C14 TP Sujet - Bases de l'électronique numérique

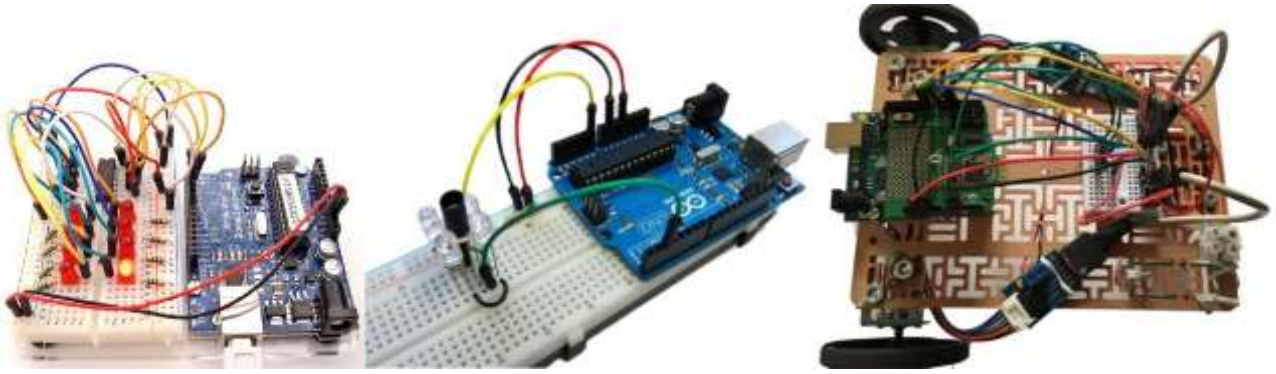


Table des matières

Ressources nécessaires 1
Objectifs 2
Activité 1 : La carte Arduino Uno 2
Activité 2 : L'environnement de programmation 3
Structure d'un programme 3
Activité 3 : Communication série : affichage d'informations sur un écran d'ordinateur 4
Activité 4 : Plaque de prototypage et montages élémentaires 5
Schéma de la carte Arduino Uno et MLI 6
Types et déclaration 7
Bloc d'instructions 7
Déclaration d'une fonction ma_fonction sans argument ni valeur retournée 7
Structure conditionnelle et opérateurs booléen 7
Structures itératives 7
Sortie écran 7
Bouton poussoir 8
Montage 8
Minimum à programmer 8
Potentiomètre 9
Montage 9
Minimum à programmer 9
Trame de programme 9
Diode 10
Montage 10
Fonctions élémentaires 10
Trame de programme en numérique 10
Trame de programme en pseudo-analogique 10
Moteur à courant continu et hacheur PmodHB5 11
Broches du hacheur PmodHB5 12
Montage sans codeur 12
Montage avec codeur 12

Ressources nécessaires

- carte **Arduino Uno** rev 3 (~20€, http://www.selectronic.fr/carte-arduino-uno.html);
- composants annexes : **diode**, **bouton poussoir**, **potentiomètre**, fils (moins de 5€ l'ensemble, http://www.selectronic.fr/) ;
- **moto-réducteur PMT1-53** avec codeur 2 pistes (~20€, http://www.lextronic.fr/P5448-moteur-reducteur-avecencodeur.html) ;
- **carte de puissance + traitement codeur PMODHB-5** (~25€, http://www.lextronic.fr/P5451-module-de-commandepour-moteur-cc.html) ;
- **logiciel Arduino** (gratuit, http://arduino.cc/en/Main/Software).

Objectifs

Connaître les bases de l'électronique numérique – acquérir, traiter et coder l'information

Il est proposé de :

- réaliser des **montages d'électronique numérique** d'acquisition et de commande ;
- de générer un **programme de commande** dans l'environnement Arduino et de l'implémenter dans un microcontroleur.

Activité 1 : La carte Arduino Uno

Le cœur Arduino est un **microcontrôleur**, un « petit » ordinateur contenu dans un seul circuit imprimé avec :

- **une unité de commande** (microprocesseur)
- de la mémoire vive et de la **mémoire FLASH** (comme les clés USB)
- **des interfaces** pour communiquer avec son environnement.

Après **compilation** sur PC, un programme est « **téléversé** » (*upload)* dans la mémoire FLASH par liaison USB. Il y reste tant qu'il n'est pas remplacé par un autre.

Le **programme** s'exécute **automatiquement** et en **continu**, **dès la mise sous tension**. Il redémarre à chaque reset.

Ces cartes possèdent des broches (pins, ports ou pattes) qui reçoivent les signaux des **entrées** (en provenance des capteurs ou des interfaces homme-machine) **ou** enverront des signaux aux **sorties** (vers les interfaces machine-homme ou préactionneurs). Les mots « entrées » et « sorties » doivent être compris du point de vue du microcontrôleur. On trouve :

14 broches pour des entrées ou sorties numériques (tout ou rien) repérées de 0 à 13 dont **6 avec sortie pseudo-analogique MLI** (Modulation de Largeur d'Impulsions, ou PWM en anglais pour Pulse Width Modulation). Les broches utilisées sont **configurées** par le programme.

Les états binaires 1 et 0 sont associés aux tensions de 5V ou de 0V.

Certaines broches disposent d'options particulières :

- les broches pouvant générer un signal MLI (3, 5, 6, 9, 10 et 11) sont repérées par le **symbole** ~ ;
- la **broche 13** est reliée à une **petite diode test** « L » directement implantée sur la carte ;
- les broches 2 et 3 sont aussi des **entrées d'interruption**. Elles peuvent être utilisées pour détecter des fronts montants et descendants ;
- les broches 0 et 1 peuvent être utilisées pour de la **communication série** (asynchrone par défaut) avec le PC (0 : Rx = réception et 1 : Tx = transmission). Ces broches 0 et 1 ne sont donc plus utilisables en parallèle d'une communication série avec le PC.

Les **6 broches des entrées analogiques** sont repérées de A0 à A5 :

- la **tension d'entrée** doit être inférieure à la tension de référence (5V généralement) ;
- la tension est **convertie en valeur numérique** sur 10 bits (entre 0 et 1023) ;

5 broches d'alimentation permettant d'alimenter d'autres composants (intensité totale <50 mA) :

- 3 broches GND : 0V de l'Arduino (Masse) ;
- 1 broche +5 V ;
- 1 broche +3,3 V.

➲ Sur le **schéma de la carte Arduino** (en document ressource) et **sur la carte réelle** à votre disposition, **repérer TOUTES ces broches**. **Repérer** celles qui peuvent être utilisées en entrée ou en entrée/sortie.

Activité 2 : L'environnement de programmation

- **Brancher la carte** au PC, via le **port USB** qui permet **d'alimenter la carte** mais également de **télécharger** le programme.

Windows peut demander alors d'installer les drivers. Lui indiquer d'aller les prendre dans le dossier */arduino/drivers*.

- Lancer **l'environnement de développement** : programme *arduino.*
- À l'aide du menu *Outils/Type de carte*, **sélectionner la carte** adéquate.
- À l'aide du menu *Outils/Port*, **sélectionner le port** de communication (**sélectionner toujours le port le plus grand**).
- A l'aide du menu *Fichier / Exemples / 0.1 Basics / Blink*, ouvrir le **croquis Blink**.
- Vérifier la bonne **compilation** du programme à l'aide de l'icône .
- **Téléverser** (charger) le programme dans l'Arduino. Le programme est chargé via la liaison USB qui fonctionne comme un port série.
- **Vérifier** le bon fonctionnement du programme.

La Led intégrée doit clignoter toutes les secondes.

Structure d'un programme

La structure usuelle et minimale d'un programme est décrite ci-dessous. Noter qu'en dehors des déclarations des variables, toutes les instructions doivent être réalisées au sein de fonctions.

```
// déclaration des variables
int i ;
int BPin=4 ;
…



// fonction setup void
setup() {
pinMode(BPpin,INPUT) ;
  Serial.begin(8600) ;
…
}

// fonction de boucle void
loop() {
…
}
```

1) Déclaration des variables.

À l'inverse du Python, toutes les variables doivent être déclarées avant d'être utilisées. Toutes les instructions se terminent par le caractère **;** (point-virgule).

2) Fonction setup exécutée une fois au début du programme.

Elle permet de **configurer les E/S** de la carte ainsi que la **communication série** avec l'ordinateur.

3) Fonction loop exécutée en continue.

C'est le cœur du programme. Dès que les instructions de cette fonction sont terminées, elle est appelée de nouveau.

- En vous aidant du document ressource sur les différences entre python et le C, repérer dans le corps du programme *blink* :

  - les blocs de commentaires ;

  - le corps et les instructions des fonctions `setup()` et `loop()` ;

Ci-dessous un rapide descriptif des fonctions de ce programme :
`pinMode(13, OUTPUT)` ; définit le mode de fonctionnement de la broche 13, ici en sortie (OUTPUT pour *sortie*)
`digitalWrite(13, HIGH)` ; met à 5V la broche 13 (LOW pour la mettre à la masse)
`delay(1000)` ; boucle à vide 1000 millisecondes.

- Pourquoi la LED clignote-t-elle en continue ?

Activité 3 : Communication série : affichage d'informations sur un écran d'ordinateur

- **Ouvrir** le fichier **Bouts_de_code.html**.
- Créer un **nouveau croquis**, dans le logiciel Arduino : Fichier / Nouveau.
- Par copier-coller à partir du fichier **Bouts_de_code.html** compléter le croquis avec le code ci-dessous :

```
void setup() {
Serial.begin(9600);  /* permet de définir la vitesse de transmission de données entre
la carte et le PC (9600 bauds = 9600 symboles transmis par seconde).
         Nécessaire pour utiliser le moniteur série et les fonctions Serial.print .println*/
}
void loop() {
Serial.println("la SII c'est super");  // affiche sur l'écran du PC
                                       // la chaîne de caractères "la SII c'est super"
Serial.println("  "); // avec un retour à la ligne
delay(500);           // attend 500 ms }
```

Analyser le programme proposé :

- La fonction **Serial.begin** n'étant appelée qu'une seule fois, elle est présente dans la fonction **setup**.
- **Vérifier la compilation** du programme à l'aide de l'icône puis **Téléverser** le programme.
- **Vérifier** le bon fonctionnement en affichant le **moniteur série** à l'aide de l'icône situé en haut à droite de la fenêtre de programmation Arduino.

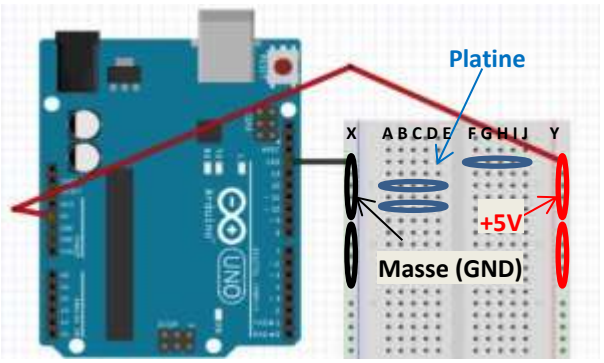
NB : les échanges d'informations sont visibles sur la carte Arduino au moyen de deux diodes : Rx pour la réception des données, et Tx pour la transmission. **L'ouverture du moniteur série réinitialise la carte.**

- Débrancher le cordon USB de la carte. Fermer le moniteur série. Attendre quelques secondes, puis rebrancher le cordon USB.
- Rouvrir le moniteur série et vérifier que le texte s'affiche.

Activité 4 : Plaque de prototypage et montages élémentaires

L'objectif est de réaliser successivement différents montages permettant d'appréhender les bases de l'électronique numérique, dont la gestion des entrées et sorties.

Les montages proposés seront réalisés sur une même **plaque de prototypage**.
Les broches des colonnes X et Y sont déjà reliées par groupement de 5, alors que les broches des colonnes A à J sont reliées par ligne également par groupement de 5 (voir photo ci-contre).



Par commodité, une broche « **masse (GND)** » de la carte sera connectée à un des groupements de la colonne X. **Ensuite, toute masse sera récupérée sur cette colonne X avec un fil noir.**

Par commodité, la broche « **+5V** » de la carte sera connectée à un des groupements de la colonne Y. **Ensuite, tout +5V sera récupéré sur cette colonne Y. On prendra toujours des fils rouges.**

En vous aidant :

- des **documents ressources,**
- des **bouts de code** présents dans le fichier html de même nom déjà ouvert, vous devez réaliser les activités suivantes,
- **dans l'ordre**,
- sans démonter le câblage entre chaque activité (sauf indication contraire), en demandant à votre **enseignant de contrôler le montage avant de brancher la carte au port USB** (qui alimente le circuit) :

1. afficher dans le moniteur série la **valeur du signal** en provenance d'un **bouton poussoir** ;
2. **commander l'allumage** et l'extinction d'une **diode** à l'aide du bouton poussoir (par programmation, pas par montage série !) ;

Supprimer le montage du bouton en laissant celui de la LED

3. afficher dans le moniteur série **l'image d'une tension** définie par le **potentiomètre** (valeur comprise entre 0 et 1023) ;
4. **commander l'intensité** lumineuse d'une **diode**, par **MLI**, à l'aide d'un **potentiomètre** (lire dans le document ressource *Schéma de la carte Arduino Uno et MLI* les informations sur le signal MLI et comprendre son fonctionnement) ;

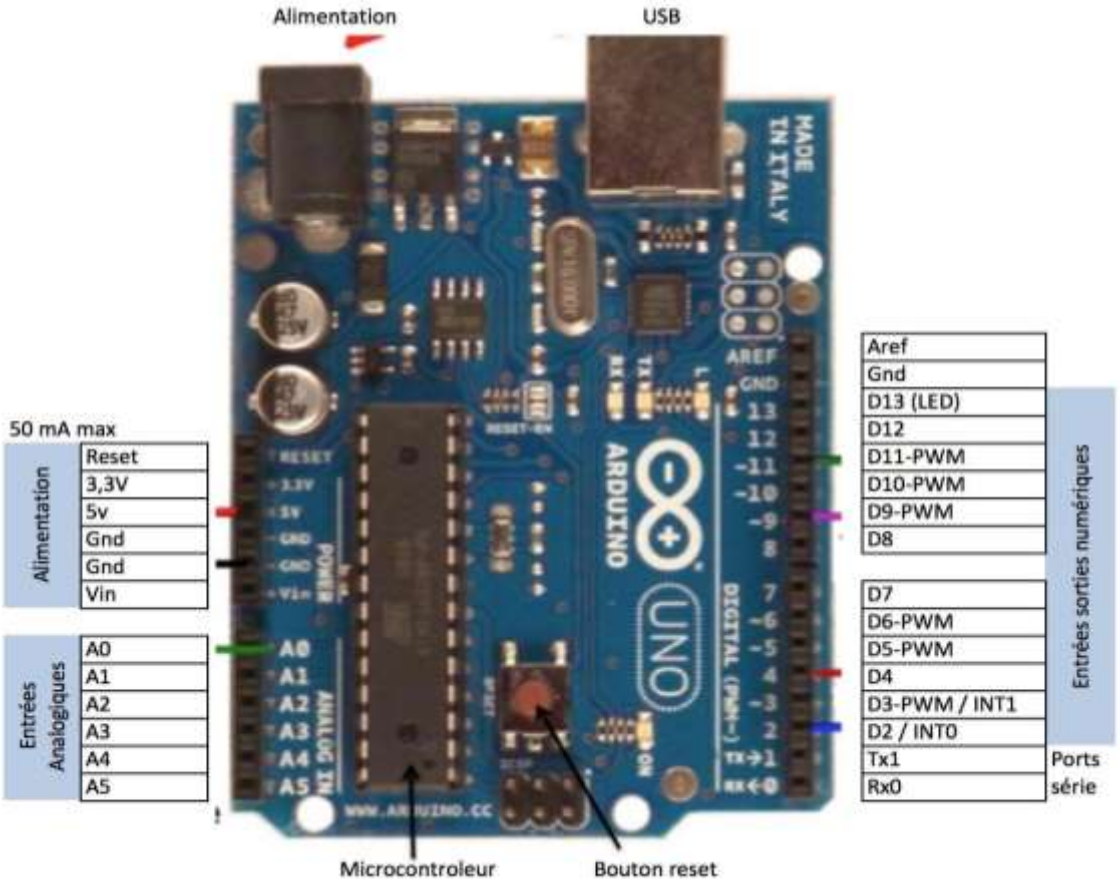
5. **commander en tension** (par MLI) un **moteur** à courant continu à l'aide d'un **potentiomètre sans gestion du sens** de rotation puis **avec gestion du sens de rotation** (arrêt du moteur sur position centrale du potentiomètre) ;

6. **commander en tension** (par MLI) un **moteur** à courant continu à l'aide d'un **potentiomètre sans** gestion du **sens de rotation**, puis avec gestion du sens de rotation et afficher dans le moniteur série **la position angulaire de la roue** (dT=0,1 s).

Pour **chaque activité** :

- **Débrancher le câble** USB
- **Réaliser le montage** sur la plaque de prototypage en vous aidant des documents ressources. **Faire valider le montage par l'enseignant avant de brancher le câble USB !!**
- **En parallèle, écrire** le programme associé au montage (éléments de code dans **Bouts_de_code.html**).
- **Téléverser** le programme et tester le.

Schéma de la carte Arduino Uno et MLI

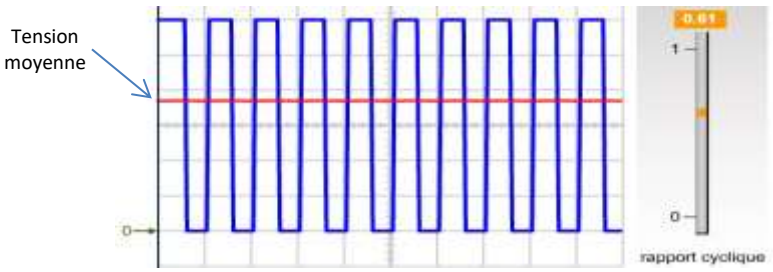


Les microcontrôleurs des cartes Arduino ne disposent d'aucun CNA (Convertisseur Numérique Analogique). Il n'est donc pas possible d'obtenir une tension image d'une donnée numérique.

Par contre, ces microcontrôleurs peuvent générer un signal pseudo-analogique : **signal logique rectangulaire de fréquence élevée qui ne peut prendre que deux valeurs 0V ou 5V**, dit PWM (Pulse Width Modulation en anglais) ou **MLI** (Modulation de Largeur d'Impulsion).

Comme la **fréquence** de ce signal est **élevée** :

- si le système connecté à la sortie PWM est plus lent (type préactionneur du moteur à courant continu), il ne voit à ses bornes que la tension moyenne du signal PWM (il fonctionne comme un filtre) ;
- si le système connecté à la sortie PWM est instantanément rapide (type diode où le temps de réponse est de l'ordre de la nanoseconde), cette dernière clignote à cette fréquence élevée. Mais ce sont nos yeux qui jouent le rôle de filtre en ne retenant que la valeur moyenne de l'intensité lumineuse perçue…



Voir http://fisik.free.fr/ressources/hacheurserie.swf (**mettre** la valeur de l'**inductance et** la valeur de la **fréquence au maximum, puis faire varier le rapport cyclique**).

Le **rapport cyclique** désigne le rapport entre la durée où le signal est au niveau haut (sur une période) sur la durée de cette même période.

Différences Python – C

Commentaires

`# commentaire court` `""" commentaire sur plusieurs lignes. """`	`// commentaire court /* commentaire sur plusieurs lignes */`

Séparateurs d'instructions

Caractère **;** ou changement de ligne	Caractère **;**

Types et déclaration

Déclaration dynamique et typage dynamique
`i=1 // déclare i et affecte la valeur 1`
 	
`types : int, float, str, bool`
	Les types précisent la taille et, pour les entiers, s'ils sont signés ou non. Les variables doivent être déclarées. Le typage est statique.
	`boolean b; // booléen`
	`char c ; // caractère c`
	`byte i=0 ; // entier non signé sur 8 bits (0-255) et affectation initiale`
	`int i ; // entier signé sur 16 bits`
	`long i ; // entier signé sur 32 bits`
	`float a,b ; // flottants sur 32 bits`
	`string s ; // chaîne de caractère`

Bloc d'instructions

Lignes ayant la même indentation :
`Instruction 1 bloc 1`
`Instruction 2 bloc 1`
`  Instruction 1 bloc 2`
`  Instruction 2 bloc 2`
	Instructions comprises **entre accolades** (l'indentation ne sert qu'à faciliter la lecture)
	`{ Instruction 1 bloc 1`
	`  Instruction 2 bloc 1`
	`  { Instruction 1 bloc 2`
	`  Instruction 2 bloc 2 } // fin bloc 2`
	`} // fin bloc 1`

Déclaration d'une fonction ma_fonction sans argument ni valeur retournée

`def ma_fonction():`
`instruction 1`
`instruction 2`
`instruction 3`
	`void ma_fonction(){ instruction 1 ; instruction 2 ; instruction 3 ; } // fin fonction`

Structure conditionnelle et opérateurs booléen

`If condition :`
`  bloc1`
`else :`
`  bloc2`
	Parenthèses obligatoires autour de la condition :
	`if (condition) {bloc 1 ; }`
	`else { bloc 2 ; }`

Valeurs booléennes : `True, False`
Opérateurs logiques : `or ; and ; not`
Comparaisons : `== ; != ; <= ; < …`
	`true ; false`
	`|| ; && ; !`
	`== ; != ; <= ; < …`

Structures itératives

`while condition :` `  bloc`	`while (condition) { bloc ; }`
`  for i in range(100) :` `  bloc`	Regroupe une initialisation, un test, un incrément ou décrément : `for (i=0 ; i<100 ; i++) { bloc ; }`

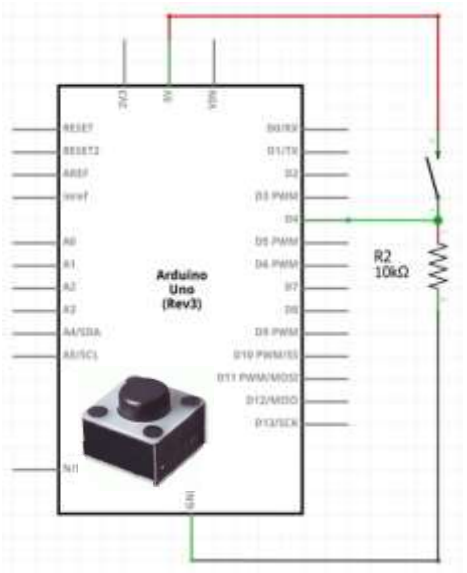
Sortie écran

`print(a,"fin\n")`	La sortie est réalisée sur le port série :
	`Serial.begin(9600) ; // définition vitesse de transmission (9600 bauds ici)`
	`Serial.print(a) ;`
	`Serial.println("fin") ; // avec retour à la ligne`

Bouton poussoir

Montage

La carte Arduino doit recevoir sur une entrée numérique (D4 dans le montage ci-contre) 5V lorsque le bouton est pressé et 0V dans le cas contraire. Pour cela, une **résistance R2 bleue foncée** est utilisée. Elle a pour fonction d'éviter les court-circuits lorsque l'interrupteur est fermé.



Fonctions élémentaires

`pinMode (BPpin,INPUT) ;`	Initialise la broche BPpin en mode entrée
`digitalRead(BPpin) ;`	Lit le niveau de tension sur la broche numérique BPpin. Retourne un booléen.

Minimum à programmer

- **déclarer des variables** associées à la broche et à l'état du bouton
- dans la fonction **setup** : configurer la broche du bouton (**pinMode**)
- dans la fonction **loop** lire l'état du bouton (**digitalRead**)

Trame de programme

```
int BPpin = 4, BPvaleur; // déclare les variables BPpin (numero de broche) et BPvaleur
// comme des entiers. La variable BPpin aura comme valeur 4
void setup() {
pinMode(BPpin,INPUT);      // configure la broche BPpin (ici 4) comme une entrée
// A COMPLETER
}
void loop() {
BPvaleur = digitalRead(BPpin); // lit la valeur de l'entrée numérique BPpin : 0 ou 1
                               // (car les valeurs 0V ou 5V sont converties en 0 ou 1),
                               // puis affecte cette valeur à la variable BPvaleur
  // A COMPLETER
}
```

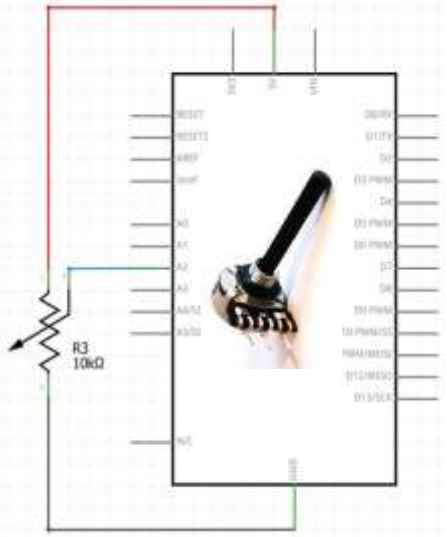
Potentiomètre

Montage

La carte Arduino peut « lire » sur une entrée analogique (A2 dans le montage cicontre) une tension variable entre 0 et 5V.
Il n'y a **pas besoin de résistance** supplémentaire car le potentiomètre est déjà une résistance fixe. Alimenté à ces bornes, la tension au point milieu, une tension comprise 0 et 5V lue ici sur l'entrée A2.

Fonction élémentaire

analogRead(BPpin) ;	Lit le niveau de tension sur la broche analogique Retourne un entier compris **entre 0 et 1023**



Minimum à programmer

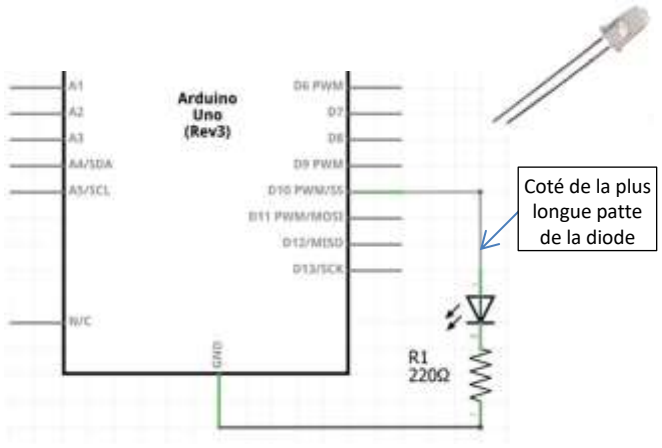
- **déclarer des variables** associées à la broche et à l'image de la tension lue
- dans la fonction **setup** : configurer la broche d'entrée (**pinMode**)
- dans la fonction **loop**, lire la tension sur la broche du potentiomètre (**analogWrite**)

Trame de programme

```
byte Potpin = A2;   // déclare la variable Potpin comme un entier non signé de valeur
A2 int Potvaleur1023;  // déclare la variable Potvaleur1023 comme un entier

void setup() {   // A COMPLETER
} void

loop() {
Potvaleur1023 = analogRead(Potpin);   // lit la valeur de l'entrée analogique Potpin :
       entre 0 et 1023, entier obtenu par un CAN sur 10 bits (convertisseur analogique numérique)
  // A COMPLETER
}
```

Diode

Montage



La carte Arduino doit envoyer sur une de ses sorties (D10 dans le montage ci-contre) une tension numérique :

- de 0 V ou 5 V en numérique ;
- entre 0 et 5 V en MLI (pseudo-analogique). Pour une commande en MLI, il faut que la broche soit associée au symbole ~.

Les diodes électroluminescentes (led) sont polarisées : le pôle "-" est relié à la cathode (la plus courte) et donc le pôle "+" à l'anode (la plus longue). Les diodes doivent toujours être associées dans un montage à un élément de limitation de leur courant. La **résistance R1 bleue claire / verte,** assure cette fonction.

Fonctions élémentaires

digitalWrite(BPpin, val) ;	Pour définir la tension sur la broche BPpin en digital : • +5 V si `val=1` ou `val=HIGH` • 0 V si `val=0` ou `val=LOW`
analogWrite(BPpin, potvaleur255)	Pour définir un signal de type MLI (modulation de largeur d'impulsion) sur la broche BPPin avec un rapport cyclique de `potvaleur255/255`. `Potvaleur255` est un entier compris **entre 0 et 255**.
map(val,min1,max1,min2,max2)	Change l'échelle [min1, max1] en [min2, max2]. Retourne un entier

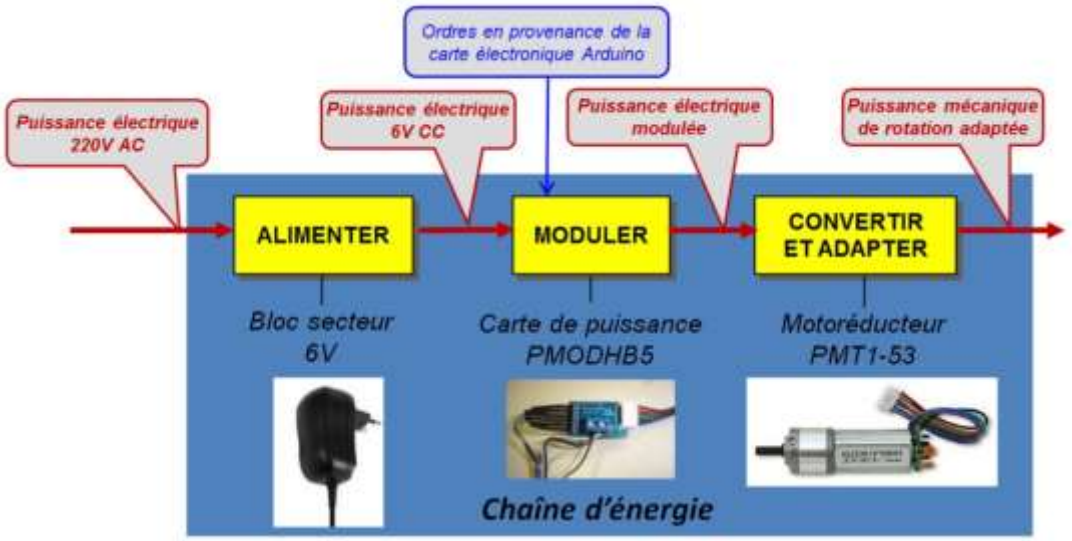
Trame de programme en numérique

```
// COMMANDE DIGITALE
int Ledpin = 10; // déclare la variable Ledpin comme un entier de valeur 10 //
A COMPLETER

void setup() {
  pinMode(Ledpin,OUTPUT);      // configure la broche Ledpin comme une sortie
  // A COMPLETER
}
void loop() {
  digitalWrite(Ledpin,1);      // alume la Led en mettant une tension de +5V sur la broche
// digitalWrite(Ledpin,0);   // éteint la Led en mettant une tension de 0V sur la broche
  // A COMPLETER
}
```

Trame de programme en pseudo-analogique

```
// COMMANDE EN PSEUDO-ANALOGIQUE
int Ledpin = 10; // déclare la variable Ledpin comme un entier de valeur 10
int Potvaleur255; // déclare la variable définissant la tension (entre 0 et 255) // A COMPLETER

void setup() {
  pinMode(Ledpin,OUTPUT);      // configure la broche Ledpin comme une sortie   // A COMPLETER
}
void loop() {   // A COMPLETER
  Potvaleur255 = map(Potvaleur1023, 0, 1023, 0, 255); //change l'échelle de 0 à 1023 en 0 à 255
                    // afin que la variable Potvaleur255 soit compatible avec les sorties MLI
analogWrite(Ledpin, Potvaleur255);   // défini le rapport cyclique de sortie MLI Ledpin
            // (nécessairement compris entre 0 et 255) (NB : 255 signifie +5V en continu)
}
```

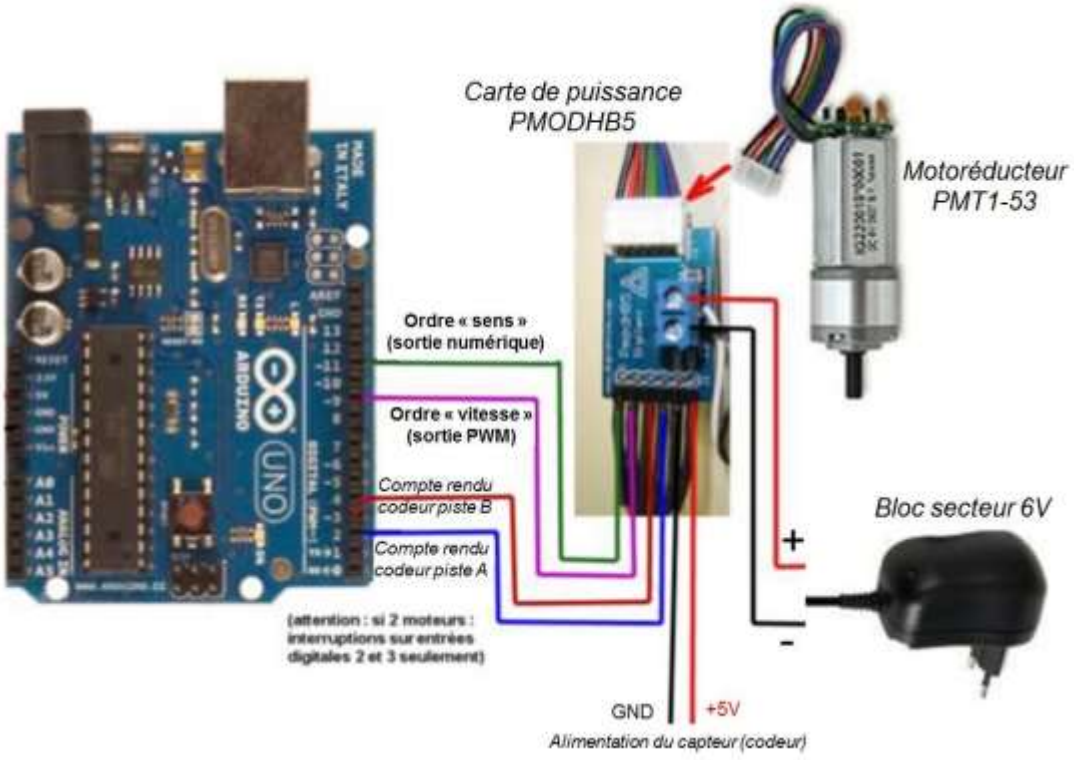
Moteur à courant continu et hacheur PmodHB5



En général, la carte arduino ne peut délivrer la puissance nécessaire à l'alimentation d'un moteur. C'est un composant de la chaîne d'information. Par contre, elle peut piloter une carte de puissance : un **hacheur**.
Ainsi, la carte de puissance PMODH5 est commandée par un signal MLI (PWM) en provenance de la carte Arduino. Le hacheur fonctionne comme un interrupteur piloté par le signal MLI, à haute fréquence donc.

La fonction MODULER est alors réalisée conjointement par le microcontrôleur (génération du signal MLI) et le hacheur.

Le motoréducteur possède également un encodeur utilisant 2 capteurs à effet hall positionnés en quadrature (détaillé plus loin).

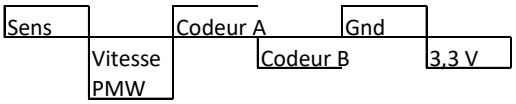


Broches du hacheur PmodHB5

Branchement moteur

Alimentation de puissance

Branchement Arduino



Montage sans codeur

Pour commander le moteur brancher :

- l'alimentation, masse et 3,3 V
- la **broche Sens** sur une sortie **logique** de la carte Arduino
- la **broche Vitesse** PWM sur une **sortie MLI** de la carte Arduino.

Ne pas brancher les connexions « compte rendu codeur piste A » et « compte rendu codeur piste B ».

Éléments pour la programmation de la commande de l'avance

À réaliser dans la fonction **setup** :

- **configuration** des deux sorties (fonction pinMode)
- **initialisation du sens** de rotation (fonction digitalWrite)
- **initialisation de la commande** du moteur (fonction analogWrite(vPin,0))

A réaliser dans la fonction **loop** :

- **Commande du moteur** (fonction analogWrite)
- si le programme le nécessite, **commande du sens de rotation** (fonction digitalWrite)

Montage avec codeur

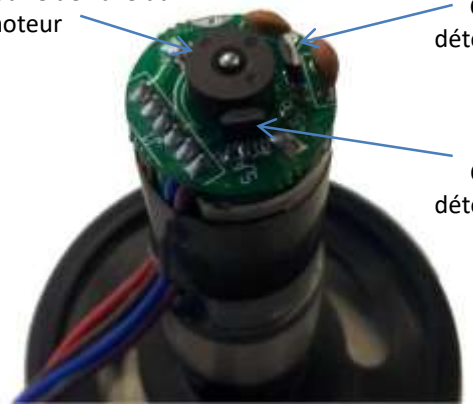
Encodeur à effet hall et logique du codeur

Le moteur comprend deux capteurs à effet hall décalés angulairement de 90°.

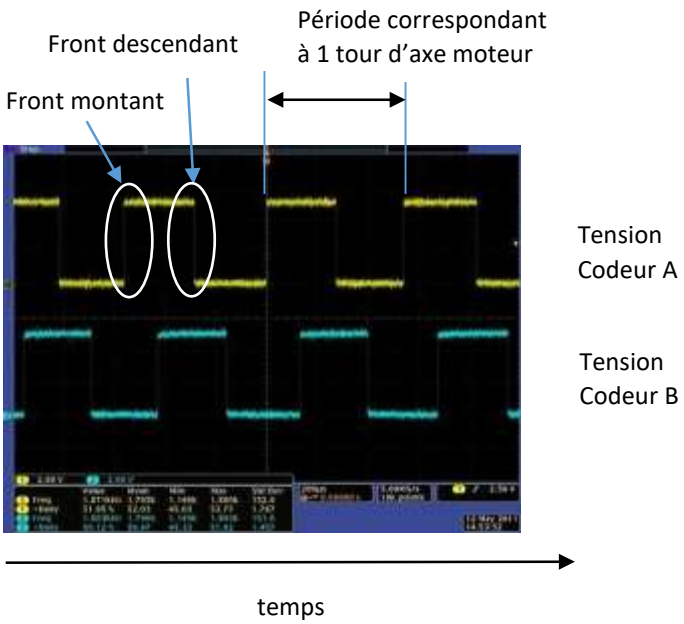
Aimant solidaire de l'axe du moteur

Capteur effet Hall A qui détecte le champ magnétique de l'aimant

Capteur effet Hall B qui détecte le champ magnétique de l'aimant



Si le moteur tourne à vitesse constante, l'électronique associée génère des signaux carrés sur les broches Codeur A et Codeur B de la carte de puissance.
Chaque période d'un signal carré correspond à 1 tour de l'axe du moteur.
Le décalage angulaire de 90° des capteurs se retrouve sur les signaux avec un déphasage de 90°.



On remarque que :

- les fronts montants et descendants d'un même capteur sont espacés de 180°. Compter ces fronts permet de compter des incréments de 180° de l'axe moteur
- dans le sens de rotation de la figure, lors d'un front montant du capteur A, la valeur du capteur B est de 0 V ; lors d'un front descendant, la valeur du capteur B est de 5 V.
- dans l'autre sens de rotation, lors d'un front montant du capteur A, la valeur du capteur B est de 5 V ; lors d'un front descendant, la valeur du capteur B est de 0 V.

D'où, en notant *n* le nombre d'incréments de 180°, on obtient la règle suivante :

- si, après un front du codeur A, état codeur A ≠ état codeur B, alors : $n = n+1$
- si, après un front du codeur A, état codeur A = état codeur B, alors : $n = n-1$

Sur une carte Arduino Uno, il y a deux broches d'interruption externe (**nommées 0 et 1**), qui correspondent respectivement aux **broches 2 et 3**. L'intérêt d'une broche interruption externe est qu'elle permet, comme son nom l'indique, d'**interrompre le déroulement du programme aux changements d'état des broches pour effectuer un traitement spécifique**. Ce traitement est défini par **une fonction** et consiste, en l'occurrence, à la mise à jour du compteur d'incréments. Après traitement, le programme principal reprend.

Montage des broches

Pour obtenir une information de position ou vitesse (en utilisant une seule interruption), brancher aussi :

- la broche Codeur A sur une entrée logique **avec interruption** (broche D2 ou D3) de la carte Arduino
- la broche Codeur B sur une entrée logique.

Éléments de programmation pour compter les incréments

À réaliser dans la fonction **setup** :

- **déclarer une variable** pour le compteur d'impulsion de type `volatile int`
- **configurer les broches** des codeurs en entrée (**modePin**)
- **attacher une fonction** à la broche d'interruption en définissant le **mode de fonctionnement** : appel sur front montant (RISING), descendant (FALLING) ou les deux (CHANGE) (**attachInterrupt**).

En **dehors des fonctions setup et loop** : définir la **fonction appelée à chaque interruption**. Celle-ci modifie le compteur d'incréments en fonction de l'état des codeurs A et B.
(code présent dans Bouts de code.html)

```
int pinCA=2, pinCB=12; // broches des codeurs A et B. Codeur A sur interruption 0.
volatile int inc=0;   // incrément (type entier long) // A COMPLETER
void setup() {
// A COMPLETER
pinMode(pinCA,INPUT);
pinMode(pinCB,INPUT);
attachInterrupt(0,fcodeur,CHANGE); // attache la fonction fcodeur à l'interruption 0.
                            // la fonction est appelée à chaque changement de tension de la broche
}
void loop() {
// A COMPLETER
}
void fcodeur() { // fonction appelée à chaque interruption. Incrémente ou décrémente inc
if(digitalRead(pinCA)==digitalRead(pinCB)) inc++ ;
else inc--;
}
```

Éléments de programmation pour déterminer une vitesse

La vitesse de rotation du moteur en incréments par seconde (inc/s) est définie comme **la variation du compteur d'incréments pendant une durée dt**. Il s'agit donc d'une **valeur moyenne** sur le **pas de temps** dt.
Au code précédent (éléments de programmation pour compter les incréments), il faut ajouter une **boucle de temporisation** qui ne bloque pas les interruptions (boucle while).

```
int pinCA=2, pinCB=12; // broches des codeurs A et B. Codeur A sur interruption 0.
volatile int inc=0;    // incrément
int vitesse;           // vitesse en incréments/seconde long
T0,T1;     // variables pour gérer le pas de temps long
dT=100000; // pas de temps en microsecondes
int ndT=10;     // nombre de pas de temps par seconde (doit être un entier) //
A COMPLETER

void setup() {
    pinMode(pinCA,INPUT); pinMode(pinCB,INPUT);
    attachInterrupt(0,fcodeur,CHANGE); // attache la fonction fcodeur à l'interruption 0 (broche
   D2).
                            // la fonction est appelée à chaque changement de tension de la broche
    Serial.begin(115200);
    // A COMPLETER
}

void loop() {
inc=0; // initialisation compteur à chaque pas de temps
    // attente du pas de temps dT (sans blocage des interruptions)
T0=micros(); // durée de fonctionnement en microsecondes
do {
    T1=micros();
    while (T1-T0<dT);    }
vitesse = inc*ndT; // calcul de la vitesse en incréments/secondes
Serial.println(vitesse);
}

void fcodeur() { // fonction appelée à chaque interruption
if(digitalRead(pinCA)==digitalRead(pinCB)) inc++ ;
else inc--;
}
```