

# Initiation à la robotique mobile

## Mise en œuvre matérielle et logicielle

---

Initiation à la robotique mobile  
Mise en œuvre matérielle et logicielle

Support réalisé par P. Gaucher et N. Monmarché  
Version 2 : novembre 2014. M&J octobre 2016



Ce document est en cours de rédaction et peut contenir des erreurs. Merci de bien vouloir les signaler afin d'en améliorer le contenu.

Reproduction et diffusion interdites sans autorisation préalable.

Contacts : [pierre.gaucher@univ-tours.fr](mailto:pierre.gaucher@univ-tours.fr) et [nicolas.monmarche@univ-tours.fr](mailto:nicolas.monmarche@univ-tours.fr)

Ressources :

Arduino :

- <http://arduino.cc>

Robotics with the board of education for Arduino :

- <http://learn.parallax.com/ShieldRobot>

## 1. Introduction : véhicule mobile autonome

De façon très générale, un véhicule mobile possède l'architecture fonctionnelle telle qu'elle est décrite par la figure ci-après (cf. Figure 1).

Le système de contrôle est le « cerveau » de notre véhicule. Il assure la gestion des différents capteurs *extéroceptifs*<sup>1</sup>, qui permettent d'obtenir des informations sur l'environnement au sein duquel le robot évolue (présence d'objets, détection d'obstacles,...). Des capteurs *proprioceptifs*<sup>2</sup> peuvent également être utilisés. Leur fonction permet d'obtenir des informations sur l'état interne du robot (par exemple le nombre de tours de roue lors d'un déplacement). Dans notre cas, ceux-ci ne seront pas mis en œuvre.

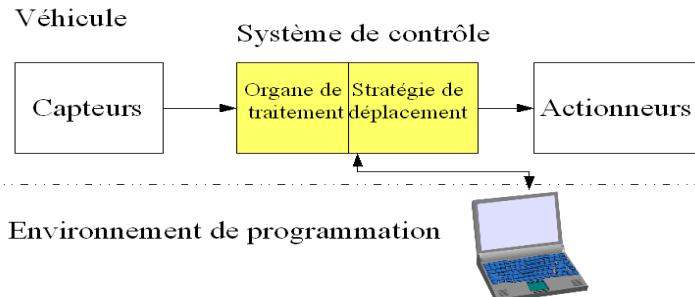


Figure 1: Architecture générale d'un robot mobile

Le traitement des informations issues des capteurs est réalisé par l'organe de traitement. Les résultats issus de ce traitement permettent d'activer les déplacements du robot conformément à la stratégie de déplacement définie au préalable. Un exemple simple de stratégie de déplacement est le suivant :

- le véhicule avance tout droit tant qu'il ne rencontre pas d'obstacle ;
- si la présence d'un obstacle est détectée, le véhicule effectue une rotation d'un quart de tour vers la droite puis repart tout droit.

Enfin, la mobilité du véhicule est assurée par des actionneurs qui, le plus souvent, sont des moteurs électriques ou des vérins.

Les trois éléments, capteurs, organe de commande et actionneurs, constituent l'ossature du robot lorsqu'il fonctionne en mode autonome.

L'environnement de programmation permet de charger la stratégie de déplacement, et de façon plus générale, le modèle comportemental du robot, au sein du système de contrôle. Cette stratégie de déplacement est codée à l'aide d'un langage de programmation compatible avec le matériel utilisé. Cet aspect de la programmation sera détaillé plus loin.

## 2. Organisation

Sous le prétexte de mettre en œuvre un robot mobile, l'objectif de cet enseignement est de sensibiliser, sous une forme ludique, à différents champs de compétences scientifiques ou technologiques. Ainsi, des aspects matériels mais aussi logiciels seront abordés, permettant de se confronter à la mise en œuvre de divers périphériques, mais aussi à la programmation en langage C.

<sup>1</sup> Un capteur extéroceptif permet de doter le robot d'une perception sur son monde extérieur. Cela permet d'adapter le comportement du robot. Les capteurs utilisés sont de nature très variée (capteur TOR, ultrasonore, infra rouge, caméra couleur ou noir et blanc,...).

<sup>2</sup> Un capteur proprioceptif permet de doter le robot d'une perception de son état interne et d'appréhender l'état dans lequel il se trouve. Il peut s'agir par exemple de la mesure de l'autonomie du robot afin de savoir si il doit trouver une borne de charge pour ses batteries.

Le guide proposé n'a que pour objectif de proposer les éléments nécessaires à la prise en main de l'environnement logiciel et de vous guider dans la mise en œuvre du robot. Au-delà des éléments présentés ci-après, il est indispensable de faire preuve d'autonomie et de curiosité afin de dépasser les exemples proposés.

Ce guide d'accompagnement est construit dans le but d'acquérir pas à pas les compétences qui permettront de s'approprier les outils logiciels et matériels afin de pouvoir au final programmer une tâche que le robot devra accomplir en autonomie. Dans un premier temps, les compétences logicielles de base seront abordées (structures de données, typage de variables, structures de sélection, structures itératives,...) au travers de l'utilisation d'une carte microcontrôleur Arduino via l'environnement de programmation proposé. Lorsque cet apprentissage sera effectué, nous pourrons alors aborder la mise en œuvre et la programmation des différents éléments matériels du robot.

### 3. Platine Arduino et environnement de programmation

#### 3.1. Matériel

Le module qui permet de contrôler le robot est constitué par la carte Arduino Uno Rev 3 (cf. Figure 2). Cette carte est architecturée autour du microcontrôleur ATMEL Atmega 328P.

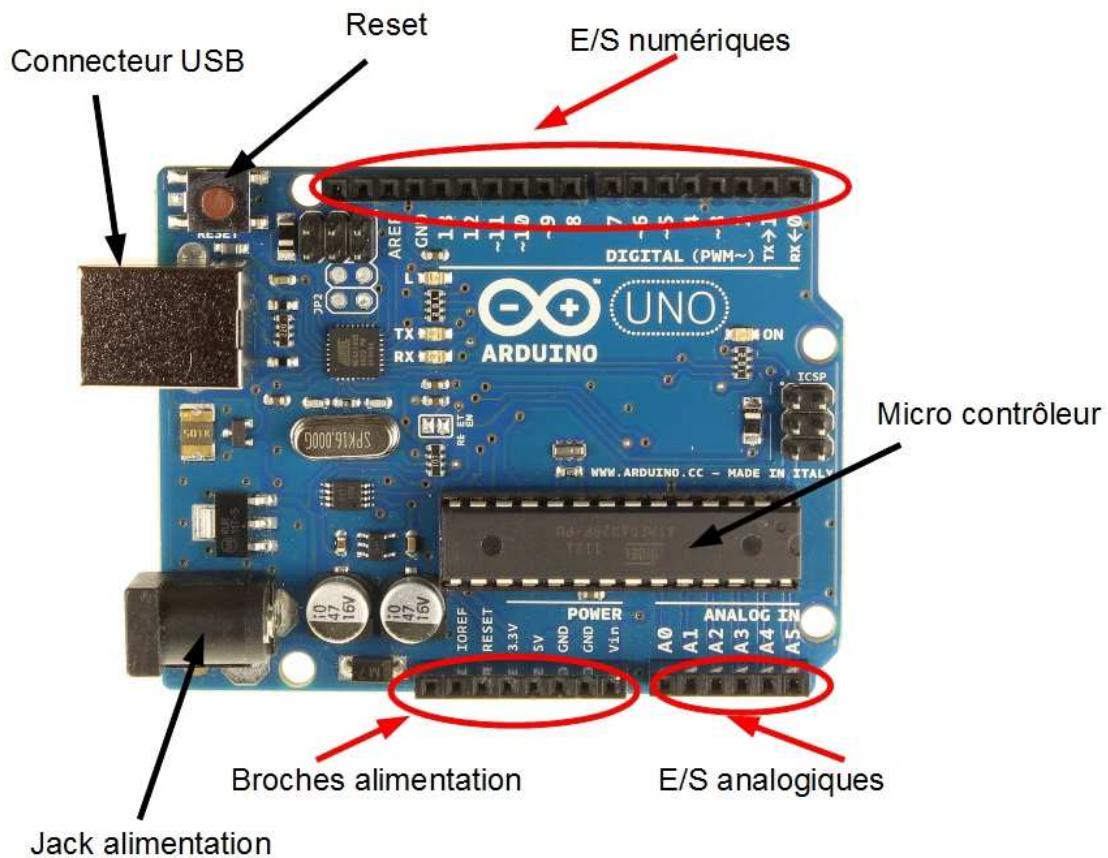


Figure 2 : Carte Arduino Uno Rev 3

Les principales caractéristiques de la carte sont résumées ci-dessous (voir aussi <http://arduino.cc/en/Main/ArduinoBoardUno>)

- 14 broches d'E/S numériques
- 6 broches d'E/S analogiques

- Alimentation par port USB ou prise jack
- Bouton de reset
- 32 Ko de mémoire flash
- Fréquence quartz : 16 MHz

### 3.2. Environnement de programmation

L'écriture de programmes ainsi que leur téléchargement nécessitent l'utilisation du logiciel open source Arduino. Il s'agit d'une surcouche logicielle en Java, qui intègre l'ensemble des outils nécessaires à l'écriture d'un programme ainsi que son téléchargement dans la mémoire du microcontrôleur. La version de l'environnement de programmation utilisée est la version 1.0.5.

#### 3.2.1. Téléchargement et installation

Les ressources logicielles peuvent être téléchargées depuis le site Arduino (cf. <http://arduino.cc/en/Main/Software>). Par la suite, nous travaillerons en environnement Windows, même si d'autres systèmes d'exploitation sont supportés. La version de l'environnement de programmation utilisée est la 1.0.5 (voire plus récente, la version 1.0.6).

Il faut d'abord télécharger l'archive *arduino-1.05-windows.zip*<sup>3</sup>, la stocker dans le répertoire de son choix, puis la décompresser. Dans le répertoire de travail choisi, on doit retrouver les répertoires et fichiers comme ci-dessous (cf. Figure 3).

Nom	Modifié le	Type	Taille
drivers	19/09/2013 16:51	Dossier de fichiers	
examples	19/09/2013 16:51	Dossier de fichiers	
hardware	19/09/2013 16:51	Dossier de fichiers	
java	19/09/2013 16:52	Dossier de fichiers	
lib	19/09/2013 16:53	Dossier de fichiers	
libraries	19/09/2013 16:51	Dossier de fichiers	
reference	19/09/2013 16:53	Dossier de fichiers	
tools	19/09/2013 16:53	Dossier de fichiers	
arduino	17/05/2013 22:26	Application	840 Ko
cygiconv-2.dll	17/05/2013 22:24	Extension de l'app...	947 Ko
cygwin1.dll	17/05/2013 22:24	Extension de l'app...	1 829 Ko
libusb0.dll	17/05/2013 22:24	Extension de l'app...	43 Ko
revisions	17/05/2013 22:24	Document texte	38 Ko
rxtxSerial.dll	17/05/2013 22:24	Extension de l'app...	76 Ko

Figure 3 : Répertoires et fichiers environnement Arduino 1.0.5

#### 3.2.2. Description des fonctions de l'éditeur

Pour ouvrir et lancer l'environnement de programmation, il faut cliquer sur l'icône, ce qui permet l'accès aux différentes fonctionnalités de l'éditeur. Selon la version de

<sup>3</sup> Il est également possible d'utiliser « l'installateur », ce qui peut être plus rapide.

Windows utilisée, il est possible qu'une fenêtre d'avertissement s'ouvre (cf. Figure 4).

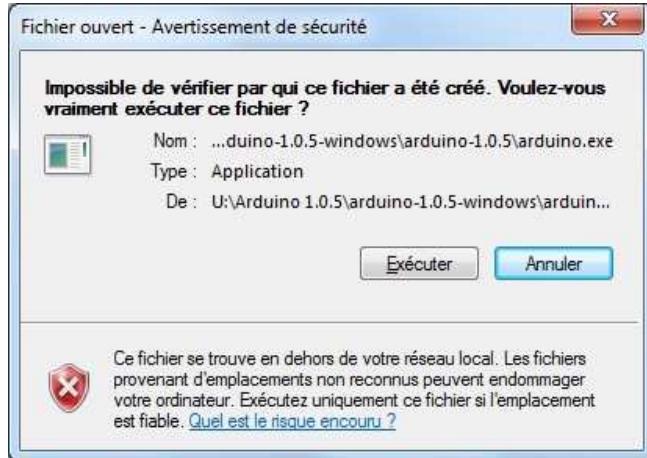


Figure 4 : Avertissement ouverture environnement Arduino

Il faut alors cliquer sur *Executer*. La fenêtre ci-dessous doit alors s'ouvrir (cf. Figure 5).

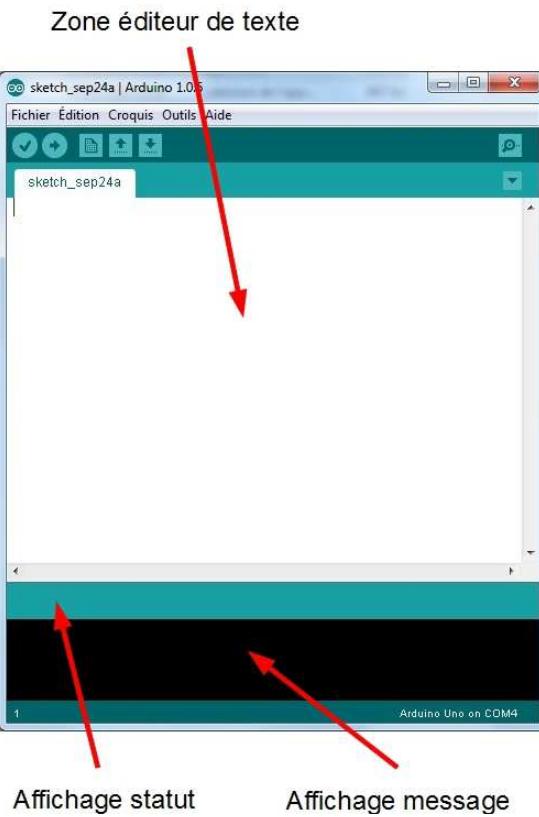


Figure 5 : Environnement de programmation Arduino

Cet environnement dispose de fonctions également accessibles par des raccourcis présents dans la barre contenant les icônes suivants (cf. Figure 6).

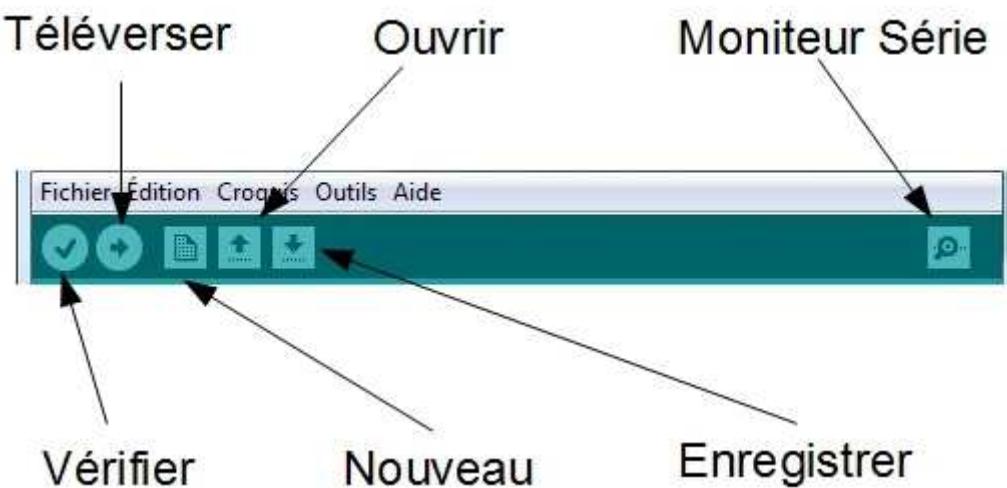


Figure 6 : Icônes raccourcis environnement de programmation Arduino

Dans l'environnement de développement Arduino, un programme est dénommé « croquis » ou « sketch »

- « *Vérifier* » : permet de détecter des erreurs de syntaxe d'un programme ;
- « *Téléverser* » : permet de transférer le code exécutable d'un programme vers la mémoire du microcontrôleur ;
- « *Nouveau* » : pour ouvrir un nouveau croquis ;
- « *Ouvrir* » : pour ouvrir un croquis déjà existant ;
- « *Enregistrer* » : pour sauvegarder un croquis ;
- « *Moniteur Série* » : permet d'ouvrir une fenêtre texte pour afficher des messages texte via le port série.

### 3.2.3. Accès aux ressources logicielles Arduino

L'environnement de développement permet d'accéder aux ressources logicielles suivantes :

- Une base de programmes selon différentes rubriques, en passant par l'onglet *Fichier* puis *Exemples* (cf. Figure 7).
- Un accès direct à des ressources documentaires internet sur le site Arduino, en passant par l'onglet *Aide*.

### 3.2.4. Configuration matérielle

Avant de pouvoir utiliser la carte Arduino et télécharger un programme, il est nécessaire de procéder au pilote de la carte, afin de pouvoir gérer le port de communication. La procédure à suivre est décrite ci-dessous :

- Connecter la carte Arduino sur un des ports USB. Lors de la première connexion, le pilote de la carte est recherché, mais sa recherche automatique échoue (cf. Figure 8). Il est donc nécessaire de procéder manuellement à son installation.
- Fermer alors la fenêtre ;
- Ouvrir le panneau de configuration Windows (cf. Figure 9)
- Cliquer sur l'icône Système (cf. Figure 10) ;
- Puis sélectionner le gestionnaire de périphériques (cf. Figure 11) ;
- Dans la partie « Autre périphérique », sélectionner « Périphérique inconnu » puis clic droit pour demander la mise à jour du pilote. Dans la fenêtre contextuelle qui s'ouvre, choisir « Mettre à jour le pilote ». (cf. Figure 12)

- Une nouvelle fenêtre s'ouvre. Elle permet de choisir le mode de recherche du pilote à installer (cf. Figure 13). Il faut sélectionner « Rechercher un pilote sur mon ordinateur » .

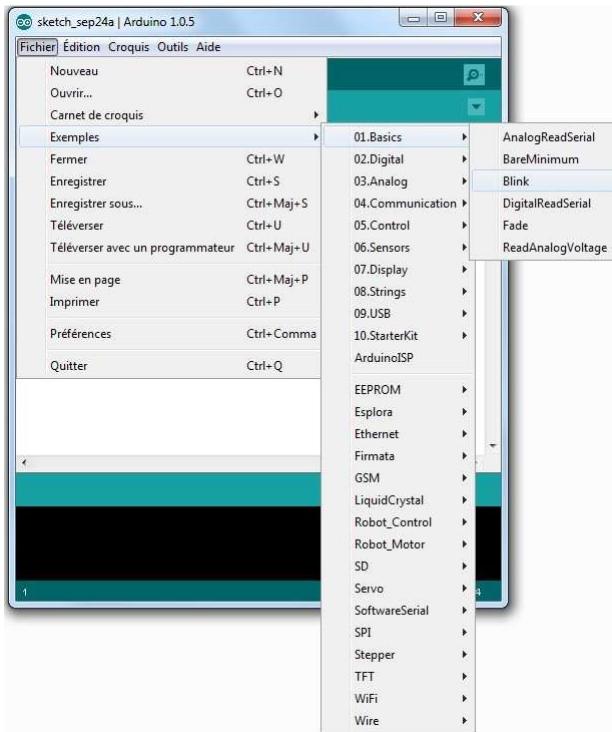


Figure 7 : Bases exemples de croquis Arduino

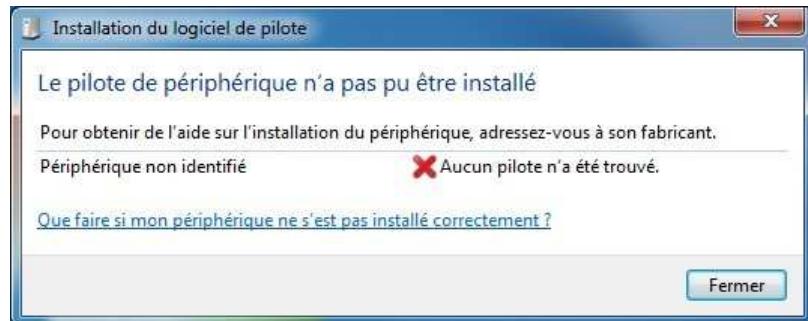


Figure 8 : installation du pilote de la carte

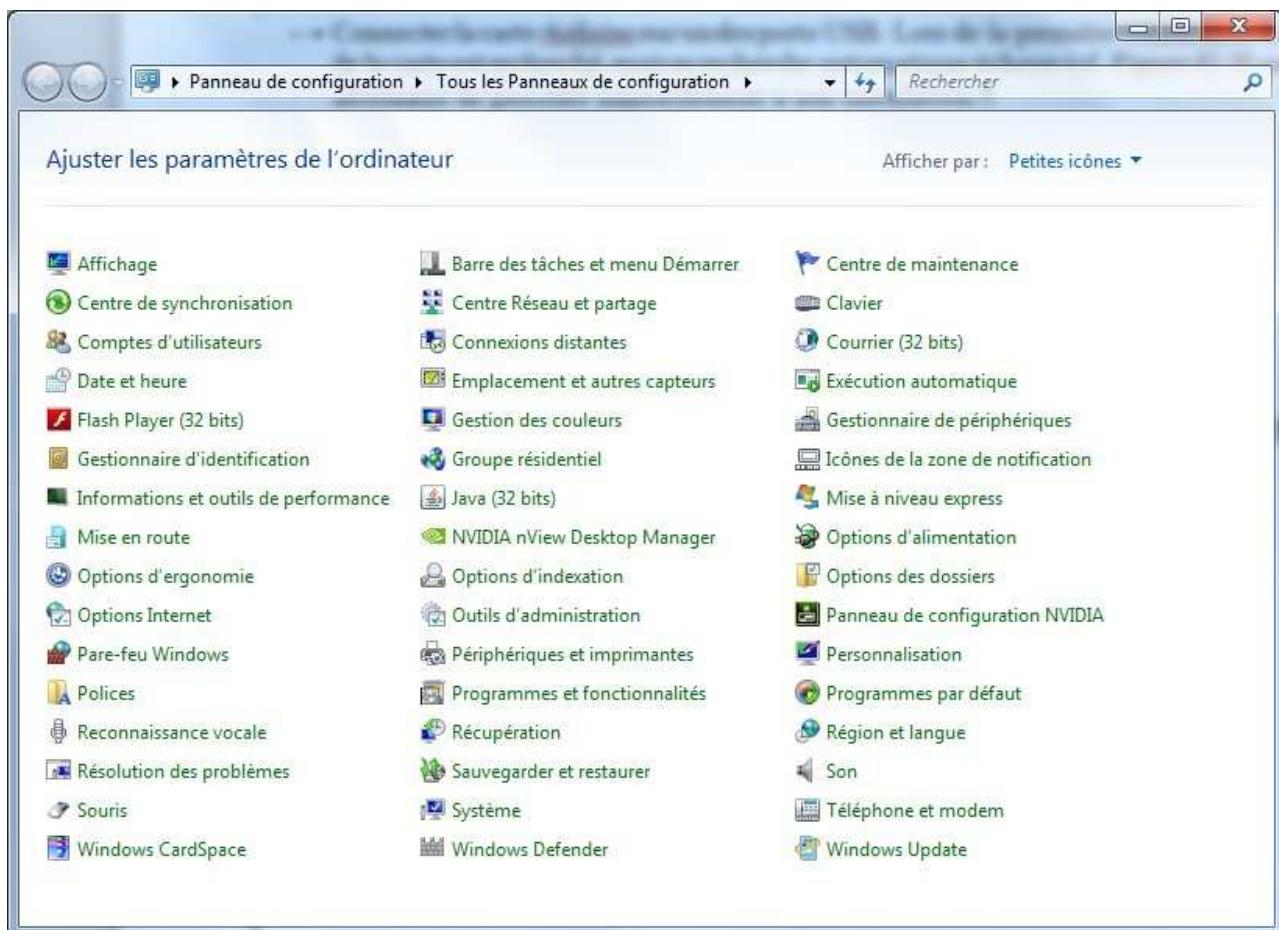


Figure 9 : Panneau de configuration Windows

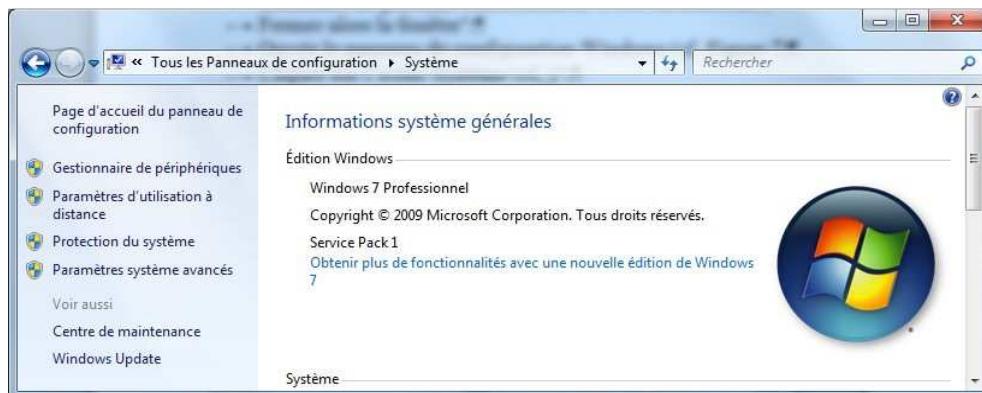


Figure 10 : Panneau de configuration Windows – Système

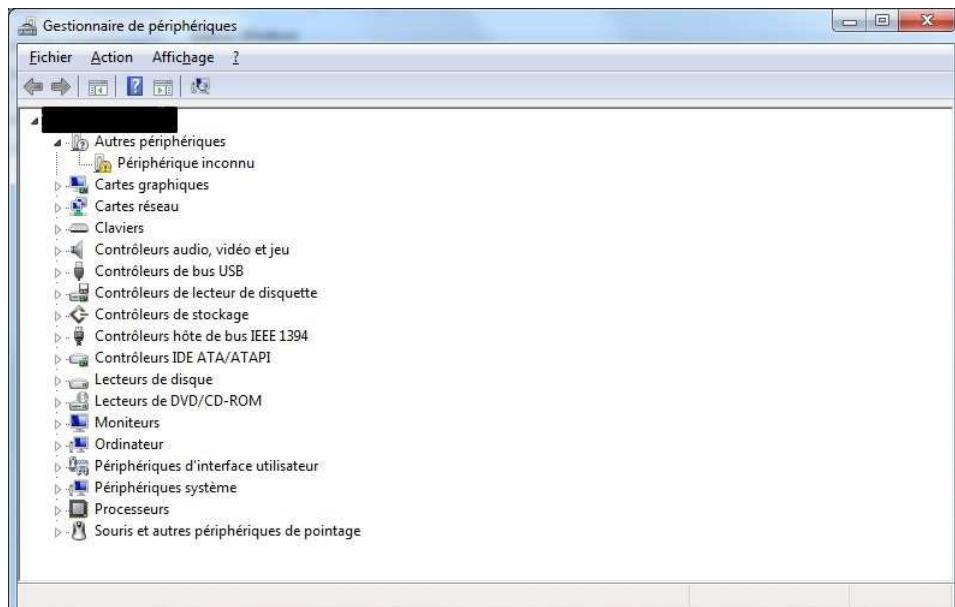


Figure 11 : Gestionnaire de périphériques

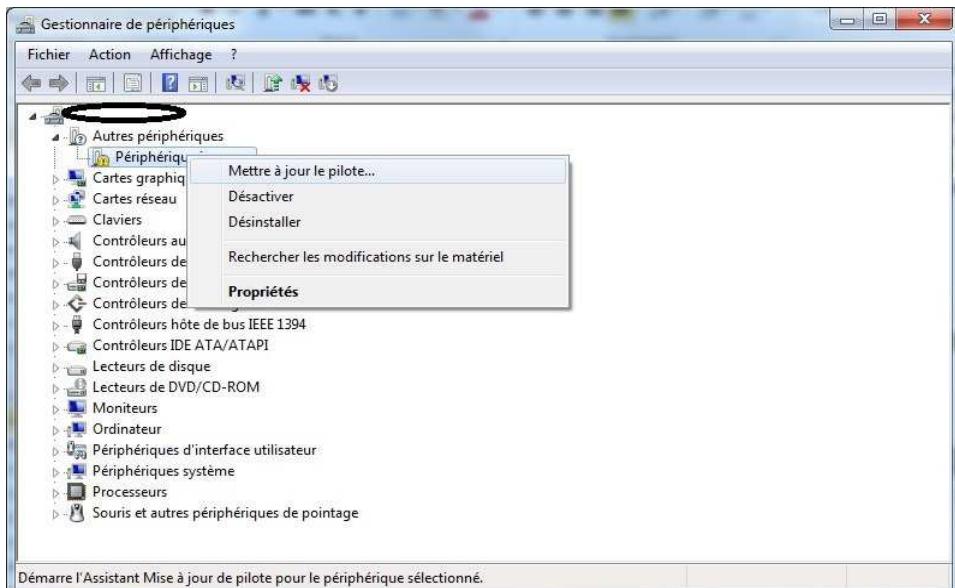


Figure 12 : Mise à jour du pilote

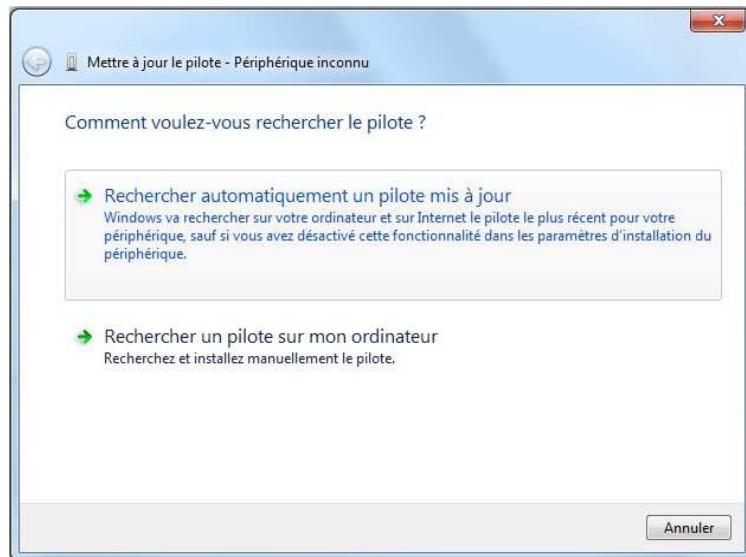


Figure 13 : Recherche du pilote

- Rechercher le répertoire « Drivers » contenu dans le répertoire d’installation de l’environnement Arduino (cf. Figure 14). Valider par « OK ». Cette validation provoque le retour à la fenêtre de mise à jour du pilote (cf. Figure 15). Pour continuer le processus d’installation, cliquer sur suivant. Une fenêtre de confirmation s’ouvre (cf. Figure 16). Pour finaliser l’installation, cliquer sur « Installer ». Lorsque le pilote de la carte est installé, une dernière fenêtre s’ouvre pour confirmer que l’installation est achevée avec succès (cf. Figure 17).



Figure 14 : Sélection du répertoire contenant le pilote

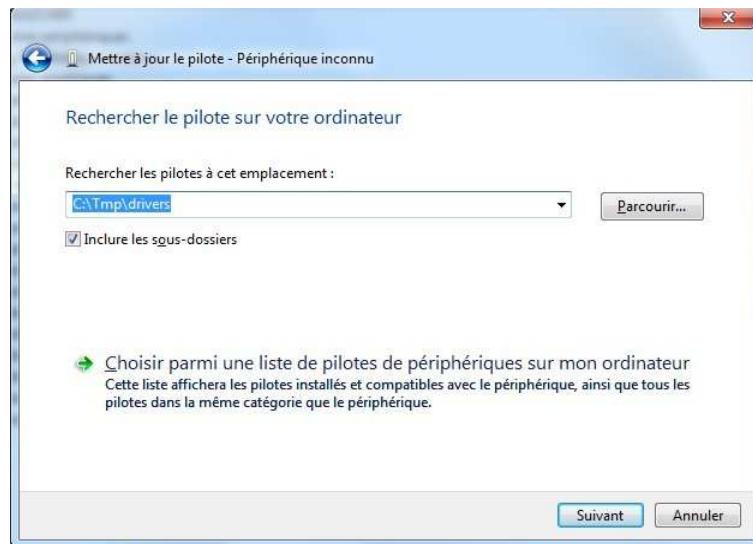


Figure 15 : Validation répertoire pilote

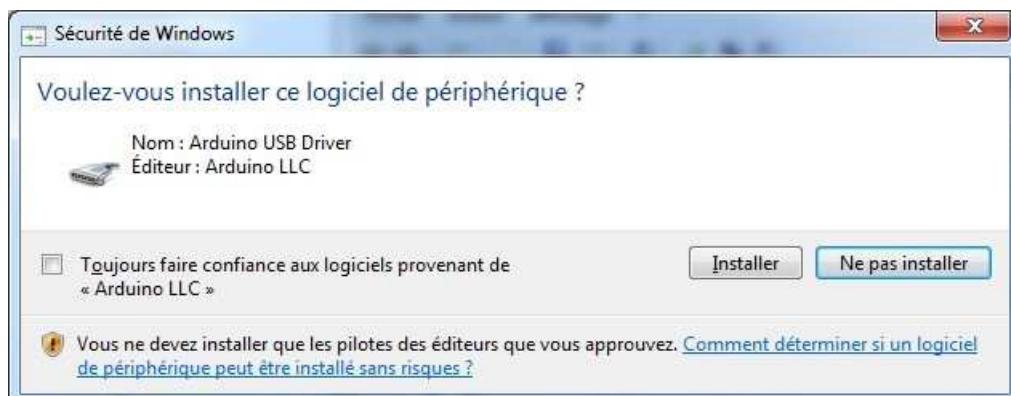


Figure 16 : Confirmer l'installation du pilote

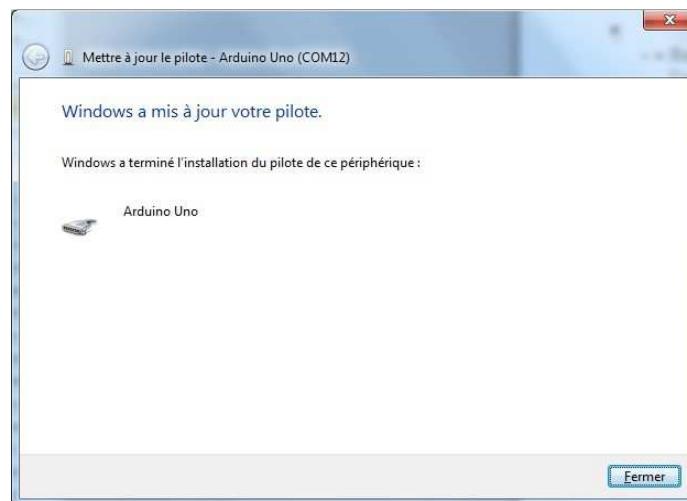


Figure 17 : Fin installation pilote

En retournant alors dans le gestionnaire de périphériques, il est possible de retrouver le port COM ouvert (ici le COM12) (cf. Figure 18).



Figure 18 : Identification port COM ouvert pour la carte Arduino

A partir de ce moment, la partie matérielle est configurée et prête à fonctionner.

### 3.3. Structure de base d'un programme Arduino

Il s'agit de mettre en place les premiers éléments de syntaxe du langage utilisé ainsi que de décrire l'organisation de base d'un programme Arduino.

**Remarque :** la syntaxe du langage utilisé est celle du langage C. Toutefois, l'environnement de programmation peut présenter certaines limites par rapport à la définition du langage C selon la norme C-ANSI. Ces limitations sont essentiellement dues à la plateforme Arduino et au type de microcontrôleur présent sur la carte.

#### 3.3.1. Programme « Hello » - Version 1

Dans la fenêtre d'édition, recopier le programme ci-dessous,

```

1   void setup()
2
3   {
4       Serial.begin(9600);
5       Serial.print("Hello!");
6   }
7
8   void loop()
9   {
10      //Portion de code exécuté indéfiniment
11
12 }
```

puis le sauvegarder sous le nom « Hello\_V1 ». (cf. Figure 19), Tous les sketch Arduino possèdent l'extension `.ino` depuis la version 1.0.

Ce programme permet d'afficher une fois la chaîne de caractères « Hello! » dans la fenêtre du moniteur série.

```

Hello_V1 | Arduino 1.0.5
Fichier Édition Croquis Outils Aide
Hello_V1
void setup()
{
    Serial.begin(9600);
    Serial.print("Hello!");
}

void loop()
{
    //Portion de code répétée indéfiniment
}

```

Enregistrement terminé.

11 Arduino Uno on COM4

**Figure 19 : Programme Hello\_V1**

Pour vérifier la syntaxe du code, cliquer sur l’icône « Vérifier » (cf. Figure 6). Si aucune erreur n’est détectée, alors il est possible de vérifier dans la fenêtre d’affichage des messages que le programme est correct du point de vue de la syntaxe uniquement (cf. Figure 20). **Le volume du programme qui sera téléchargé est également fourni par la taille du croquis exprimé en octets.**

```

Hello_V1 | Arduino 1.0.5
Fichier Édition Croquis Outils Aide
Hello_V1 §
void setup()
{
    Serial.begin(9600);
    Serial.print("Hello!");
}

void loop()
{
    //Portion de code exécuté indéfiniment
}


```

Compilation terminée

Taille binaire du croquis : 1 834 octets (d'un max de 32 256 octets)

11 Arduino Uno on COM11

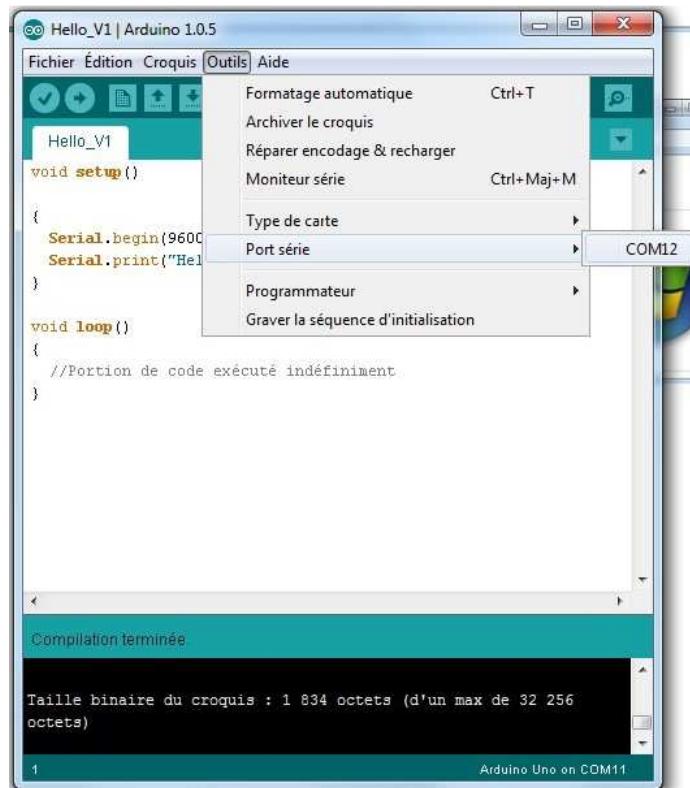
**Figure 20 : Vérification de la syntaxe**

### 3.3.2. Exécution du programme

Pour exécuter le programme, il est nécessaire de le télécharger. Auparavant, il faut configurer l’environnement de programmation en spécifiant :

- Le type de carte Arduino cible au programme ;
- Le port COM de communication.

Cette configuration s'effectue en sélectionnant l'onglet « Outils » puis « Type de carte » et en choisissant « Arduino Uno ». Pour le choix du port de communication, il suffit de sélectionner « Outils » puis « Port série » et dans la liste qui apparaît, le numéro de port COM attribué lorsque la carte Arduino a été connectée sur un des ports USB (cf. Figure 21).



**Figure 21 : Configuration matérielle environnement Arduino**

En cliquant sur le raccourci « Téléverser », cela provoque la compilation du programme ainsi que le téléchargement du code exécutable dans la mémoire de programme du microcontrôleur. Dès que le téléchargement est achevé (cf. Figure 22), le programme s'exécute sur le microcontrôleur. En ouvrant le moniteur série, il est alors possible de vérifier le résultat de cette exécution (cf. Figure 23).

**Remarque :** pour relancer une exécution du programme chargée dans la mémoire du microcontrôleur, il suffit d'appuyer sur le bouton RESET de la carte Arduino.



Figure 22 : Téléversement du programme Hello\_V1

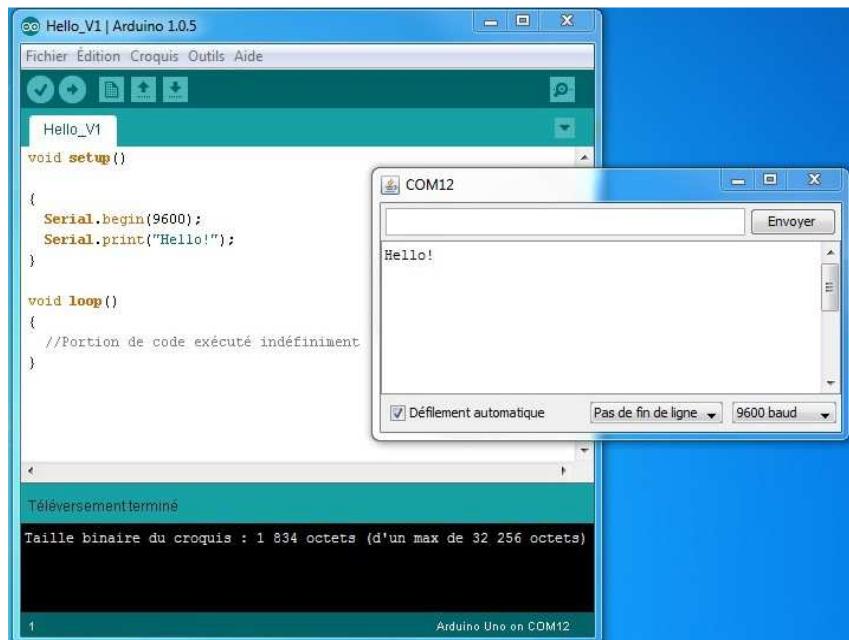


Figure 23 : Exécution de Hello\_V1

### 3.3.3. Structure de base d'un programme Arduino

Un programme Arduino est structuré autour de deux fonctions de base, *setup* et *loop*.

La fonction *setup* est exécutée une seule fois lors du lancement du programme. Elle contient toutes les initialisations de paramètres.

La fonction *loop* est exécutée de façon répétitive. C'est en fait une « boucle infinie ». Elle contient le corps du programme, lorsque cela se justifie.

## Syntaxe : écriture d'une fonction

Toute fonction est constitué d'un en tête et du corps de la fonction

L'en tête permet de définir :

- Le type de valeur renvoyée par la fonction
- L'identificateur de la fonction, c.à.d. son nom
- La liste des paramètres

Sous la forme : type Identificateur (liste des paramètres)

Le corps de la fonction est délimité par les symboles { et }. Il contient les instructions de la fonction.

Dans le programme Hello\_V1, au sein de la fonction *setup* se trouve l'appel aux fonctions *Serial.begin* et *Serial.print* :

- *Serial.begin* (9600) : définir la vitesse de transmission des caractères avec le moniteur série ;
- *Serial.print* ("Hello !") : afficher la chaîne de caractères Hello! sur le moniteur série.

Pour la description de ces deux fonctions, voir aussi : <http://arduino.cc/en/Reference/HomePage>

Il est recommandé de placer des commentaires dans le code source afin de documenter chaque fonction. Tout commentaire est précédé des symboles //. Il est également possible de faire apparaître des commentaires en les délimitant par /\* (marque de début de commentaire) et \*/ (marque de fin de commentaire). Les deux syntaxes sont supportées.

Exemples :

```
// Ceci est un premier commentaire
/* Ceci est un autre commentaire */
```

### 3.3.4. Programme « Hello » - Version 2

Il s'agit de modifier le programme Hello\_V1 afin que la chaîne de caractères Hello! s'affiche en continu. Il suffit pour cela de déporter l'instruction *Serial.print*("Hello") ; au sein de la fonction *loop*.

Afin de pouvoir suivre l'affichage des messages "Hello!" sur le moniteur série, nous introduisons un délai entre chaque affichage avec la fonction *delay*(valeur). Cette fonction *delay* permet de définir une temporisation exprimée en ms (voir aussi <http://arduino.cc/en/Reference/HomePage>).

Reprendre le programme Hello\_V1 et le modifier pour arriver au code ci-dessous. Enregistrer le nouveau croquis dans le fichier Hello\_V2.

```
1 void setup()
2 {
3     Serial.begin(9600);
4 }
5
6
7
8 void loop()
9 {
10    //Portion de code exécuté indéfiniment
11    Serial.print("Hello!");
12    delay(1000); // delay de 1s ou 1000ms
13 }
14
```

Exécuter le code de ce programme et observer ce qui se passe dans la fenêtre du moniteur série.

### 3.3.5. Programme « Hello » - Version 3

A l'aide de la fonction *print*, il est possible d'effectuer des mises en format de résultats et de sauter des lignes afin de rendre plus lisibles les messages qui s'affichent. Si l'on souhaite sauter une ligne après chaque impression, il faut utiliser la fonction *Serial.println* au lieu de *Serial.print*.

Modifier le code du programme Hello\_V2 afin d'obtenir sur chaque ligne du moniteur série l'affichage "Hello !".

```
1 void setup()
2 {
3     Serial.begin(9600);
4 }
5
6
7
8 void loop()
9 {
10    //Portion de code exécuté indéfiniment
11    Serial.println("Hello!");
12    delay(1000); // delay de 1s ou 1000ms
13 }
14 }
```

Sauver ce nouveau programme dans le croquis Hello\_V3, puis lancer l'exécution du programme ainsi modifié. Observer l'effet de l'exécution dans la fenêtre du moniteur série.

**Remarque :** le respect de la casse (respect minuscules –majuscules) est impératif concernant le nom de fonctions utilisées ainsi que le nom des variables utilisées par la suite. Le non-respect de cette règle conduira simplement à des erreurs lors de la compilation du fichier source.

## 3.4. Manipulation de variables

La plupart des programmes qui seront écrits par la suite nécessitent de pouvoir stocker des valeurs numériques entières ou réelles. Cette manipulation s'appuie sur la notion de variable.

### 3.4.1. Définition d'une variable

Variable : zone de stockage / emplacement mémoire

Identificateur : nom associé à la zone mémoire associée à une variable.

Une variable permet donc de manipuler une donnée, selon un format caractérisé par le type de donnée. Le typage de la variable définit aussi la plage des valeurs qui peuvent être stockées ou manipulées.

### 3.4.2. Typage des variables

Le tableau ci-dessous résume les types de base utilisés au sein de l'environnement de programmation Arduino (cf. Tableau 1). Il est aussi possible de consulter le lien <http://arduino.cc/en/Reference/HomePage> pour accéder à la description des différents types de données.

Type	Identificateur	Taille – Occupation mémoire (octets)	Valeurs
	boolean	1	{0, 1}
Entier	char	1	{-128, ..., 127}
	byte	1	{0, ..., 255}
	int	2	{-32768, ..., 32767}
	word	2	{0, ..., 65535}
	short	2	{-32768, ..., 32767}
	lon	4	{-2147483648, ..., 2147483647}
Réel	float	4	[-3.4028235E+38, 3.4028235E+38]
	double	4	Idem type float pour Arduino
Sans type	void		Uniquement lors de la définition de fonction

Tableau 1 : Type de base sous l'environnement Arduino

Le modificateur de type est un attribut qui permet de préciser la représentation des entiers. Il affecte également le domaine de définition. Les modificateurs utilisables sont :

- *signed*
- *unsigned*

Le modificateur *signed* permet de préciser explicitement que l'on utilise une variable entière dans une représentation signée (valeur pouvant être positive ou négative).

Le modificateur *unsigned* permet de préciser explicitement que l'on utilise une variable entière dans une représentation non signée (valeur positive uniquement). Il peut être utilisé sur les types *char*, *int* et *long*.

#### Remarques :

- Les types de variables ainsi que les plages de valeurs associées ne sont valides ici que pour l'environnement de programmation Arduino.
- Il n'existe pas de type spécifique pour les chaînes de caractères. Une chaîne de caractères sera gérée sous la forme d'un tableau dont les éléments seront de *char*.

#### 3.4.3. Déclaration d'une variable

Avant toute utilisation, une variable doit être déclarée.

#### Syntaxe :

```
type Identificateur 1, Identificateur 2, Identificateur 3,..., Identificateur N ;
```

Exemples :

```
int a,b,c ;
char c = 42 ;
char c1= 'a' ;
```

#### Remarques : Constitution d'un identificateur

- Suite de caractères, majuscules ou minuscules, de chiffres ou \_
- Jamais un mot réservé
- Premier caractère une lettre ou \_, jamais un chiffre

- Taille max d'un identificateur : 31 caractères
- **Distinction minuscules / majuscules !!!!**

### 3.4.4. Portée d'une variable

#### 3.4.4.1. Définition

- Portée d'une variable : définit la portion ou la zone d'un programme où une variable et son contenu sont accessibles ;
- Notion qui s'applique quel que soit le typage de(s) variable(s) ;
- La portée d'une variable débute à la fin de sa déclaration.
- Trois types de variables :
  - o Variables locales
  - o Paramètres de fonction
  - o Variables globales

**Remarque :** dans un programme, c'est la place de la déclaration de la variable qui définit sa portée.

#### 3.4.4.2. Catégories de variables

##### Variable locale

- Toute variable déclarée au sein d'une fonction ;
- Toute variable déclarée au sein d'un bloc d'instructions { }

##### Portée :

- En dehors du bloc où elle est déclarée, une variable locale est inconnue ;

##### Création / destruction :

- A l'exécution d'un bloc d'instructions ;
- Entrée dans le bloc : création des variables ;
- Sortie du bloc : destruction des variables.
- Utilisation de la pile pour stocker le contenu des variables.

##### Propriétés :

- Le contenu n'est pas conservé entre 2 exécutions successives d'un même bloc d'instructions ;
- Aucun effet de bord, pas de modification non explicite du contenu d'une variable locale.

##### Variable globale

- Toute variable déclarée en dehors de tout bloc d'instructions.

##### Portée :

- Visibilité étendue à l'ensemble du code situé après la déclaration.

##### Propriétés :

- Allocation fixe, pérenne durant toute l'exécution du code ;
- Créer automatiquement en début du programme.

### Remarque :

- Possibilité d'effets de bord, et donc de modification intempestive du contenu d'une variable au sein de plusieurs fonctions.

### Remarque : Structuration du code

- Zone de déclaration des variables locales : au début du corps d'une fonction juste après {
- Zone de déclaration des variables globales : avant la fonction *setup()*, en dehors de toute autres fonctions.

### 3.4.5. Exemples de mise en œuvre

#### 3.4.5.1. Exemple 1 : programme Variables\_1

Cet exemple illustre la notion de **variables locales**. Les cinq variables déclarées au sein de la fonction *setup()* ne sont visibles et accessibles uniquement sein de cette fonction.

Recopier le programme ci-dessous. Dans la fenêtre du moniteur série, observer les valeurs affichées des variables.

```
1  /* Programme Variables_1
2  /* Manipulation de variables locales */
3
4  void setup()
5
6  {
7      char c = 'A';
8      int a = 10;
9      float r = 25.2;
10     float s = 25.0 * 2;
11     float t = 25 / 2;
12
13     Serial.begin(9600);
14     Serial.print("Valeur de la variable c : ");
15     Serial.println(c);
16     Serial.print("Valeur de la variable a : ");
17     Serial.println(a);
18     Serial.print("Valeur de la variable r : ");
19     Serial.println(r);
20     Serial.print("Valeur de la variable s : ");
21     Serial.println(s);
22     Serial.print("Valeur de la variable t : ");
23     Serial.println(t);
24 }
25
26 void loop()
27 {
28     //Portion de code exécuté indéfiniment
29 }
30
```

#### 3.4.5.2. Exemple 2 : programme Variables\_2

Cet exemple illustre la notion de **variables globales**. Les variables sont maintenant déclarées en dehors de la fonction *setup()*. Elles deviennent alors visibles et accessibles au sein de toutes les fonctions du programme.

Recopier le programme ci-dessous. Dans la fenêtre du moniteur série, observer les valeurs affichées des variables.

```
1  /* Programme Variables_2 */  
2  /* Manipulation de variables globales */  
3  
4  // Déclarations de variables globales  
5  char c = 'A';  
6  int a = 10;  
7  float r = 25.2;  
8  float s = 25.0 * 2;  
9  float t = 25 / 2;  
10  
11 void setup()  
12 {  
13     Serial.begin(9600);  
14 }  
15  
16 void loop()  
17 {  
18     //Portion de code exécuté indéfiniment  
19     Serial.print("Valeur de la variable c : ");  
20     Serial.println(c);  
21     Serial.print("Valeur de la variable a : ");  
22     Serial.println(a);  
23     Serial.print("Valeur de la variable r : ");  
24     Serial.println(r);  
25     Serial.print("Valeur de la variable s : ");  
26     Serial.println(s);  
27     Serial.print("Valeur de la variable t : ");  
28     Serial.println(t);  
29     delay (1000); // Temporisation de 1000ms  
30 }
```

### 3.4.5.3. Exemple 3 : opérateurs arithmétiques de base

Le langage proposé permet d'utiliser les opérateurs arithmétiques de base +, -, /, \*, % ainsi que l'opérateur = qui est l'opérateur d'affectation du contenu d'une variable.

L'ensemble des opérateurs arithmétiques est disponible en annexe.

A partir du squelette du programme ci-dessous, modifier le code afin d'utiliser les autres opérateurs. Que constatez-vous ?

```

1  /* Programme Operateurs                               */
2  /* Opérateurs arithmétiques                         */
3
4  void setup()
5  {
6      char c = 'A';
7      int d = 10.5;
8      char r1;
9      int r2;
10     float r3;
11
12     r1 = c + d;
13     r2 = c + d;
14     r3 = c + d;
15
16     Serial.begin(9600);
17     Serial.print("Valeur de c + d : ");
18     Serial.println(r1);
19     Serial.print("Valeur de c + d : ");
20     Serial.println(r2);
21     Serial.print("Valeur de c + d : ");
22     Serial.println(r3);
23 }
24
25 void loop()
26 {
27     //Portion de code exécuté indéfiniment
28 }
```

#### 3.4.5.4. Exemple 4 : fonctions mathématiques

Le programme ci-dessous permet de calculer le périmètre d'un cercle ainsi que la surface d'un disque de même rayon. Il permet d'illustrer l'utilisation de quelques fonctions mathématiques disponibles dans les bibliothèques de l'environnement de programmation Arduino (voir aussi <http://arduino.cc/en/Reference/HomePage>).

```

1  /* Programme Fonctions_Math
2  /* Calcul du périmètre et de la surface d'un disque */
3
4  // Déclaration de constante symbolique
5 #define PI 3.14159
6
7  // Déclarations de variables globales
8 float rayon, perimetre, surface;
9
10 void setup()
11 {
12     Serial.begin(9600);
13
14     rayon = 0.25;
15     perimetre = 2.0 * PI * rayon;
16     surface = PI * pow (rayon,2);
17
18     Serial.print("Valeur du rayon : ");
19     Serial.println(rayon);
20     Serial.print("Perimetre = ");
21     Serial.println(perimetre);
22     Serial.print("Surface = ");
23     Serial.println(surface);
24 }
25
26 void loop()
27 {
28     //Portion de code exécutée indéfiniment
29 }
```

### 3.5. Structures de décision

Il est souvent nécessaire de pouvoir décider d'une action à accomplir selon qu'une condition est vérifiée ou non. La prise en compte de ce type de comportement s'appuie sur la structure de sélection de type *si ... alors...sinon*.

Convention :

- une assertion logique est considérée fausse si la valeur retournée est égale à 0
- une assertion logique est considérée vraie si la valeur retournée est différente de 0

Mots réservés associés : if else

Structure algorithmique :

**if** (condition) expression ;

**if** (condition) expression 1;  
**else** expression 2;

**if** (condition)  
{  
 expression1;  
 expression2;  
}  
**else** expression3;

Il est également possible d'imbriquer des structures de sélection **if ... else**.

Ces différentes syntaxes sont illustrées par les exemples de programmes ci-dessous.

```
1  /* Structure de sélection si ... alors */  
2  /* Programme Selection_1 */  
3  void setup()  
4  {  
5      Serial.begin(9600);  
6  
7      int a = 89;  
8      int b = 42;  
9  
10     if(a > b)  
11     {  
12         Serial.print("a est plus grand que b");  
13     }  
14     else  
15     {  
16         Serial.print("a est plus petit que b");  
17     }  
18 }  
19  
20 void loop()  
21 {  
22     // //Portion de code exécutée indéfiniment  
23 }
```

```
1  /* Structure de sélection si ... alors */  
2  /* Programme Selection_2 */  
3  
4  void setup()  
5  {  
6      Serial.begin(9600);  
7  
8      int a = 89;  
9      int b = 42;  
10  
11     if(a > b) Serial.print("a est plus grand que b");  
12     else if(a < b) Serial.print("b est plus grand que a");  
13     else Serial.print("a est égal à b");  
14 }  
15  
16 void loop()  
17 {  
18     // //Portion de code exécutée indéfiniment  
19 }
```

Il est également possible de construire des conditions multiples afin de prendre en compte plusieurs conditions simultanément. Pour cela, il est possible d'utiliser l'opérateur logique `&&` (ET) ainsi que l'opérateur `||` (OU). Ainsi, pour tester si une valeur est dans un intervalle de valeurs valides, nous pouvons construire l'expression :

If (`a >= seuil1`) `&&` (`a <= seuil2`) afin de vérifier que la valeur de `a` est bien dans l'intervalle `[seuil1 , seuil2]`.

Les différents opérateurs de comparaison ainsi que les opérateurs logiques disponibles sont fournis en annexe.

### 3.6. Structures itératives

Ces structures permettent de répéter une action plusieurs fois. Le nombre d’itérations à effectuer peut être connu au préalable ou non. L’action peut aussi être répétée tant qu’une condition est vraie. Nous pourrons utiliser par la suite les structures itératives suivantes :

- la boucle **for**
- La boucle **while**
- La boucle **do while**

### 3.6.1. Boucle for

Exemple : afficher 5 fois le message « Action effectuée à l’itération x », x variant de 0 à 4

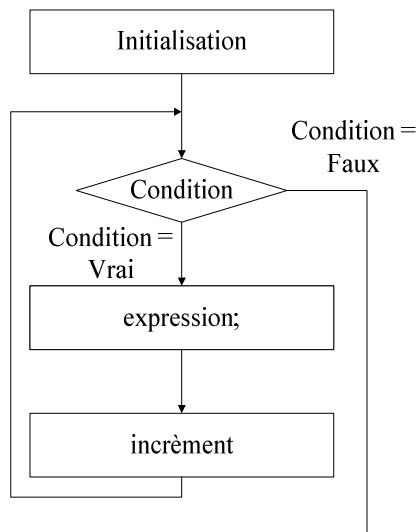
```
for (i = 0 ; i < 5 ; i++)
{
    Serial.print ("Action effectuée à l'itération ");
    Serial.println(i);
}
```

Syntaxe de la boucle **for** :

**for** (initialisation ; condition ; increment) expression;

- *initialisation* :
  - o expression permettant d’initialiser le compteur de la boucle ;
  - o peut aussi être une expression basée sur l’opérateur séquentiel ,
- *condition* : expression permettant de contrôler l’exécution de la prochaine itération et / ou l’évaluation de l’instruction
  - o si condition == Faux, alors sortir de la boucle
  - o si condition == Vrai, alors évaluer expression
- *incrément* :
  - o expression permettant de faire évoluer la valeur du compteur de la boucle ;
  - o peut aussi être une expression basée sur l’opérateur séquentiel ,

Comportement de la boucle **for** :



Le programme ci-dessous illustre l'utilisation d'une boucle for.

```
1  /* Programme Boucle_for                               */
2  /* Illustration syntaxe boucle for                   */
3
4  void setup()
5  {
6      int i;
7
8      Serial.begin(9600);
9
10     for (i = 0 ; i < 5 ; i++)
11     {
12         Serial.print ("Action effectuée - iteration ");
13         Serial.println(i) ;
14     }
15 }
16
17 void loop()
18 {
19     //Portion de code exécuté indéfiniment
20 }
```

La sortie obtenue sur la fenêtre du moniteur série est présentée ci-dessous (cf. Figure 24).

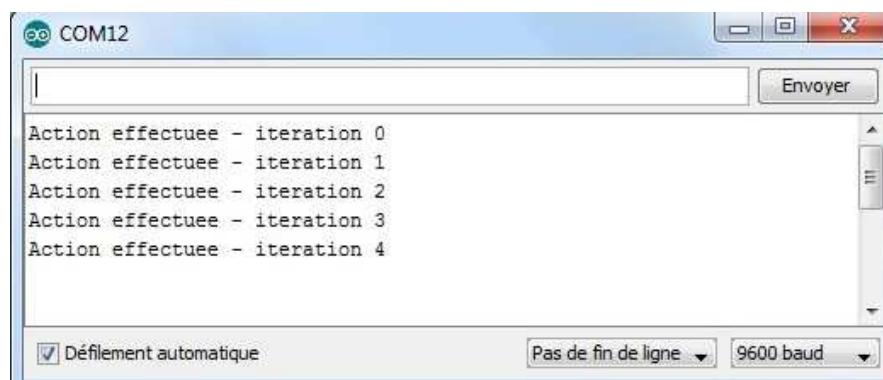


Figure 24 : Sortie moniteur série itérations boucle for

### 3.6.2. Boucle while

La boucle while est une variante de la boucle for. L'initialisation est effectuée alors avant de rentrer dans la structure de la boucle, et la gestion du compteur de boucle est réalisée au sein du corps de la boucle. Du point de vue algorithmique, ces deux structures itératives sont identiques. Si nous reprenons l'exemple précédent, il sera alors traité comme suit :

```
i = 0 ;
while (i < 5)
{
    Serial.print ("Action effectuée à l'itération ");
    Serial.println(i) ;
    i++
}
```

Syntaxe de la boucle **while** :

```
while (condition) expression;
```

```
while (condition)
{
    Bloc d'instructions
}
```

Comportement de la boucle **while** :

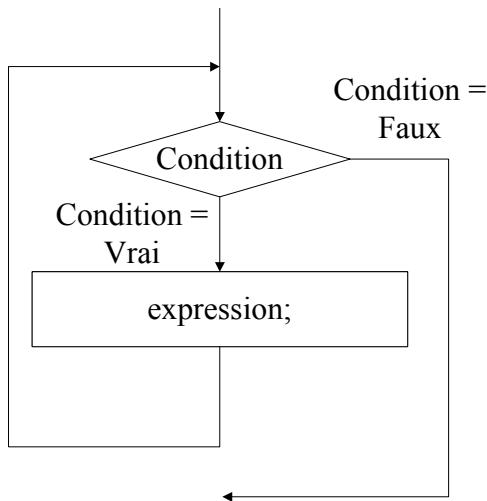


Illustration sous Arduino :

```
1  /* Programme Boucle_while
2  /* Illustration syntaxe boucle while
3
4  void setup()
5  {
6      int i;
7
8      Serial.begin(9600);
9
10     i = 0;
11     while (i < 5)
12     {
13         Serial.print ("Action effectuée - iteration ");
14         Serial.println(i);
15         i++;
16     }
17 }
18
19 void loop()
20 {
21     //Portion de code exécuté indéfiniment
22 }
```

### 3.6.3. Boucle do while

Contrairement aux deux structures itératives précédentes, la boucle **do while** procure l'assurance qu'au moins une itération sera effectuée.

L'exemple précédent peut être implémenté comme suit :

```
i = 0 ;  
do  
{  
    Serial.print ("Action effectuée à l'itération ");  
    Serial.println(i);  
    i++  
} while (i < 5) ;
```

Syntaxe de la boucle **do while**

```
do  
{  
    expression;  
}  
while (condition);
```

Comportement de la boucle **do while**

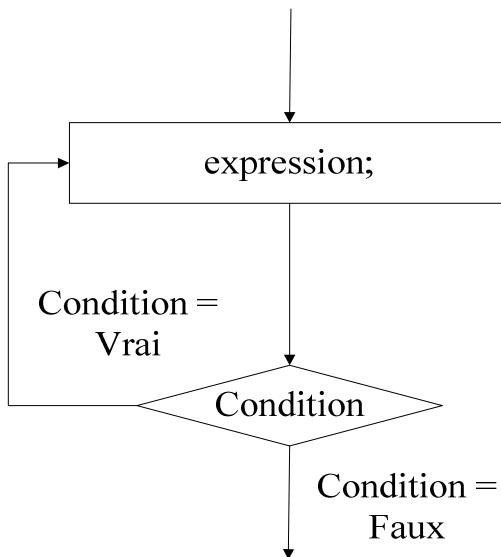


Illustration sous environnement Arduino :

```

1  /* Programme Boucle_dowhile
2  /* Illustration syntaxe boucle do while
3
4  void setup()
5  {
6      int i;
7
8      Serial.begin(9600);
9
10     i = 0;
11     do
12     {
13         Serial.print ("Action effectuée - iteration ");
14         Serial.println(i);
15         i++;
16     }while (i < 5);
17 }
18
19 void loop()
20 {
21     //Portion de code exécuté indéfiniment
22 }
```

## 3.7. Constantes - Commentaires

### 3.7.1. Constantes

- **Les constantes littérales**
  - o Valeur numériques directement introduites dans le code ;
  - o Exemples :
    - 100 est une constante entière ;
    - 100. ou 100.0 sont des constantes réelles ;
  - o Possibilité de définir des constantes littérales entières en binaire
    - B00000111 correspondant à la valeur 7 en décimal

Il est possible de forcer le type utilisé pour représenter une valeur constante. Ainsi, il est possible de choisir une représentation non signée ou bien de forcer la constante dans le type long. Ceci s'effectue en utilisant les suffixes dont la liste est donnée ci-dessous :

Type	Format numérique	Suffixe	Exemple
int	entier	aucun	31415
unsigned int	entier	U	31415U
long int	entier	L	31415L
unsigned long int	entier	UL	31415UL

Exemples :

45U  
2337LU  
25485L

- **Les constantes symboliques**

- o Représentée par un nom
- o Syntaxe :

#define NOM valeur
--------------------

- o Par convention, NOM est en majuscules

- Par convention, la définition de constantes symbolique s'effectue en début de fichier source.
- Exemple : **#define NBLIGNE 10**

- **Les variables constantes**

- Utilisation du modificateur **const** ;
- Ce modificateur est appliqué lors de la déclaration de la variable :  
const int seuil = 128 ;

### 3.7.2. Commentaires

Les commentaires permettent de documenter le code des programmes réalisés. A ce titre, ils constituent une source documentaire, et un moyen de structurer le code écrit. Deux formes de commentaires sont utilisables :

- Forme 1 : basée sur l'encadrement du commentaire par /\* pour marquer le début et \*/ pour marquer la fin du commentaire.

```
/* Ceci est le début d'un commentaire
Ici la suite du commentaire
Et la fin du commentaire */
```

- Forme 2

Il est également possible de placer des commentaires, ponctuellement sur une ligne de code, et uniquement sur cette ligne. Dans ce cas, on utilise uniquement la marque // pour indiquer que le texte qui suit est un commentaire.

```
// Ceci est un commentaire associé à une ligne
```

Vous trouverez des illustrations de l'utilisation de ces différentes de commentaires dans les exemples de programmes fournis ci-dessus.

## 3.8. Structures générales d'un programmes Arduino

Cette partie a pour objet de vous amener à structurer vos différents programmes Arduino, afin de les rendre davantage lisibles, compréhensibles, maintenables voire modifiables plus facilement. La structure générale d'un programme Arduino suit celle décrite par la figure ci-dessous (cf. Figure 25).

### Les directives du préprocesseur

- Instructions particulières, prises en compte par le préprocesseur ;
- Sont caractérisées par le caractère #
- Deux directives importantes :
  - Directives d'inclusion de fichiers : **#include**
  - Directives de définition de symboles **#define**

### Définition de types de données

- Déclaration de nouveaux types de données composés ;
- Déclaration de nouveaux noms de type de données.

## Prototypes de fonctions

- Toute fonction doit être déclarée ou définie avant de pouvoir être utilisée ;
- Introduction de mécanisme de vérification sur le typage des arguments de fonctions.

## Déclarations de variables globales

- Les variables globales du fichier source sont déclarées ;

## Code des fonctions

- Une fonction ne contient que des instructions et des variables locales ;

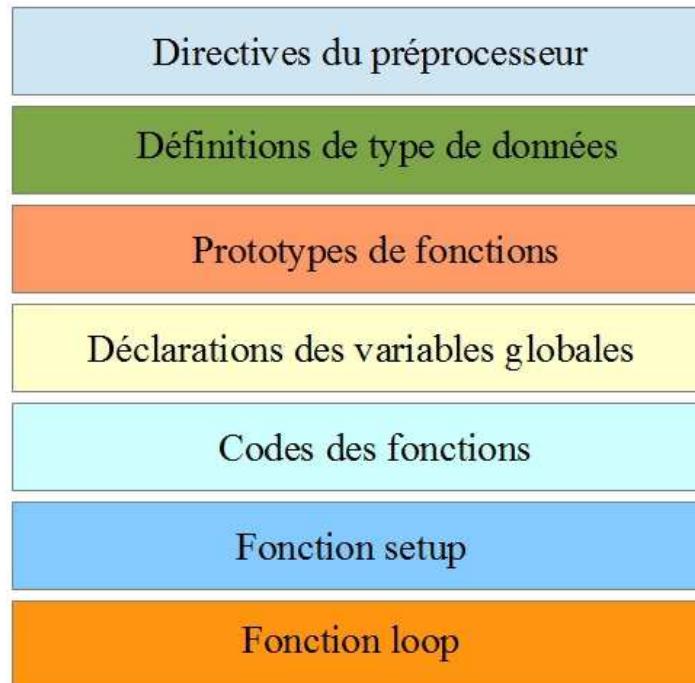


Figure 25 : Structure générale d'un programme Arduino

## 4. Montage du shield Boe et de l'arduino

### 4.1. Utilisation de Led

#### 4.1.1. Commander la Led de la carte Arduino

La carte Arduino possède une Led déjà câblée, et reliée à la broche d'E/S numérique 13. Nous allons dans cette partie voir comment commander cette Led.

Nous souhaitons allumer puis éteindre la Led, selon le diagramme temporel ci-dessous (cf. Figure 26). Pendant la durée  $t_1$ , la Led sera allumée, tandis que sur la durée  $t_2$ , elle sera éteinte.

Afin de réaliser la commande de la Led, il faut :

- Préciser que la Led est connectée sur la broche 13 ;
- Indiquer que la broche 13 est en sortie ;
- Pouvoir modifier l'état de la broche et la faire passer à l'état haut (1) ou à l'état bas (0) ;
- Gérer les durées  $t_1$  et  $t_2$ .

L'environnement de programmation et les bibliothèques de fonctions fournies vont permettre de réaliser ces différentes opérations.

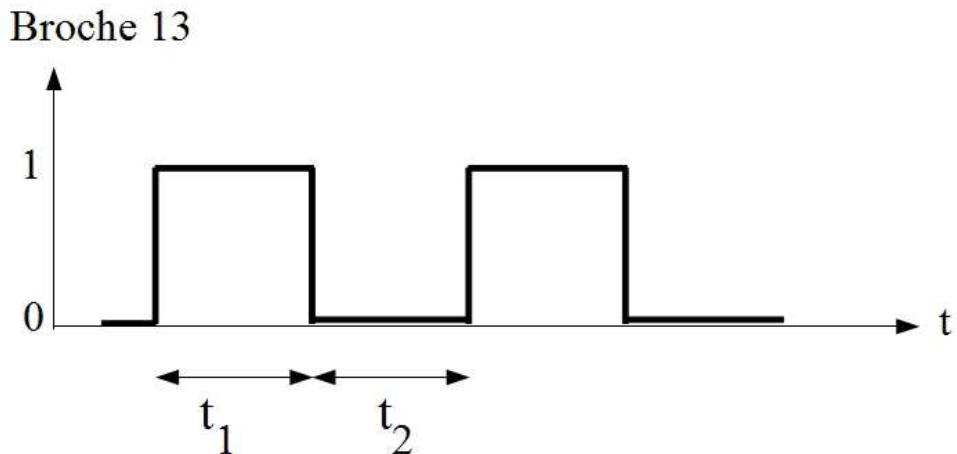


Figure 26 : Signal de commande de la Led

Pour associer la Led à la broche 13, et en même temps, préciser que cette broche est utilisée en sortie, il faut utiliser la fonction *pinMode* (pin, mode) où :

- pin désigne le numéro de la broche utilisée ;
- mode permet de définir si la broche est en entrée (lecture de son état) ou en sortie (écriture de l'état). Le paramètre mode peut être défini à l'aide des constantes INPUT ou OUTPUT.

Pour modifier l'état d'une broche définie en sortie numérique, il faut utiliser la fonction *digitalWrite* (pin, value) où :

- pin désigne le numéro de la broche utilisée ;
- value permet de forcer le niveau de la broche soit à l'état bas, soit à l'état haut. Deux constantes symboliques peuvent être utilisées LOW et HIGH.

La gestion des durées  $t_1$  et  $t_2$  peut être effectuée de diverses façons explicitées ci-dessous.

#### 4.1.2. Faire clignoter la Led – Version 1

La première façon de gérer les durées  $t_1$  et  $t_2$  est d'utiliser une structure itérative, au sein de laquelle aucune action est effectuée. Ce sera donc le nombre d'itérations effectuées qui fixera la durée de chaque période.

Dans cette première version, le cycle de durée  $t_1$  et  $t_2$  sera effectué seulement 10 fois.

Une implémentation est proposée ci-dessous. Afin de rendre le code lisible et facilement interprétable, 3 constantes symboliques sont définies :

- LED\_pin pour indiquer que la Led est branchée sur la broche 13
- TEMPO\_High pour définir la durée  $t_1$
- TEMPO\_Low pour définir la durée  $t_2$

Pour finir, seule la fonction *setup()* est utilisée.

Essayer de trouver les valeurs des deux constantes symboliques TEMPO\_High et TEMPO\_Low afin d'avoir des durées de  $t_1$  et  $t_2$  respectivement de l'ordre de 1s et 1.5s

```

1  /* Programme Led_V1
2  /* Faire clignoter la Led de la carte Arduino */
3  /* Temporisations gérées par boucles for */
4
5  #define LED_pin 13 // Led carte Arduino sur broche 13
6  #define TEMPO_High 500000L // Durée état haut - LED allumée
7  #define TEMPO_Low 750000L // Durée état bas - LED éteinte
8
9  int i;
10 long int tempo;
11
12 void setup ()
13 {
14     Serial.begin(9600);
15     pinMode(LED_pin, OUTPUT); // Broche 13 en sortie
16     digitalWrite(LED_pin,LOW); // Broche 13 état bas et LED éteinte
17
18     for (i = 0; i < 10; i++)
19     {
20         Serial.print("\t : "); Serial.println (i);
21         digitalWrite(LED_pin,HIGH); // Broche 13 état haut et LED allumée
22         Serial.println("LED etat haut");
23         for (tempo = 0; tempo < TEMPO_High; tempo++) ; // Temporisation état haut
24         digitalWrite(LED_pin,LOW); // Broche 13 état bas et LED éteinte
25         Serial.println("LED etat bas");
26         for (tempo = 0; tempo < TEMPO_Low; tempo++) ; // Temporisation état bas
27     }
28 }
29
30 void loop ()
31 {
32     // Ne rien faire pour l'instant
33 }
```

#### 4.1.3. Faire clignoter la Led – Version 2

Dans cette seconde version du programme précédent, nous allons maintenant gérer plus finement les durées  $t_1$  et  $t_2$ . Pour cela, nous disposons de la fonction *delay* qui admet en paramètre une durée exprimée en ms ( $10^{-3}$  s). On peut se référer à <http://arduino.cc/en/Reference/HomePage> pour le descriptif de cette fonction.

Il suffit alors de modifier le programme Led\_V1 en apportant les corrections suivantes :

- Redéfinir la valeur des constantes symboliques permettant de fixer les durées de  $t_1$  et  $t_2$  pour avoir respectivement 500 ms et 750 ms.
- Remplacer les boucles *for* par l'appel à la fonction *delay*.

A partir du programme Led\_V1, appliquer ces modifications afin d'obtenir le programme Led\_V2. Vérifier que l'exécution du programme est celle attendue.

```

1  /* Programme Led_V2 */  

2  /* Faire clignoter la Led de la carte Arduino */  

3  /* Temporisations gérées avec la fonction delay */  

4  

5  #define LED_pin 13 // Led carte Arduino sur broche 13  

6  #define TEMPO_High 500 // Durée état haut ms - LED allumée  

7  #define TEMPO_Low 750 // Durée état bas ms - LED éteinte  

8  

9  int i;  

10 long int tempo;  

11  

12 void setup ()  

13 {  

14     Serial.begin(9600);  

15     pinMode(LED_pin,OUTPUT); // Broche 13 en sortie  

16     digitalWrite(LED_pin,LOW); // Broche 13 état bas et LED éteinte  

17  

18     for (i = 0; i < 10; i++)  

19     {  

20         Serial.print("\t : "); Serial.println (i);  

21         digitalWrite(LED_pin,HIGH); // Broche 13 état haut et LED allumée  

22         Serial.println("LED etat haut");  

23         delay (TEMPO_High); // Temporisation état haut  

24         digitalWrite(LED_pin,LOW); // Broche 13 état bas et LED éteinte  

25         Serial.println("LED etat bas");  

26         delay (TEMPO_Low); // Temporisation état bas  

27     }  

28 }  

29  

30 void loop ()  

31 {  

32     // Ne rien faire pour l'instant  

33 }

```

#### 4.1.4. Faire clignoter la Led – Version 3

Il s'agit maintenant de faire clignoter la Led selon le cycle défini plus haut (cf. Figure 26). Pour cela, il faut maintenant placer le code permettant d'allumer puis d'éteindre la Led au sein de la fonction *loop*, qui est une structure itérative qui s'exécute indéfiniment. Les modifications apportées conduisent alors au programme Led\_V3.

Effectuer ces modifications et vérifier que le cycle de la Led s'effectue bien de façon continue, y compris sur le moniteur série.

```

1  /* Programme Led_V3                      */
2  /* Faire clignoter la Led de la carte Arduino   */
3  /* Temporisations gérées avec la fonction delay  */
4  /* Répéter le cycle de commande de la Led        */
5
6  #define LED_pin 13 // Led carte Arduino sur broche 13
7  #define TEMPO_High 500 // Durée état haut ms - LED allumée
8  #define TEMPO_Low 750 // Durée état bas ms - LED éteinte
9
10 long int tempo;
11
12 void setup ()
13 {
14     Serial.begin(9600);
15     pinMode(LED_pin,OUTPUT); // Broche 13 en sortie
16     digitalWrite(LED_pin,LOW); // Broche 13 état bas et LED éteinte
17 }
18
19 void loop ()
20 {
21     digitalWrite(LED_pin,HIGH); // Broche 13 état haut et LED allumée
22     Serial.println("LED etat haut");
23     delay (TEMPO_High); // Temporisation état haut
24     digitalWrite(LED_pin,LOW); // Broche 13 état bas et LED éteinte
25     Serial.println("LED etat bas");
26     delay (TEMPO_Low); // Temporisation état bas
27 }

```

## 4.2. Assemblage shield Boe et Arduino

Nous abordons dans cette partie l'assemblage du robot et la gestion des premiers éléments périphériques.

### 4.2.1. Montage

Nous procédons ici à l'assemblage de la carte Arduino et du shield Board of Education. Le matériel nécessaire est constitué par (cf. Figure 27) :

- La carte Arduino Rev3
- Le shield Boe
- 4 entretoises en aluminium de 2.5 cm de hauteur
- 4 vis tête bombée de 7 mm



Figure 27 : Eléments d'assemblage carte Arduino et shield Boe

La première étape consiste à assembler la carte Arduino et le shiel Boe. La carte Arduino se fixe sous le shield Boe, en prenant soin d'aligner les connecteurs des deux cartes. Il faut prendre comme points de repères les broches Rx et Analog In – A5 (cf. Figure 28).

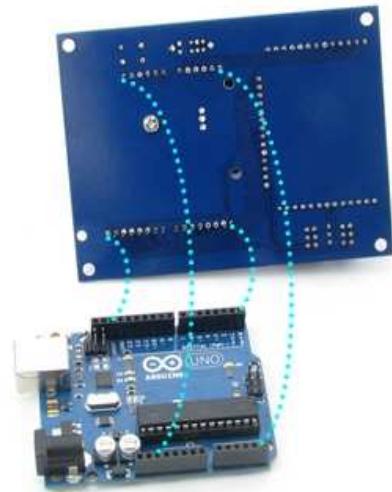


Figure 28 : Alignement carte Arduino et Shield Boe

Sans tordre les pattes des connecteurs, « clipser » les deux cartes pour arriver au résultat de la figure ci-dessous (cf. Figure 29).



Figure 29 : Assemblage final carte Arduino et Boe

L'étape suivante consiste à placer les 4 entretoises et à les fixer sur la Boe (cf. Figure 30).

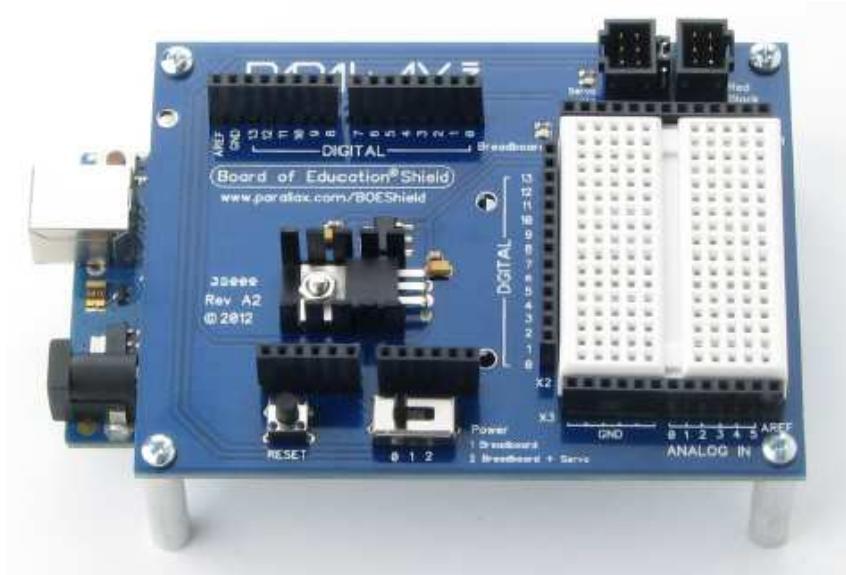


Figure 30 : Carte Arduino et Boe et entretoises

#### 4.2.2. Plaque de prototypage

Parmi les éléments de la Boe se trouve la plaque de prototypage, qui permet d'effectuer la réalisation de petits montages électroniques (cf. Figure 31). Elle est entourée de trois connecteurs qui permettent d'accéder aux ports d'E/S du microcontrôleur ainsi qu'aux différentes tensions d'alimentation.

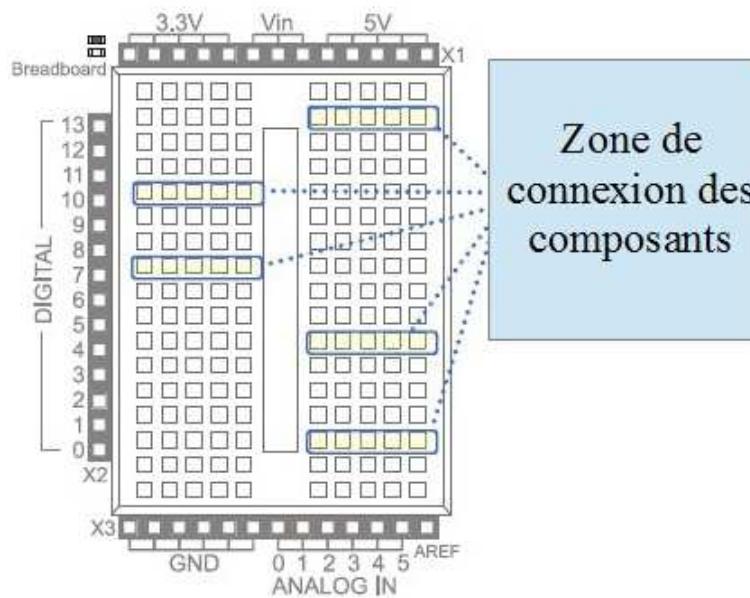


Figure 31 : Plaque de prototypage

#### 4.2.3. Piloter une Led

Il s'agit de reprendre la version précédente du programme Led\_V3 dans lequel la Led était commandée, mais cette fois, en utilisant les ressources de la Boe. Au lieu d'utiliser la Led déjà

présente sur la carte Arduino, nous allons voir comment câbler le circuit électronique de la Led sur la plaque de prototypage. Pour cela il est nécessaire de savoir comment reconnaître une résistance et quelles sont les caractéristiques d'une Led.

#### 4.2.3.1. Résistance et code couleur

Une résistance est un composant qui permet de contrôler le courant au sein d'un composant ou d'un circuit. Il est caractérisé par la valeur de sa résistance exprimée en Ohm (symbole :  $\Omega$ ). Dans un schéma, la représentation d'une résistance s'appuie sur le symbole ci-dessous (cf. Figure 32), accompagné de la valeur de la résistance.

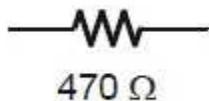


Figure 32 : Symbole de représentation d'une résistance

Une résistance se présente comme ci-dessous (cf. Figure 33).



Figure 33 : Exemples de résistances

Avec un code couleur qui permet de connaître la valeur de la résistance. Les deux premiers anneaux donnent les chiffres significatifs (le premier donne la dizaine et le second l'unité). Le troisième donne le multiplicateur (la puissance de 10 par laquelle faut multiplier les chiffres significatifs). Le quatrième la tolérance (les incertitudes sur la valeur réelle de la résistance donnée par le constructeur) (cf. Figure 34).

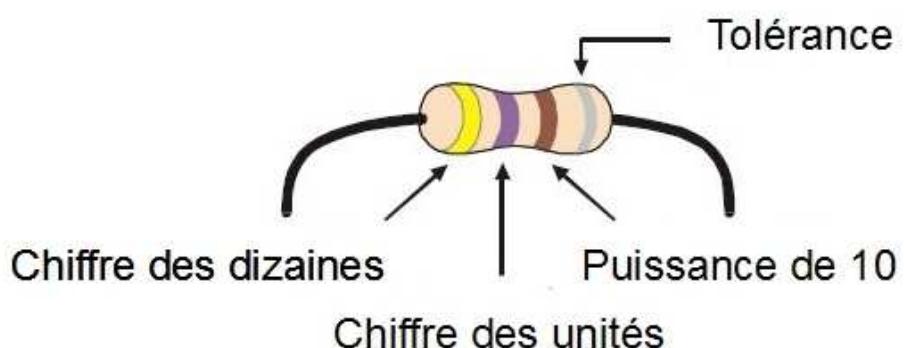


Figure 34 : Codage de la valeur résistance

Le tableau ci-dessous résume la valeur attribuée à chaque couleur (cf. Tableau 2).

	1° anneau gauche	2° anneau gauche	Dernier anneau gauche	Anneau droite
	1er chiffre	2e chiffre	Multiplicateur	Tolérance
Noir	0	0	1	
Marron	1	1	10	1%
Rouge	2	2	102	2%
Orange	3	3	103	
Jaune	4	4	104	
Vert	5	5	105	0.5%
Bleu	6	6	106	0.25%
Violet	7	7	107	0.1%
Gris	8	8	108	0.05%
Blanc	9	9	109	
or			0.1	5%
argent			0.01	10%

Tableau 2 : Tableau de codage des valeurs de résistance

Ainsi, les couleurs Jaune – Violet – Marron représente une résistance d'une valeur de  $470\Omega$ .

#### 4.2.3.2. Led : repérer les broches

Une LED (Light-Emitting Diode) ou diode électro luminescente (DEL) est une diode qui émet de la lumière lorsqu'un courant la parcourt. C'est un composant qui ne laisse passer le courant que dans un seul sens. Une led possède deux broches, l'anode et la cathode. Elle est symbolisée par le schéma ci-dessous (cf. Figure 35).

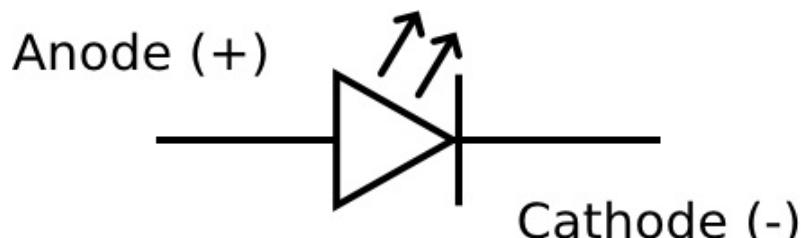


Figure 35 : Schéma de représentation d'une Led

Le courant circule de l'anode vers la cathode.

Lorsque l'on manipule le composant, en général, l'anode correspond à la broche la plus longue et la cathode à la plus courte. Si les 2 pattes sont de la même longueur, alors il faut se repérer par rapport à un méplat présent sur la partie plastique du composant (cf. Figure 36).

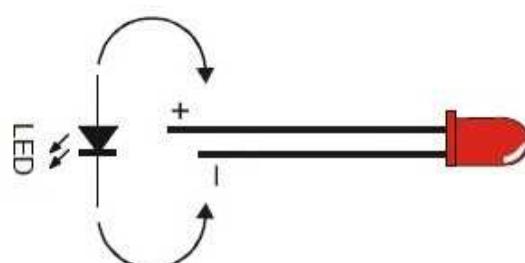


Figure 36 : Identifier les broches d'une Led

Nous allons maintenant mettre en place sur la plaque de prototypage le circuit de la Led. Ce circuit pourra être utilisé par la suite pour signaler des événements particuliers par la suite.

Avant, il est impératif que l'interrupteur soit sur la position 0.

#### 4.2.3.3. Montage

Le circuit à réaliser est représenté par le schéma ci-dessous (cf. Figure 37).

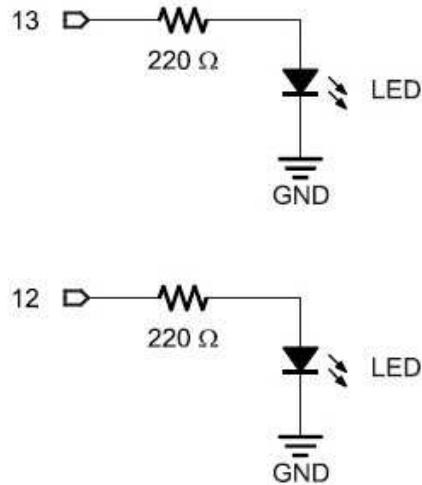


Figure 37 : Schéma de câblage de deux Leds sur la carte Boe

Chacune des Led sera commandée respectivement depuis les ports 12 et 13. Le courant qui les traverse sera limité par une résistance de  $220\Omega$ .

La réalisation du circuit sur la plaque de prototypage s'effectuera comme sur la figure ci-dessous (cf. Figure 38).

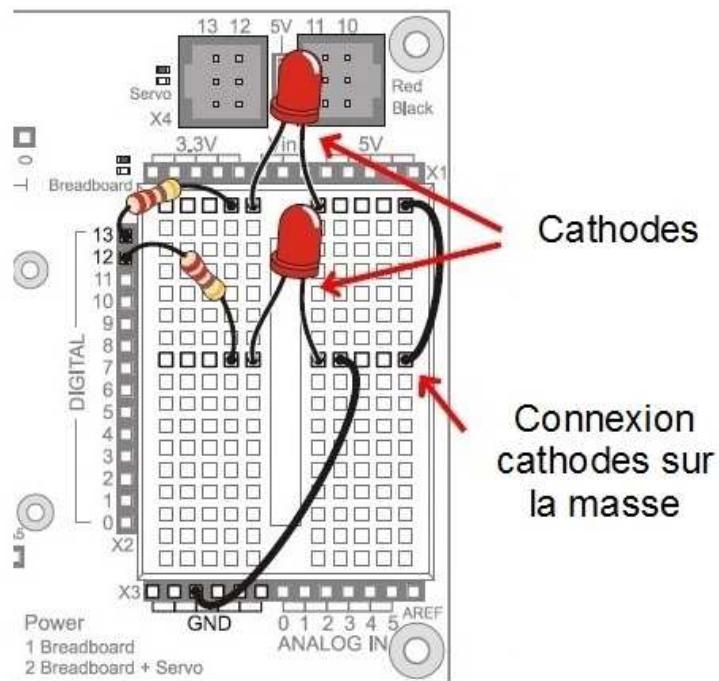


Figure 38 : Câblage de deux Leds sur platine de prototypage

**Remarque :** Par convention, les fils reliés à la masse (GND) seront choisis de couleur noire

#### 4.2.3.4. Reprise des programmes Led

Après avoir terminé le câblage des deux Leds, reprendre le programme Led\_V3 et le télécharger. Vérifier que la Led connectée sur la broche 13 clignote selon le même cycle que précédemment.

Modifier maintenant le programme Led\_V3 pour effectuer le même cycle d'allumage / extinction sur la Led connectée sur la broche 12. Sauvegarder votre programme sous Led\_V4.

On souhaite maintenant commander les deux Leds simultanément selon le cycle défini par le diagramme ci-dessous (cf. Figure 39).

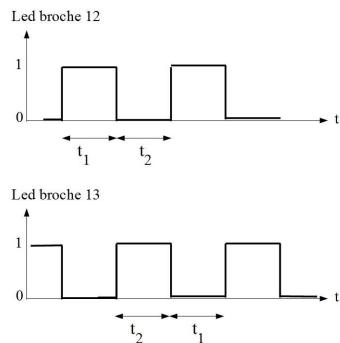


Figure 39 : Signal de commande des deux Leds

Ecrire le programme Led\_V5 permettant de piloter chacune des deux Leds avec les valeurs de  $t_1$  et  $t_2$  telles que  $t_1 = t_2 = 500$  ms. Vérifier que vous obtenez le résultat attendu.

#### 4.2.4. Vers la commande des servo moteurs

Nous allons maintenant considérer le signal de commande uniquement de la Led connectée sur la broche 13, ayant la forme définie par le diagramme ci-dessous (cf. Figure 40).

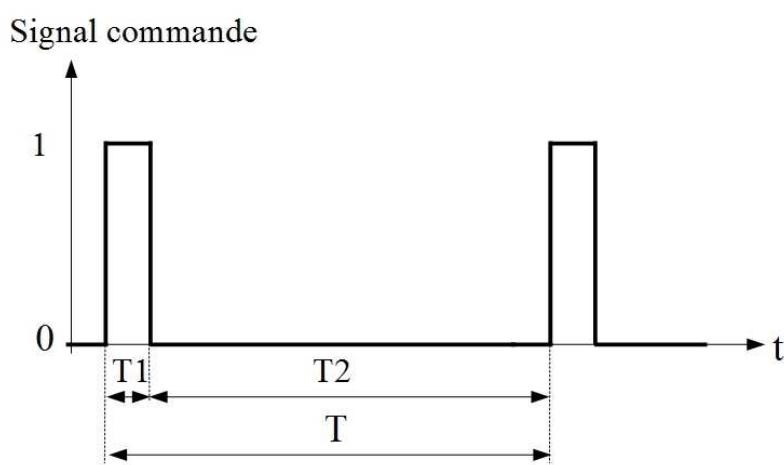


Figure 40 : Signal de commande Led broche 13 – version 2

La durée  $T$  désigne la période du signal de commande et  $T_1$  correspond à la durée du signal à l'état haut. On souhaite maintenant reprendre le programme Led\_V3 en prenant en compte les couples de valeurs ( $T_1$ ,  $T_2$ ) définis dans le tableau ci-dessous, en ms

	T	T1	T2 = T - T1
Cas 1	2000	150	1850
Cas 2	200	15	185
Cas 3	20	2	18

Reprendre alors le programme Led\_V5 pour les trois cas avec les valeurs de T1 et T2 correspondant. Qu'observe ton sur le cas 2 et 3 ?

En fait, la forme du signal décrite ci-dessus (cf. Figure 40) correspond à celle nécessaire pour commander un servo moteur. Afin de pouvoir contrôler finement la vitesse de rotation, il est nécessaire de pouvoir définir de manière plus précise les durées T1 et T2, tout en maintenant la somme T1 + T2 constante.

### 4.3. Mise en œuvre des servo moteurs

Le signal de commande des servo moteurs est défini par une durée T (la période du signal de commande) de 20 ms. La durée T1 est de l'ordre de 1.5 ms (cf. Figure 41). Ce signal de commande est appliqué sur une des broches du servo moteur, la broche de commande. Les deux autres broches permettent d'effectuer l'alimentation électrique du servo moteur.

Afin de pouvoir générer le signal de commande de chaque servo moteur, nous allons utiliser une bibliothèque de fonctions disponibles sous l'environnement Arduino. Cette bibliothèque regroupe les fonctions de base et les fonctions utiles sont disponibles au sein de la bibliothèque *Servo*. La description des fonctions au sein de cette bibliothèque est accessible depuis la page <http://arduino.cc/en/Reference/Libraries> où toutes les librairies de fonctions sont décrites.

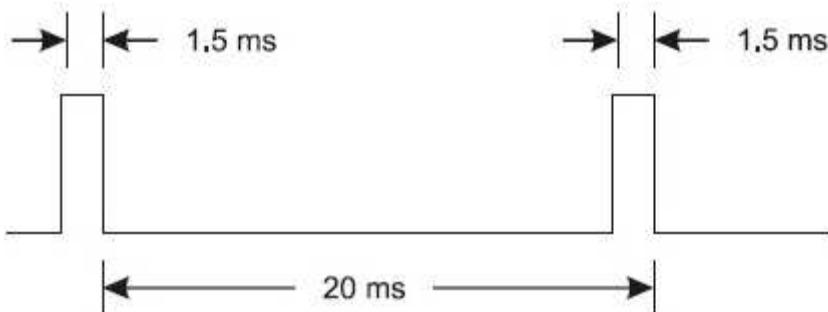


Figure 41 : Signal de commande d'un servo moteur

#### 4.3.1. Fonction de la bibliothèque *Servo*

Les fonctions dont nous aurons besoin par la suite sont listées ci-dessous :

- *attach ()*
- *writeMicroseconds ()*

Il est d'abord nécessaire de déclarer et définir une variable correspondant à l'objet *Servo*. Par exemple, pour déclarer une variable correspondant au servo moteur droit, il faudra effectuer la déclaration suivante :

```
Servo Servo_droit ;
```

Ensuite, il faut définir quelle sera la broche de la carte Arduino sur laquelle sera générée le signal de commande. Pour cela, il faudra utiliser la fonction `attach()`. Si l'on souhaite que ce soit la broche 12 qui soit connectée sur le servo moteur droit, alors on utilisera l'instruction :

```
Servo_droit.attach(13);
```

La dernière étape consiste à générer le signal de commande et surtout contrôler la durée T1. Il faut maintenant utiliser la fonction `writeMicroseconds()` qui permet de gérer une durée exprimée en micro secondes ( $1\mu\text{s} = 10^{-6}\text{ s}$ ). Donc pour générer un signal de commande avec une durée de 1.5 ms sur la broche 13, nous aurons :

```
Servo_droit.writeMicroseconds(1500);
```

Enfin, afin de pouvoir accéder aux fonction de la bibliothèque `Servo`, il faut inclure le fichier `Servo.h` avant la fonction `setup()`, en utilisant la directive :

```
#include <Servo.h>
```

Ecrire alors le programme `Servo_V1` afin de piloter le servo moteur droit, qui sera commandé via la broche 13, le servo moteur gauche via la broche 12. Pour chaque servo moteur, la durée T1 est fixée à 1.5 ms. Les instructions seront pour l'instant placées au sein de la fonction `setup()`.

#### 4.3.2. Mise en place des servo moteurs

##### 4.3.2.1. Description des servo moteurs

Les servo moteurs utilisés sont des servo de la marque Parallax, à **rotation continue**. Ils sont constitués des éléments décrits sur la figure ci-dessous (cf. Figure 42).

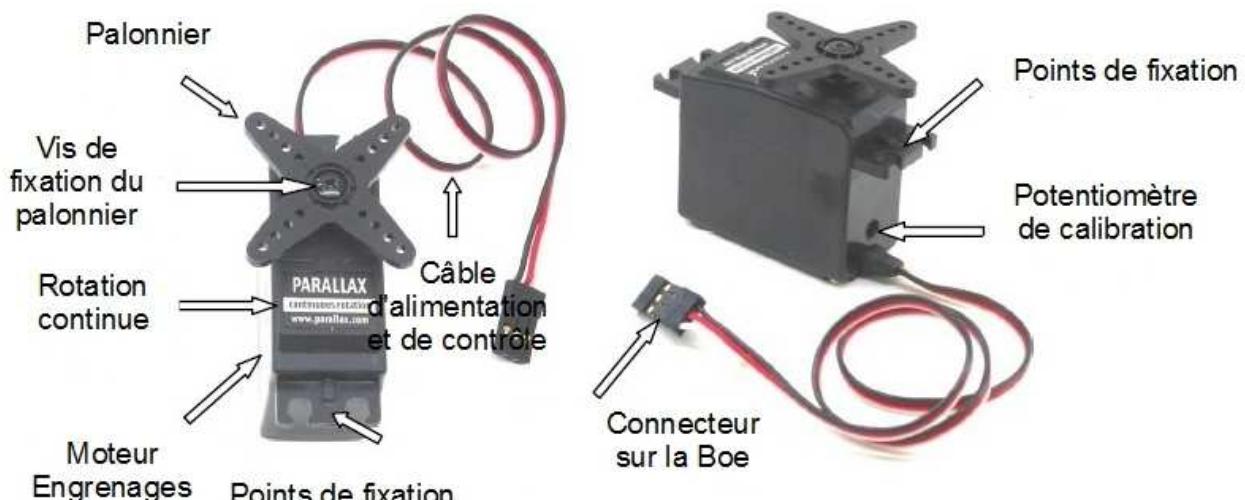


Figure 42 : Elements constitutifs d'un servo moteur Parallax

##### 4.3.2.2. Connexion des servo moteurs sur la carte Boe

#### Précautions

- Débrancher au préalable le câble USB de la carte Arduino.
- Vérifier que l'interrupteur de la Boe est sur la position 0.

- Vérifier que le cavalier d'alimentation de la Boe est sur la position 5V (cf . Figure 43).

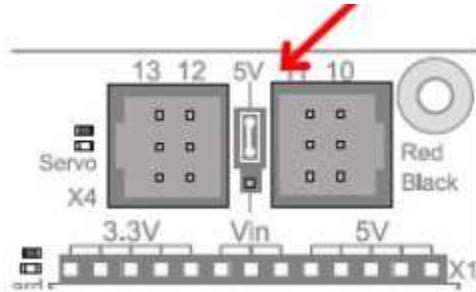


Figure 43 : Position du cavalier d'alimentation de la Boe

Le schéma de connexion des deux servo moteurs est donné ci-dessous (cf. Figure 44).

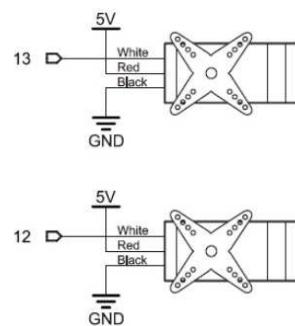


Figure 44 : Schéma de connexion des servo moteurs sur la Boe

Concrètement, il faut enficher les deux connecteurs de chaque servo moteur sur les emplacement correspondant aux repères 12 et 13 comme sur la figure ci-après (cf. Figure 45).

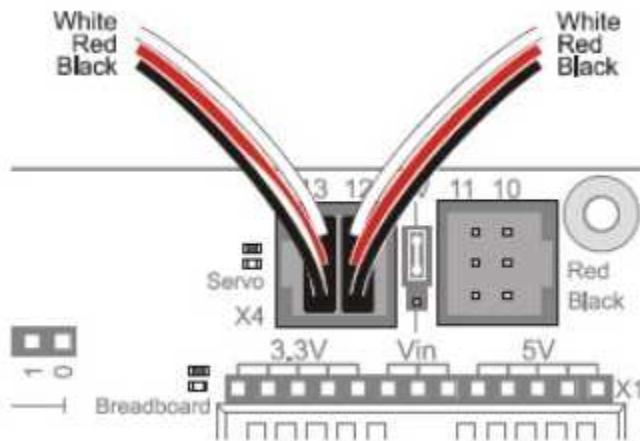


Figure 45 : Connexion des servo moteurs sur la Boe

#### 4.3.2.3. Mise en place du pack de batteries

Le robot sera alimenté par un pack de 5 batteries. Ayant auparavant chargé les batteries, il faut les insérer dans le pack, puis connecter ce dernier pour arriver au montage final décrit par la figure ci-dessous (cf. Figure 46). Le circuit des deux Led sera conservé afin de vérifier que le signal de commande de chaque servo moteur est bien généré.

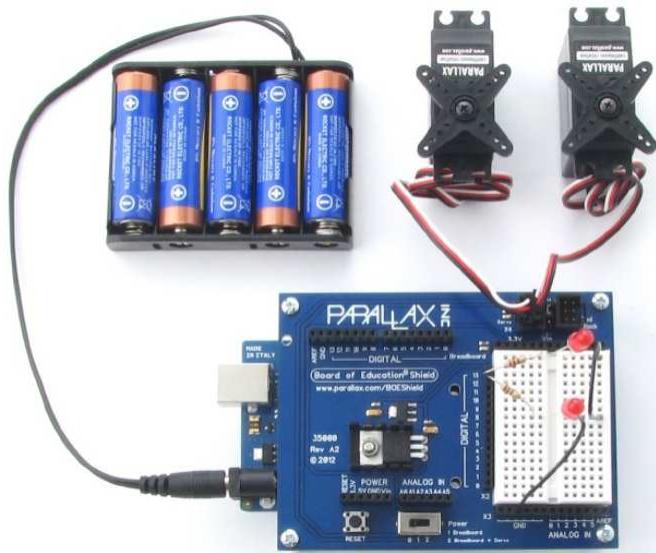


Figure 46 : Boe et servo et pack batteries

#### 4.3.3. Calibration des servo moteurs

Lorsque les servo moteurs n'ont jamais été utilisés, il est nécessaire de les calibrer. Cette calibration permet d'associer à la valeur de  $T_1 = 1.5$  ms le fait que le servo moteur ne tourne pas. Il s'agit en fait du point zéro ou du point neutre. Pour effectuer cette calibration, reprenez le programme Servo\_V1 et le télécharger sur la carte Arduino. Pour cela il faut reconnecter le câble USB, puis mettre l'interrupteur sur la position 1. Après avoir effectué cette opération, basculer l'interrupteur de la carte Boe sur la position 2. Cela permet d'alimenter les servo moteurs. Vous pouvez alors constater qu'ils se mettent à tourner (peu importe le sens pour l'instant).

La calibration consiste alors à ajuster la position du potentiomètre rotatif à l'aide du tournevis fourni (cf. Figure 47). En tournant dans un sens puis dans l'autre (cf. Figure 48), vous devez constater que le servo moteur change de sens de rotation. Il faut alors trouver **la position du potentiomètre** telle que le servo moteur ne tourne plus. Effectuer cette opération sur les deux servo moteurs.

#### Remarques

- Ne pas appuyer fort sur la tête d'entraînement du potentiomètre.
- Ne pas forcer lorsque le potentiomètre est en butée



Figure 47 : Accès au potentiomètre rotatif



Figure 48 : Recherche du point neutre du servo moteur

#### 4.3.4. Gérer le sens de rotation des servo moteurs

La gestion du sens de rotation s'effectue en déterminant la valeur de la durée T1 par rapport à la valeur de T1 correspondant au point neutre du servo moteur. Pour mémoire, cette valeur de référence est égale à 1500 µs. Selon que la durée T1 est inférieure ou supérieure à 1500µs, le servo moteur tournera dans un sens ou dans l'autre, comme spécifié sur les deux figures ci-dessous (cf. Figure 49 et Figure 50).

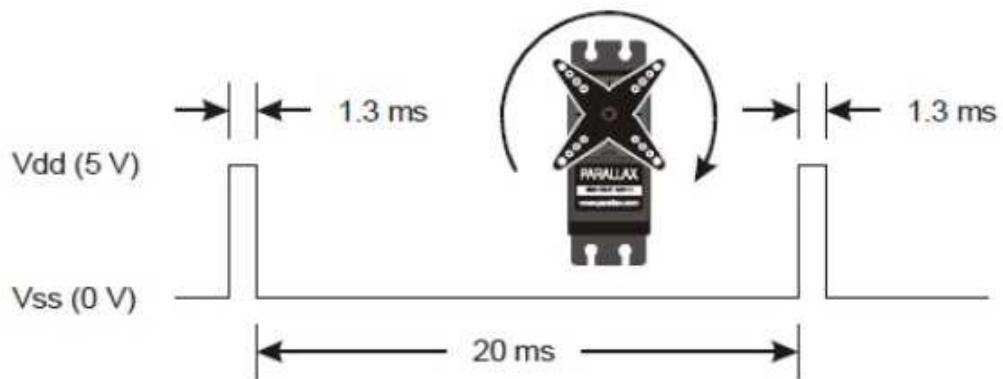


Figure 49 : Rotation servo moteur sens horaire

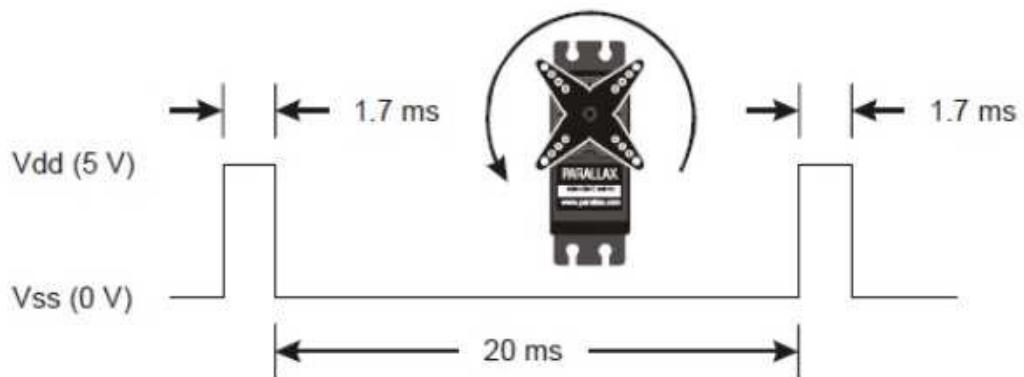


Figure 50 : Rotation servo moteur sens trigonométrique

Avec ces deux valeurs de T1, le servo moteur tourne à une vitesse de l'ordre de 50 à 60 tours par minutes.

Ecrire le programme Servo\_V2 qui permet de faire tourner le servo moteur droit dans un sens puis d'ans l'autre.

Faire de même pour le servo moteur gauche avec le programme Servo\_V3.

#### 4.3.5. Contrôler la vitesse et le sens de rotation des deux servo moteurs

Il est désormais possible de contrôler simultanément les deux servo moteurs en les faisant tourner selon des sens opposés. Pour cela, il suffit d'affecter au premier une commande avec une valeur de T1 égale à 1700 $\mu$ s et pour le second servo moteur une commande avec une valeur de T1 égale à 1300  $\mu$ s par exemple.

Ecrire le programme Servo\_V4 qui permet de faire tourner les deux servo moteurs dans un sens pour l'un et l'autre sens pour l'autre.

En fonction des différents paramètres de rotation utilisés pour faire tourner chaque servo moteur, remplissez le tableau ci-dessous observer ce qui se passe.

Servo gauche	Servo droit	Description	Observation
T1 = 1300 $\mu$ s	T1 = 1700 $\mu$ s		
T1 = 1700 $\mu$ s	T1 = 1300 $\mu$ s		
T1 = 1700 $\mu$ s	T1 = 1700 $\mu$ s		
T1 = 1300 $\mu$ s	T1 = 1300 $\mu$ s		
T1 = 1500 $\mu$ s	T1 = 1500 $\mu$ s	Servo moteurs à l'arrêt	
T1 = 1500 $\mu$ s	T1 = 1700 $\mu$ s		
T1 = 1700 $\mu$ s	T1 = 1500 $\mu$ s		
T1 = 1300 $\mu$ s	T1 = 1400 $\mu$ s		
T1 = 1400 $\mu$ s	T1 = 1300 $\mu$ s		

#### 4.3.6. Contrôler la durée d'activité des servo moteurs

En plus de pouvoir contrôler désormais le sens de rotation, il est également possible de contrôler la durée d'activité de chaque servo moteur. Pour illustrer cet aspect, nous allons considérer le cycle suivant :

- 1. Faire tourner les deux servo moteurs dans **le sens horaire** pendant 3s ;
- 2. Faire tourner les deux servo moteurs dans **le sens trigonométrique** pendant 2.5s
- 3. Arrêter les deux servo moteurs.

Entre chaque étape de ce cycle, il suffit d'insérer une temporisation avec la fonction *delay*, comme le montre le programme Servo\_V5 ci-dessous.

Pour arrêter les servo moteurs, il est aussi possible d'utiliser la fonction *detach ()* de la bibliothèque *Servo* en écrivant l'instruction :

```
Servo_droit.detach () ;
```

ce qui a pour effet de stopper la rotation du servo moteur droit.

```

1  /* Programme Servo_V5                               */
2  /* Commande des servo moteurs droit et gauche      */
3  /* Contrôle de la durée d'activité des servo moteurs */
4  /* Leds connectées sur broches 12 et 13             */
5
6  #include <Servo.h>
7
8  // Définition de constantes symboliques
9  #define SERVO_GAUCHE_PIN 12 // Commande servo droit sur broche 12
10 #define SERVO_DROIT_PIN 13 // Commande servo droit sur broche 13
11 #define T1_1 1300 // Durée T1 en us, sens horaire
12 #define T1_2 1700 // Durée T1 en us, sens trigonométrique
13 #define T1_3 1500 // Durée T1 en us, arrêt
14
15 // Variables globales
16 Servo Servo_droit;
17 Servo Servo_gauche;
18
19 void setup ()
20 {
21     Servo_droit.attach(SERVO_DROIT_PIN);
22     Servo_gauche.attach(SERVO_GAUCHE_PIN);
23
24     Servo_droit.writeMicroseconds (T1_1);
25     Servo_gauche.writeMicroseconds (T1_1);
26     delay (3000);
27     Servo_droit.writeMicroseconds (T1_2);
28     Servo_gauche.writeMicroseconds (T1_2);
29     delay (2500);
30     Servo_droit.writeMicroseconds (T1_3);
31     Servo_gauche.writeMicroseconds (T1_3);
32 }
33
34 void loop ()
35 {
36
37 }
```

## 5. Assembler le robot

Avant de programmer le robot et de gérer ses déplacements, nous allons procéder à son montage final, pour arriver au résultat présenté ci-dessous (cf. Figure 51).

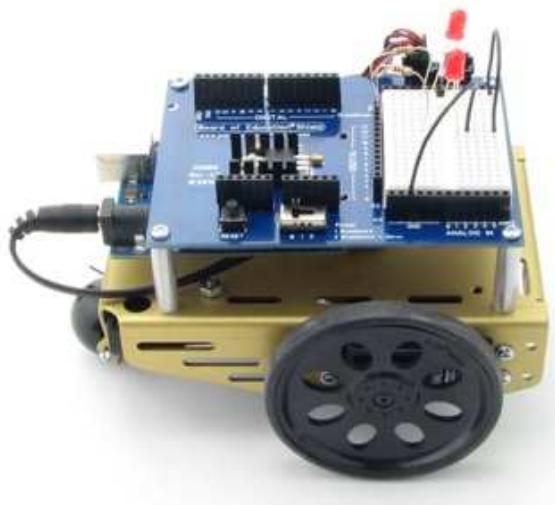


Figure 51 : Montage du robot achevé

Avant de procéder au montage, il est impératif de débrancher la pack de batteries ainsi que le câble USB.

### 5.1. Montage des servo moteurs et du pack de batteries

Commencer par débrancher les servo moteurs de la Boe, puis retirer de chaque servo moteur le cabestan en dévissant la vis. Penser à conserver la vis, car elle sera utilisée plus tard pour monter les roues (cf. Figure 52).



Figure 52 : Préparation des servo moteurs

Avant de monter les servo moteurs sur le châssis, il est nécessaire de mettre en place la bague de protection en caoutchouc sur le trou au milieu du châssis ainsi que les deux entretoises situées à l'avant du robot (cf. Figure 53).



Figure 53 : Protection du passage de câbles sur le châssis

Puis il faut monter les servo moteurs sur le châssis. Il y a deux options possibles, comme présentées sur la figure suivante (cf. Figure 54). Ces deux options sont presque équivalentes, à ceci près que l'une d'entre elles permettra de recalibrer les servo moteurs sans avoir à les démonter. D'autre part, la mobilité des deux bases n'est pas tout à fait identique. Pour finir, repérer le servo moteur droit et le gauche

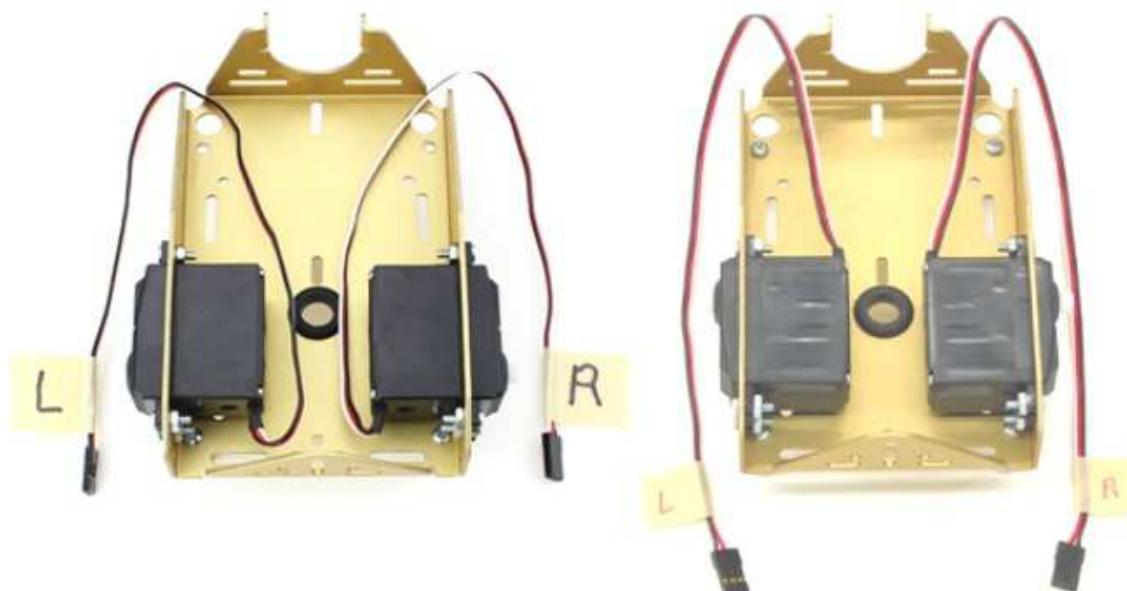


Figure 54 : Montage des servo moteurs sur le châssis

L'étape suivante consista à positionner le support des batteries. Il faut utiliser les vis dont la tête est fraîsée. Le résultat est donné ci-dessous (cf. Figure 55).



Figure 55 : Montage du pack de batterie

Pour finir, faire remonter les câbles des servo moteurs et celui du pack batterie sur le dessus du châssis (cf. Figure 56).

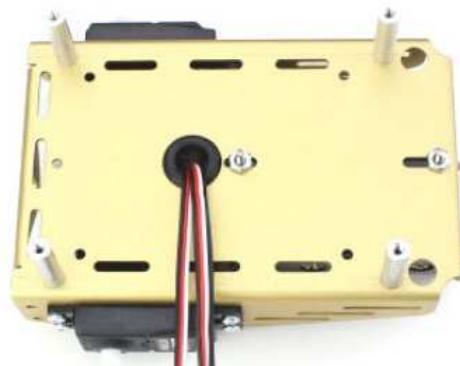


Figure 56 : Passage des câbles

## 5.2. Montage des roues

Le montage des roues nécessite les éléments présents sur la figure ci-dessous (cf. Figure 57).



Figure 57 : Eléments pour le montage des roues

Les roues latérales se placent sur l'axe de chaque servo moteur et sont bloquées avec la vis du cabestan. Il faut ensuite placer un bandage en caoutchouc sur chacune des roues afin d'éviter qu'elles ne glissent. Quant à la roue arrière, elle est fixée à l'aide de la goupille que sert d'axe de rotation. Lorsque la goupille est totalement engagée, il faut la bloquer en retournant ses deux extrémités (cf. Figure 58).

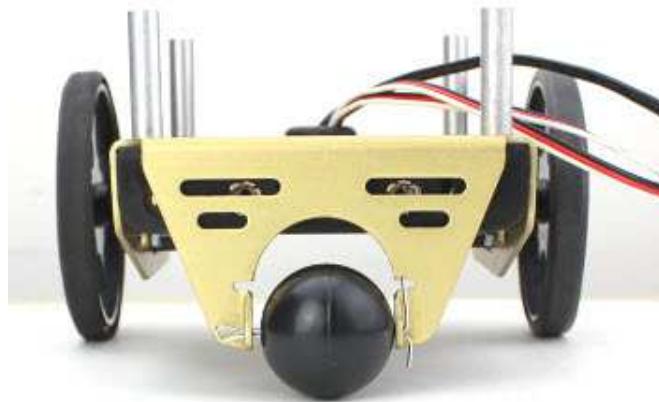


Figure 58 : Montage roue arrière

## 5.3. Mise en place de la carte Arduino et de la Board of Education

Il faut placer puis fixer la Boe sur ses entretoises, puis câbler les deux servo moteurs respectivement sur les ports 12 et 13. L'avant du robot se trouve du côté de la plaque de prototypage et l'arrière du côté de la roue folle.

Lorsque le montage est achevé, il faut tester de nouveau les servo moteurs droit et gauche en considérant le cycle suivant pour chaque servo moteur :

- Tourner dans le sens horaire 3s
- Arrêt pendant 2.5 s
- Tourner dans le sens trigonométrique 2 s

## - Arrêt du servo moteur

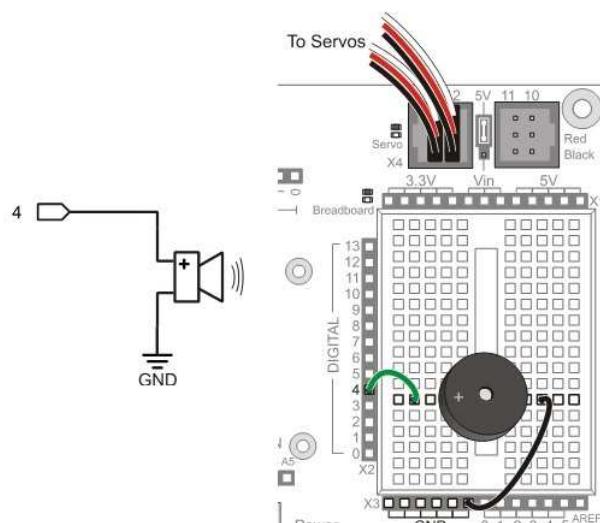
### 5.4. Mise en œuvre du buzzer

Parmi les ressources disponibles, nous pouvons utiliser un buzzer, dispositif piezzo électrique qui permet d'émettre des sons dont la tonalité est définie par la fréquence du signal numérique appliquée à ses bornes. Ce buzzer peut être utilisé comme indicateur pour indiquer le début ou la fin d'une tâche par exemple. Il est schématisé par le pictogramme ci-dessous (cf. Figure 59). **Il faut remarquer que ce composant est polarisé.**



Figure 59 : Schéma et représentation d'un buzzer

La mise en œuvre du buzzer s'effectue à l'aide du montage décrit ci-dessous (cf. Figure 60).



产生声音，3000hz, 2s;暂停1s后重新开始；  
附加题：产生音乐

Figure 60 : Câblage du buzzer sur la Boe

La commande du buzzer s'effectue à l'aide de la fonction `tone()` décrite dans <http://arduino.cc/en/Reference/Tone>. La seconde fonction qui sera utilisée est `noTone()` qui permet de stopper la génération du son.

A l'aide de ces fonctions, réalisez le programme **Buzzer\_V1** permettant de générer un son, à la fréquence de **3000Hz**, pendant 2s. Après une pause de 1s, reprendre la génération du même son.

Question supplémentaire : générer une mélodie.

### 5.5. Courbe de variation de vitesse de rotation des servo moteurs

Il s'agit de tracer la courbe permettant de connaître la vitesse de rotation d'une servo moteur en fonction de la largeur de l'impulsion du signal de commande. Pour cela, il faut renseigner le tableau ci-dessous (cf. Tableau 3) pour ensuite tracer la courbe correspondante (vous pouvez utiliser un

tableur type Excel ou le tableur d'Open Office pour effectuer automatiquement le tracé de la courbe demandée).

<b>Tableau de variation de la vitesse de rotation en fonction de la largeur du signal générée</b>							
Largeur d'impulsion en µs	Vitesse en tours par minute	Largeur d'impulsion en µs	Vitesse en tours par minute	Largeur d'impulsion en µs	Vitesse en tours par minute	Largeur d'impulsion en µs	Vitesse en tours par minute
1300		1400		1500		1600	
1310		1410		1510		1610	
1320		1420		1520		1620	
1330		1430		1530		1630	
1340		1440		1540		1640	
1350		1450		1550		1650	
1360		1460		1560		1660	
1370		1470		1570		1670	
1380		1480		1580		1680	
1390		1490		1590		1690	
						1700	

**Tableau 3 : Variation de la vitesse de rotation des servo moteur en fonction de la largeur de l'impulsion**

Le programme ci-dessous permet d'effectuer le relevé des mesures qui serviront à établir la courbe de la vitesse de rotation en fonction de la largeur de l'impulsion du signal de commande du servo moteur.

```

1  /* Programme Etalonnage des servo moteurs          */
2  /* Vitesse de rotation en fonction de la largeur d'impulsion   */
3  /* Envoi d'un caractère depuis le moniteur série pour démarrer */
4  /* la rotation du servo moteur                         */
5  /* Le servo moteur doit tourner pendant 6 s           */
6  /* Débuter à 1300 us et finir à 1700 us par pas de 10 us */
7
8  #include <Servo.h>
9
10 #define BUZZER_PIN 4 // Buzzer commandé par broche 4
11 #define SERVO_GAUCHE_PIN 12 // Commande servo droit sur broche 12
12 #define SERVO_DROIT_PIN 13 // Commande servo droit sur broche 13
13
14 #define DUREE 6000
15
16 Servo Servo_droit;
17 Servo Servo_gauche;
18
19 int largeur_impulsion;
20
21 void setup()
22 {
23     tone(BUZZER_PIN, 3000);
24     delay(1000);
25     noTone(BUZZER_PIN);
26
27     Serial.begin(9600);
28
29     Servo_droit.attach(SERVO_DROIT_PIN);
30     // Servo_gauche.attach(SERVO_GAUCHE_PIN);
31     Servo_droit.writeMicroseconds(1500);
32     // Servo_gauche.writeMicroseconds(1500);
33 }
34
35 void loop()
36 {
37
38     for(largeur_impulsion = 1300; largeur_impulsion <= 1700; largeur_impulsion += 10)
39     {
40         Serial.print("Largeur_impulsion = ");
41         Serial.println(largeur_impulsion);
42         Serial.println("Appuyer sur une touche et valider");
43         Serial.println("Envoi pour demarrer le servo moteur...");
44
45         while(Serial.available() == 0);
46         Serial.read();
47
48         Serial.println("Running...");
49         Servo_droit.writeMicroseconds(largeur_impulsion);
50         // Servo_gauche.writeMicroseconds(largeur_impulsion);
51         delay(DUREE);
52         Servo_droit.writeMicroseconds(1500);
53         // Servo_gauche.writeMicroseconds(1500);
54     }
55 }

```

Le programme ci-dessus permet d'effectuer l'ensemble des mesures de vitesse de rotation. Télécharger ce programme et effectuer les relevés de mesure pour ensuite tracer les courbes de vitesse de rotation en fonction de la largeur de l'impulsion pour les servo moteurs droit et gauche.

## 6. Navigation du robot

Il s'agit dans cette partie de mettre en place les méthodes qui permettront de contrôler les déplacements du robot. Lorsque ces méthodes seront maîtrisées, alors nous écrirons les fonctions de base pour gérer les différents déplacements, afin de pouvoir, dans les chapitres suivants les réutiliser lorsque nous prendrons en compte des informations issues de différents capteurs. Les principaux déplacements à maîtriser concernent la marche avant, arrière, ainsi que le fait de pouvoir tourner à droite ou à gauche (cf. Figure 61).

### 6.1. Déplacement en avant

Afin que le robot se déplace en avant, il est nécessaire que la roue droite tourne dans le sens horaire et la roue gauche dans le sens trigonométrique. Ecrire le programme qui permet au robot d'avancer pendant 3 s à la vitesse maximum, et le sauver dans le fichier **Avance\_V1**. Il s'appuie sur l'utilisation de la bibliothèque *Servo* comme précédemment. Vérifier que le programme écrit réalise ce qui est attendu.

### 6.2. Déplacement en arrière

Par rapport au déplacement en avant, il suffit d'inverser le sens de rotation des roues droite et gauche. Ecrire le programme **Recule\_V1**. Vérifier que le robot se déplace bien en marche arrière.

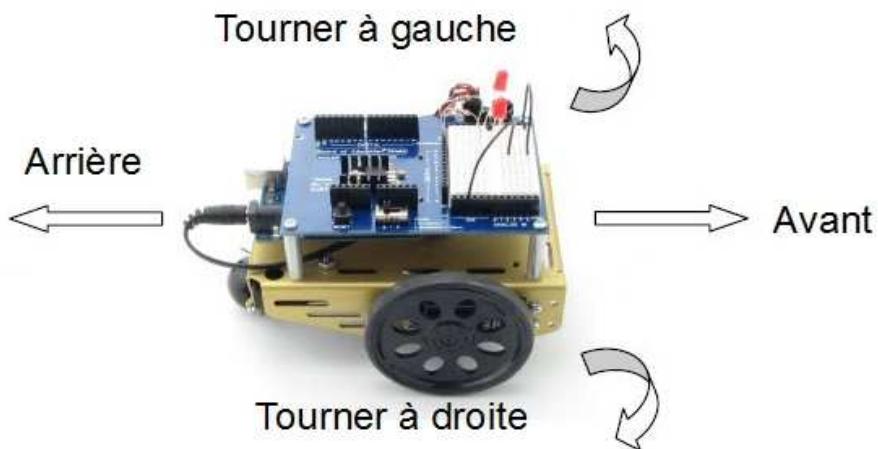


Figure 61 : Déplacements du robot

### 6.3. Pivoter et tourner

Il est également possible de faire **pivoter** sur lui-même le robot. Le centre de rotation se situe au milieu du segment passant par l'axe des deux roues. Pour faire pivoter le robot vers la droite, il faut que les roues droite et gauche tournent dans le même sens, et de plus le sens trigonométrique. Modifier le programme **Recule\_V1** et le sauver dans le programme **Pivote\_droit\_V1**.

Pour que le robot pivote sur lui-même vers la gauche, il faut que les deux roues tournent alors dans le sens horaire. Modifier le programme **Pivote\_droit\_V1** et le sauver dans le fichier **Pivote\_gauche\_V1**. Vérifier de nouveau que le comportement du robot est bien celui attendu.

Le deuxième type de déplacement est d'avancer ou de reculer tout en tournant. Dans ce cas, la vitesse de rotation des roues doit être différente à droite et à gauche. Par exemple, vous pouvez

utiliser des vitesses de rotation définies par les largeurs d'impulsion (1700, 1600) ou (1300, 1400). A partir des couples de valeurs, déterminer pour les roues droite et gauche les valeurs de largeurs d'impulsion qui permettent :

- Avancer et tourner à droite
- Avancer et tourner à gauche
- Reculer et tourner à droite
- Reculer et tourner à gauche

## 6.4. Gérer les déplacements avec des fonctions

La gestion des déplacements nécessite de retenir les durées des largeurs d'impulsion à appliquer sur la commande des servo moteurs droit et gaucher. Maintenant que ces paramètres sont identifiés, nous allons écrire des fonctions qui permettront une gestion des déplacements du robot plus simple à gérer.

### 6.4.1. Retour sur l'écriture d'une fonction

Dans nos différents programmes, nous avons déjà rencontré l'utilisation de fonctions (par exemple *delay*, *tone*,...). Nous allons voir ici comment écrire nos propres fonctions, puis celles nous permettant ensuite de gérer nos déplacements.

Toute fonction est constitué d'un en tête et du corps de la fonction

L'en-tête permet de définir :

- Le type de valeur renournée par la fonction
- L'identificateur de la fonction, c.à.d. son nom
- La liste des paramètres

sous la forme : type Identificateur (liste des paramètres)

Le corps de la fonction est délimité par les symboles { et }. Il contient les instructions de la fonction.

Chaque fonction est accompagnée de son prototype, défini par l'en tête, suivi d'un point-virgule.

Exemple : écriture de la fonction Ma\_fonction, ne retournant aucune valeur et admettant comme paramètre un entier de type int.

```
void Ma_fonction (int mon_parametre)
```

dont le prototype sera : void Ma\_fonction (int mon\_parametre) ;

Une fois l'écriture de la fonction réalisée, il est possible de l'utiliser en effectuant l'appel à la fonction par l'instruction :

```
Ma_fonction (2000) ;
```

par exemple.

### 6.4.2. Fonctions des déplacements de base

Par la suite, nous aurons besoins des fonctions suivantes :

- Avance
- Recule

- Pivot\_droite
- Pivot\_gauche
- Arret

qui admettrons comme paramètre la durée du déplacement (à l'exception de la fonction Arret).

Ainsi, la fonction Avance sera écrite comme ci-dessous :

```
void Avance (int duree_deplacement)
{
    Servo_droit.writeMicroseconds(1300);
    Servo_gauche.writeMicroseconds(1700);
    delay (duree_deplacement);
}
```

Et la fonction permettant de stopper le robot pourra s'écrire :

```
void arret (void)
{
    Servo_droit.detach ();
    Servo_gauche.detach ();
}
```

Dans les deux cas, il est impératif de déclarer en variables globales les deux servo moteurs droit et gauche par les déclarations :

```
Servo Servo_droit;
Servo Servo_gauche;
```

et d'inclure le fichier *Servo.h*.

A partir de ces deux exemples, écrire les fonctions Recule, Pivot\_droite et Pivot\_gauche. Reprendre alors les programmes Avance\_V1, Recule\_V1, Pivot\_droit\_V1, Pivot\_gauche\_V1 en utilisant les fonctions que vous venez d'écrire pour réaliser les programmes Avance\_V2, Recule\_V2, Pivot\_droit\_V2, Pivot\_gauche\_V2.

## 7. Navigation et détection d'obstacles

Maintenant que nous disposons des fonctions de base pour déplacer le robot, nous allons le munir de capteurs qui vont lui permettre de prélever des informations sur l'environnement au sein duquel il évolue. Le premier capteur que nous allons mettre en œuvre est un capteur tactile. Il se présente sous la forme de « moustache » et permettra de détecter des obstacles. En fonction des informations fournis par ces capteurs, nous pourrons adapter le comportement du robot. Le robot muni de ses « moustaches » est représenté ci-dessous (cf. Figure 62).

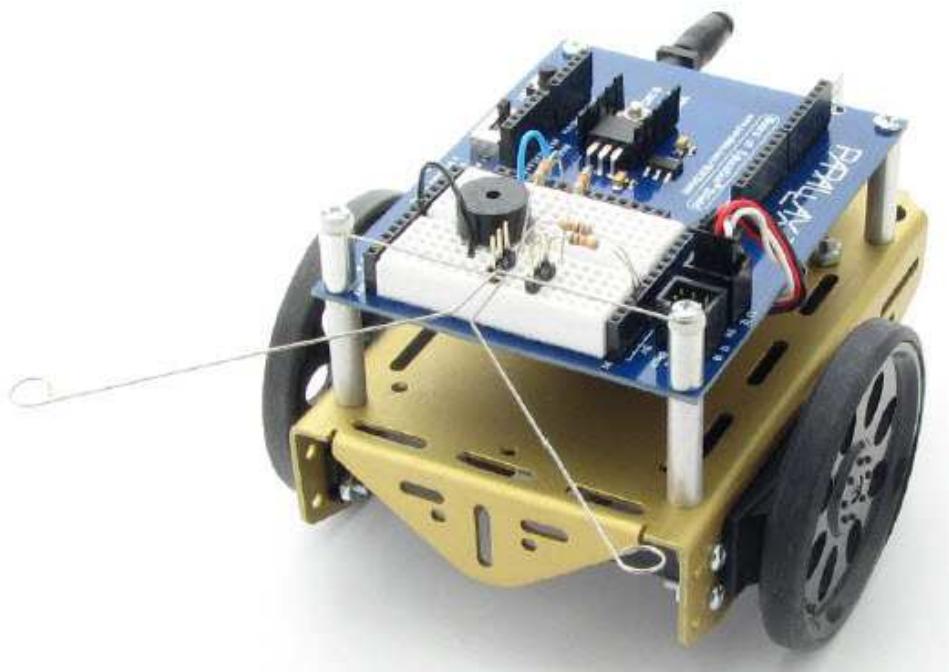


Figure 62 : Robot et ses moustaches

### 7.1. Construction des moustaches

La mise en place des moustaches nécessite les éléments ci-dessous, dont deux résistances de  $220\Omega$  et  $10k\Omega$  (cf. Figure 63).

A partir de ces éléments, le montage des moustaches est explicité par la figure suivante (cf. Figure 64).



Figure 63 : Eléments constitutifs des moustaches

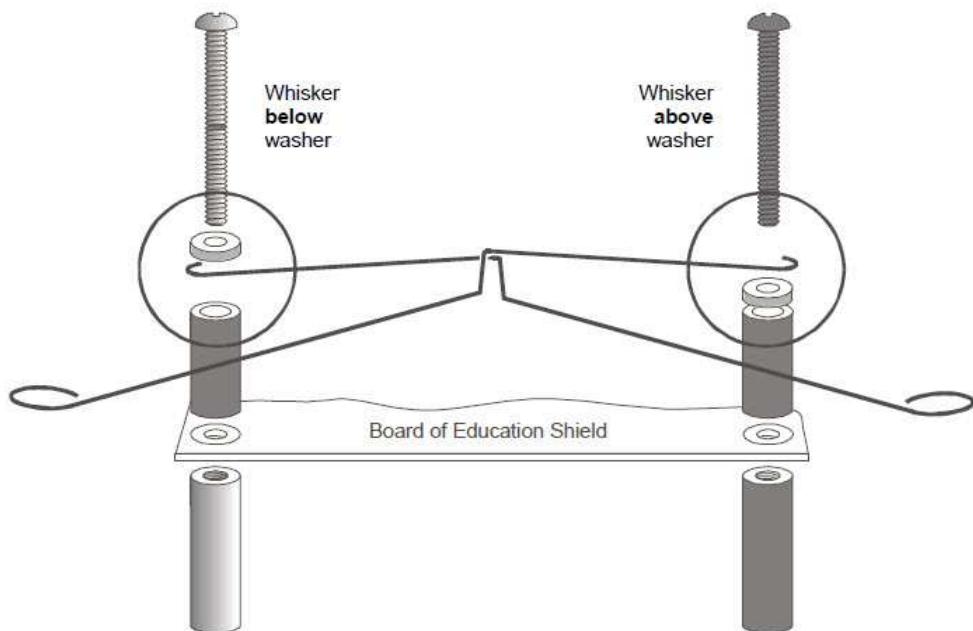


Figure 64 : Montage des moustaches

Une fois les moustaches assemblées, il faut câbler le circuit qui permettra de recueillir le signal issu de chaque moustache. Ce circuit est défini ci-dessous (cf. Figure 65).

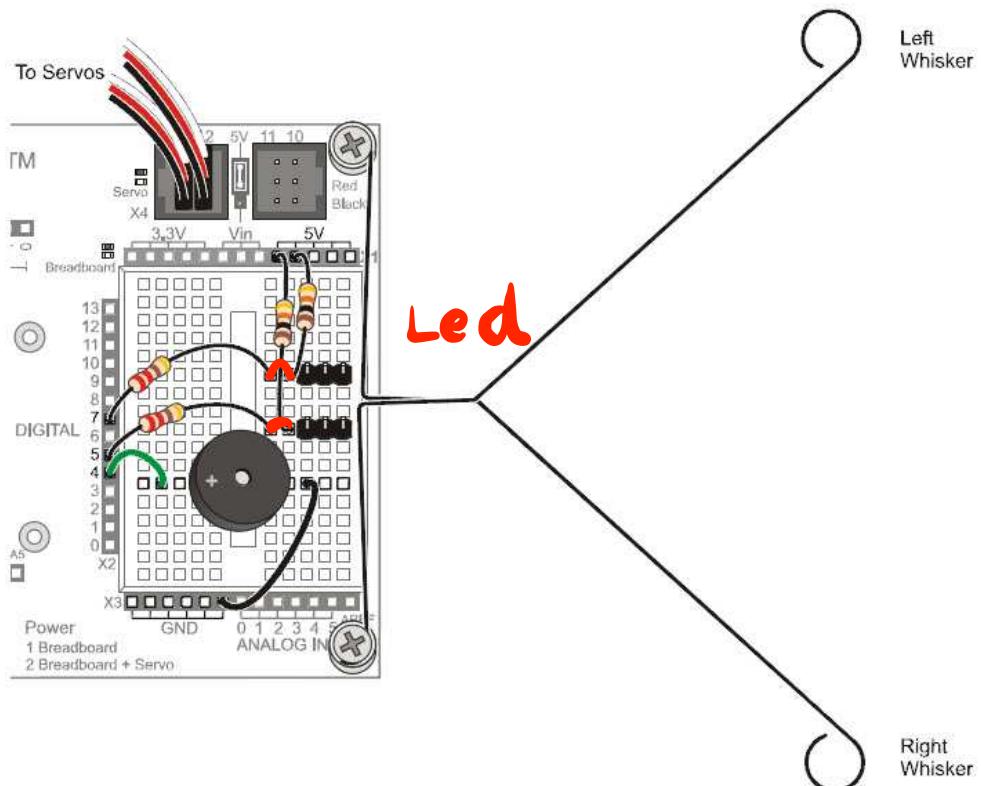


Figure 65 : Montage du circuit associé aux moustaches

## 7.2. Principe de fonctionnement des moustaches

Le schéma ci-dessous décrit le principe de fonctionnement de chacune des moustache (cf. Figure 66).

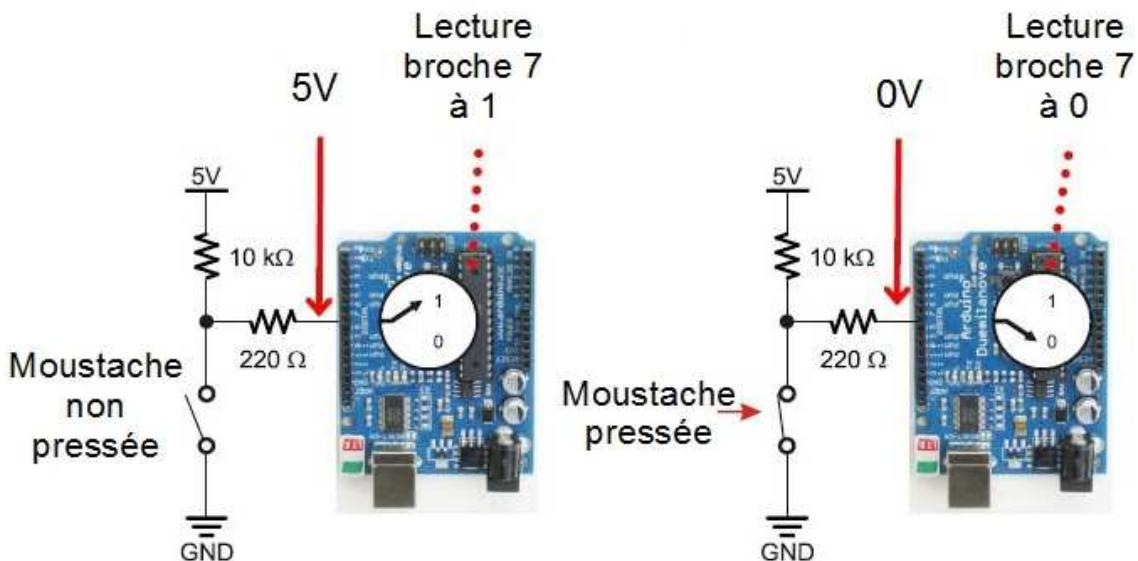


Figure 66 : Principe de fonctionnement des moustaches

Chaque moustache se comporte comme un interrupteur. Lorsque la moustache n'est pas pressée (elle n'est pas en contact avec un objet), alors la tension lue sur la broche correspond à 5V, dans le cas contraire (un objet est au contact de la moustache), alors la tension lue est égale à 0V. Cela suppose donc que l'on puisse lire l'état d'une broche, et que l'on puisse également par la suite stocker l'état de chaque moustache au sein d'une variable.

## 7.3. Test de l'état des moustaches

La lecture de l'état de la moustache gauche s'effectue sur la broche 5 tandis que pour la moustache droite, la lecture de son état s'effectue sur la broche 7. Les niveaux lus sont des niveaux logiques numériques correspondant à LOW et HIGH. Cela implique également que ces deux broches doivent être en entrée.

Pour définir les broches 5 et 7 en entrée, il faut utiliser la fonction *pinmode* comme suit :

```
pinMode (5, INPUT); // Pour la moustache gauche  
pinMode (7, INPUT); // Pour la moustache droite
```

Pour lire l'état de chacune de ces broches, il faut utiliser la fonction *digitalRead* avec la syntaxe :

```
digitalRead (5);  
digitalRead (7);
```

Voir aussi <http://arduino.cc/en/Reference/HomePage> pour la description de ces deux fonctions.

La fonction *digitalRead* renvoie la valeur lue sur la broche correspondante, soit HIGH ou LOW. Il est alors possible de stocker la valeur renournée dans une variable pour une utilisation ultérieure.

Afin de tester l'état des moustaches, nous allons réaliser le programme **Moustaches\_V1**. En ouvrant le moniteur série, selon que les moustaches sont pressées ou non, vous devez voir évoluer l'état logique des broches 5 et 7. Tant que ce n'est pas le cas, vous devez rechercher les causes de dysfonctionnement.

#### 7.4. Visualiser l'état des moustaches avec les Leds

Afin de pouvoir identifier quelles est la moustache qui est pressée lorsque le robot rencontre un obstacle, il est possible d'utiliser les **Leds** pour visualiser l'état de chaque moustache. Pour cela, nous allons compléter le circuit des moustaches par le circuit de Leds ci-dessous (cf. Figure 67).

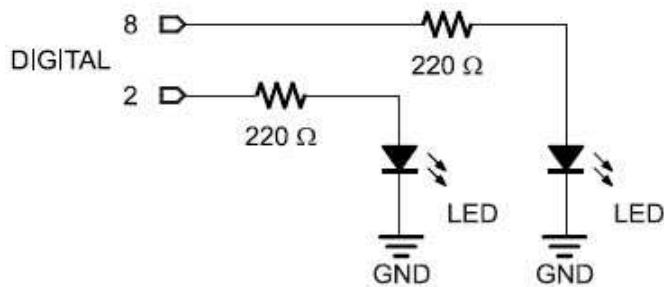


Figure 67 : Circuit de Leds et moustaches

La Led connectée sur la broche 8 sera associée à la moustache gauche, tandis que celle connectée sur la broche 2 sera associée à la moustache droite. Lorsque par exemple, la moustache gauche sera pressée, alors la Led connectée sur la broche 8 sera allumée, et éteinte dans le cas contraire. La gestion des Leds s'effectuera comme suit :

**si** (moustache gauche est pressée) **alors** allumée Led de la broche 8  
**sinon** éteindre Led de la broche 8

ce qui peut se traduire par le code :

```
if (mgauche == LOW) digitalWrite (MGAUCHE_PIN, HIGH);
else digitalWrite (MGAUCHE_PIN, LOW);
```

en reprenant les variables utilisées dans le programme Moustaches\_V1.

Inclure cette gestion des Leds dans le programme précédent, et le sauver dans le fichier Moustache\_V2. Vérifier que le résultat obtenu est cohérent avec le fonctionnement des moustache.

#### 7.5. Déplacement du robot et gestion des moustaches

Nous allons maintenant inclure la gestion des moustaches dans la stratégie de déplacement du robot. Pour cela, nous considérons le comportement suivant :

**si** (aucune moustache pressée) alors avancer pendant 50 ms  
**sinon si** (moustache gauche pressée) **alors**  
{  
    Reculer pendant TBack secondes  
    Pivoter vers la droite de 60 degrés environ  
} **sinon si** (moustache droite pressée) **alors**

```

{
    Reculer pendant TBack secondes
    Pivoter vers la gauche de 60 degrés environ
} sinon si (moustaches droite et gauche pressées) alors
{
    Reculer pendant TBack secondes
    Pivoter de 180 degrés environ
}

```

Il va donc falloir utiliser les fonctions associées aux déplacements de base telles que nous les avons considérés dans le chapitre précédent, associées à la prise en compte de l'état des moustaches.

Ecrire le programme **Moustache\_V3** qui vous permettra d'implémenter cette stratégie de déplacement.

Vérifier que le comportement du robot est conforme à cette stratégie de déplacement.

## 8. Navigation et capteurs de lumière

Nous allons mettre en œuvre un second type de capteur, qui permettra de prendre en compte la **luminosité ambiante** au sein de l'environnement où le robot évolue. Après avoir précisé les caractéristiques du capteur, et défini sa mise en œuvre matérielle et logicielle, nous verrons comment prendre en compte les informations délivrées par ce capteur pour contrôler les déplacements du robot.

### 8.1. Description du capteur utilisé

Le capteur sur lequel nous allons nous appuyer est un photo transistor. Fonctionnellement, s'agit d'un transistor bipolaire, dont le courant qui le traverse est contrôlé par la lumière qui éclaire la base. La représentation d'un photo transistor<sup>4</sup> est conforme à celle de la figure ci-dessous (cf. Figure 68Figure 68 : Représentation d'un photo transistor). Il est constitué de la base (B), de l'émetteur (E) et du collecteur (C).

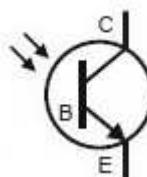


Figure 68 : Représentation d'un photo transistor

Lorsque la base est éclairée, le photo transistor est traversé par un courant proportionnel à l'éclairage de la base. Il s'agit du courant de collecteur qui est transmis vers l'émetteur. Lorsque la base n'est pas éclairée, le photo transistor se comporte comme un interrupteur ouvert. Aucun courant ne le traverse.

Le photo transistor qui sera utilisé est sensible à la lumière ambiante, et plus spécifiquement à la composante du rayonnement infrarouge contenu dans la lumière. Vous trouverez en annexe le spectre des couleurs en fonction de la longueur d'onde exprimée en nm (nano mètre –  $10^{-9}$  m).

---

<sup>4</sup> Le transistor a été inventé par trois chercheurs des laboratoires Bell en 1947, John Bardeen, William Shockley et Walter Brattain). Ces trois chercheurs ont reçu le prix Nobel de physique en 1956.

Les photo transistors utilisés par la suite sont plutôt sensibles au rayonnement infrarouge (IR). Ils possèdent toutefois une certaine sensibilité dans le spectre visible. Il faut également souligner que les sources d'éclairage telles que les lampes à incandescence ou les néons génèrent de la lumière infrarouge. Les capteurs utilisés pourront donc être mis en œuvre en environnement intérieur.

## 8.2. Mise en œuvre sur le robot

Le photo transistor utilisé est représenté sur la figure ci-dessous (cf. Figure 69). Il ne faut pas le confondre avec une Led IR dont le sommet est davantage arrondi.

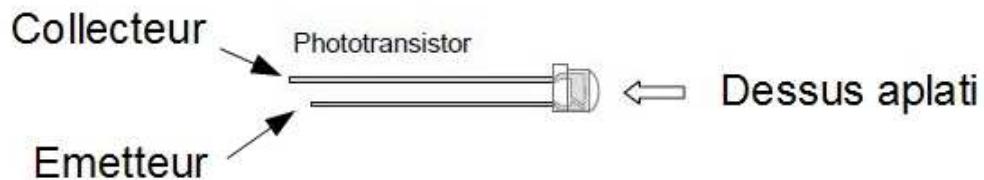


Figure 69 : Photo transistor - composant

Sa mise en œuvre est décrite sur le schéma (cf. Figure 70) et son câblage sur la plaque de prototypage est illustré ci-après (cf. Figure 71).

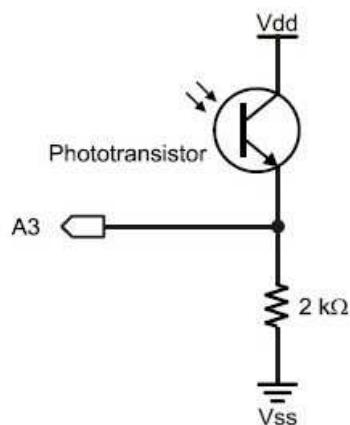


Figure 70 : Schéma de câblage du photo transistor

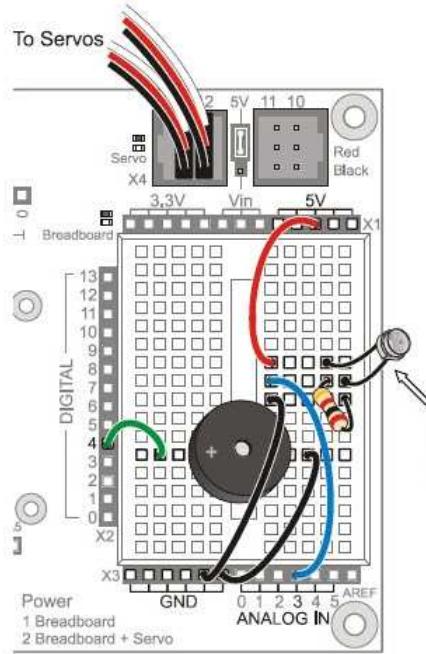


Figure 71 : Câblage du photo transistor sur la Boe

Le signal issu du photo transistor est renvoyé vers la broche A3 du microcontrôleur. Il correspond à la tension aux bornes de la résistance placée en série de l'émetteur du photo transistor. Plus le photo transistor est éclairé, plus le courant I qui le traverse est également important. Ce même courant I traverse aussi la résistance R. En appliquant la loi d'Ohm, le signal issu sur la broche A3 correspond à la tension  $V_{A3} = R \cdot I$ , pouvant évoluer entre 0V et 5V (Vdd). C'est donc cette tension qui est appliquée en entrée de la broche A3 du microcontrôleur. Cette broche A3 est connectée à un convertisseur analogique – numérique qui permet de quantifier la tension appliquée en entrée sur 10 bits, soit 1024 valeurs sur l'ensemble {0, ..., 1023}.

Dans un premier temps, nous allons afficher dans la fenêtre du moniteur série à la fois la valeur numérique lue sur la broche A3, mais aussi la valeur de la tension aux bornes de la résistance. Pour cela, nous disposons de la fonction *analogRead* (cf. <http://arduino.cc/en/Reference/AnalogRead>) qui permet de lire la tension aux bornes d'une entrée analogique et de quantifier cette valeur sur 10 bits. A partir de cette valeur numérique, il est possible de retrouver la tension aux bornes de la résistance en utilisant la relation suivante :

$$V_R = V_{A3} = \text{Valeur numérique} * 5 / 1024$$

Le programme **Phototransistor\_V1** permet d'afficher la valeur de la tension aux bornes de la résistance ainsi que la valeur numérique correspondante. L'affichage est rafraîchi toutes les secondes. Reprendre ce programme et vérifier les résultats obtenus.

```

1  /* Programme Photo_transistor_V1           */
2  /* Mise en oeuvre photo transistor        */
3  /* Prise en compte de la luminosité ambiante */
4  /* Entrée sur broche A3                  */
5
6 #define PHOTO_TRANSISTOR 3 // Sortie photo transistor sur broche 3 <=> A3
7
8 // Prototypes de fonctions
9 float D_To_A (int V_num);
10
11 /**
12 * Conversion numérique (10 bits) vers analogique
13 float D_To_A (int V_num)
14 {
15     return (float) V_num * 5.0 / 1024.0;
16 }
17 /**
18 void setup ()
19 {
20     Serial.begin (9600);
21 }
22
23 void loop ()
24 {
25     int V_num_A3;
26
27     V_num_A3 = analogRead (PHOTO_TRANSISTOR);
28     Serial.print ("A3 : ");
29     Serial.print (V_num_A3);
30     Serial.print (" A3 : ");
31     Serial.print (D_To_A (V_num_A3));
32     Serial.println ("Volts");
33     delay (1000);
34 }

```

La sensibilité du montage basé sur l'utilisation du photo transistor dépend de la valeur de la résistance qui est utilisée. Selon les conditions d'éclairage, il sera peut être utile de changer la résistance. En reprenant le programme précédent, évaluer l'influence de la valeur de cette résistance.

### 8.3. Gestion des déplacements - Approche 1

我们将考虑, 只要机器人在亮度太低的区域进化, 它就会移动, 一旦亮度变得更加重要, 机器人就会陷入停滞

Il s'agit maintenant de prendre en compte l'information concernant la luminosité ambiante afin de contrôler les déplacements du robot. Nous allons considérer que tant que le robot évolue dans une zone où la luminosité est trop faible il se déplace et dès que cette luminosité devient plus importante, le robot s'immobilise. Dès que la luminosité redévient plus faible et inférieure au seuil défini précédemment, le robot se remet en mouvement. Il est donc nécessaire de caractériser la luminosité qui provoquera l'arrêt du robot par la valeur numérique LUMINOSITE\_SEUIL. En fonction des conditions d'éclairage, définir cette valeur de seuil. Elle pourra ensuite être ajustée. Ecrire le programme Phototransistor\_V2 qui permet d'implémenter ce comportement du robot. Vous pouvez spécifier le type de déplacement que le robot effectue par la séquence de mouvements de votre choix.

### 8.4. Mesure de luminosité - Approche 2

La mesure de luminosité effectuée précédemment s'effectue sur une échelle de valeurs limitées à {0,..., 1023}. D'autre part, cette première approche ne permet pas de considérer de faibles différences d'éclairage au sein d'une même pièce par exemple. Afin d'améliorer la finesse de la mesure de l'éclairage ambiant, nous pouvons utiliser un circuit de type RC série avec le photo

transistor. Dans ce cas, la mesure de luminosité se traduira par la mesure d'une durée. Plus la durée sera faible, et plus la luminosité observée sera importante.

?

#### 8.4.1. Circuit RC : quelques résultats

Soit le circuit RC série ci-dessous (cf. Figure 72).

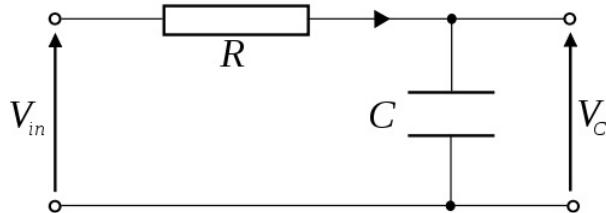


Figure 72 : Circuit RC série

La réponse du circuit à un échelon de tension  $V_{in}$  constant est de la forme :

$$V_c(t) = V_{in} [1 - e^{-t/\tau}]$$

où  $\tau$  désigne la constante du circuit, soit  $\tau = RC$ . Pour  $t = 3\tau$ , alors  $V_c(t) = 0.95 V_{in}$  et pour  $t = 5\tau$  alors  $V_c(t) = 0.993 V_{in}$ .

Lorsque l'on considère la décharge du condensateur C dans la résistance R, l'évolution de la tension  $V_c(t)$  est régie par la relation :

$$V_c(t) = V_{in} e^{-t/\tau}$$

où  $V_{in}$  désigne la tension aux bornes de la capacité C à  $t = 0$ .

#### 8.4.2. Mise en œuvre avec le photo transistor

Le circuit qui permettra d'effectuer une mesure de la luminosité ambiante sur le robot est décrit ci-dessous (cf. Figure 73).

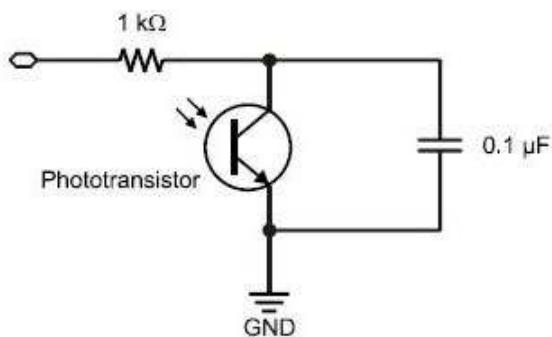


Figure 73 : Circuit RC et photo transistor sur Boe

Il s'appuie sur le circuit RC décrit précédemment, et aussi sur l'utilisation du photo transistor. Le principe d'utilisation du circuit est le suivant :

- Le condensateur de  $0.1\mu F$  est chargé pendant  $T_{Charge}$  secondes ;
- Une fois que le condensateur est suffisamment chargé, alors le décharger
  - \* Attendre que la tension aux bornes du condensateur soit inférieure à  $2.1V$
  - 等到电容量测电压小于2.1v

\* Relever le temps nécessaire pour atteindre cette tension de 2.1V, soit T\_Décharge  
La valeur de T\_Décharge est alors une mesure de la luminosité ambiante.

Ce principe de mesure prend en compte le fait que le photo transistor est plus ou moins passant selon la luminosité à laquelle il est exposé. Plus la luminosité sera élevée, et plus vite le condensateur se déchargera.

#### 8.4.3. Mise en œuvre sur le robot

Il s'agit maintenant de mettre en place 2 circuits indépendants de mesure de la luminosité ambiante, selon le schéma ci-dessous (cf. Figure 74 ).

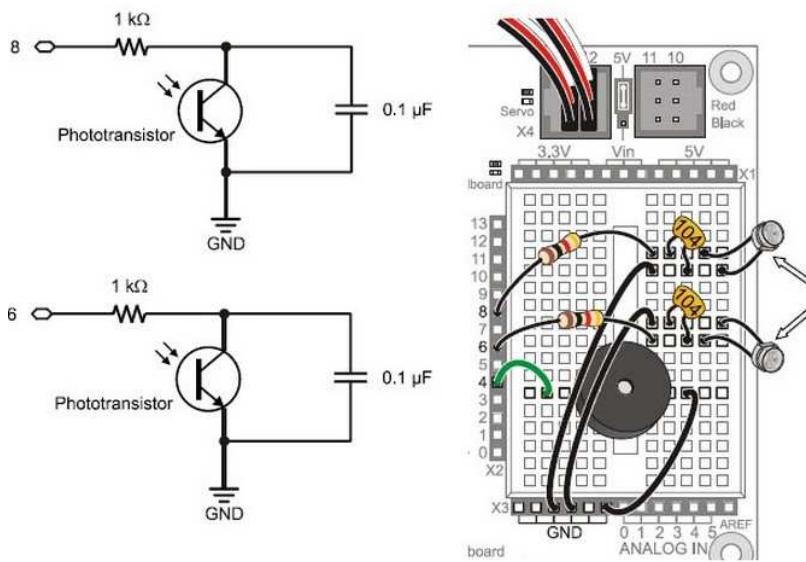


Figure 74 : Câblage des circuits de mesure de la luminosité

Il sera nécessaire d'ajuster l'orientation les deux photo transistors en les inclinant de 45°.

#### 8.4.4. Test des circuits

Le schéma général de test des circuits suit les étapes suivantes :

1. Placer la broche 6 (respectivement 8) en sortie ;
2. Mettre à l'état haut la broche 6 (respectivement 8) à l'état haut, ce qui permet de charger le condensateur ;
3. Attendre une durée T\_Charge afin que le condensateur ai le temps de se charger ;
4. Basculer la broche 6 (respectivement 8) en entrée ;
5. Effectuer une mesure de temps, soit T0 ;
6. Le condensateur se décharge. Attendre que la tension sur la broche 6 (respectivement 8) passe sous le seuil de 2.1V ;
7. Effectuer une mesure de temps, soit T1
8. Calculer  $T_{\text{Décharge}} = T1 - T0$  et constitue une mesure de la luminosité.

Cette mesure de temps impliquera l'utilisation de la fonction *micros ()* dont la description est rappelée ci-dessous (voir aussi <http://arduino.cc/en/Reference/Micros>) :

`unsigned long micros (void) ;`

La fonction `micros()` renvoie, en  $\mu$ s, la durée écoulée depuis le lancement du programme qui s'exécute.

Afin de pouvoir gérer les deux capteurs, on peut s'appuyer sur la fonction `rcTime` décrite ci-dessous :

```
1 long rcTime(int pin)                                // Mesure du temps de décharge du circuit RC
2 {                                                 
3     unsigned long int time;                        
4
5     pinMode(pin, OUTPUT);                         // Configurer la broche en sortie
6     digitalWrite(pin, HIGH);                      // Mettre la sortie à l'état haut
7     delay(1);                                    // Attendre 1 ms
8     pinMode(pin, INPUT);                          // Configurer la broche en entrée
9
10    digitalWrite(pin, LOW);                       // Mesure de T0
11    time = micros();                            // Attendre que la tension soit < à 2.1V
12    while(digitalRead(pin));                     // Calculer T_Décharge = T1 - T0
13    time = micros() - time;                      // Retourner T_Décharge
14
15 }
```

Ecrire alors le programme `Photo_transistor_V3` qui permet d'afficher dans la fenêtre du terminal texte les valeur retournées par les capteurs de luminosité gauche et droit.

## 9. Navigation et capteurs IR

En cours

## 10. Application – Mise en œuvre libre

A partir des ressources utilisées précédemment, vous avez la possibilité de définir et mettre en œuvre un comportement du robot, que vous définirez.

## 11. Annexe

### 11.1. Codage des caractères

// Codage des caractères : table Ascii

[http://fr.wikipedia.org/wiki/American\\_Standard\\_Code\\_for\\_Information\\_Interchange](http://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange)

<http://www.catonmat.net/images/ascii-cheat-sheet.png>

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	'
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	\$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

## 11.2. Opérateurs arithmétiques

Affectation et ...	Symbole	Equivalence	Exemple
Addition	$+=$	$x = x + y$	$x += y$
Soustraction	$-=$	$x = x - y$	$x -= y$
Multiplication	$*=$	$x = x * y$	$x *= y$
Division	$/=$	$x = x / y$	$x /= y$
Modulo	$\% =$	$x = x \% y$	$x \% = y$
Décalage à droite	$>=$	$x = x > y$	$x >= y$
Décalage à gauche	$<=$	$x = x < y$	$x <= y$
Complémentation	$^=$	$x = x ^ y$	$x ^= y$
Et bit à bit	$\&=$	$x = x \& y$	$x \&= y$
Ou bit à bit	$ =$	$x = x   y$	$x  = y$

## 11.3. Opérateurs de comparaison et logiques

Opérateur	Symbol	Description	Syntaxe
Egalité	$==$	Test d'égalité	$x == y$
Inférieur	$<$	Test d'infériorité	$x < y$
Supérieur	$>$	Test de supériorité	$x > y$
Inférieur ou égal	$<=$	Test d'égalité et de supériorité	$x <= y$
Supérieur ou égal	$>=$	Test d'égalité et d'infériorité	$x >= y$

Différent	<b>!=</b>	Test de différence	x <b>!=</b> y
-----------	-----------	--------------------	---------------

Opérateur	Symbol	Description	Syntaxe
et	<b>&amp;&amp;</b>	Les 2 expressions sont elles vraies	Exp1 <b>&amp;&amp;</b> Exp2
ou	<b>  </b>	Une des 2 expressions est elle vraie	Exp1 <b>  </b> Exp2
non	<b>!</b>	L'expression est elle fausse	<b>!</b> Exp

## 11.4. Spectre de la lumière

Décomposition de la lumière et spectre associé en fonction de la longueur d'onde.

