



# Samedis bénévoles spécial Arduino

## Workshop n° 1

### FICHE F4 - COMMANDER DES SERVOMOTEURS CLASSIQUES ET A ROTATION CONTINUE

Contact : Paula Bruzzone Rouget

Secrétariat général :

6, rue Emmanuel Pastré 91000 Evry - Tél. : 01 64 97 82 34 - [idf@planete-sciences.org](mailto:idf@planete-sciences.org)

## SOMMAIRE

1. Les servomoteurs .....	3
2. Piloter un servomoteur avec l'Arduino .....	3
3. Piloter un servomoteur classique .....	5
3.1. Principe .....	5
3.2. Application : pas à pas .....	5
3.2.1. Montage .....	5
3.2.2. Codage .....	7
3.3. Autres exemples .....	9
4. Piloter un servomoteur à rotation continue .....	10
4.1. Principe .....	10
4.2. Application : pas à pas .....	11
4.2.1. Montage .....	11
4.2.2. Codage .....	11
5. A-t-on vraiment besoin d'un Arduino pour faire ça? .....	14

# 1. Les servomoteurs

Les servomoteurs sont des actionneurs. Très utilisés en modélisme et dans l'industrie, ils ont comme caractéristique principale leur « couple », c'est-à-dire la force de rotation qu'ils peuvent exercer. Plus un servomoteur aura de couple et plus il pourra actionner des « membres » lourds comme déplacer un bras qui porte une charge.

Pour la robotique de loisirs, les servomoteurs ont en général peu de couple et sont de taille réduite, bien adaptée à un encombrement minimal et à une énergie disponible limitée.



Les servomoteurs sont pilotés par un fil de commande et alimentés par deux autres fils. Habituellement, ces 3 fils sont rassemblés dans une prise au format standard.



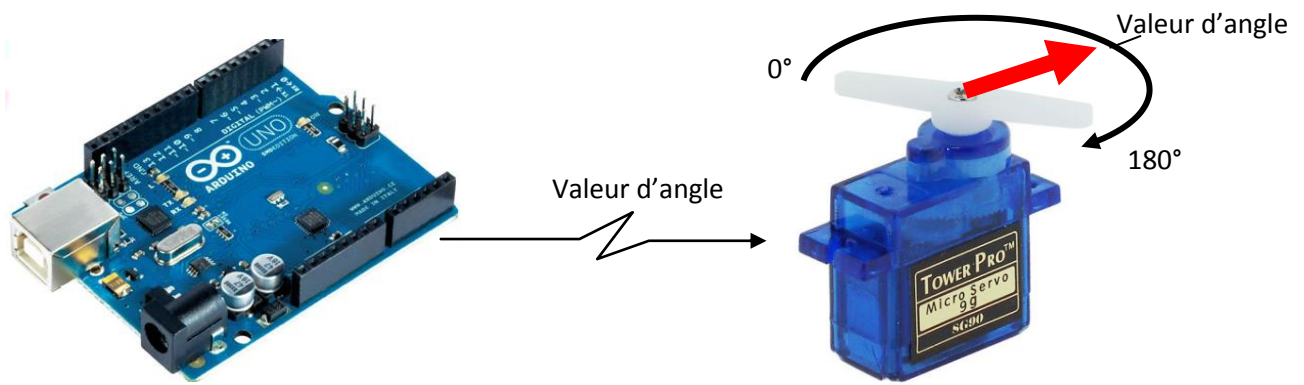
Un fil rouge est relié à l'alimentation positive (+5 ou +6 V selon le servo), le fil noir est relié à la masse (GND) et le fil jaune est utilisé pour la commande.

Il y aurait beaucoup à dire sur le fonctionnement d'un servomoteur, ses composants, son moteur et le petit potentiomètre qui permet de connaître sa position mais cette fiche va droit au but et se limitera à son utilisation avec l'Arduino.

## 2. Piloter un servomoteur avec l'Arduino

Le mode de commande d'un servomoteur est standardisé : on envoie sur son fil de commande une impulsion dont la durée correspond à l'angle désiré. Historiquement, cette impulsion était délivrée par un circuit oscillateur. Le circuit intégré NE555 est un exemple vedette du circuit utilisé.

Avec la programmation de l'Arduino, ce circuit n'est plus nécessaire. Une bibliothèque (library) dédiée, la bibliothèque « servo », permet de piloter un servomoteur en lui transmettant simplement l'angle sur lequel il souhaite se positionner.



Ce qui permet de réaliser simplement des mouvements qui peuvent être complexes et de les automatiser.



Les connaissances nécessaires au pilotage d'un servomoteur sont :

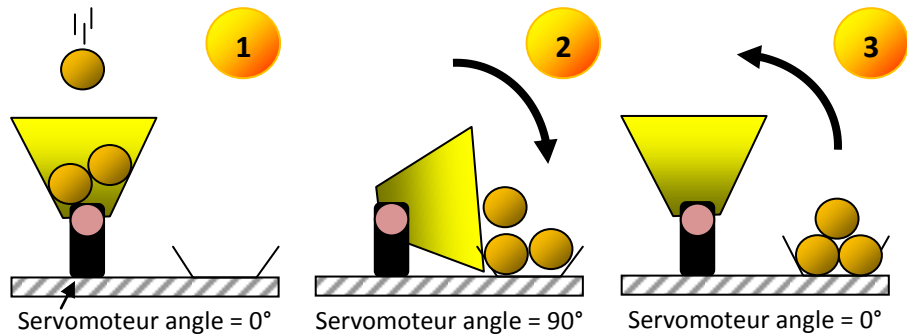
<code>#include &lt;Servo.h&gt;</code>	L'utilisation de la librairie <code>servo.h</code> qui prend en charge la communication de bas niveau entre l'Arduino et le servomoteur. Les commandes précédées par <code>#</code> sont des ordres particuliers donnés au compilateur
<code>Servo myservo;</code>	La création d'un objet de type <code>Servo</code> , appelé ici <code>myservo</code>
<code>myservo.attach(9);</code>	Le fil de commande de ce servo sera connecté au PIN 9 et l'objet <code>myservo</code> pilotera ce PIN
<code>myservo.write(90);</code> <code>myservo.write(Angle);</code>	Demander au servomoteur de se déplacer à l'angle désiré, soit de façon absolue en lui indiquant une valeur entière (90°) dans le premier cas, soit en lui passant le contenu d'une variable ( <code>Angle</code> ) compris entre 0 et 180, ce qui peut être utile par exemple pour donner une progressivité au déplacement en faisant varier l'angle d'un pas fixe (quelques degrés) par une boucle
<code>myservo.read(Angle);</code>	Pour lire la valeur de l'angle du servomoteur

# 3. Piloter un servomoteur classique

## 3.1. Principe

Prenons un exemple simple pour commencer : le vidage d'un bac ou d'une benne.

Un bac est monté sur un axe solidaire de l'axe d'un servomoteur. Ce bac se remplit lentement et on souhaite le vider en appuyant sur un bouton



Dans cet exemple, nous avons passé au servomoteur les commandes suivantes :

1. Angle = 90°, le servomoteur a pivoté à angle droit vers la droite et vidé les balles
2. Angle = 0°, le servomoteur a pivoté en sens inverse et retrouvé sa position de départ

Entre les étapes 1 et 2, nous aurions pu ajouter un petit temps d'attente ou de retard pour laisser le temps au bac de se vider complètement.

L'algorithme de cette action est donc assez simple :

- Début
  - Lorsque le bouton de vidage de bac est appuyé
    - Alors mettre le servo à l'angle = 90°
    - Attendre 1 seconde
    - Mettre le servomoteur à l'angle = 0°
- Fin

Bien évidemment, au lieu d'un bouton de vidage, nous aurions pu positionner un capteur qui aurait détecté que le bac était plein.

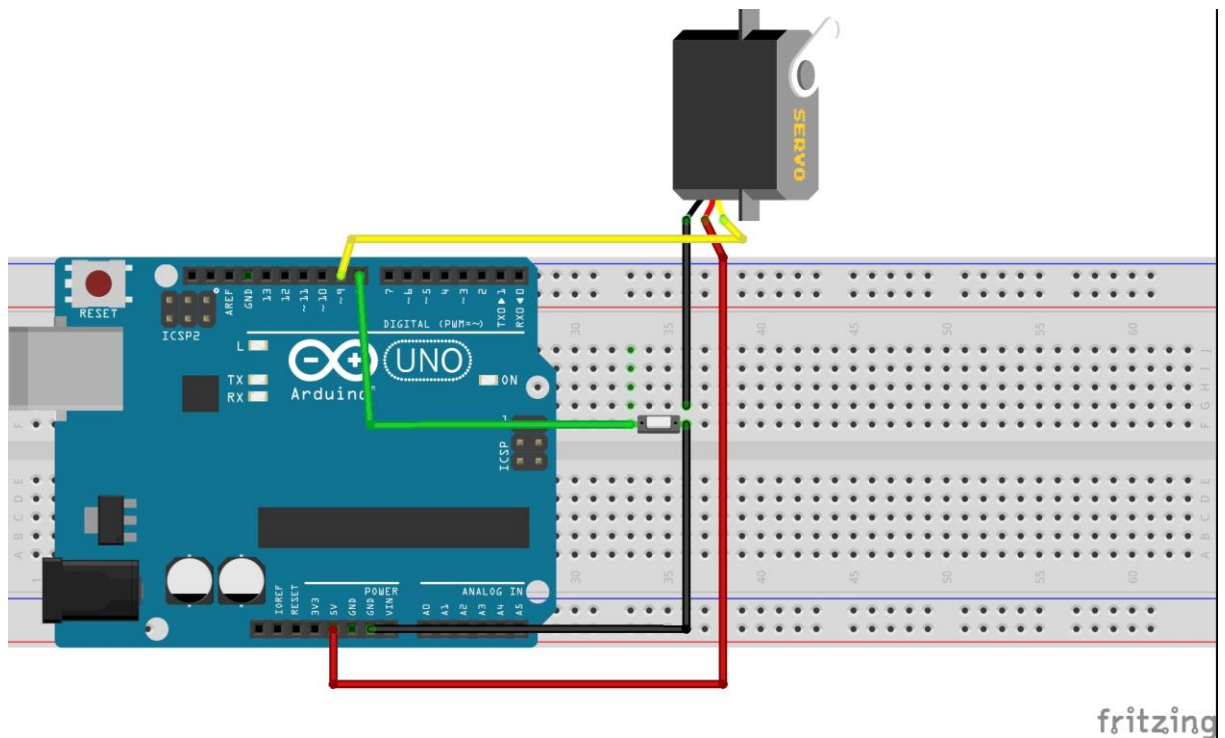
## 3.2. Application : pas à pas

### 3.2.1. Montage

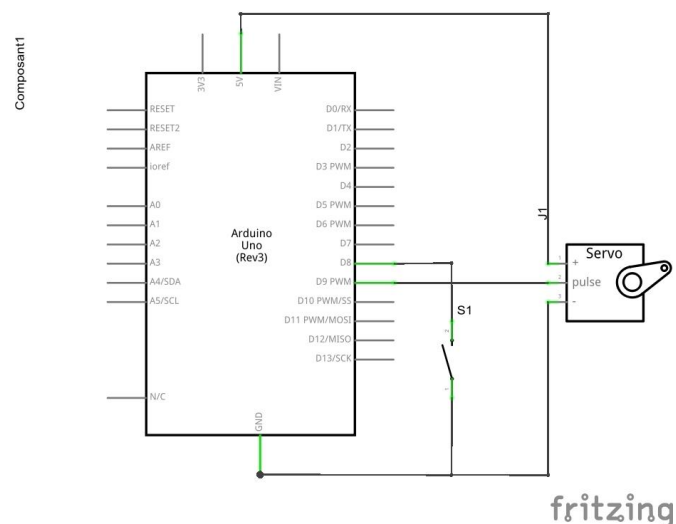
Matériel nécessaire : 1 plaque d'expérimentation, 1 module Arduino, 1 servomoteur « classique », un bouton poussoir ou un connecteur mâle-mâle.

Le module Arduino peut être alimenté par son câble USB et alimenter un servomoteur sans dommage. Au-delà, il est prudent de calculer la consommation totale car le port USB ne pourra délivrer que 500mA au maximum. Une alimentation séparée et correctement dimensionnée est conseillée.

1. Connecter le fil rouge du servo au +5V de l'Arduino et le fil noir sur la masse GND
2. Connecter le fil jaune du servo sur le PIN 9
3. Connecter un fil du bouton poussoir sur le PIN 8 et l'autre sur GND

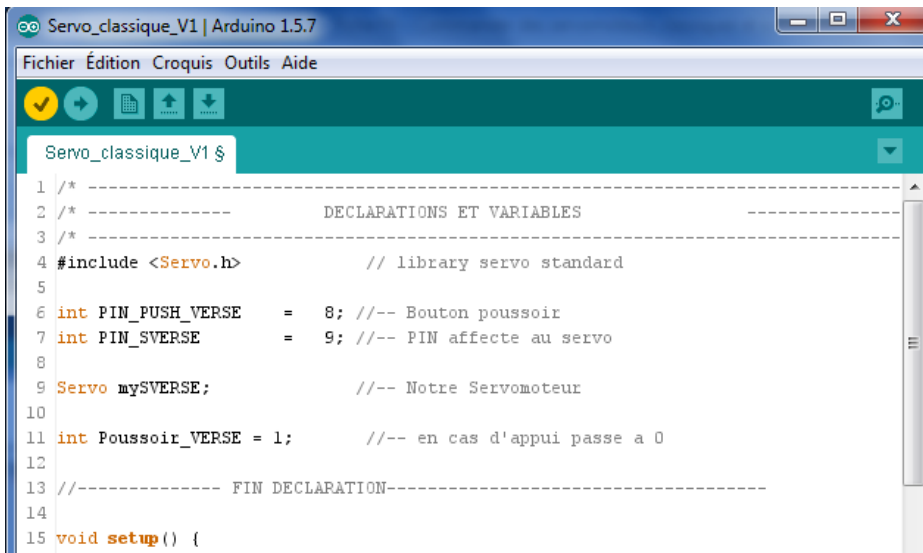


Le montage sur la plaque d'expérimentation devrait ressembler à ça.



## 3.2.2. Codage

1. Lancer l'IDE de l'Arduino et créer un nouveau croquis ou ouvrir un nouvel onglet
2. Créer les variables en leur donnant un nom, un type et une valeur



```
1 /* -----  
2 /* -----      DECLARATIONS ET VARIABLES      -----  
3 /* -----  
4 #include <Servo.h>           // library servo standard  
5  
6 int PIN_PUSH_VERSE    =  8; //-- Bouton poussoir  
7 int PIN_SVERSE       =  9; //-- PIN affecte au servo  
8  
9 Servo mySVERSE;         //-- Notre Servomoteur  
10  
11 int Poussoir_VERSE = 1;   //-- en cas d'appui passe a 0  
12  
13 //----- FIN DECLARATION-----  
14  
15 void setup() {
```

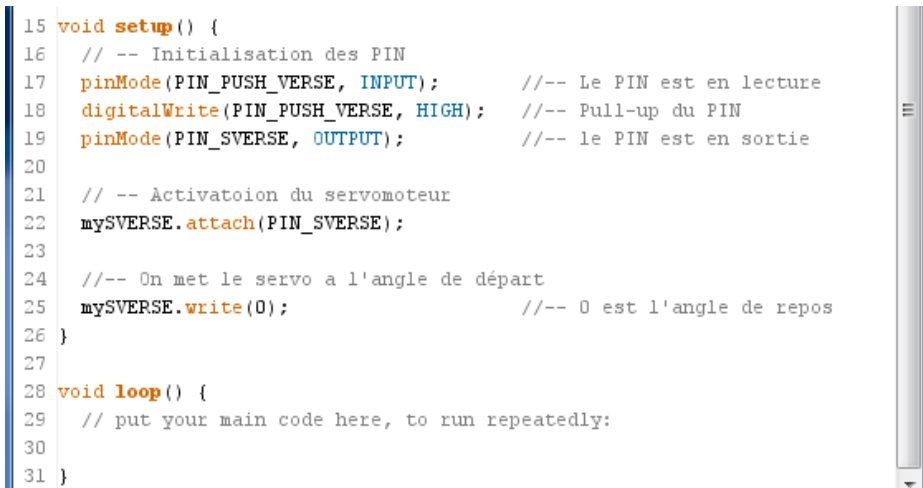
Avant le setup, il faut déclarer les variables que l'on va utiliser, soit :

- La librairie servo
- Une variable servo : mySVERSE
- Une variable PIN\_SVERSE pour le PIN affecté au servo
- Deux variables pour le bouton poussoir : une pour le PIN et une pour tester si le poussoir est pressé

Utiliser une variable pour la valeur du PIN facilite la mise au point du code et son utilisation dans des modules différents : il suffira de personnaliser le code en affectant les numéros de PIN aux variables du code dans la partie déclaration.

Notez les commentaires abondants qui éclaireront sur ce que fait le code lors de relectures futures ou d'une analyse par quelqu'un d'autres que le développeur.

### 3. Initialiser les variables et le servo dans le setup



```
15 void setup() {  
16 // -- Initialisation des PIN  
17 pinMode(PIN_PUSH_VERSE, INPUT); //-- Le PIN est en lecture  
18 digitalWrite(PIN_PUSH_VERSE, HIGH); //-- Pull-up du PIN  
19 pinMode(PIN_SVERSE, OUTPUT); //-- le PIN est en sortie  
20  
21 // -- Activation du servomoteur  
22 mySVERSE.attach(PIN_SVERSE);  
23  
24 //-- On met le servo a l'angle de départ  
25 mySVERSE.write(0); //-- 0 est l'angle de repos  
26 }  
27  
28 void loop() {  
29 // put your main code here, to run repeatedly:  
30  
31 }
```

Rappel : PinMode permet de définir le sens d'utilisation d'un PIN : en entrée (INPUT) ou en sortie (OUTPUT)

**Le « pull-up » donne une valeur explicite à l'état du PIN qui peut être aléatoire sinon du fait d'interférences électriques**

L'activation du servo est réalisée par l'instruction **attach**

#### 4. Coder l'algorithme dans la boucle (loop())

```
30 /*-----  
31 /*-----          BOUCLE PRINCIPALE          -----  
32 /*-----  
33 void loop() {  
34  
35   //-- commande le vidage dubac  
36   Poussoir_VERSE = digitalRead(PIN_PUSH_VERSE); //-- Lecture de l'etat  
37   if (Poussoir_VERSE == LOW) {           //-- Test de sa valeur  
38     //-- Le poussoir est appuye donc il faut verser!  
39     mySERVE.write(90); // Rotation de 90 degres pour liberer les pop corns  
40     delay(2000);      // Attendre 2 secondes (2000ms)  
41     mySERVE.write(0); // Retour a la position de depart  
42  
43   }  
44   //------- FIN DE BOUCLE -----
```

Lorsque le bouton de vidage de bac est appuyé :

- A. Alors mettre le servo à l'angle = 90°
- B. Attendre 1 seconde
- C. Mettre le servomoteur à l'angle = 0°

Initialement le PIN du poussoir est l'état haut. Lorsqu'on le presse, on relie la masse (GND) au PIN et il passe à l'état bas.

Avant de tester sa valeur, il faut la lire, c'est pourquoi on affecte à la Poussoir\_VERSE la valeur lue par `digitalRead` sur le PIN correspondant.

Ensuite, on teste cette valeur en la comparant à `LOW` et dans ce cas on exécute les 3 instructions du bloc. Dans le cas contraire, on ne fait rien.

#### 5. Il ne reste plus qu'à relire et compiler :



```
Servo_classique_V1 $
1 /* -----
2 /* -----          DECLARATIONS ET VARIABLES          -----
3 /* -----
4 #include <Servo.h>           // library servo standard
5
6 int PIN_PUSH_VERSE      =  8; //-- Bouton poussoir
7 int PIN_SVERSE         =  9; //-- PIN affecte au servo
8
9 Servo mySVERSE;         //-- Notre Servomoteur
10
11 int Poussoir_VERSE = 1;    //-- en cas d'appui passe a 0
12
13 //-----          FIN DECLARATION-----
14
15 void setup() {
16   // -- Initialisation des PIN
17   pinMode(PIN_PUSH_VERSE, INPUT);    //-- Le PIN est en lecture
18   digitalWrite(PIN_PUSH_VERSE, HIGH);  //-- Pull-up du PIN
19   pinMode(PIN_SVERSE, OUTPUT);      //-- le PIN est en sortie
20
Compilation terminée.
Le croquis utilise 2 654 octets (8%) de l'espace de stockage de programmes. Le
maximum est de 30 720 octets.
Les variables globales utilisent 60 octets (2%) de mémoire dynamique, ce qui laisse
1 988 octets pour les variables locales. Le maximum est de 2 048 octets.
24 Arduino Nano, ATmega328 on COM4
```

6. Puis à téléverser et à tester !

### 3.3. Autres exemples

Dans les exemples livrés avec l'IDE, vous trouverez deux croquis intéressants :

- 1) Sweep, qui montre comment faire un balayage du servo dans les deux sens avec une boucle **for** :

```
for(pos = 0; pos <= 180; pos += 1) // deplace le servo de 0 a 180 degres
{
  myservo.write(pos);              // le servo se positionne a la valeur de la variable pos
  delay(15);                       // laisse 15ms de delai pour que le servo aille a la
  // nouvelle position
}
```

Le balayage du croquis sweep peut être réutilisé dans deux situations :

- la recherche d'un déplacement progressif du servo, selon un pas fixe, en degrés ;

```
angleCible = 75;
for(pos = 0; pos <= angleCible; pos += 1) // deplace le servo de 0 a un angle cible
{
    // par pas de 1 degre
    myservo.write(pos); // le servo se positionne a la valeur de la variable pos
    delay(15); // laisse 15ms de delai pour que le servo aille a la
                // nouvelle position
}
```

- Le balayage de la zone située devant le robot par un sonar ou un capteur-émetteur infrarouge, pour détecter des obstacle.

2) Knob, qui permet de contrôler la position d'un servomoteur à l'aide d'un potentiomètre.

```
void loop()
{
    val = analogRead(potpin); // lit la valeur du potentiomere connecte sur le PIN potpin
                              // cette valeur est comprise entre 0 et 1023 et stockee dans val
    val = map(val, 0, 1023, 0, 180); // la fonction map remplace cette valeur entre 0 et 180)
    myservo.write(val); // le servo se positionne sur cete valeur d'angle
    delay(15); // on laisse le temps au servo d'aller a cette valeur
}
```

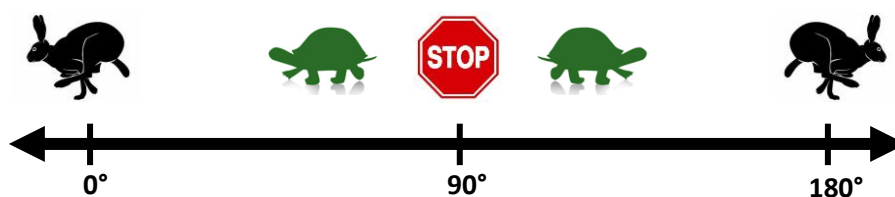
Le croquis knob illustre la commande d'un servo avec un potentiomètre qui peut être celui d'un joystick, ce qui est très pratique pour la commande d'un bras robotisé, par exemple. Une autre application est dans le réglage d'un servomoteur à rotation continue, objet de la section suivante...

## 4. Piloter un servomoteur à rotation continue

### 4.1. Principe

Un servomoteur à rotation continue est une évolution du servomoteur classique et son principe est similaire : déclaration des variables et envoi d'un angle par l'instruction `myServo.write(angle)`.

Par contre, la valeur de l'angle a une grande importance sur le déplacement : elle permet de régler le sens et la vitesse de déplacement :



Il faut cependant relativiser : à pleine vitesse, le déplacement reste encore assez lent. Mais c'est suffisant pour motoriser un petit robot autonome ou un actionneur.

Par exemple, pour collecter des balles de ping-pong qui sortent d'un distributeur, on peut déclencher par l'appui sur un bouton la rotation d'un servomoteur muni d'une palette qui les fera tomber une par une dans un bac...

## 4.2. Application : pas à pas

### 4.2.1. Montage

Le montage est identique à celui du 3.2.1 : il n'y a pas de différence matérielle avec un servomoteur classique du point de vue du pilotage.

L'appui sur le bouton doit déclencher la rotation continue du servomoteur vers la droite, à pleine vitesse, et un relâchement du bouton doit entraîner son arrêt.

### 4.2.2. Codage

#### 1. Rechercher la valeur du point mort du servomoteur

Il s'agit de gagner un peu de temps en recherchant la vraie valeur du point mort du servomoteur. En effet, les contraintes de fabrication et surtout de coût font que ce point mort ne se situe que rarement à la valeur exacte de 90°. Le servomoteur tourne alors qu'il est supposé être à l'arrêt, ce qui pose problème (c'est même agaçant !).

Deux solutions sont envisageables :

- 3) Rechercher la vraie valeur de point mort en observant son arrêt et en relevant la valeur de l'angle correspondant ;
- 4) Ajuster la valeur de l'angle par un décalage réglé par un potentiomètre.

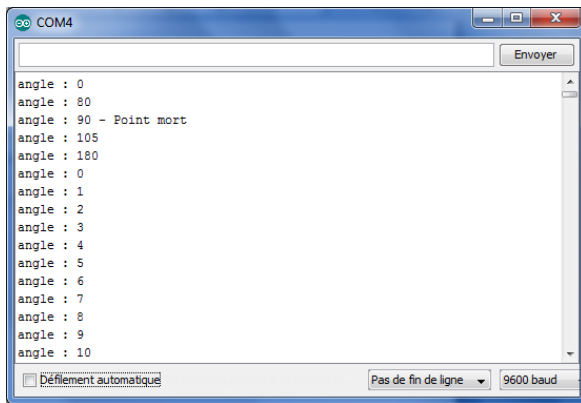
La première solution offre l'avantage de tenir compte définitivement du décalage et de ne pas mobiliser un PIN de l'Arduino pour le potentiomètre. Nous vous conseillons de noter la vraie valeur du point mort sur le servomoteur une fois trouvée.

Le petit programme ci-dessous est dérivé du croquis « sweep » et permet de concrétiser :

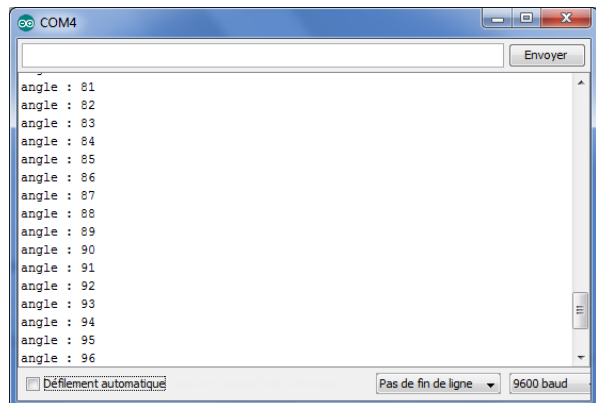
```

1 #include <Servo.h>
2
3 Servo myservo; //-- On cree un objet de type servo
4
5 int pos = 0; //-- on definit une variable de stockage de la position du servo
6
7 void setup()
8 {
9     myservo.attach(7); //-- l'objet servo sera attaché au PIN 7
10    Serial.begin(9600) ; //-- on active le moniteur serie pour lire les valeurs de pos
11 }
12
13 void loop()
14 {
15     //-- Tests de 5 valeurs d angle
16     myservo.write(0);
17     Serial.println("angle : 0");
18     delay(5000);
19     myservo.write(80);
20     Serial.println("angle : 80");
21     delay(5000);
22     myservo.write(90);
23     Serial.println("angle : 90 - Point mort");
24     delay(10000);
25     myservo.write(105);
26     Serial.println("angle : 105");
27     delay(5000);
28     myservo.write(180);
29     Serial.println("angle : 180");
30     delay(5000);
31
32     for(pos = 0; pos <= 180; pos += 1) //-- On parcourt toute la gamme de vitesse du servà
33     { //-- de 0 a 180 degres avec un pas de 1 degre
34         myservo.write(pos); //-- le servo se positionne sur la valeur d angle pos
35         Serial.print("angle : "); //-- on ecrit sa valeur sur le moniteur serie
36         Serial.println(pos);
37         delay(500); // on laisse 1/2s pour changer de position |
38     }
39
40 }

```



Comme on peut le constater (au moins sur le servomoteur utilisé pour ce test, avec un angle de 90° réputé être le point mort, il tourne encore



En observant la valeur qui correspond à l'arrêt complet du servomoteur, on peut donc en déduire la vraie valeur du point mort, soit 92°

La valeur du point mort sera stockée dans une constante (une variable qui ne varie pas !). Comme le servomoteur de collecte sera baptisé `SCOLLECTE` dans notre code, nous appellerons cette variable `mySCOLLECTE_Repos` et nous la déclarerons de la manière suivante :

```
int mySCOLLECTE_Repos = 92;
```

Pour stopper le servomoteur, il suffira de lui envoyer cet angle par l'instruction :

```
mySCOLLECTE.write(mySCOLLECTE_Repos);
```

## 2. Définir les variables, leur type et leur affecter une valeur

De façon classique, les variables suivantes sont créées :

La librairie servo.h	<code>#include &lt;Servo.h&gt;</code>
L'affectation du PIN 9 au bouton poussoir	<code>int PIN_PUSH_COLLECTE = 9;</code>
L'affectation du PIN 7 au servomoteur	<code>int PIN_SCOLLECTE = 7;</code>
La création d'un objet servo	<code>Servo mySCOLLECTE;</code>
La création de deux constantes : une pour le point mort et l'autre pour la vitesse max à droite	<code>int mySCOLLECTE_Repos = 92;</code> <code>int mySCOLLECTE_Action = 180;</code>
Une variable pour le bouton poussoir	<code>int Poussoir_COLLECTE = 1;</code>

## 3. Initialiser dans la fonction `setup()`

Ces initialisations ne sont pas fastidieuses mais sécurisent le comportement du robot.

Le PIN utilisé par le poussoir est déclaré en entrée et initialisé à <code>HIGH</code> par un pull-up	<code>pinMode(PIN_PUSH_COLLECTE, INPUT);</code> <code>digitalWrite(PIN_PUSH_COLLECTE, HIGH);</code>
Le servomoteur est attaché au PIN déclaré en constante	<code>mySCOLLECTE.attach(PIN_SCOLLECTE);</code>
Ce PIN est déclaré en sortie	<code>pinMode(PIN_SCOLLECTE, OUTPUT);</code>
Le servomoteur est mis au point mort	<code>mySCOLLECTE.write(mySCOLLECTE_Repos);</code>

#### 4. Détecter l'appui sur le bouton et déclencher la rotation

Lors d'un appui sur le poussoir, le PIN correspondant va passer à l'état **LOW** puisqu'il sera relié à la masse (**GND**). Il suffit de tester cette condition à chaque itération de la boucle, après avoir lu la valeur du poussoir soit

```
Poussoir_COLLECTE = digitalRead(PIN_PUSH_COLLECTE);  
if (Poussoir_COLLECTE == LOW) {
```

Ensuite, soit la valeur est resté à **HIGH**, ce qui signifie que le poussoir n'a pas été actionné, soit elle est passée à **LOW**, et dans ce cas le servomoteur doit se mettre à tourner.

```
void loop() {  
  
  //-- commande la rotation du collecteur de pop-corn  
  Poussoir_COLLECTE = digitalRead(PIN_PUSH_COLLECTE);  
  if (Poussoir_COLLECTE == LOW) {  
    mySCOLLECTE.write(mySCOLLECTE_Action); ;  
  }  
  else  
  {  
    mySCOLLECTE.write(mySCOLLECTE_Repos);  
  }  
}
```

Donc :

- 5) On lit la valeur du poussoir
- 6) Si elle est passée à **LOW**, le servomoteur reçoit la valeur d'angle qui permet sa rotation
- 7) Sinon, il reçoit la valeur du point mort

## 5. A-t-on vraiment besoin d'un Arduino pour faire ça?

Ces exemples sont très simples et seraient réalisés plus facilement avec un peu d'électronique. On peut s'interroger sur le nombre de variables à déclarer et la taille du code qui sont nécessaires pour faire tourner un seul servomoteur.

Mais dans la plupart des applications robotiques, plusieurs servomoteurs et capteurs sont utilisés et l'électronique se complique tout en restant dans une logique cablée. Un Arduino Uno peut piloter jusqu'à 12 servomoteurs et sa programmation permet de gérer simplement des capteurs plus subtils que des boutons poussoirs. Encore qu'il reste à dire sur la gestion des poussoirs et notamment sur la suppression de l'effet rebond qui peut induire un comportement inattendu... Une fiche est dédiée à ce sujet.