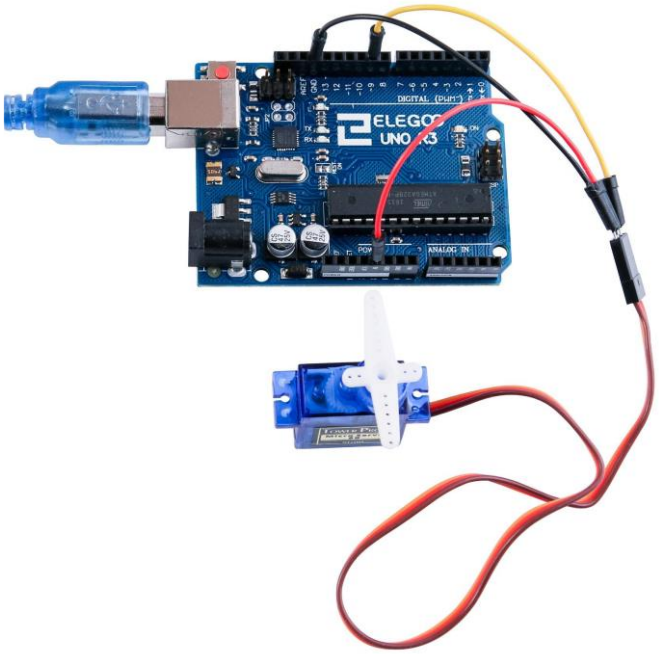
Introduction à la programmation sur ARDUINO



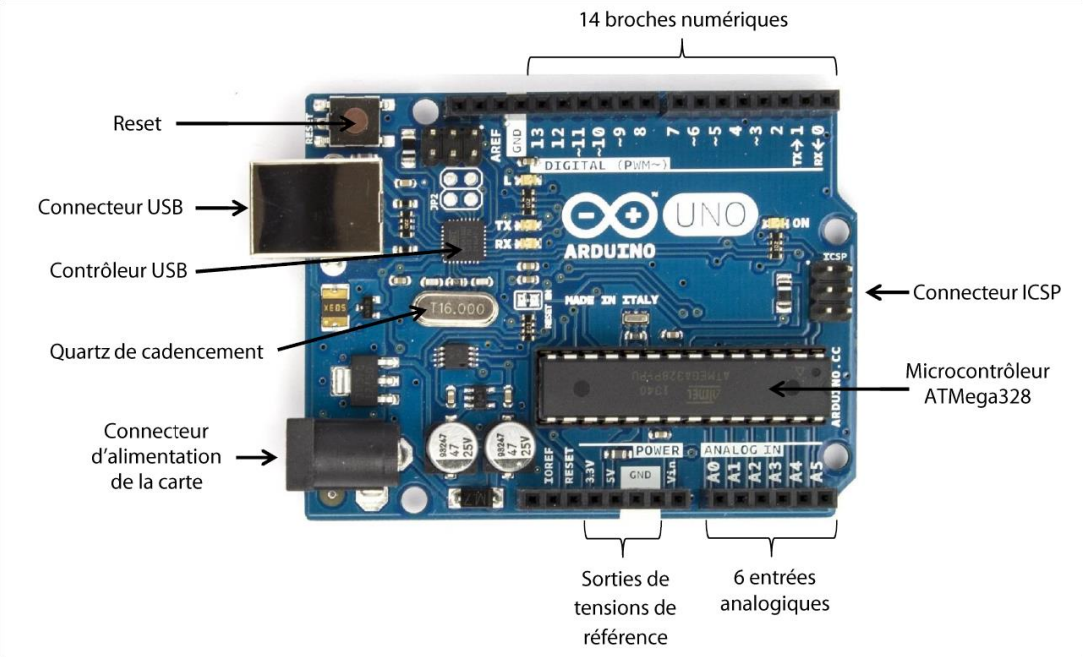
Introduction

De nombreux systèmes industriels sont automatiques : ils exécutent des tâches plus ou moins complexes de manière autonome. Chacune de ces tâches peuvent très souvent être décomposées en tâches élémentaires (que l'on peut décrire avec la chaîne d'information/énergie).

Dans ce TP, on s'intéresse plus particulièrement aux composants de la chaîne d'information, et notamment à la manière dont l'unité de traitement gère les ordres et informations.

Le microcontrôleur Arduino

Les unités de traitement existent sous différentes formes. Les microcontrôleurs en font partie. Leurs couts et les fonctionnalités qu'ils proposent dépendent du cahier des charge du système à concevoir. Pour des raisons de coûts et de simplicité de manipulation, nous utiliserons une carte de type Arduino. Le site officiel www.arduino.cc propose de multiples ressources sur les produits Arduino. La carte de « base » est la carte Arduino Uno. Cette carte est fondée sur un microcontrôleur ATMega328 cadencé à 16 MHz. Les connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires. Cette carte peut se programmer avec le logiciel Arduino, grâce à un cordon de connexion USB. Les entrées/sorties de la carte Arduino sont détaillées sur la figure suivante.



Cette carte dispose :

- d'un microcontrôleur ATMega328
- de 14 broches numériques d'entrées/sorties (dont 6 peuvent être utilisées en sorties PWM 2)
- de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques)
- d'un quartz 16MHz (horloge de cadencement)
- d'une connexion USB (avec son contrôleur associé) qui permet la programmation du microcontrôleur ainsi que l'alimentation de la carte
- d'un connecteur d'alimentation jack (nécessaire si le cordon USB est déconnecté après programmation)
- d'un connecteur ICSP(programmation "in-circuit")
- d'un bouton de réinitialisation (reset)

La démarche de réalisation d'un programme

La carte Arduino est une carte programmable, à volonté, à cela près que le langage de base n'est malheureusement pas Python, mais le langage Arduino (un mix entre le langage C et C++). La syntaxe est bien plus lourde que celle offerte par Python, mais aussi bien plus rigoureuse.

La démarche de conception d'un programme est donnée ci-dessous :

1. On utilise le logiciel Arduino pour écrire son programme.

2. On télécharge (téléverse) le programme dans la carte (il y a une étape de compilation).

3. Le programme étant téléchargé, la carte fonctionne en autonomie si elle a sa propre source d'énergie, ou reste connectée au PC sinon. Il est également possible d'avoir une communication avec le PC pour renvoyer des données mesurées (via la liaison série par exemple).

Prise en main

 Brancher la carte Arduino à l'ordinateur puis démarrer le logiciel Arduino.

Une fois le logiciel démarré, aller dans *Outils*, *Type de carte* puis vérifier que *Arduino Uno* est bien coché (le cocher sinon). Ensuite, vérifier que la carte est bien reconnue en allant dans *Outils*, *Port*. La carte est reconnue si un port est affiché, par exemple « *Com 3* ».

Structure du programme

A l'ouverture du logiciel, on peut remarquer qu'un bout de code est déjà fourni :

```
void setup() {
// put your setup code here, to run once:
}
```

`setup()` est une fonction de type `void`, c'est à dire que cette fonction ne renvoie rien. Il s'agit d'une **procédure**.

Tout le code compris entre les accolades ne sera exécuté qu'une seule fois.

```
void loop() {
// put your main code here, to run repeatedly:
}
```

`Loop()` est également une procédure (type `void`). Elle contiendra le code principal.
Le corps de cette procédure sera exécuté en boucle.

Vérification de la connexion avec la carte Arduino

On désire afficher, sur l'écran de l'ordinateur, une seule fois "La connexion avec la carte Arduino est correcte".

Dans ce cas, la carte communique avec le PC. On utilisera la communication série RS232. Une nouvelle liaison série (plus rigoureusement un nouvel objet de communication série RS232) s'initialise dans le setup avec `Serial.begin(9600);`

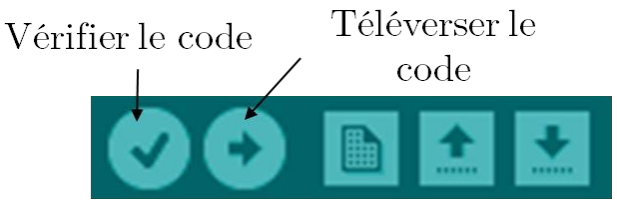
Ensuite, pour afficher un message, on utilise l'instruction `Serial.println("La connexion avec la carte Arduino est correcte");`

A l'aide du logiciel Arduino, créer le programme suivant :

```
void setup() { Serial.begin(9600); Serial.println("La connexion avec la carte
Arduino est correcte ");
}

void loop() {
// put your main code here, to run repeatedly:
}
```

Une fois le code créé, cliquer sur l'icône de vérification du code :



Téléverser ensuite le programme dans la carte, puis sélectionner *Outils/Moniteur série*. Si tout se passe bien, le message apparaît.

Remarque : Dans la commande `Serial.begin(9600);` la valeur 9600 correspond aux caractéristiques de la communication entre la carte Arduino et le PC. Cette valeur peut être modifiée selon le type de communication souhaitée (Bluetooth, …).

Les différents types et les précautions qui vont avec

Contrairement à Python, Arduino requiert qu'on lui renseigne un type à chaque variable créée. Par exemple, si on veut créer une variable a, dont est sûr qu'elle est entière et comprise entre 0 et 255, alors on pourra écrire :

```
byte a=20;
```

Remarque : La valeur maximale est de 255 car le système code en 8 bits, ce qui donne $2^8 = 256$. Les valeurs vont donc de 0 à 255.

Si on affecte la valeur 300 à *a*, on peut observer ce qu'il se passe :

```
void setup() { Serial.begin(9600); byte a=300; Serial.println(a);
}

void loop() {
}
```

Le programme renvoie 44 : il y a eu un dépassement de capacité de la variable. Il faut donc être très vigilant sur le type utilisé.

Remarque : Ici, pour la valeur 300, on 300 = 256 + 44. Vu que l'on travaille ici en base 256 (on travaille sur 8 bits), la carte renvoie donc 44.

Bien heureusement, on peut stocker dans une variable de très "grandes" valeurs, cependant elles prendront plus de place en mémoire. Les types les plus utilisés sont regroupés dans le tableau suivant :

Type de variable	Intervalle	Commentaires
boolean	*True* ou *False*	Occupe 1 octet en mémoire.
char	Caractère ASCII	Occupe 1 octet en mémoire.
String	mots	Occupe 1 octet/caractère.
byte	[0 ;255]	Représentation partielle de N^+ Occupe 1 octet en mémoire.
int	[-32,768;32,767]	Représentation partielle de N Occupe 2 octets en mémoire.
unsigned int	[0 ;65535]	Représentation partielle de N^+ Occupe 2 octets en mémoire.
long	[-2 147 483 648;2 147 483 647]	Représentation partielle de N Occupe 4 octets en mémoire.
unsigned long	[0;4 294 967 295]	Représentation partielle de N^+ Occupe 4 octets en mémoire.
float	$[-3,4028235.10^{38}\ ;3,4028235.10^{38}]$	Représentation partielle de R Occupe 4 octets en mémoire.

Elements de syntaxe

Boucle *for*

Indice croissant

```
for(int i=0;i<=10;i++){
//corps de la boucle
}
```

Indice décroissant

```
for(int i=10;i>=0;i--){
//corps de la boucle
}
```

Boucle *while*

Indice croissant

```
int i=0; while(i<=10){
//corps de la boucle i++;
}
```

Indice décroissant

```
int i=10; while(i>=0){
//corps de la boucle i--;
}
```

Structure conditionnelle if

```
int a=0; int b=2;
if(a==b){ Serial.println("a=b");
}
else if(a>b){ Serial.println("a>b");
}
else{ Serial.println("a<b");
}
```

Remarque : il faut terminer chaque instruction par un " ;".

Affection des Entrees/Sorties

Les affectations des entrées/sorties sont à placer dans la procédure setup. Pour affecter une entrée sur une broche :

`pinMode(8,INPUT)` : Affectation de la broche 8 (logique) en tant qu'entrée.

`pinMode(A0,INPUT)` : Affectation de la broche A0 (analogique) en tant qu'entrée.

`pinMode(8,OUTPUT)` : Affectation de la broche 8 (logique) en tant que sortie.

`pinMode(9,OUTPUT)` : Affectation de la broche 9 (sortie PWM car marquée par le symbole **~** sur la carte) en tant que sortie.

Lire les entrées

Lorsqu'une broche « digitale » ou logique est configurée en entrée, il est possible de récupérer la valeur correspondante en entrée de cette broche via :

```
digitalRead(numero_de_broche)
```

Cette fonction renvoie 1 si la tension en entrée de la broche est supérieure à 3 V, et 0 V si la tension est inférieure.

Il est possible de lire une tension sur une broche analogique, via un convertisseur analogique/numérique, en appelant la fonction :

```
analogRead(numero_de_broche)
```

La tension lue doit varier entre 0 V et 5 V. La valeur renvoyée est un entier contenu entre 0 et 1023.

Remarque : Les valeurs vont de 0 à 1023 car les valeurs analogiques sont codées en 10 bits (2^{10} = 1024)

Imposer une tension en sortie

Sur une broche logique configurée en sortie, on peut imposer soit 0 V soit 5 V :

```
digitalWrite(LOW) // impose  0 V
digitaleWrite(HIGH) //  impose  5 V
```

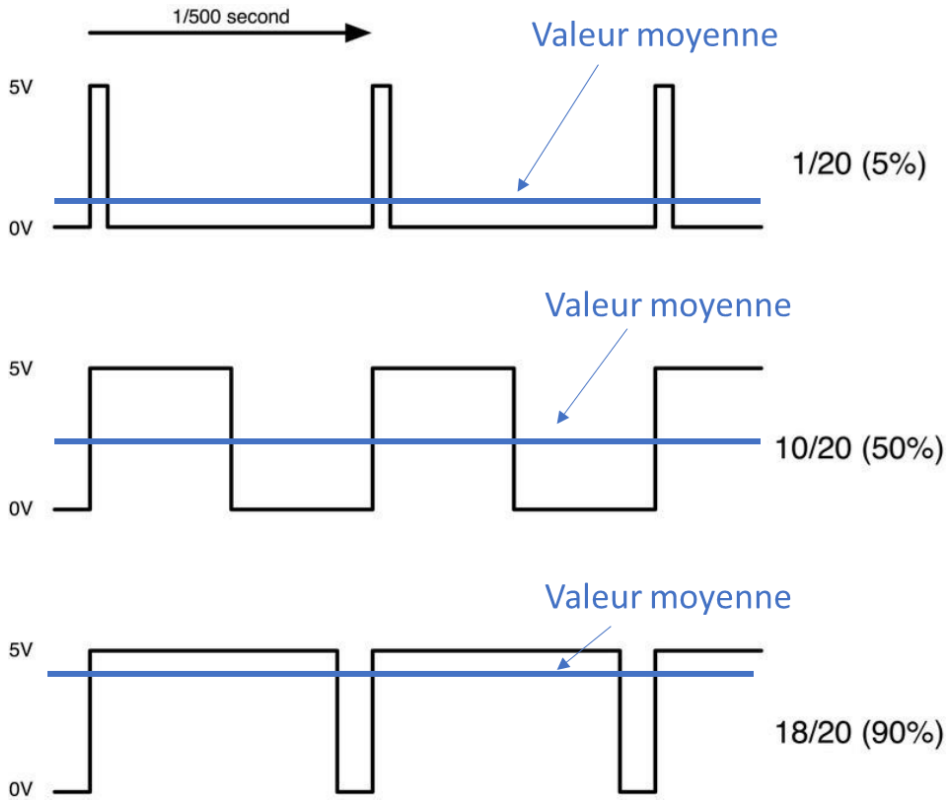
Certaines broches spéciales configurées en sortie, et suivies du symbole ~, peuvent délivrer une tension moyenne variable grâce à la fonction :

`analogWrite(val)` avec val une valeur entière comprise entre 0 et 255. Si `val=0`, la valeur moyenne vaut 0V, si `val=255`, la valeur moyenne vaut 5V. La sortie est obtenue par modulation de largeur d'impulsion (MLI ou PWM en anglais).

PWM (Pulse Width Modulation) :

Pulse Width Modulation (PWM) ou modulation de largeur de pulsation est une technique pour contrôler la puissance d'une sortie. Les PWM permettent par exemple de contrôler la vitesse de rotation d'un MCC ou de contrôler par exemple la luminosité d'une LED.

Les diagrammes suivants illustrent le fonctionnement des sorties PWM de la carte UNO R3 :

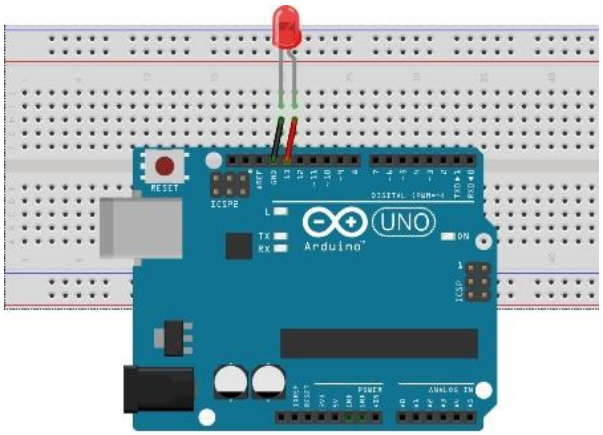


Toutes les 1/500 de seconde, les sorties PWM vont produire une impulsion. La longueur de ces impulsions peut être contrôlée avec la fonction '`analogWrite`'.

Ainsi '`analogWrite(0)`' ne produira pas d'impulsion et '`analogWrite(255)`' produira une impulsion qui durera jusqu'au prochain déclenchement.
Il est donc possible en ajustant la valeur entre 0 et 255 de moduler la puissance en sortie en 0% et 100%. Si la sortie est de 5V sur 100% du temps, en modulant à 90%, nous obtiendrons 90% de 5V.

Les PWM ont un fonctionnement identique à la sortie fournie par un hacheur.

Montage “Led Blink”



- Patte courte de la LED sur **GND**
- Patte longue de la LED sur **D13**

On souhaite réaliser un programme, associé à un montage, qui fait clignoter une LED à intervalles de temps réguliers. Le circuit est donné ci-contre.

Réaliser le montage suivant avec le matériel à disposition, puis compléter le programme fourni ‘Clignoter_LED’ afin de faire clignoter la LED en permanence toutes les 500 ms (LED allumée pendant 250 ms, LED éteinte pendant 250 ms).

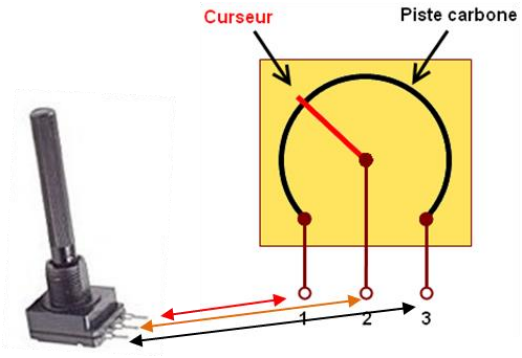
On pourra utiliser la fonction `digitalWrite(numéro_de_pin,LOW ou HIGH)`, pour imposer 0 V ou 5 V aux bornes de la LED.

On pourra également utiliser la fonction `delay()`. Cette fonction permet d’attendre un temps défini. Par exemple, `delay(250)` permet d’attendre 250 ms.

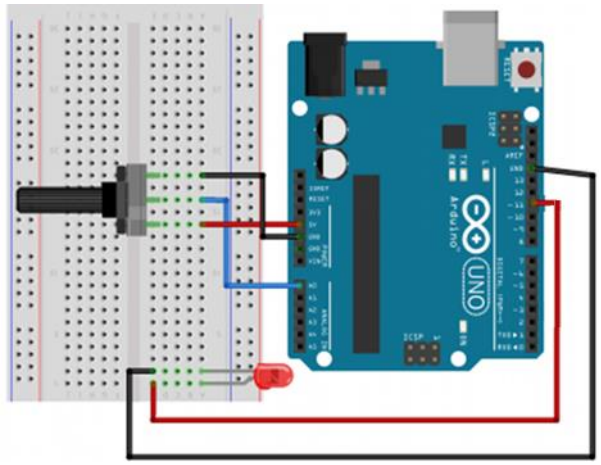
Améliorer ensuite le programme pour qu’il affiche sur le terminal l’état de la LED (allumée ou éteinte).

Variation de luminosité d'une LED

On souhaite réaliser un programme, associé à un montage, qui fait varier la luminosité d'une LED, à partir d'un potentiomètre, dont le fonctionnement est donné.



Principe de fonctionnement d'un potentiomètre rotatif



- Broche 1 du potentiomètre sur **5V**
- Broche 3 du potentiomètre sur **GND**
- Broche 2 du potentiomètre sur **A0**
- Patte courte de la LED sur **GND**
- Patte longue de la LED sur **D11**

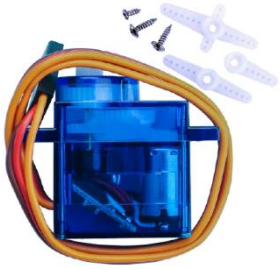
Réaliser le montage suivant avec le matériel à disposition, puis compléter le programme 'Variation_LED' afin d'afficher la valeur lue en sortie du potentiomètre.

Attention : Ici la LED doit être connectée sur **D11** pour utiliser le PWM.

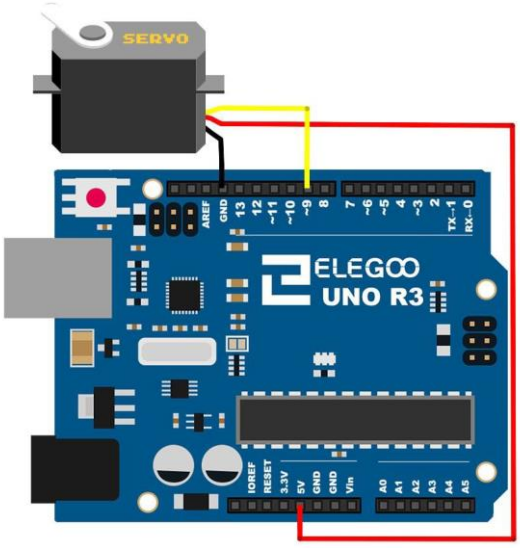
Compléter le programme précédent pour que la luminosité de la LED varie avec la rotation du potentiomètre.

On rappelle que la fonction `analogRead` renvoie des valeurs comprises entre 0 et 1023, alors que la fonction `analogWrite` ne prend que les valeurs comprises entre 0 et 255 en argument. On pourra utiliser la fonction `map` pour régler le problème (ex : `map(val,0,1023,0,255)`).

Montage servomoteur



Un **servomoteur** (souvent abrégé en « servo ») est un **moteur électrique intégré dans une boucle d'asservissement en position**, il s'agit donc d'un système asservi. Sur ce type de moteur, la position est définie avec une limite de débattement d'angle de 180°.



- Fil rouge du servomoteur sur la broche **5V**.
- Fil noir sur la broche **GND**.
- Fil jaune (ou blanc en fonction des constructeurs) sur la broche **D9**.

Après avoir testé le bon fonctionnement du programme 'ServoMoteur', modifier le code pour faire apparaitre la position du servomoteur dans le *Moniteur série*. Afficher la courbe de la position en temps réel grâce au *Traceur série* (aller dans *Outils*/*Traceur série*)

Vous pouvez laisser le servomoteur branché sur la carte, il va vous resservir par la suite.

Montage capteur à ultrasons HC-SR04

Le capteur ultrasons est parfait pour tous les projets nécessitant de faire de la mesure de distances, en évitement d'obstacles par exemple. Le HC-SR04 est un produit peu onéreux et facile à utiliser.

Le capteur permet une mesure d'un objet situé à une **distance allant de 2 cm à 4 m** et fourni une mesure avec une **précision de 3 mm**. Cône de propagation des ondes : **15 °**

Principe de fonctionnement :

1. Un signal de déclenchement (trigger) est émis pendant 10 µs.
2. Le module émet 8 signaux (minimum) à 40 kHz.
3. Le module se met à l'écoute d'un signal de retour.
4. Le temps entre l'émission et la réception est le temps nécessaire au signal pour faire l'aller et le retour vers l'objet qui a réfléchi celui-ci.

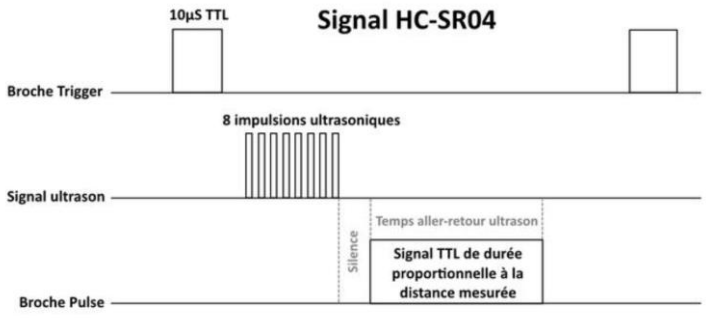
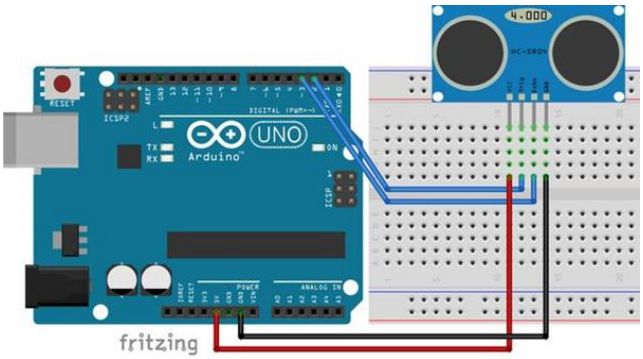


Illustration du signal TRIGGER et ECHO

On a donc :

Distance = (Temps_mesuré * Vitesse_du_son)/2 *(340 m/s pour la vitesse du son)*

Attention : La vitesse du son dans l'aide dépend de la température.



- L'alimentation **5V** de la carte Arduino va sur la broche VCC du capteur.
- La broche **GND** de la carte Arduino va sur la broche GND du capteur.
- La broche **D2** de la carte Arduino va sur la broche TRIGGER du capteur.
- La broche **D3** de la carte Arduino va sur la broche ECHO du capteur.

Après avoir testé le bon fonctionnement du programme 'Capteur_Ultrasons', modifier le code pour faire apparaitre uniquement la distance en mm dans le *Moniteur série*.

Détection d'obstacles

Combiner le montage du servomoteur et du capteur à ultrasons.

En utilisant la pièce transparente supplémentaire, fixer le capteur à ultrasons sur le servomoteur.

Modifier le code pour faire apparaitre les deux paramètres (position angulaire et distance) dans le *Moniteur série.*

Remarque : Le format des données sera le suivant : Sur chaque ligne, les valeurs seront séparées par des VIRGULES.

En récupérant les données et en les copiant dans un fichier texte, tracer la distance mesurée en fonction de la position angulaire. Pour cela, utiliser le programme Python 'Courbe_polaire.py'.

Répéter cette opération avec des obstacles placés à différentes positions.

Vérification de la précision du capteur à ultrasons

Réaliser un protocole expérimental permettant de déterminer la précision du capteur à ultrasons en fonction de la distance entre le capteur et l'objet à détecter.

Vérifier si les données du constructeur sont valides.