Équipe Robotique d'Exploration — EROBI

Travaux Dirigés

Introduction à Arduino

Laurent Beaudoin & Loïca Avanthey

Épita 2018

1 Introduction

1.1 Contexte et objectifs

Depuis quelques années, une **révolution** (parmi d'autres) est en marche : celle des ***makers***. Elle a pour objectif de permettre à **tout un chacun** de réaliser soi-même des **prototypes** électroniques, robotiques ou domotiques très facilement, c'est-à-dire avec juste un minimum de connaissances et du matériel facilement accessible. Le prototypage peut inclure des parties **mécaniques** (imprimables en 3D ou découpables par une fraiseuse à commande numérique par exemple), **électroniques** ou **informatiques**. L'ensemble de ces composantes permet de réaliser rapidement des projets dont la seule vraie limite est celle de notre **imagination**.

Ainsi, dans le cadre de ce module, nous allons nous intéresser à **réaliser un premier robot d'exploration** bio-inspiré et à le **mettre en œuvre sur un parcours**, le tout en expérimentant des solutions de prototypage rapide grand public de type Arduino. L'objectif du module n'est pas de vous fournir des connaissances approfondies en robotique mais de vous démontrer que c'est **à votre portée** et de vous **donner envie** d'aller plus loin.

Pour commencer, ce premier TD est consacré aux manipulations de base de la carte Arduino.

1.2 Quelques mots sur les cartes Arduino

Une carte Arduino (ou équivalent) est une carte dont l'élément central est un **micro-contrôleur** Atmel AVR cadencé à 16 MHz. Comme nous l'avons vu en introduction du module, un micro-contrôleur est composé d'une unité de calcul (processeur), ainsi que de la mémoire (registres), des interfaces d'entrées/sorties, des protocoles de communication et des périphériques intégrés. Une carte Arduino permet d'utiliser facilement toutes ces capacités pour **divers usages**.

Il existe **plusieurs** types de cartes **Arduino** dont les **spécificités** dépendent du nombre d'entrées/sorties et de la puissance dont on a besoin.

Plus on monte en gamme, plus la carte est grande, plus elle coûte cher et consomme d'électricité. On pourra trouver toutes ces informations sur [Ard18a] et [Wik18a]. Cependant, pour toutes les cartes, on trouvera :

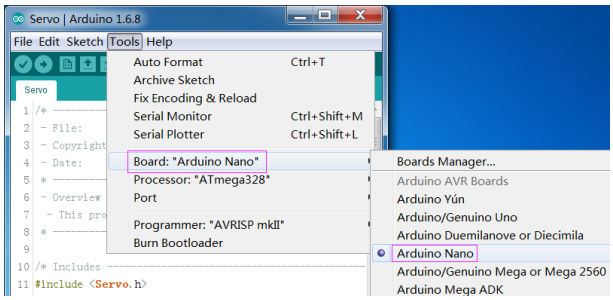
- une **entrée** de type **USB** qui permet d'**alimenter** la carte et de **dialoguer** avec l'ordinateur,
- une série de **connecteurs** numérotés (mâles ou femelles selon les cartes) qui permettent une **interaction** avec l'extérieur (entrées/sorties),
- un **bouton reset** qui permet de **relancer** l'exécution du programme,
- des **LEDs** pour afficher un **état**, comme la mise sous tension ou la communication série,
- une **LED de test** programmable intégrée.

Les **entrées/sorties** peuvent être :
— des entrées/sorties **numériques** (ou digitales). Elles admettent et délivrent des **signaux logiques** compatibles TTL (*Transistor Transitor Logic*), c'est-à-dire composés uniquement de 2 états logiques : un bas à 0 V et un haut à 5 V. Leur numéro de port commence par un `D` sur la carte (de `D2` à `D13` pour la version Arduino nano),
— des entrées **analogiques**. Elles admettent n'importe quelle tension comprise entre 0 V et 5 V et la numérise sur 10 bits (donc entre 0 et 1023). Leur numéro de port commence par un `A` sur la carte (de `A0` à `A7`).
L'**alimentation** électrique de l'Arduino peut au choix se faire :
— via le câble **USB** (5 V),
— via une **source externe** comprise entre 7 V et 12 V et connectée sur les entrées GND (Ground, borne -) et Vin (borne +).

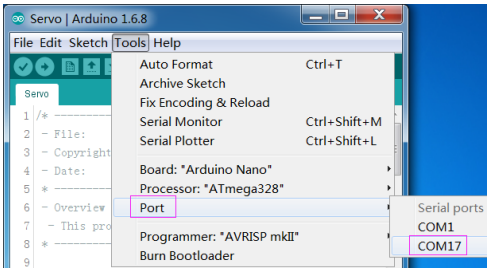
1.3 Installation et configuration logicielle

Pour pouvoir programmer la carte Arduino, il faut **installer l'environnement de développement**. Rendez-vous à l'**adresse** suivante (version anglophone par défaut, paramétrable en français après l'installation) : https://www.arduino.cc/en/Main/Software

Il faut ensuite **configurer** le logiciel Arduino pour qu'il puisse **dialoguer** avec nos cartes électroniques : dans ce TD, nous utilisons des **Arduino Nano**.



1) **Ouvrez** l'onglet `Outils -> Type de carte` (`Tools -> Board`) et **sélectionnez** `Arduino Nano`.
2) **Ouvrez** l'onglet `Outils -> Processeurs -> ATmega328P (Old Bootloader)` (`Tools -> Processors -> ATmega328P (Old Bootloader)`).



3) **Ouvrez** l'onglet `Outils -> Port` (ou `Tools -> Port`) et **sélectionnez** `COMXX` où `XX` est un nombre entier qui dépend du port usb où est branché le câble.

Si vous **rebranchez** la carte sur le **même port USB**, elle sera **automatiquement reconnue**.

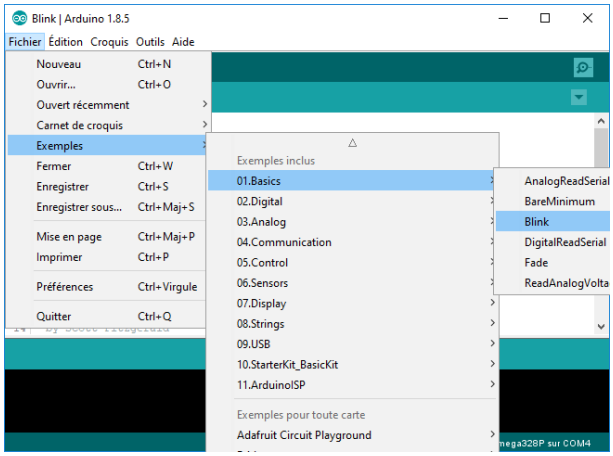
Si le port n'apparaît pas dans la liste, c'est peut-être que le **pilote** de la puce CH340 qui gère la communication USB n'est pas installé sur votre ordinateur.

1.4 Hello World ! ! !

Maintenant que tout l'environnement de développement est installé et que votre carte Arduino est connectée, commençons à programmer.

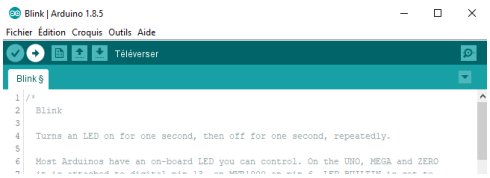
Comme vous l'avez constaté, il n'y a **pas d'écran** sur la carte : on ne peut donc pas commencer par le traditionnel Hello Word de l'informaticien.

On pourrait utiliser le lien entre l'ordinateur et la carte, c'est-à-dire une liaison série sur le port USB, pour utiliser l'écran de l'ordinateur pour afficher des messages, mais c'est un peu trop compliqué pour un premier programme qui doit être simple. On va donc faire un `Hello Word` d'électronicien qui va utiliser un composant électronique tout simple pour communiquer visuellement : une LED (*Light Emitting Diode* ou diode électroluminescente) ! Ainsi, notre premier programme va faire clignoter une LED de test déjà intégrée à la carte !



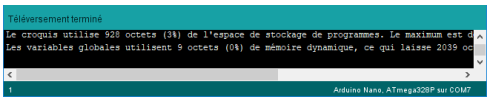
1) **Ouvrez** l'onglet `Fichier -> Exemples -> 01.Basics -> Blink` (`File -> Examples -> 01.Basics -> Blink`).

Les exemples en ligne permettent de démarrer rapidement un prototype de code utilisant les principales fonctionnalités de la carte.



2) **Téléverser** le code de l'ordinateur à la carte électronique en **cliquant** sur le bouton indiqué sur l'image ci-dessus.

L'étape de téléversement permet de **compiler** le code et de le **déposer** à distance dans le **micro-contrôleur**.



3) Lorsque le message `Téléversement terminé` (`Done uploading`) apparait à l'écran, le programme est exécuté dans la foulée sur la carte et le sera automatiquement dès que la carte sera sous tension. Si tout s'est bien passé, vous devriez avoir la LED de la carte qui se met à clignoter à un rythme d'une seconde éteinte, une seconde allumée !

Le langage de programmation utilisé dans l'environnement de développement est proche du `C/C++` [Wik18b, GL11]. Un programme s'écrit dans un **croquis** (*sketch*) qui est le répertoire du projet. Ce programme est au minimum composé des deux fonctions suivantes :

— `void setup()` : cette fonction est **exécutée une seule fois** au moment du **lancement** du programme. Elle contient toutes les **initialisations** indispensables au bon fonctionnement de la carte (définition des entrées, des sorties, le niveau logique au démarrage, la vitesse de fonctionnement d'un port série etc.).
— `void loop()` : cette fonction contient toutes les **instructions** qui seront **indéfiniment répétées** tant que la carte est **sous tension**.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

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

Dans ce programme `Blink.ino`, on retrouve bien nos 2 fonctions `setup` et `loop`. Dans la première, la fonction `pinMode` définit que le port `LED_BUILTIN` (c'est une constante qui dépend de la carte Arduino et pour la carte nano elle vaut 13) est une sortie (`OUTPUT`). Dans la seconde, la fonction `digitalWrite` met le port `LED_BUILTIN` à l'état haut (`HIGH`) ce qui concrètement signifie qu'une tension de 5 V va être émise par le micro-contrôleur. Puis, la fonction `delay` va mettre le programme en pause pendant 1000 ms. Enfin, la fonction `digitalWrite` met le port `LED_BUILTIN` à l'état bas (`LOW` soit 0 V).

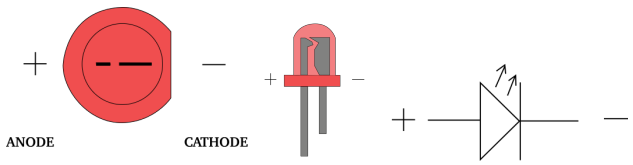
Vous trouverez à la section 3 un récapitulatif des constantes et des fonctions utiles pour réaliser ce TD.

2 Bravo, c'est brillant !

2.1 LED

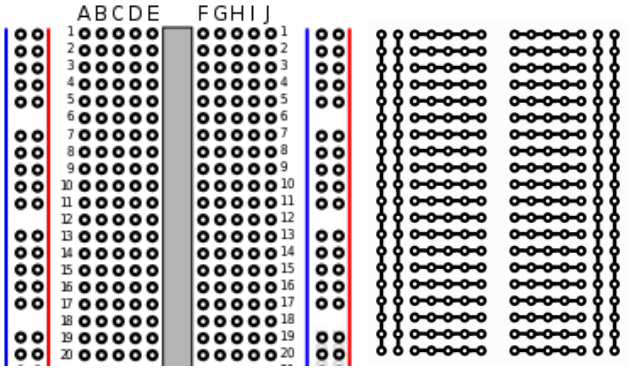
Exercice 1 : LED de test clignotante Écrire une version minimaliste de l'`Hello world` précédent permettant de faire clignoter la LED de test deux fois plus vite (c'est-à-dire qu'elle change d'état chaque $0.5s$).

Exercice 2 : LED clignotante sur un port quelconque Nous allons maintenant faire clignoter une LED autre que la LED de test.



Une LED est un élément polarisé. La tension coté anode (coté patte longue) doit être supérieure à la cathode (coté patte courte et méplat sur l'ampoule)

Prenez une LED rouge parmi celles qui se trouvent dans la boite. Identifiez l'anode et la cathode. Insérez la LED sur la plaque de développement (breadboad) fournie en vérifiant que l'anode et la cathode sont sur 2 lignes différentes.



Sur une breadboard, les 5 trous consécutifs sur une même ligne sont connectés entre eux. Toutes les lignes sont isolées les unes des autres. Les trous appartenants à la colonne + (ou à la −) sont connectés entre eux.

Maintenant, il va falloir alimenter la LED pour l'allumer (état haut).

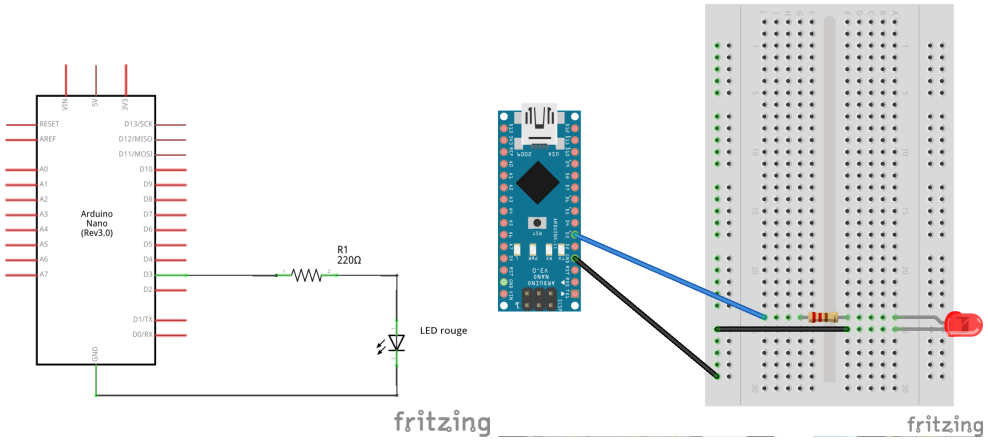
Si on connecte directement la LED aux ports 5 V et GND, ça ne va pas bien se passer… Il faut limiter le courant qui circule dans le LED pour qu'elle ne grille pas !

Pour savoir quel est le courant maximal qui peut traverser la LED sans l'abîmer, on doit consulter la documentation technique (RTFM !). Dans notre cas, la référence de la LED rouge est RL5-R5015. On lit que la **tension d'usage** (*forward voltage*) est de 2 V pour un **ampérage d'usage** mesuré de 20 mA (continuous forward current). Or, la carte ne peut fournir que du 5 V ! Dans ce cas, le courant va être trop fort et va griller le composant. Il faut donc limiter le courant entrant en utilisant une résistance qui va provoquer une chute de tension.

Petit rappel sur le calcul de la valeur de la résistance adéquate :

- La valeur de la résistance R se déduit de la loi d'Ohm $U = R \times I$ avec U la tension en V et I l'intensité en A. Dans notre cas, on utilisera $R = U/I$.
- Des données techniques, la tension U à dissiper par la résistance est de $5 - 2 = 3$ V avec 5 V la tension fournie par l'Arduino et 2 V la chute due à la LED.
- Des données techniques, l'intensité de la LED doit être de 0.02 A.
- La résistance doit donc être de $R = 3/0.02$ soit 150 ohms. On prendra donc une résistance de 220 ohms qui est la valeur immédiatement supérieure à la valeur théorique disponible dans votre kit.

Une résistance n'est pas polarisée. Vous pouvez donc la brancher dans n'importe quel sens.



Réalisez le montage illustré par le schéma précédent. Modifiez le code de l'exercice précédent pour que la LED rouge connectée sur le port 3 clignote.

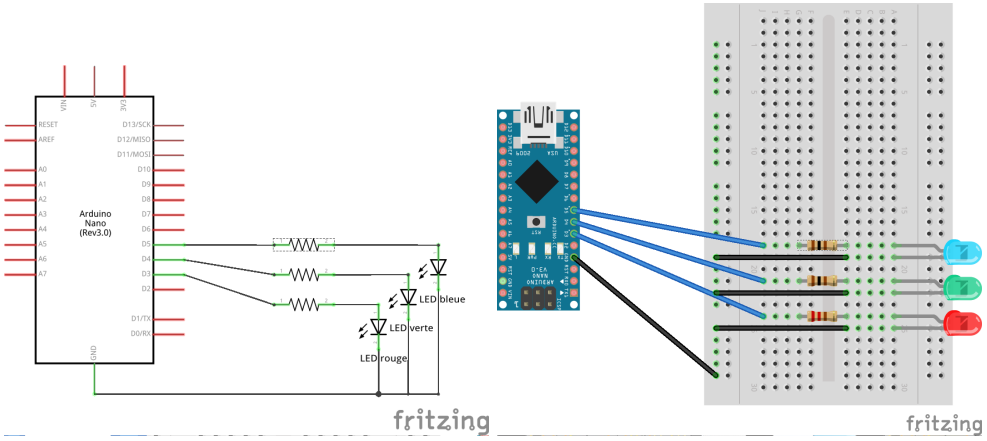
Ampérage maximum par entrée/sortie : 40 mA

Chaque entrée/sortie ne peut fournir ou absorber que 40 mA au maximum ! Au delà, vous risquez de griller l'Arduino !

Dans notre cas, on est bien en dessous de cette consommation maximale sur le port 3.

Exercice 3 : 3 LEDs clignotantes On souhaite maintenant compléter le montage précédent en ajoutant une LED verte sur le port 4 et une LED bleue sur le port 5. Il faut donc calculer la valeur de la résistance à utiliser pour chacune de ces LEDs comme on l'a fait à l'exercice précédent. Pour cela, consultez la documentation technique pour les LEDs verte et bleue (RL5-R5015) et complétez le tableau suivant.

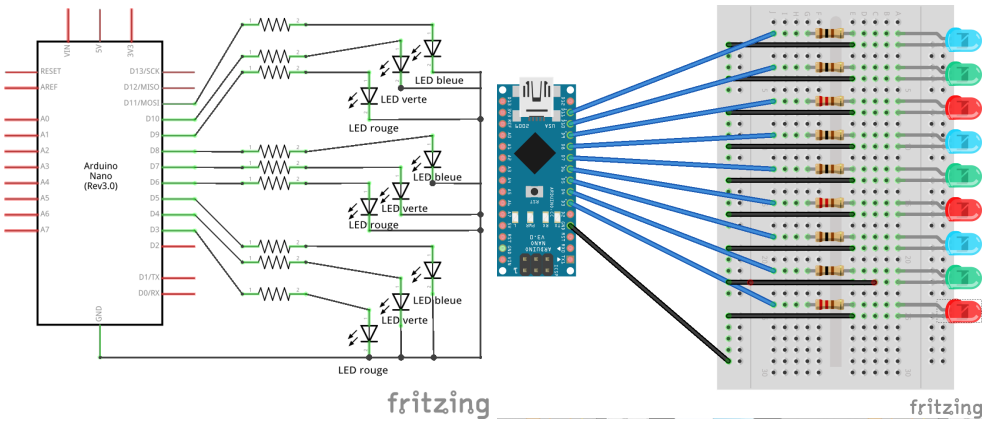
Résistance de	Valeur en ohms
la LED verte	
la LED bleue	



Faites le montage illustré avec les valeurs de résistances calculées. Écrire le code qui fait indéfiniment la boucle suivante :

- allumez la LED rouge, LED verte et bleue éteintes, pendant 0.5 seconde,
- allumez la LED verte, LED rouge et bleue éteintes, pendant 0.5 seconde,
- allumez la LED bleue, LED rouge et verte éteintes, pendant 0.5 seconde.

Exercice 4 : le chenillard à 9 LEDs clignotantes On souhaite maintenant ajouter au montage précédent deux jeux supplémentaires de 3 LEDs rouges, vertes et bleues entre les ports 6 et 11.



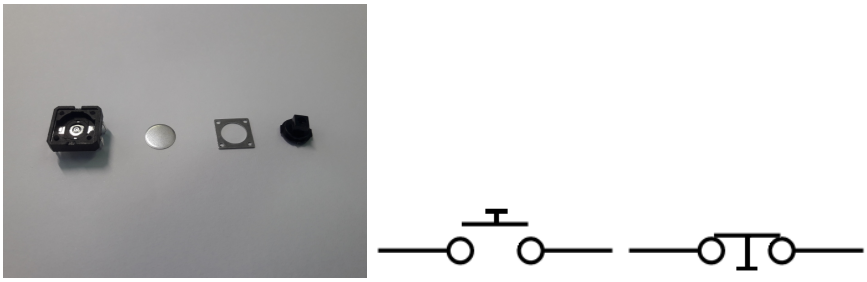


Figure 1 – *Le bouton poussoir de votre kit à gauche et les symboles de bouton poussoir normelement ouvert (centre) et fermé (droite).*

Tout en gardant le même séquencement qu'à l'exercice précédent (rouge puis vert puis bleu), faites clignoter toutes les LEDs les unes après les autres de manière à avoir indéfiniment un chenillard multicolore.

Ampérage maximum en entrée/sortie pour la carte : 200 mA
Le courant total fournit ou absorbé par l'ensemble des entrées/sorties ne doit pas excéder 200 mA !

Dans notre cas, il n'y a pas plus d'une LED allumée à la fois, donc on est loin de la valeur critique. Par contre, si on allume toutes LEDS en même temps, combien peut-on avoir de de LEDs connectées au maximum sur une même carte ?

2.2 Bouton poussoir

Un **bouton poussoir** est un autre composant électronique très classique et très utilisé dans les montages électroniques. Il permet d'interagir avec un circuit (c'est le pendant électronique du "appuyez sur une touche" des programmes informatiques). Il ne faut cependant pas confondre le bouton poussoir avec un interrupteur. Ce dernier a 2 états stables : soit il est ouvert (pas de courant), soit il est fermé (le courant passe). Il peut rester indéfiniment dans l'un ou l'autre de ces 2 états. Le bouton poussoir lui n'a qu'**un seul état stable**. Pour passer à l'autre état, il faut maintenir un effort pour appuyer sur le bouton. C'est donc un état instable : si l'effort s'arrête, le bouton revient à son état stable.

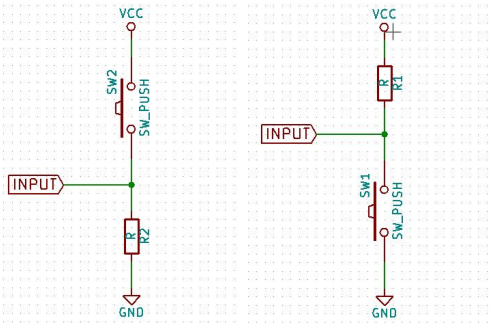
Il existe 2 type de boutons poussoirs. Ceux qui sont **normalement ouverts** (à l'état stable, le bouton ne laisse pas passer le courant) et ceux qui sont **normalement fermés** (à l'état stable, le bouton laisse passer le courant). Dans votre kit, vous avez un bouton qui, une fois démonté, est composé des éléments de la figure 1. Le composant noir à droite appuie sur la rondelle qui est maintenue en place par le carré troué métallique. La rondelle fait alors contact entre les pattes de gauche et celles de droite. Les encoches du bouton délimitent une ligne qui permet d'identifier facilement les pattes qui sont toujours connectées entre elles. Puisque dans notre cas il y a contact que lorsque le bouton est appuyé, il s'agit donc d'un bouton normalement ouvert.

Pour la suite de ce TD, nous allons utiliser l'information "*le bouton écouté sur le port X a été appuyé*" pour modifier le comportement des branches de circuits connectées à d'autres ports. Par exemple, j'éteins la LED du port 3 lorsque le bouton écouté par le port 12 est appuyé. On va donc mesurer l'état du port où est branché le bouton. En effet, lorsque l'on **appuie sur le bouton**, on **change** de tension et donc de **niveau logique**. Ce **port** est donc nécessairement configuré en **entrée** pour pouvoir **écouter**.

Sur le principe, il suffit donc de poser le bouton entre la sortie 5V et la masse pour qu'il y ait du courant qui circule dans le bouton et de brancher n'importe où sur ce circuit le port qui "écoute" le bouton.

Une broche configurée en entrée doit être forcée (tirée) à un état logique connu (haut ou bas) car sinon elle se comporte comme une antenne et donc ne va mesurer que des parasites !

Pour cela, on va utiliser une **résistance de tirage** qui a une valeur assez élevée (10 k ohms par exemple). Le principe est simple : on connecte l'entrée à une patte de la résistance de tirage et l'autre patte de la résistance soit à la masse (**montage pulldown**) soit à l'alimentation (**montage pullup**). L'avantage de cette dernière est qu'il y a toujours un courant qui circule ce qui limite le risque d'avoir des parasites. C'est donc ce type de montage qu'on utilisera pour les boutons poussoirs.



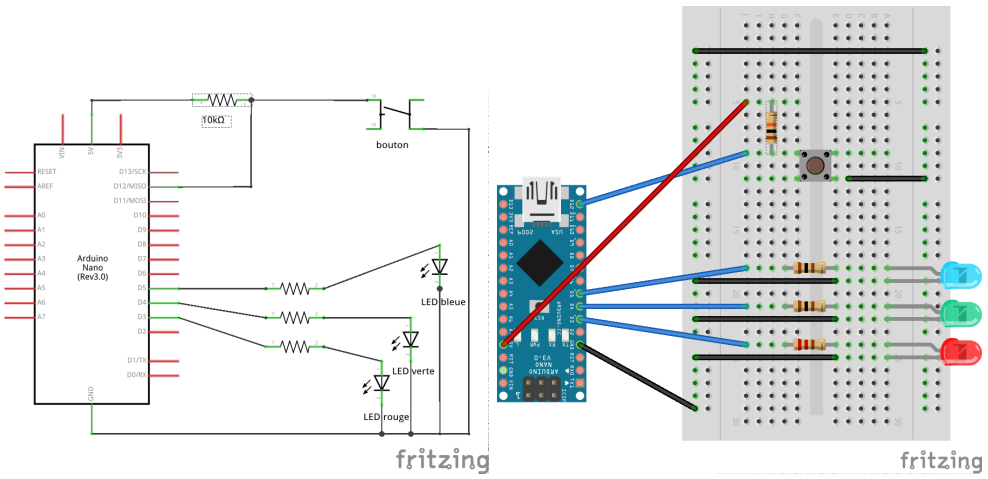
Si la résistance de tirage est liée à la masse, le montage du bouton est dit en pulldown (à gauche). Si la résistance de tirage est liée à l'alimentation, le montage du bouton est dit en pullup (à droite).

Les cartes Arduino ont une résistance de pullup de 20 ou 50 K ohms intégrée à chaque patte et qui sont activables logiciellement.

Jusqu'à présent, on a utilisé les sorties numériques pour alimenter notre circuit. C'est un peu du gâchis de sacrifier une telle sortie numérique si on a juste besoin d'alimenter un bout de circuit en permanence. Pour éviter cela, les cartes Arduino ont une sortie qui délivrent directement du courant en 5 V (et même une autre en 3.3 V).

Attention ! Il ne faut pas dépasser l'ampérage maximal (de l'ordre de 50mA) sur la sortie d'alimentation !

Exercice 5 : Bouton pour commander une LED



On partira pour cet exercice du montage fait pour l'exercice 3, c'est-à-dire avec la LED rouge connectée sur le port 3, la verte sur le port 4 et la bleue sur le port 5 (et sans oublier les résistances là-encore). Au départ, toutes les LEDS seront éteintes.

Sur un bouton, les pattes sur une même diagonale sont forcément déconnectées.

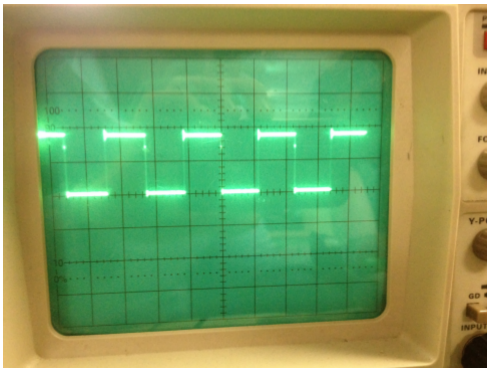
Branchez à la sortie 5V une résistance de 10 K ohms, le bouton et connectez à la masse. Brancher le port 12 à la sortie de la résistance pour obtenir un montage pullup.

Une fois le montage fait, écrire le code qui permet de changer l'état de la LED rouge lorsque l'on appuie sur le bouton poussoir. Pour cela, vous pouvez vous inspirer du code `Fichier -> Exemples -> 02.Digital -> Button`. Attention, dans notre cas, comme on utilise un montage en pullup, on détecte que l'on appuie sur le bouton quand la variable `buttonState` est dans l'état `LOW`.

Exercice 6 : Bouton pour changer l'état d'une LEDs de manière permanente On souhaite maintenant que l'état de la LED rouge change d'état quand on appuie sur le bouton et reste dans cet état quand on le relâche. Par exemple, on allume la LED rouge au démarrage. Lorsque l'on appuie sur le bouton, la LED s'éteint et reste éteinte jusqu'au prochain appui. Essayez le code naïf suivant :

```
int ledR1 = 3;
int buttonPin = 12;
int buttonState = LOW;
int ledState = HIGH;

void setup() {
  pinMode(ledR1, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

    // check if the pushbutton is pressed. If it is, the buttonState is LOW:
  if (buttonState == LOW) {
    ledState = !ledState;
  }

  if (ledState == HIGH) {
    digitalWrite(ledR1, HIGH);
  } else {
    digitalWrite(ledR1, LOW);
  }
}
```

Normalement, vous devriez avoir l'impression que votre montage est plein de faux contacts : un coup, l'appui sur le bouton fonctionne, un coup pas. En fait, ce n'est pas le cas ! Revoyons ce qui se passe au ralenti au niveau de la LED, ou mieux sur un oscilloscope.



On s'apperçoit que l'état de la LED n'arrête pas de changer mais à une vitesse (de l'ordre de 500 Hz) si rapide que l'œil ne le détecte pas ! Du coup, lorsque l'on relâche le bouton, la LED se stabilise dans le dernier état et statistiquement, un coup c'est en état haut, un autre en état bas.

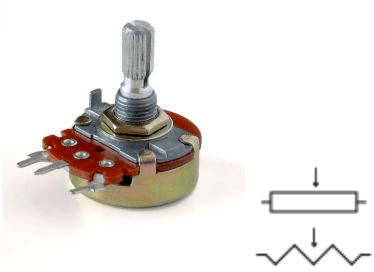


Figure 2 – *Potentiomètre à gauche et son symbole pour les schéma électronique en norme européenne (en haut à droite) et anglosaxone (en bas à droite).*

En fait, on a mal posé le problème. Ce que l'on souhaite vraiment faire c'est détecter quand on a appuyé sur le bouton et non pas si le bouton est appuyé. Dit autrement, le seul événement qui compte c'est quand l'état du bouton change, c'est-à-dire quand on passe de l'état du bouton relâché (état haut pour nous) à l'état du bouton appuyé (état bas). Pour détecter ce changement faut donc introduire une nouvelle variable `lastButtonState` qui contient l'état précédent du bouton et comparer avec l'état actuel du bouton (variable `buttonState`). Si on est en phase de transition, lors de l'appui, l'état `lastButtonState` doit être à haut et celui de `buttonState` doit être à bas. On peut alors changer l'état de la LED (et uniquement dans ce cas là).

Écrire le code qui change l'état de la LED rouge quand on appuie sur le bouton et laisse la LED dans cet état quand on le relâche.

Bien que non perceptible par l'humain, il y a toujours des vibrations de tension (appelées rebonds) au moment du contact physique par le bouton. Pour éliminer ces rebonds, on peut soit mettre un condensateur de 100nF en parallèle du bouton, soit plus simplement attendre 25 à 150ms que les rebonds disparaissent avant de faire la lecture.

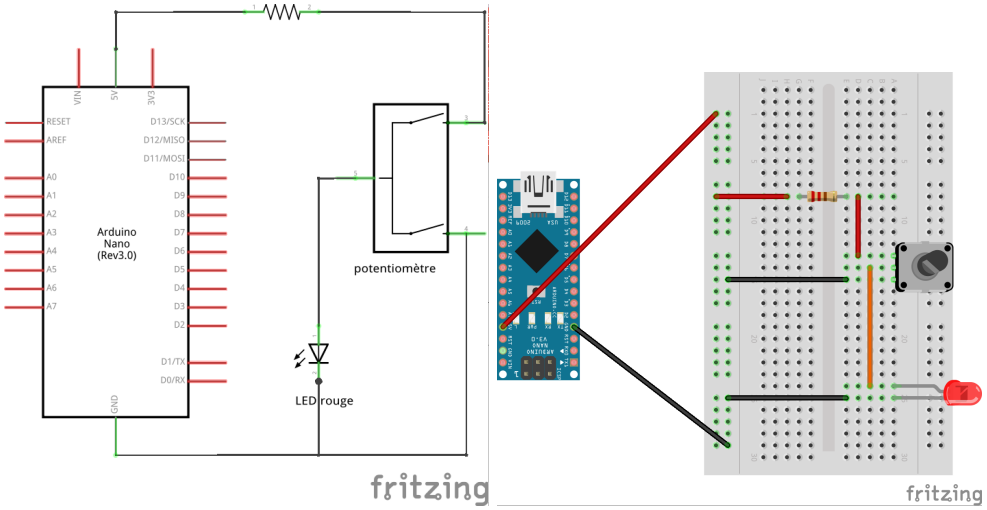
Exercice 7 : Bouton pour commander 3 LEDs On partira du montage de l'exercice 5 avec cette fois la LED rouge allumée au démarrage. Modifier le code pour que l'appui sur le bouton change la LED allumée pour que l'on ait infiniment la séquence LED rouge, puis verte, puis bleue.

2.3 Potentiomètre

Jusqu'à présent, nous avons travaillé sur des entrées et des sorties numériques. Mais les cartes Arduino peuvent aussi prendre en entrée un signal analogique (i.e. continu) et le numériser pour pouvoir être utilisé par le microcontrôleur. Les entrées qui permettent cette numérisation sont les **entrées analogiques** dont le nom commence par un `A` sur la carte. Il y en a 8 (de `A0` à `A7`). La plupart des composants électroniques étant analogiques, il est donc important de pouvoir interagir avec eux. Dans cet exercice, on utilisera un potentiomètre (figure 2).

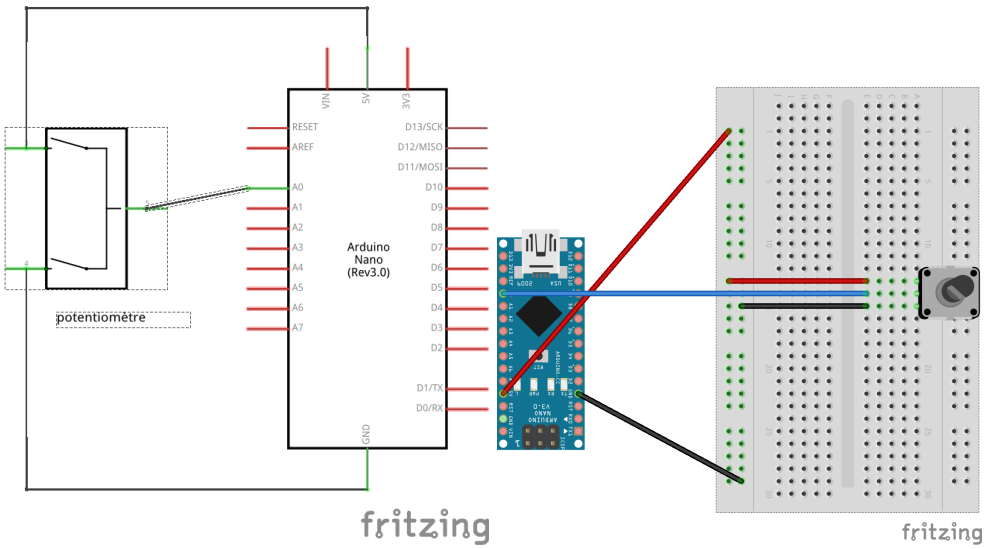
Un potentiomètre est une résistance variable. Il a 3 pattes. Le courant rentre par une patte à l'extérieur et sort par la patte centrale. La dernière patte est à relier à la masse. La patte centrale est reliée à un curseur qui se déplace sur une piste résistante. La position physique du curseur étant variable, la résistance du composant entre une patte externe et la patte centrale l'est aussi.

Un potentiomètre n'est pas un composant polarisé. Il n'y a donc pas de sens de branchement à respecter.



Faites le montage et tournez le potentiomètre pour faire varier la résistance et donc l'intensité du courant qui traverse la LED et par voie de conséquence l'éclat lumineux de celle-ci.

Exercice 8 : clignotement variable Dans cet exercice, on va utiliser un potentiomètre pour faire varier manuellement la période de clignotement de la LED de test.



Branchez le potentiomètre à 5v et à la masse pour les pattes externes et la patte centrale sur l'entrée `A0`. Faites varier le clignotement de la LED de test avec un période correspondant à la mesure digitalisée sur le port `A0`. Pour numériser un signal analogique, on utilisera la fonction `analogRead` (voir section 3).

Exercice 8b : clignotement (vraiment) variable Dans l'exercice précédent, la période de clignotement est directement la valeur lue, ce qui est contraignant. Regardez la fonction `map` dans la documentation de référence et faites varier la période entre 100 et 2000ms.

3 Langage Arduino

Dans cette section, on résume les principales constantes et fonctions utilisées dans ce TD. La documentation de référence est disponible en ligne [Ard18b].

3.1 Constantes

`INPUT`	configure le port en entrée via la fonction `pinMode`
`OUTPUT`	configure le port en sortie via la fonction `pinMode`
`LED_BUILTIN`	contient le numéro du port où est connectée la LED de test intégrée à la carte
`HIGH`	configure un port en niveau haut (voir la fonction `digitalWrite`)
`LOW`	configure un port en niveau bas (voir la fonction `digitalWrite`)
`true`	état logique vrai
`false`	état logique faux

3.2 Fonctions

`analogRead(pin)`	convertit la tension entrante sur le port `pin` entre 0 (pour 0V) et 1023 (pour 5V) de manière linéaire
`delay(ms)`	met le programme en pause pendant `ms` millisecondes
`digitalRead(pin)`	lit le niveau logique sur le port `pin` et retourne la valeur `HIGH` ou `LOW`
`map(value, fromLow, fromHigh, toLow, toHigh)`	conversion par une règle de 3 (linéairement) de la valeur `value` comprise entre `fromLow` et `fromLight` à sa mise à l'échelle entre `toLow` à `toHigh`
`digitalWrite(pin, value)`	Si `pin` a été configurée en `OUTPUT`, alors cette fonction émet une tension à 5V (`value` à `HIGH`) ou à 0V (`value` à `LOW`). Si `pin` a été configurée en `INPUT`, `value` à `HIGH` active la résistance interne de pullup alors que `LOW` la désactive.
`pinMode(pin, mode)`	configure la patte `pin` en entrée (si `mode` vaut `INPUT`) ou en sortie (`OUTPUT`)

Références

[Ard18a] Arduino. Site officiel arduino. https ://www.arduino.cc/, 2018. 1

[Ard18b] Arduino. Site officiel langage processing pour arduino. https ://www.arduino.cc/reference/en/, 2018. 12

[GL11] Jean-Michel Géridan and Jean-Noël Lafargue. Processing, le code informatique comme outil de création. Pearson, 2011. 3

[Wik18a] Wikipédia. Site wikipédia arduino. https ://fr.wikipedia.org/wiki/Arduino, 2018. 1

[Wik18b] Wikipédia. Site wikipédia processing. https ://fr.wikipedia.org/wiki/Processing, 2018. 3