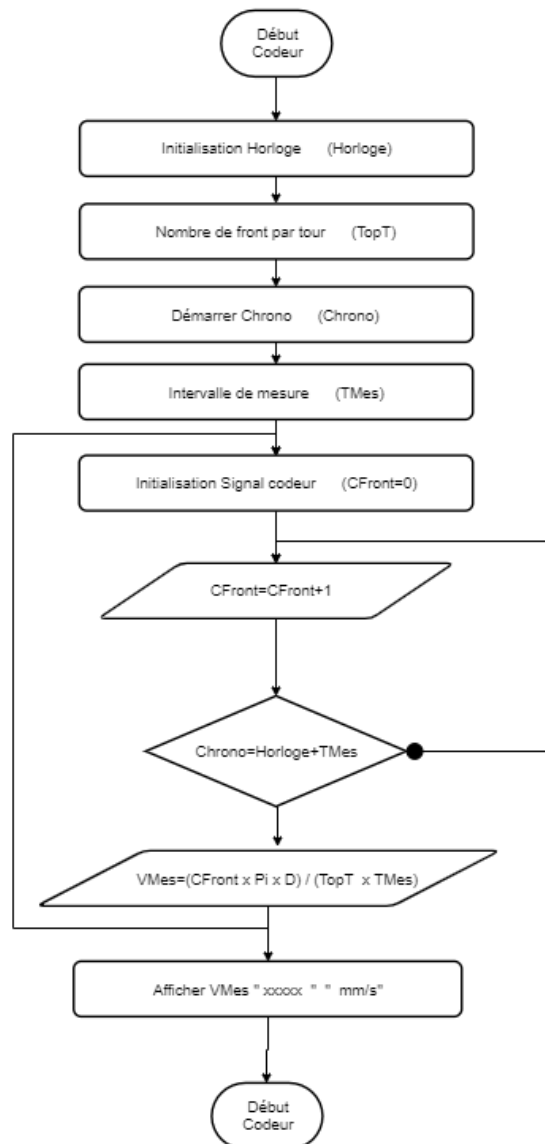
void setup() {
  lcd.init();
  Serial.begin(9600);
}

void loop() {
  ValeurConsigne = analogRead(Consigne);
  outputValue = map(ValeurConsigne 0, 1023, 0, 100);
  analogWrite(analogOutPin, SortieValeur);
  Serial.print("sensor = ");
  Serial.print(ValeurConsigne);
  Serial.print("\t output = ");
  Serial.println(SortieValeur);
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("tension moteur:12Vcc ");
  lcd.setCursor(0, 1);
  lcd.print("courant moteur:1A");
  lcd.setCursor(0, 2);
  lcd.print("consigne:");
  lcd.print(SortieValeur);
  lcd.print("%");
  lcd.print("  ");
  lcd.setCursor(0, 3);
  lcd.print("vitesse:0,1m/min ");
  delay(20);
}

```


7.12. Le codeur

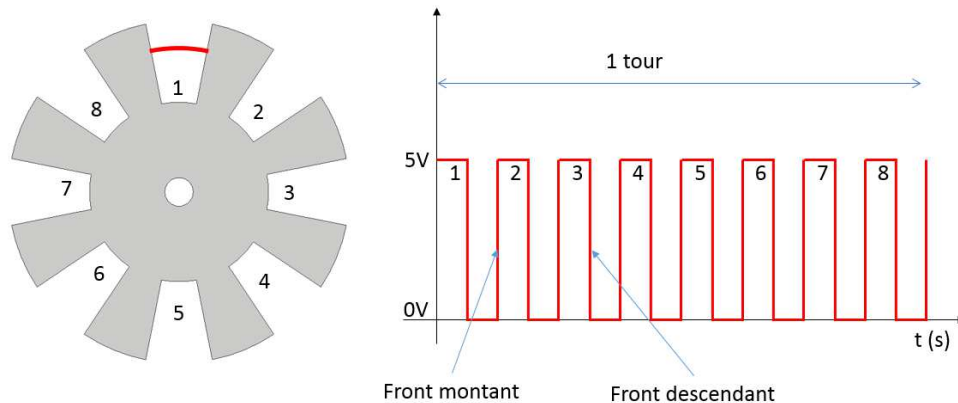
7.12.1. Algorithme



7.12.2. Principe

Au démarrage, une horloge est prise comme référence et un intervalle de mesure est donné. C'est une horloge interne dont la commande est « millis ». Elle débute à 0 milliseconde. Sa durée maximum est de 50 jours.

L'encodeur est muni d'un certain nombre d'encoches qui provoquent le signal. Les microcontrôleurs permettent de compter les fronts (montant, descendant) ou de compter le nombre de changement d'état.



Dans le cas du matériel proposé, le disque encodeur possède huit encoches, huit fronts montants, huit fronts descendants soit seize changements d'état.

Le programme propose de compter le nombre de changements d'état dans un temps défini (TMes).

7.12.3. Le code

```

/*
AC-2019-Codeur Cytron- Codeur optique 8 encoches
Le signal est branché sur la broche numérique 2.
La fonction millis est une horloge qui débute au démarrage
du programme. Durée maxi 50 jours.
*/

int CFront = 0;
// initialisation de la variable de comptage des fronts à zéro
volatile unsigned int Chrono;
long TMes = 500;
// le calcul s'effectuera tous les 500 ms
float VMes;
// Vitesse mesurée en mm/s

void setup ()
{
  Serial.begin(9600);
  attachInterrupt(0, count, CHANGE);
  // compte le changement d'état pin 2
  Chrono = millis();
}

void loop ()
{
  if ( millis() >= Chrono + TMes)
  {

```

```

VMes = (CFront*(XXXXXX))/TMes;//XXXX est la distance parcourue pour
un tour
Serial.print(VMes);
Serial.println(« mm/s ») ;
CFront = 0;
Chrono = millis();
    }
}

void count()
{
CFront++;
}

```

7.13. Les variables

7.13.1. char : caractère.

Une variable du type **char** peut contenir une valeur entre -128 et 127 et elle peut subir les mêmes opérations que les variables du type **short**, **int** ou **long**.

| description | domaine min | domaine max | nombre d'octets |
|-------------|-------------|-------------|-----------------|
| caractère | -128 | 127 | 1 |

7.13.2. int : entier standard.

Sur chaque machine, le type **int** est le type de base pour les calculs avec les entiers. Le codage des variables du type **int** est donc dépendant de la machine. Sur les IBM-PC sous MS-DOS, une variable du type **int** est codée dans deux octets.

| description | domaine min | domaine max | nombre d'octets |
|-----------------|-------------|-------------|-----------------|
| entier standard | -32768 | 32767 | 2 |

7.13.3. short : entier court.

Le type **short** est en général codé dans 2 octets. Comme une variable **int** occupe aussi 2 octets sur notre système, le type **short** devient seulement nécessaire, si on veut utiliser le même programme sur d'autres machines, sur lesquelles le type standard des entiers n'est pas forcément 2 octets.

| description | domaine min | domaine max | nombre d'octets |
|--------------|-------------|-------------|-----------------|
| entier court | -32768 | 32767 | 2 |

7.13.4. long : entier long

Le type **long** est codé dans 4 octets.

| description | domaine min | domaine max | nombre d'octets |
|-------------|-------------|-------------|-----------------|
| entier long | -2147483648 | 2147483647 | 4 |

7.13.5. unsigned

Si on ajoute le préfixe **unsigned** à la définition d'un type de variables entières, les domaines des variables est positif.

| définition | domaine min | domaine max | nombre d'octets |
|----------------|-------------|-------------|-----------------|
| unsigned char | 0 | 255 | 1 |
| unsigned short | 0 | 65535 | 2 |
| unsigned int | 0 | 65535 | 2 |
| unsigned long | 0 | 4294967295 | 4 |

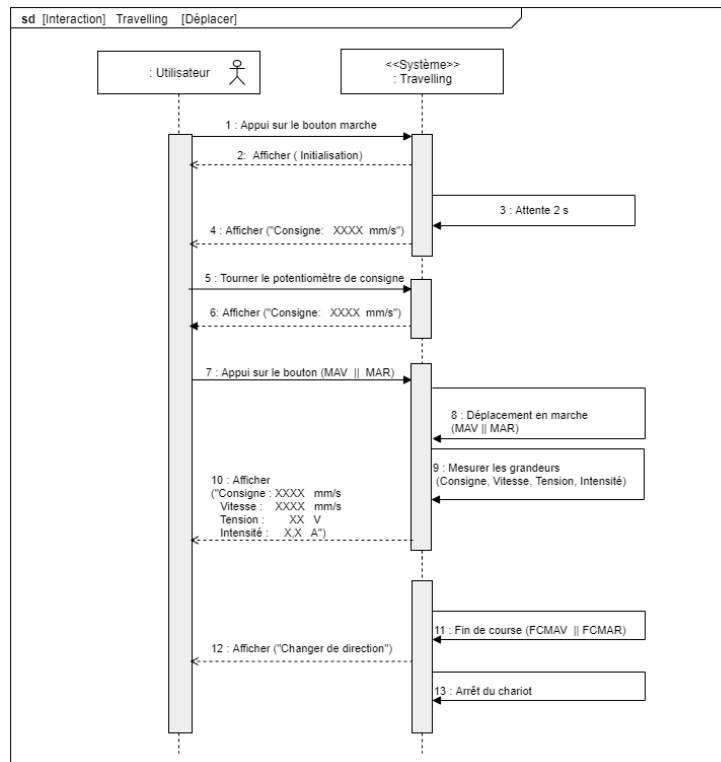
7.13.6. volatile

C'est un mot clé associé à une variable pour modifier la façon dont le compilateur et le programme traitent la variable.

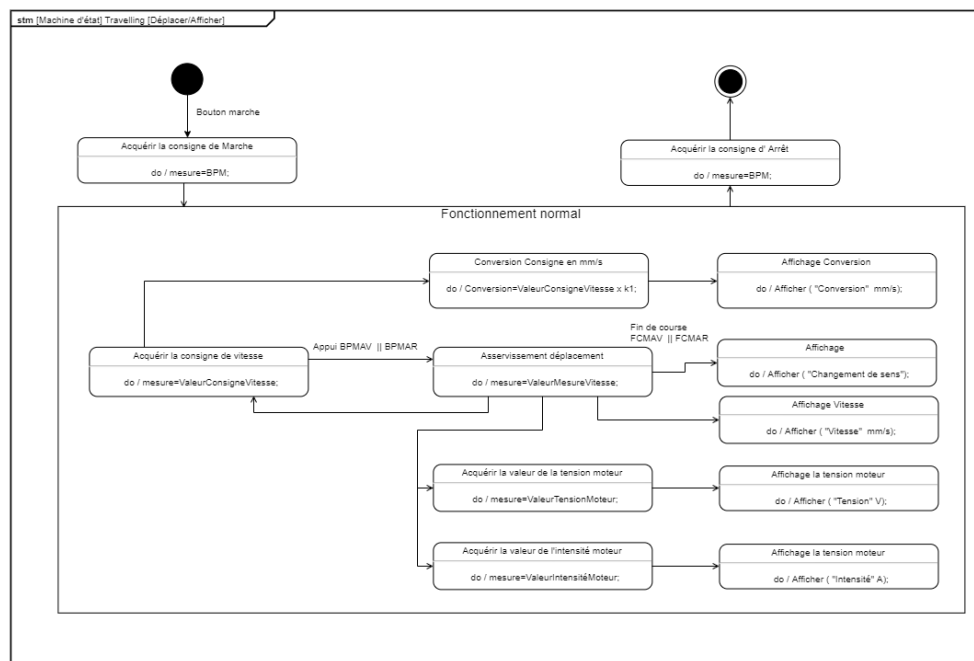
C'est une directive pour le compilateur qui a pour but de charger la variable à partir de la RAM et non à partir d'un registre de stockage, qui est un emplacement de mémoire temporaire où les variables de programme sont stockées et manipulées.

8. Le banc de travelling

8.1. Diagramme de séquence



8.2. Diagramme d'état

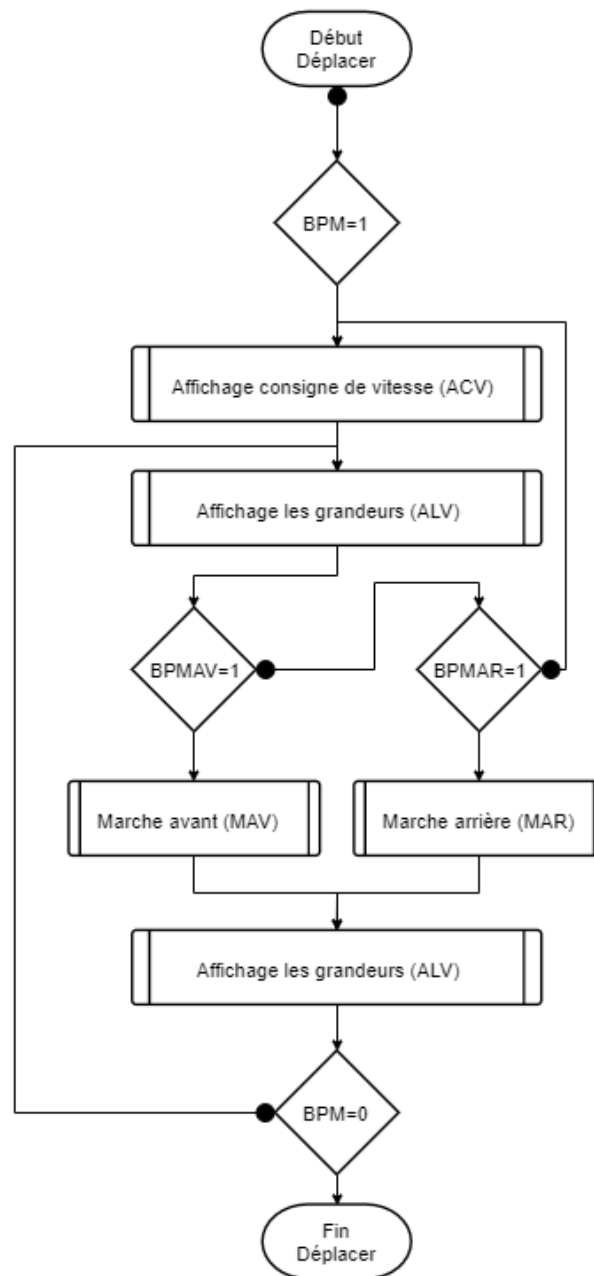


8.3.Algorigrammes

8.3.1. Déplacer

La mise en marche sera effective dès lors que la mise sous tension sera effectuée. Un bouton trois positions sera utilisé :

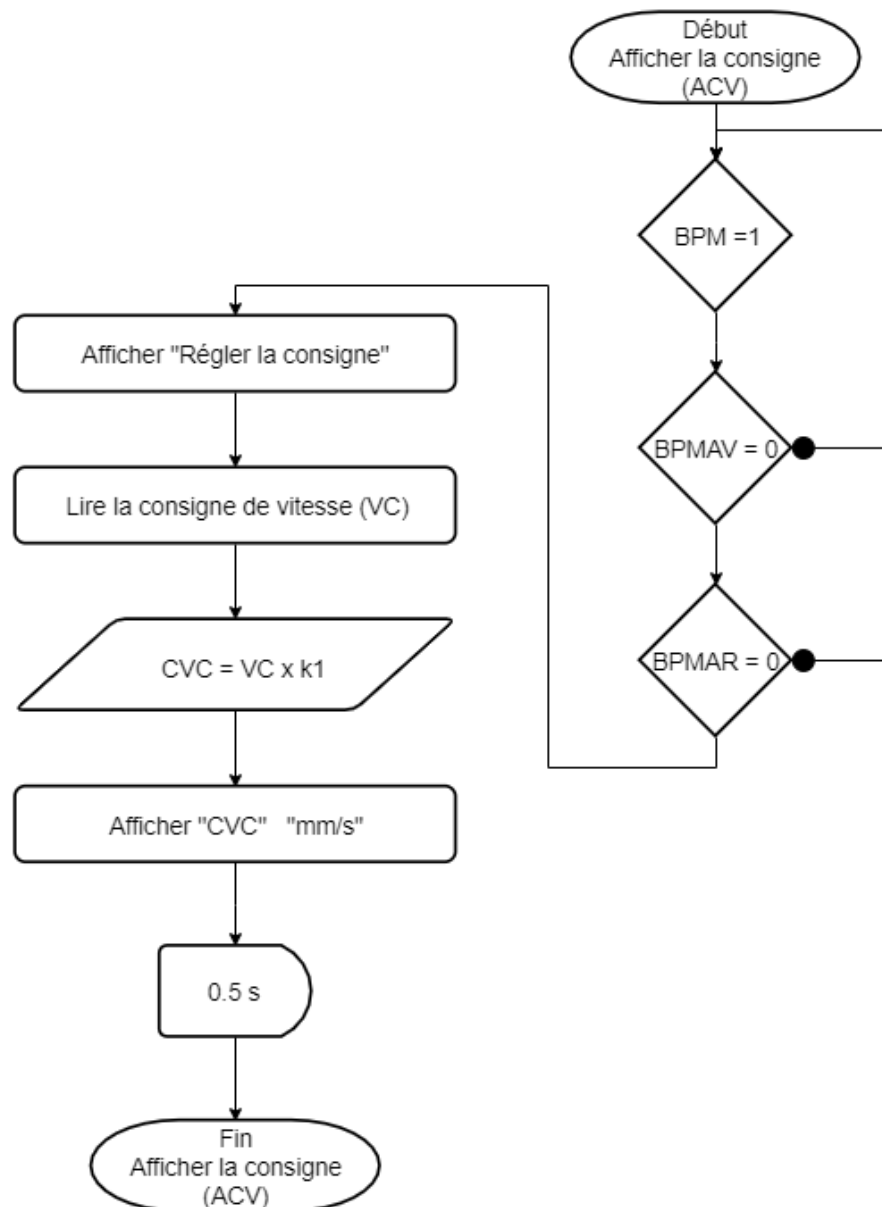
- BPM : La position centrale permettra d'effectuer le réglage de la consigne de vitesse.
- BPMAV : une position qui permettra la marche avant (MAV)
- BPMAR : une position qui permettra la marche arrière (MAR)



8.3.2. Afficher la consigne de vitesse

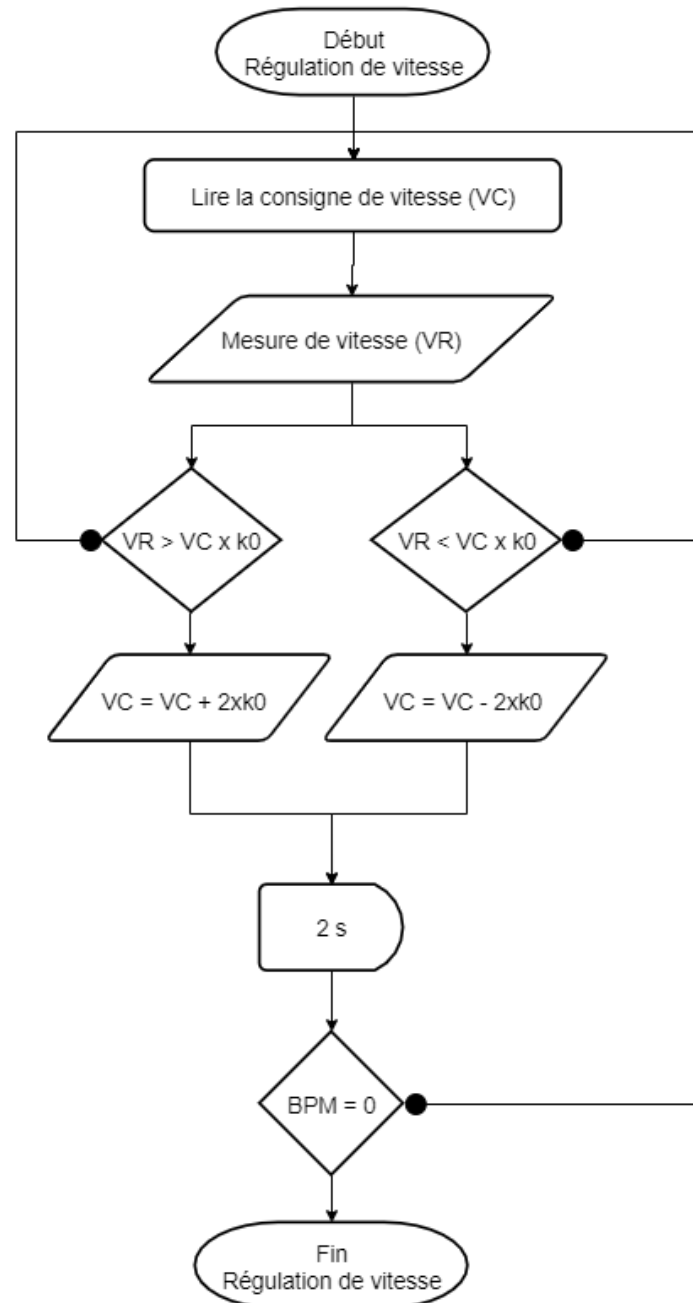
L'affichage de la consigne de vitesse (ACV) ne pourra s'effectuer que lorsque le banc est à l'arrêt (BPM=0). Elle restera affichée mais non modifiable lors des déplacements (MAV et MAR).

- BPM : Bouton de mise en marche
- BPMAV : Bouton de mise ne marche avant
- BPMAR : Bouton de mise en marche arrière
- k_1 : coefficient à déterminer en fonction de la technologie utilisée.



8.3.3. Régulation de vitesse

Un codeur permettra la mesure de la vitesse réelle en sortie (VR). Cette valeur sera comparée à la valeur de la consigne (VC). Si le système est trop lent, la vitesse sera accélérée (incrément +2), si le système est trop rapide la vitesse sera ralentie (incrément -2). Ce contrôle sera effectué lors des déplacements. La tolérance sera calculée avec un coefficient k_0 .

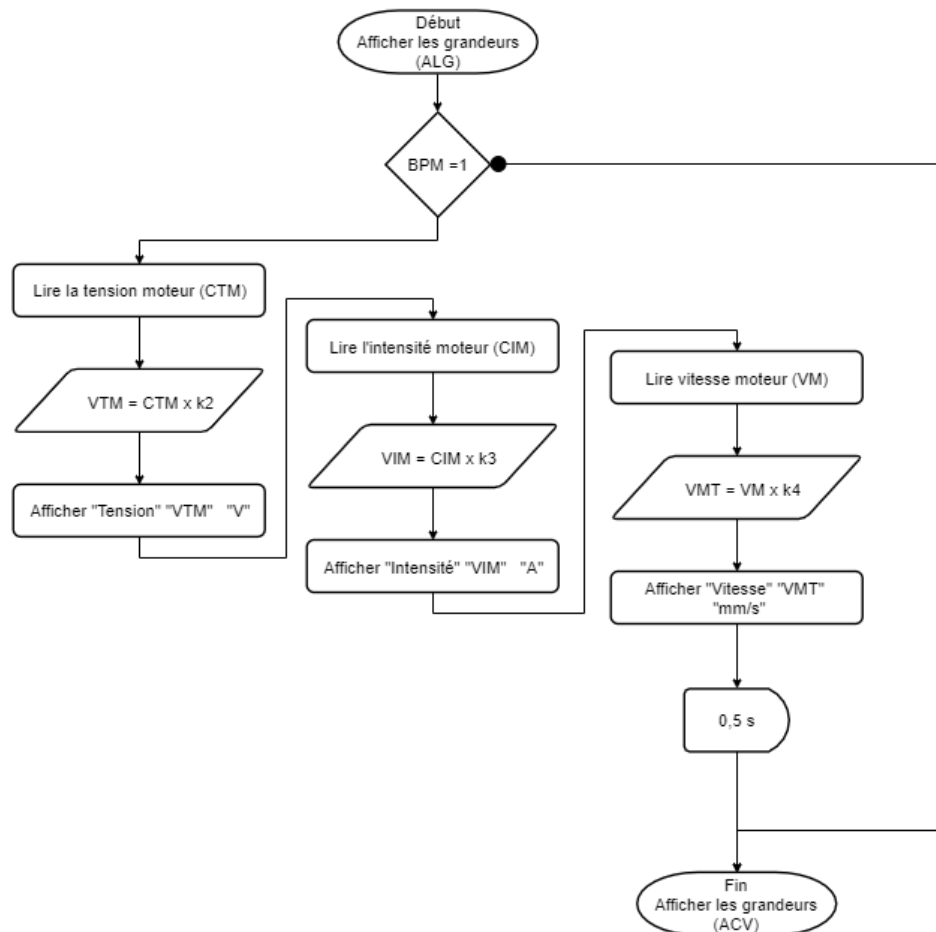


8.3.4. Afficher les grandeurs mesurées

L'affichage des grandeurs mesurées s'effectuera dès la mise en marche du système. Les mesures effectuées sont :

- La tension aux bornes du moteur en V
- L'intensité consommée par le moteur en A
- La vitesse réelle en mm/s

Les coefficients k2, k3, et k4 seront à déterminer selon les technologies utilisées.



8.3.5. Marche avant et marche arrière

La sélection des marches s'effectuera avec :

- BPMAV : Bouton de marche avant
- BPMAR : Bouton de marche arrière

Les fins de course seront assurées par :

- FCMAV : Fin de course marche avant
- FCMAR : Fin de course marche arrière

