Arduino, réalisation pratique : Fabriquer un objet connectée

Service du SCRIPT

Librement inspiré du mooc : Fabriquer un objet connecté, à partir du travail effectué par Maxime CASTELLI, Laurent TOUTAIN, Laurent MATTLÉ, Baptiste GAULTIER, Emmanuel GOUDOT, Simon LANDRAULT...

Rédigé par Jacques TOLA, programmation faite avec l’aide précieuse de Cédric DEVILLERS .

Mis à jour en Mai 2017

I] Introduction

II] Installation du logiciel Arduino, exemple sous Ubuntu

III] Réalisation de la partie électronique

IV] Programmes sur le logiciel Arduino

V] Installation du logiciel «Processing»

I] Introduction

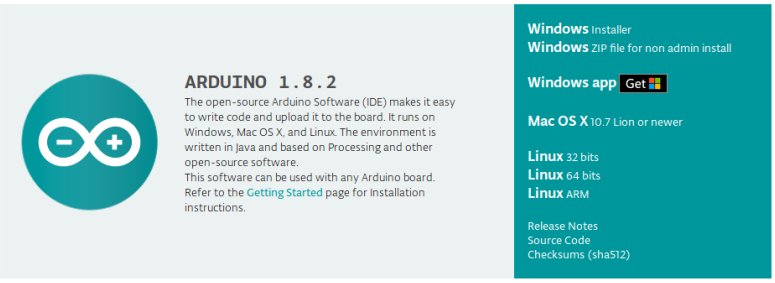
Ce document a pour but d'aider les utilisateurs de notre Fablab à trouver un moyen amusant, même si cela leur demandera un certain investissement personnel, de se familiariser avec les outils mis à leur disposition. Ceci au travers d'une réalisation amusante et innovante car faisant appel à la technologies des objets connectée, et à l'électronique des circuits spécialisés comme l'**ESP8266**. Qui est un circuit qui se programme comme un Arduino, et qui possède un module **wi-fi**.

L'utilisateur devra utiliser entre autres la **paillasse électronique** pour la modification du servomoteur, l'**imprimante 3d**, pour l'impression du boîtier, du bras du montage, et de la roue crantée qui donne du mouvement au bras.

A mon sens il n'est pas nécessaire de lire attentivement ce document avant de se lancer dans la réalisation du projet, car plusieurs chose sont d'avances disponibles comme les fichiers pour l'impression, les programmes, même si l'utilisateur, doit réaliser ses propres programmes, c'est le but de cette initiation. Néanmoins parcourez le au moins au début, pour ensuite le lire plus attentivement.

II] Installation du logiciel Arduino, exemple sous Ubuntu

Télécharger le logiciel Arduino sur le site https://www.arduino.cc/en/Main/Software la dernière version à la date ou est fait ce document est la **1.8.2** :



Choisir celle correspondant à votre système d'exploitation . Dans notre cas nous allons prendre l'exemple pour le système Ubuntu sur un PC 64 bits. Donc on clique sur la version Linux 64 bits.

On télécharge une archive compressée qui se nomme : **arduino-1.8.2-linux64.tar.xz** . Si le fichier est sur le **bureau**, on le décompresse dans un dossier nommé **arduino-1.8.2** , qui se situera, dans notre cas sur le Bureau de l'utilisateur, via l'instruction : **tar Jxvf ~/Bureau/arduino-1.8.2-linux64.tar.xz**

Une fois le dossier créé on se met à l'intérieur via l'instruction **cd** et on lance l'installation du logiciel. Il n'est pas nécessaire d'être root ou d'utiliser sudo les droits de l'utilisateur courant suffisent pour les étapes décrites. Sachant que le logiciel sera installé uniquement pour lui, donc on fait :

cd ~/Bureau/arduino-1.8.2/
~/Bureau/arduino-1.8.2$ ./install.sh

Si l'installation ne s'effectue pas et signale des erreurs. c'est que le bug qui à été constatées sur cette version du logiciel à la date ou j'écris ce document n'a pas été réparée, à la date ou vous l'utiliser.

Il faut pour y remédier, modifier la ligne numéro neuf du fichier **install.sh** (que l'on peut ouvrir via un éditeur de texte comme nano, gedit, vim, et autres…)

si la ligne en question se présente de la forme suivante :

RESOURCE_NAME=cc.arduino.arduinoide

Alors il faut l 'effacer (ou la commenter avec un # au début de la ligne), et la remplacer par la ligne suivante :

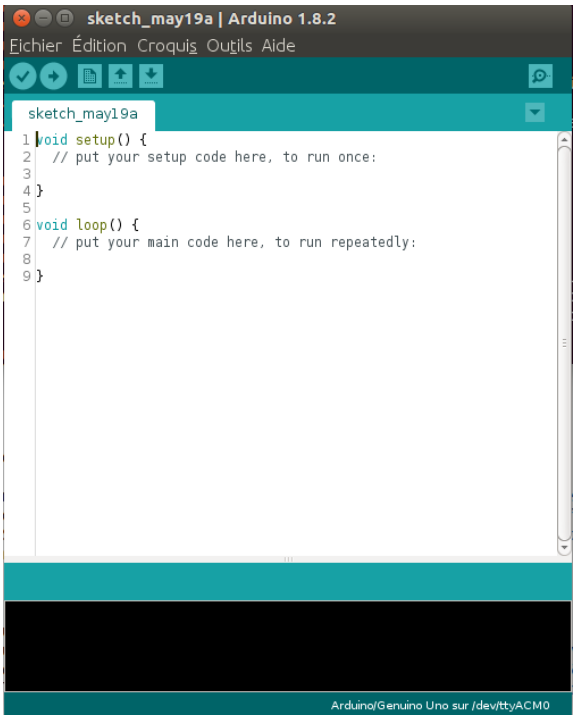
RESOURCE_NAME=arduino-arduinoide

Une fois ce changement, (qui une fois de plus ne concerne que Linux car je n'ai pas tester les autres système) est fait, on peut relancer l'installation via l'instruction :

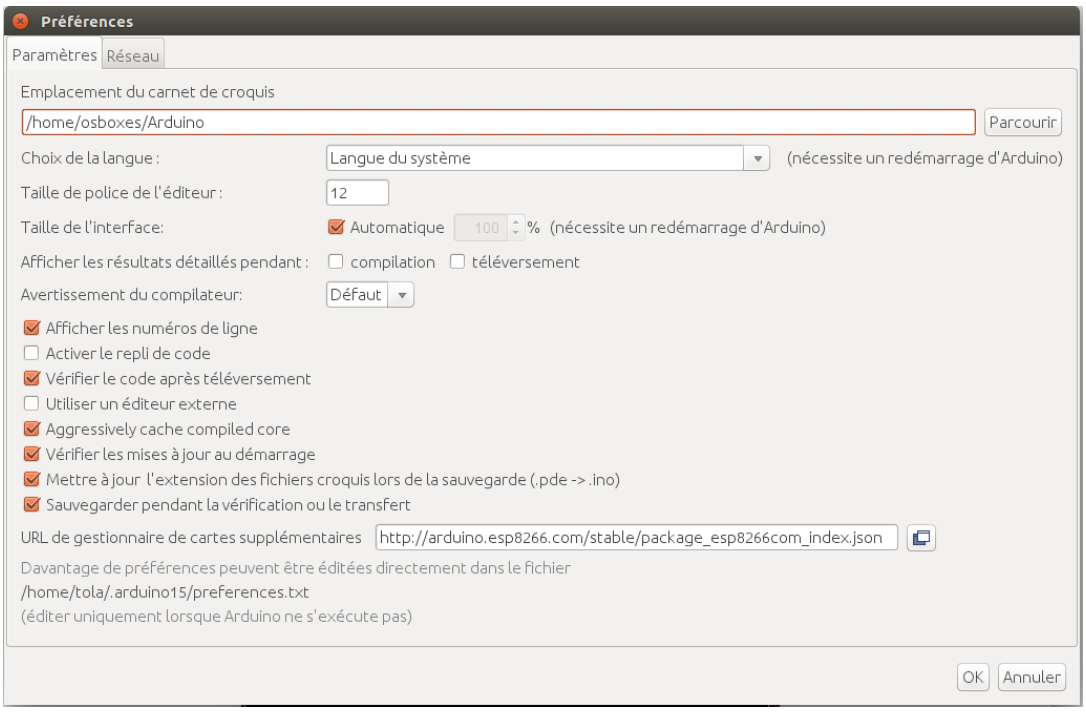
./install.sh

Cela va créer une raccourci sur le bureau ainsi que dans la barre de lancement de l'utilisateur courant.

Une fois le logiciel installé on peut le lancer via l'instruction :
~/Bureau/arduino-1.8.2/arduino
ou en étant dans le répertoire :
./arduino



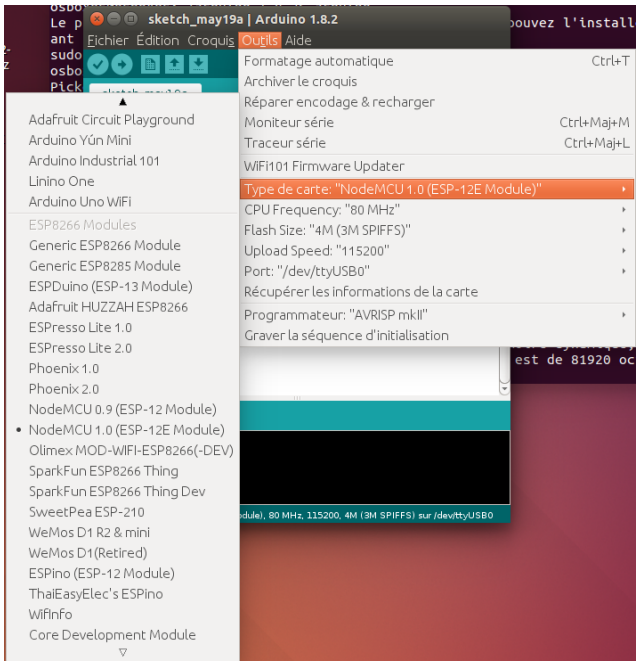
on navigue dan **Fichier->Préférences** :



dans la fenêtre "URL de gestionnaire de cartes supplémentaires" on tape :
http://arduino.esp8266.com/stable/package_esp8266com_index.json

On clique sur "OK"

On va ensuite dans **Outils->Types de carte->Gestionnaire de carte** :

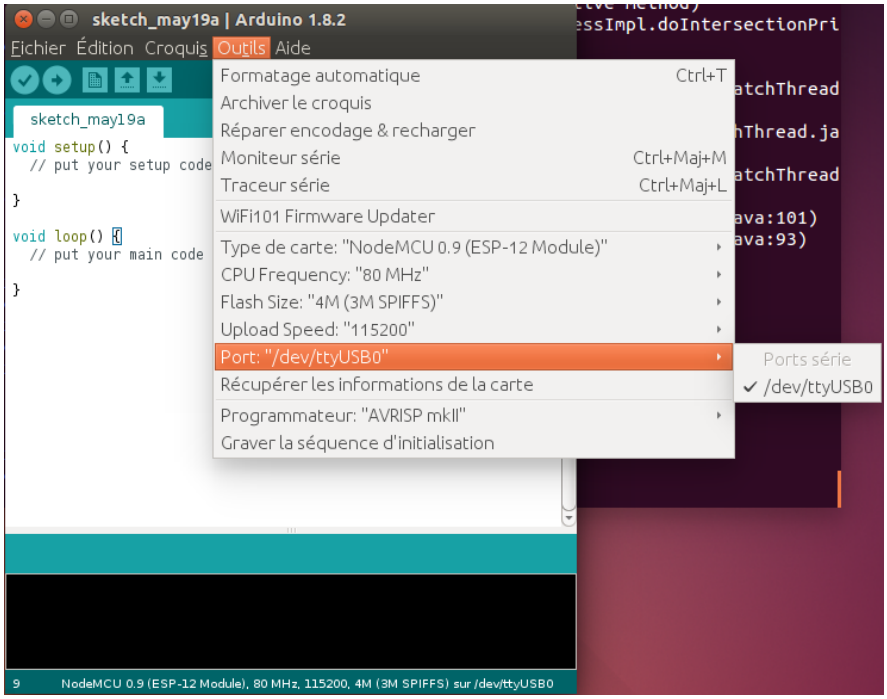


Une liste des softs disponibles s'affiche :

On clique sur la partie qui nous intéresse : "**esp8266 by ESP8266 Community**" et on clique sur "Installation". Ce qui entraînera le téléchargement du soft propre à cette carte .

Une fois que c'est terminé, on va dans **Outils->Type de carte** :
et dans la longue liste de carte que l'on a, on choisi, vers la bas "**NodeMCU 1,0 (ESP12EModule)**" :

Il faut s'assurer que le bon port série est activé via l'onglet : **Outils->Port** . Pour une distribution Ubuntu et plusieurs distribution Linux, ce sera : **/dev/ttyS0** ou **/dev/ttyUSB0…**



Les fichier qui sont utilisés par le logiciel portent l'extension "**.ino**" . Il y a des fichiers d'exemple avec lesquels on peut s'entraîner, en ayant au préalable branché le circuit comme nous le verrons plus loin. Ils sont dans **Fichier->Exemples**.

III] Réalisation de la partie électronique

Matériel Nécessaire pour 1 seul kit :

Pour 1 montage nelson (page 12 , 5 matériels à commander) :

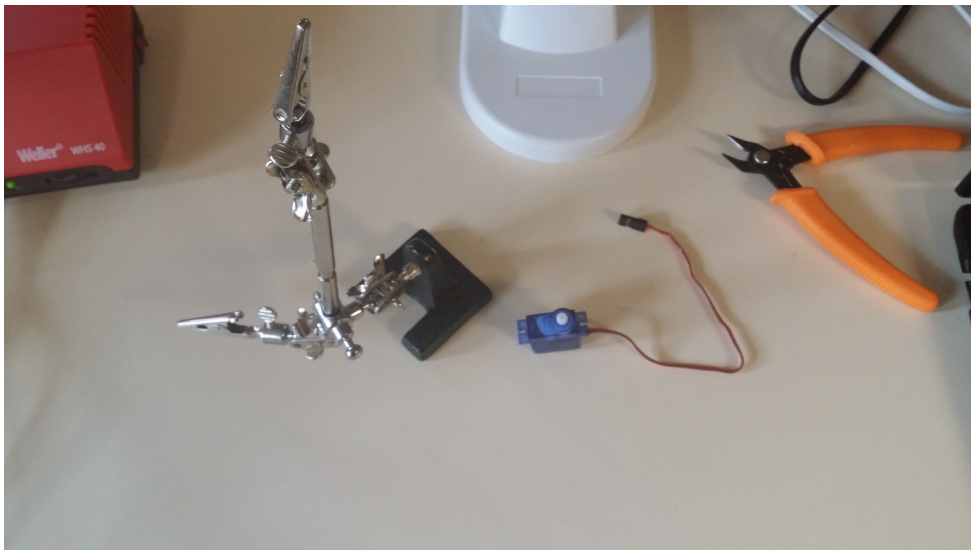
_Module **NodeMCU ESP12E** Amica (ou équivalent) ×1
_Mini platine de prototypage (170 trous 4,5×3,5 cm) ×1
_Câbles de prototypages (mâle/mâle) ×1
_Mini **servomoteur** à retour d'information (Adafruit/Batan S1123) ×1
_Câble **USB ↔ micro USB** ≥1m (ou équivalent) ×1
_Matériel optionnel
_Pièces mécaniques : Nelson imprimées en 3D (les fichiers .stl sont téléchargeables à l'adresse : http://www.thingiverse.com/thing:1571631 , il sont aussi sur le PC connecté à l'imprimante 3D)
_Adaptateur secteur 220V vers USB (type smartphone)

Un fil est soudé par l'utilisateur sur la partie électronique du servomoteur afin d'obtenir le retour d'information. Le câblage est obtenu comme suit :

Je conseille d'utiliser la loupe binoculaire du fablab, et les pinces de loupe afin de fixer le servomoteur.

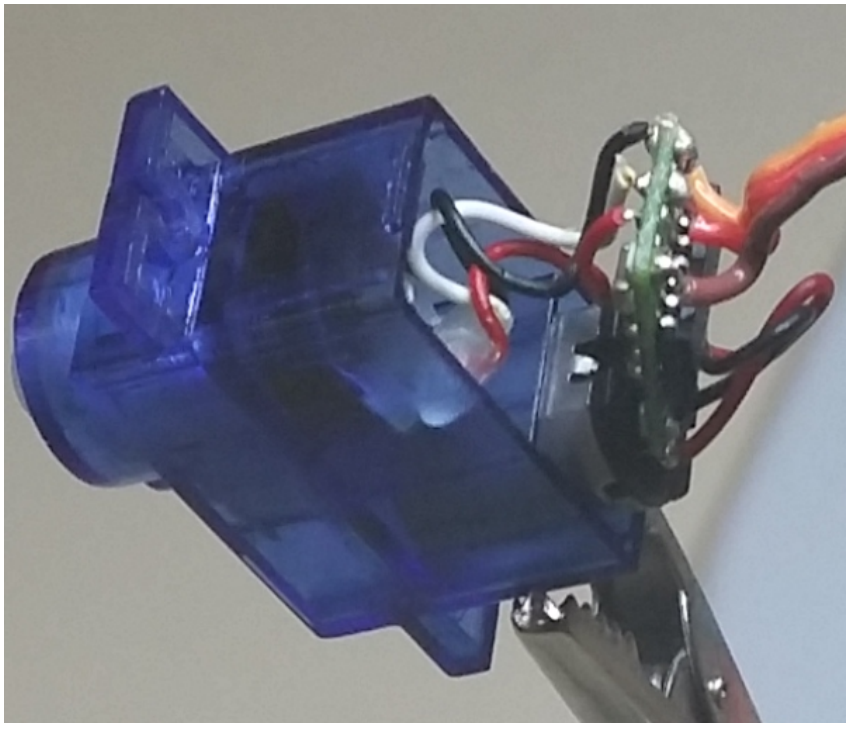
L'endroit où il faut souder est relié a un potentiomètre qui est à l'intérieur du servomoteur. On voit sur l'image ci dessous les éléments qui seront utilisés pour ce bricolage . Les pinces sur bras, le

servomoteur, Le fer à souder (coin rouge), la loupe binoculaire (base blanche) . La pince coupante pour couper le fil qui va être souder.

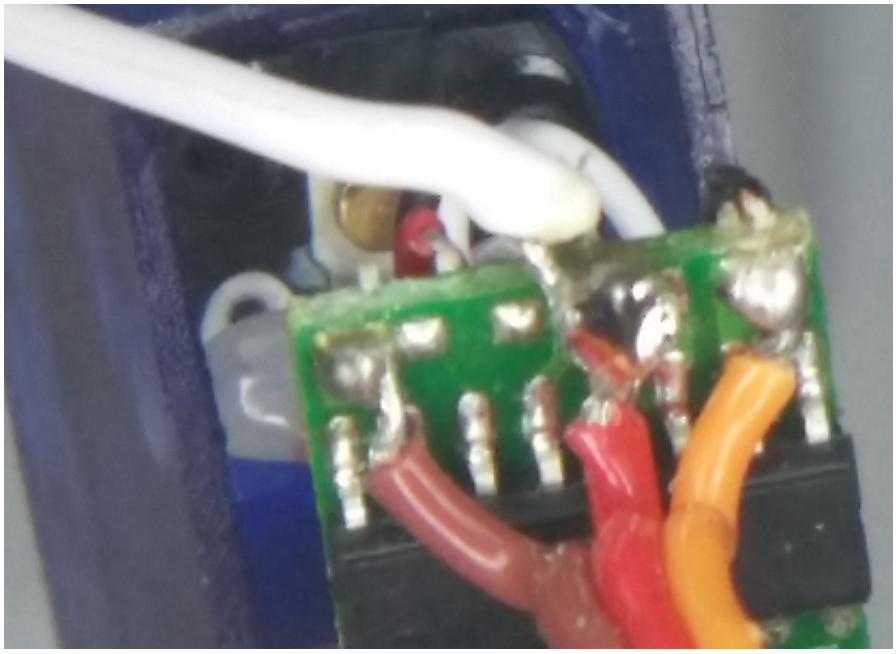


On dévisse avec un petit tournevis cruciforme pour électronicien, les 4 petites vis à la base du servomoteur, et on enlève cette base, on soulève doucement la petite carte électronique. Il y 3 fils visibles dont le fil blanc qui est relié au potentiomètre. C'est sur la soudure de ce fil que nous allons souder le notre :

Le dispositif avant notre soudure :



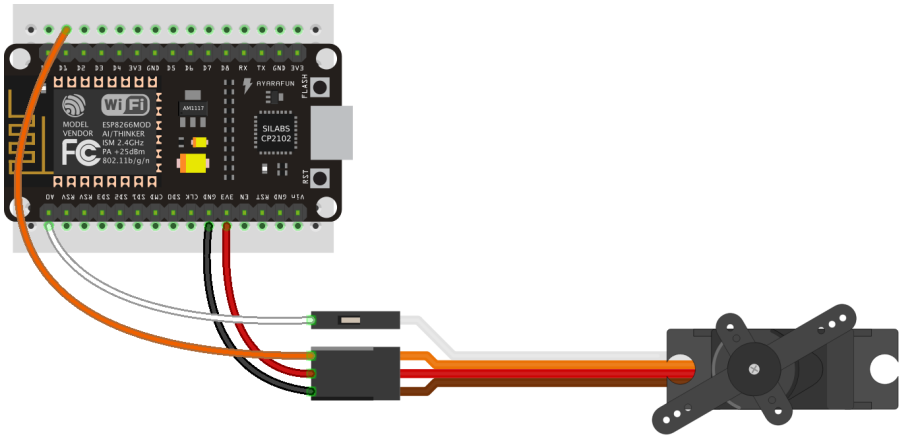
Le dispositif après notre soudure :



Bien veiller à ne pas faire de court-circuit entre les fils. Le fil blanc que nous avons rajouter ne vient qu'en contact du fil blanc qui passe en dessous de la petite carte.

Une fois cette modification faite on referme le tout en faisant les fils passer ensemble. Une minuscule entaille faite au cutter dans le plastique du servomoteur pour agrandir le passage des trous pour faire passer le quatrième fil s'avère nécessaire. Il ne faut surtout pas faire une trop grande entaille de façon à ce que les fils soient maintenus par le boîtier lorsqu'on le referme.

connexion du module ***NodeMCU ESP12E*** à la platine et au servomoteur, avec dans ce cas le retour d'information qui est branché (fil blanc), il ne devra pas l'être dans tous les exemples que nous allons voir :



Fil rouge -> broche (3v3)
fil noir -> broche (GND)
fil blanc-> broche (A0)
fil orange -> broche (D1)

IV] Programmes sur le logiciel arduino

Programmes de test et de prise en main, le programme **Blink_ESP8266.ino** qui fait clignoter une des leds intégrée à la carte que nous utilisons : **le NodeMCU1.0 ESP8266** .

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
 */


// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(16, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(16, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(16, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Programme utilisant le retour de position : **ESP8266MOOCSweep.ino**

Il faut relier le quatrième fil du servomoteur que nous avons câblé sur une entrée analogique de l'ESP8266 (**l'entrée A0**) .

```
/* Sweep
 by BARRAGAN <http://barraganstudio.com>
 This example code is in the public domain.

 modified 28 May 2015
 by Michael C. Miller
 modified 8 Nov 2013
 by Scott Fitzgerald
```

```
 http://arduino.cc/en/Tutorial/Sweep

 Adapté pour le MOOC "Fabriquer un objet connecté" 2016
  par Glenn Smith.   License : CC-BY-SA
*/

#include <Servo.h>  // Bibliothèque de prise en charge des servomoteurs

#define Maximum 170  // Valeur angulaire max. avant "grésillements"
#define Minimum 0   // Idem minimum

Servo myservo;  // Création de l'objet "myservo" de type "Servo"

int pos, angle, dir;  // Mes variables

void setup()
{
  Serial.begin(115200);
  dir = 1;
  angle = 0;
}

void loop()
{
  myservo.attach(D4);  // Connecter au servo sur broche XX
  myservo.write(angle);  // Donner la valeur angulaire
  delay(60);  // Attente pour que le servo se positionne
  myservo.detach();  // Se déconnecter (minimiser les grésillements)

  angle += dir;  // Incrémenter (ou desincrémenter - regarde bien) valeur
  if ( angle >= Maximum ) dir = -1;  // Gérer les limites de course
  if ( angle <= Minimum ) dir = 1;

  pos = analogRead(A0); // lire valeur analogique (potentiomètre du servomoteur)
//  pos = map(pos, 0, 1023, 0, 179); // (facultative) convertir en degrés
angulaires
  Serial.println(pos);  // Permet avec le 'traceur' de tracer une courbe
}
```

Autre programme de prise en main "Sweep3.ino". Qui produit une rotation du moteur dans les 2 sens. Avec à chaque fin de course, une suite de 3 clignotements d'une des led intégrés à la carte **ESP8266NodeMCU**. Pour cette manipulation le fil de retour de signal **ne doit pas être branché**.

```
/* Sweep
 by BARRAGAN <http://barraganstudio.com>
 This example code is in the public domain.

 modified 8 Nov 2013
 by Scott Fitzgerald
 http://www.arduino.cc/en/Tutorial/Sweep
Ne pas brancher la broche de retour d'info
dans ce cas de figure .
*/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards
```

```
int pos = 0;    // variable to store the servo position
int pos2 = 142;    // variable to store the max value

void setup() {
  //myservo.attach(13);  // attaches the servo on pin 9 to the servo object
myservo.attach(D1);  // le servo est branché sur la broche D1
 // initialize digital pin 16 as an output.
  pinMode(16, OUTPUT);
}

void loop() {
//  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    for (pos = 0; pos <= pos2; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable
'pos'
    //delay(15);                       // waits 15ms for the servo to reach the
position
    delay(50);                       // delais de 50 ms puisque je suis en 3.3
volts
  }

for (int i = 0; i <= 3; i += 1) { // je clignote 3 fois quand je suis arrivé au
bout
  digitalWrite(16, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);               // wait for a second
  digitalWrite(16, LOW);    // turn the LED off by making the voltage LOW
  delay(500);               // wait for a second
}

   //for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    for (pos = pos2; pos >= 0 ; pos -= 1) { // goes from pos2 degrees to 0
degrees
    myservo.write(pos);              // tell servo to go to position in variable
'pos'
   // delay(15);                       // waits 15ms for the servo to reach the
position
   delay(50);                       // delais de 50 ms puisque je suis en 3.3
volts
  }

for (int i = 0; i <= 3; i += 1) { // je clignote 3 fois quand je suis arrivé au
bout
  digitalWrite(16, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);               // wait for a second
  digitalWrite(16, LOW);    // turn the LED off by making the voltage LOW
  delay(500);               // wait for a second
}
}
```

Autre programme : "**sweep_5_Test_potar.ino**" , pour tester le retour de position avec affichage des valeurs liées à la position du potentiomètre , et graphe de sortie de celui-ci, en utilisant les fonctions "**moniteur série**" et "**traceur série**" , obtenu via le menu outils . Les deux fonctions ne peuvent pas être utilisées en même temps.

On peut utiliser ces deux fonctions pour voir la différence entre lorsque le retour de position est branché et lorsqu'il ne l'est pas .

```
/* Sweep
 by BARRAGAN <http://barraganstudio.com>
 This example code is in the public domain.
```

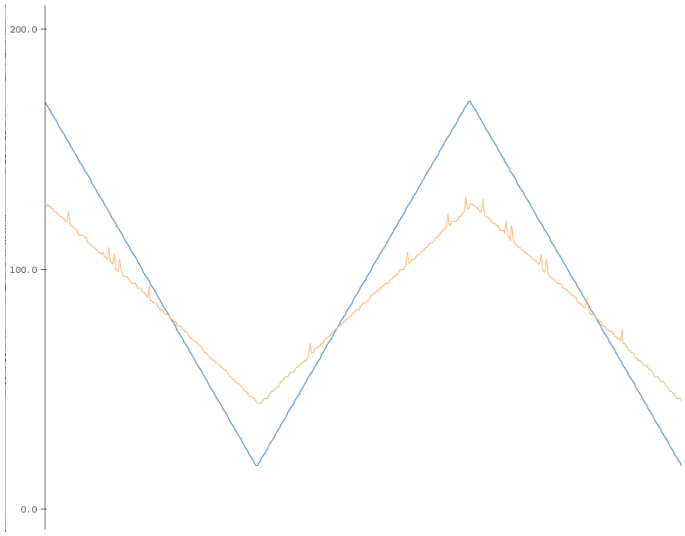
```
 modified 8 Nov 2013
 by Scott Fitzgerald
 http://www.arduino.cc/en/Tutorial/Sweep

 modified 24 Oct 2016
 by Jacques TOLA
Brancher la broche de retour d'info
dans ce cas de figure. Utiliser uniquement le 3,3 volt,
et pas la sortie Vin de 5 volts.
*/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position
int pos2 = 170; /* variable to store the max value, normalement 179, mais j'ai
vu lors des tests que la valeur
n'augmentait plus après cette valeur de 170.*/
int pos3; // variable pour récupérer et mettre à l'échelle le retour
d'information.

void setup() {
   myservo.attach(D1);  // le servo est branché sur la broche D1
 // initialize digital pin 16 as an output.Led de la carte.
  pinMode(16, OUTPUT);
//void setup()
// On initialise la sortie série à 115200 baud
{
  Serial.begin(115200);
 }

}

void loop() {
   // pos3 = analogRead(A0);                 // lit le potentiomètre du servo
(valeur comprise entre 0 et 1023)
   // pos3 = map(pos3, 0, 1023, 0, 179);

//  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    for (pos = 18; pos <= pos2; pos += 1) { /* goes from 18 degrees to pos2
degrees; normalement 0 à  pos2, mais mauvaises réponse du servo
      noir entre 8 et 17 degrés où la valeur fourni par le potentiomètre reste
bloquée à 41, et ou il y a un sursaut du servomoteur */
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable
'pos'
    pos3 = analogRead(A0);                 // lit le potentiomètre du servo
(valeur comprise entre 0 et 1023)
    pos3 = map(pos3, 0, 1023, 0, 179);
    //delay(15);                       // waits 15ms for the servo to reach the
position
    delay(15);                         // delais de 10 ms puisque je suis en 5
volts (fil rouge sur vin)
    Serial.print(pos);
     Serial.print("\t");               // prints a tab
    Serial.println(pos3);
  }

for (int i = 0; i <= 3; i += 1) { // je clignote 3 fois quand je suis arrivé au
bout
  digitalWrite(16, HIGH);   // turn the LED off (HIGH is the voltage level)
  delay(500);              // wait for a second
```

```
  digitalWrite(16, LOW);    // turn the LED on by making the voltage LOW
  delay(500);               // wait for a second
  digitalWrite(16, HIGH);    // turn the LED off by making the voltage HIGH
}

    for (pos = pos2; pos >= 18 ; pos -= 1) { // goes from pos2 degrees to 0
degrees
    myservo.write(pos);              // tell servo to go to position in variable
'pos'
    pos3 = analogRead(A0);                // lit le potentiomètre du servo
(valeur comprise entre 0 et 179)
    pos3 = map(pos3, 0, 1023, 0, 179);
   // delay(15);                       // waits 15ms for the servo to reach the
position
   delay(15);                        // delais de 10 ms puisque je suis en 5
volts (fil rouge sur vin)
   Serial.print(pos);
    Serial.print("\t");               // prints a tab
   Serial.println(pos3);
  }

for (int i = 0; i <= 3; i += 1) { // je clignote 3 fois quand je suis arrivé au
bout
  digitalWrite(16, HIGH);   // turn the LED off (HIGH is the voltage level)
  delay(500);               // wait for a second
  digitalWrite(16, LOW);    // turn the LED on by making the voltage LOW
  delay(500);               // wait for a second
  digitalWrite(16, HIGH);    // turn the LED off by making the voltage HIGH
}
}
```

On voit dans les figures ci-dessus la correspondance entre la position en degrés de la tête du moteur et la valeur mise à l'échelle pour être comprise entre **0 et 179** , renvoyée par le potentiomètre qui est branché sur la broche **A0** .

Les deux sont des signaux triangulaires en fonction du temps.

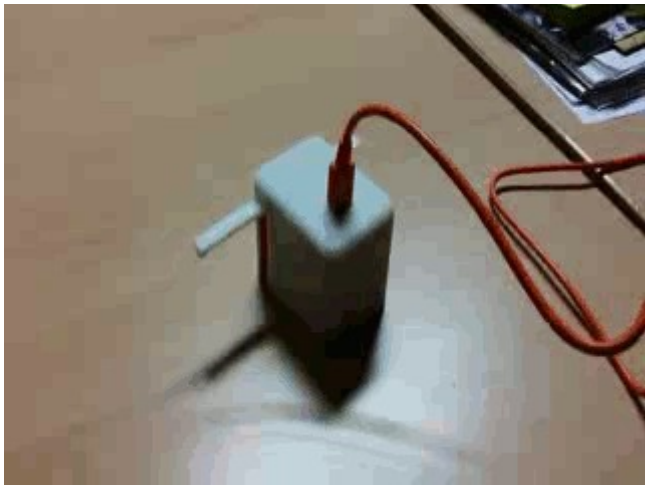
18	52
19	52
20	55
21	52
22	46
23	44
24	47
25	47
26	49
27	48
28	49
29	50
30	50
31	50
32	50
33	51
34	52
35	52
36	53
37	54
38	54
39	55
40	55
41	56
42	57



L’utilité de ce potentiomètre à la vue de ces courbes est évidente car, si la courbe bleue renvoi les valeurs qui sont renvoyées par les deux boucles. La courbe orange renvoie la valeur liée à la position du potentiomètre, qui on le voie, est en relation étroite avec celle de la variable de notre boucle, donc de la position du servomoteur. Dans cette fonction en triangle lorsque la courbe va de l’extrémité supérieure à l’extrémité inférieure, elle a balayé toute les positions du servomoteur.

Ainsi le potentiomètre permet de connaître la position du moteur, par la valeur qu’il transmet à l’entrée A0 de l’ESP8266 . Ce qui s’avérera utile lorsque l’on voudra mettre la tête du servomoteur autrement dit le bras de notre montage, dans une position précise, au départ du mouvement qu’il doit effectuer, ou à un moment particulier.

Pour cette réalisation pratique j’ai utilisé deux types de servomoteurs identiques, mais de marques différentes, et j’ai remarqué que les valeurs du potentiomètre varient suivant la marque même si elles restent linéaires.



Les API (A*pplication Programming Interface*)

Utilisation du format JSON

On utilise pour cet exercice un fichier de format JSON (**JavaScript Object Notation**) qui est retourné par le site openweathermap.org sur lequel il faut au préalable s’inscrire pour obtenir une **clé api** (**apikey**), que nous allons utiliser dans le programme qui va suivre, ce programme utilise ma clé api, mais l’utilisateur qui veut comprendre comment fonctionne ce programme devra utiliser sa propre clé api obtenu sur le site openweathermap.org .

Ce site fourni des données météorologique en ligne , entre autres la direction du vent, dans un lieu donné (ici la ville de paris), que nous allons utiliser dans ce programme.

Il se compose de la sorte :

```
/* TP semaine 2 MOOC FabOC : la servogirouette

 créé par Baptiste Gaultier le 17 Mai 2016
 modifié par Jacques TOLA le 06 Novembre 2016
 Ce code est en CC0 1.0 Universal
```

```
*/


#include <Servo.h>
#include <ESP8266WiFi.h>


// valeurs pour le WiFi de mon portable samsung comme routeur

const char* ssid     = "<ssid_de_votre_wifi>";
const char* password = "<mot_de_passe_de_votre_wifi>";


// valeurs pour le serveur Web
const char* host     = "api.openweathermap.org";
const char* apikey   = "dfd50be174df1325712b4a18476acab1";
const char* town     = "Paris,fr";

String keyword = String("\"deg\":"); //chaîne que l'on recherche dans le JSON

Servo myservo;  // création d'un objet myservo issu de la librairie Servo

// déclaration d'une variable qui stockera la position (= l'angle) du servo
int deg = 0 ;

// drapeau indiquant pendant l'analyse de la réponse du serveur
// si on est dans l'en-tête HTTP (false) ou dans le contenu de
// la ressource.
bool inBody = false;

void setup()
{
  // initialisation du port série
  Serial.begin(9600);

  // on affiche le wifi sur lequel on veut se connecter
  Serial.print("Connexion au WiFi ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);  // on se connecte

  while (WiFi.status() != WL_CONNECTED) { // on attend d'être connecté
    delay(500);
    Serial.print(".");
  }

  Serial.println("");  // on affiche les paramètres
  Serial.println("WiFi connecte");
  Serial.print("Adresse IP du module ESP: ");
  Serial.println(WiFi.localIP());
  Serial.print("Adresse IP de la box : ");
  Serial.println(WiFi.gatewayIP());
}

void loop()
{
  Serial.print("connexion au serveur : ");
  Serial.println(host);

  // On se place dans le rôle du  client en utilisant WifiClient
  WiFiClient client;

  // le serveur Web attend traditionnellement sur le port 80
  const int httpPort = 80;
```

```
  // Si la connexion échoue cela sera pour la prochaine fois
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // La connexion a réussi on forme le chemin
  // URL complexe composée du chemin et de deux
  // questions contenant le nom de ville et l'API key

  String url = String("/data/2.5/weather?q=") + town + "&appid=" + apikey;

  Serial.print("demande URL: ");
  Serial.println(url);

  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");

  // On attend 10 secondes
  delay(10000);

  inBody = false; // on est dans l'en-tête

  // On lit les données reçues, s'il y en a
  while(client.available()){
    String line = client.readStringUntil('\r');

    if (line.length() == 1) inBody = true; /* passer l'en-tête jusqu'à une ligne
vide */
    if (inBody) {  // ligne du corps du message, on cherche le mot clé
       int pos = line.indexOf(keyword);

      if (pos > 0) { /* mot clé trouvé */
       // indexOf donne la position du début du mot-clé, en ajoutant sa longueur
       // on se place à la fin.
       pos += keyword.length();

       Serial.println (&line[pos]);

       deg = atof(&line[pos]);

     } /* fin récupération du flottant */
    } /* fin de la recherche du mot clé */
  } /* fin data available */

  Serial.println();Serial.print ("Direction = ");
  Serial.println(deg); // wind.deg : Wind direction, degrees (meteorological)

  // si la valeur est > 180 alors on soustrait 179
  if(deg > 180)
    deg = deg -179;

  // et on l'envoie dans le servo :
  myservo.attach(D1);
  myservo.write(deg);

  Serial.println("connexion fermee");

  delay(10 * 1000); // 10 secondes

}
```

Ce programme une fois téléversé permettra à la fois de voir sur le moniteur série les informations concernant la direction du vent, et aussi de voir les changements de position du bras de notre montage en fonction des variations de la direction du vent.

On notera et c'est visible dans le programme que la direction du vent varie sur 360 degrés. Alors que la position du bras de notre montage varie suivant une rotation sur 180 degrés de notre servomoteur. C'est pour cela que dans notre programme il y a un test ou la valeur de la direction du vent est remise à cette même valeur moins 179 si la valeur reçu est supérieure à 180 degrés. Cela permet de rester dans la zone de balayage possible pour notre servomoteur.

Connexion au montage via un serveur web

Nous utilisons pour cet exercice un serveur qui s'appelle **tom** , celui-ci utilise un format proche du format **json** , le format **jsonp** , ce changement semble être récent car le mooc sur lequel je me base, et qui est maintenant à l'état d'archive sur la plateforme FUN , spécifie que ce serveur utilise le format **json** , avec l'aide de Cédric nous avons modifié certains programmes pour qu'ils puissent reconnaître le format **jsonp** .

Je vais identifier de la sorte mon nelson sur **`tom`** en lui donnant un nom , choisi par moi : **`mkwil3ghhtc5v`**

Pour créer un nelson sur le serveur tom , il faut déjà se connecter sur celui-ci, et sur la page où on créée les nelsons : **http://api.tom.tools/nelsons/create/**

Pour donner une latitude et une longitude à son nelson, on clique sur la ville on on se trouve. Ainsi je clique sur Paris, je zoom pour avoir précisément la "**Halle aux Farines**" , et j'ai la latitude (48,829459) et le la longitude (2,381946) correspondantes, fourni par le logiciel du site.

On met le programme suivant dans le nelson que l'on a créé sur tom en y incluant l'identité de notre nelson, afin que le serveur tom le reconnaisse :

```
        /*
  MOOC Fabriquer un objet connecté : semaine 3

 Faire une requête GET sur l'API Tom

 Le montage :
 * Juste un ESP

 créé le 24 Mai 2016
 par Baptiste Gaultier

 Ce code est en CC0 1.0 Universal
Modifié le 04 Novembre 2016 par Jacques TOLA

 */

#include <ESP8266WiFi.h>

const char* ssid     = "<ssid_de_votre_wifi>";
const char* password = "<mot_de_passe_de_votre_wifi>";

const char* host = "api.tom.tools";
```

```
const char* nelsonName = "mkwil3ghhtc5v";

void setup() {
  Serial.begin(115200);
  delay(10);

  // connexion au wifi
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  Serial.print("connecting to ");
  Serial.println(host);

  // on utilise la classe WiFiClient pour gérer les connexions avec Tom
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // We now create a URI for the request
  String url = "/nelsons/";
  url += nelsonName;
  //url += "/?format=json";
  url += "/?format=jsonp";


  Serial.print("Requesting URL: ");
  Serial.println(url);

  // on envoie la demande au serveur
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }

  // on affiche toute la réponse du serveur ligne par ligne dans le moniteur
série

 while(client.available()){
```

```
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");

  // on attend un peu pour ne pas écrouler Tom
  delay(5000);
}
```

Les **ssid** et les **password** que j'ai mis dans ce programme et que je mets dans les programmes suivants sont ceux correspondant à ma connexion wifi . Il est clair que les utilisateurs de ce document devront mettre leurs propres **paramètres de connexion wifi**. En n'oubliant pas les problèmes de sécurité. Surtout s'ils partagent leur programmes. Ils devront aussi donner à leur montage un nom suffisamment complexe pour éviter les problèmes de piratage de leur nelson.

Une fois le programme téléversé dans le nelson (**via Croquis → Téléverser , ou via la petite flèche verte en haut à gauche**) , on lance le moniteur série (**via Outils→ Moniteur série, ou le petit signe à droite au dessus de la flèche)** , en mettant la bonne vitesse pour la liaison série (**via Outils → Upload Speed → 115200**) . On doit avoir la réponse suivante :

```
connecting to api.tom.tools
Requesting URL: /nelsons/mkwil3ghhtc5v/?format=jsonp
HTTP/1.0 200 OK
Date: Sat, 05 Nov 2016 18:14:26 GMT
Server: WSGIServer/0.1 Python/2.7.9
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN
Content-Type: application/javascript; charset=utf-8
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS

callback({"name":"mkwil3ghhtc5v","latitude":"48.829459","longitude":"2.381946","
position":26,"last_activity":"2016-11-05T16:28:55.009682Z"});
closing connection
```

On doit pouvoir avoir son nelson en format **json** via la commande **curl** utilisée de la sorte :

$ curl -v GET "http://api.tom.tools/nelsons/mkwil3ghhtc5v?format=jsonp"

qui nous renvois la réponse suivante :

```
* Rebuilt URL to: GET/
* Could not resolve host: GET
* Closing connection 0
curl: (6) Could not resolve host: GET
*   Trying 51.255.35.69...
* Connected to api.tom.tools (51.255.35.69) port 80 (#1)
> GET /nelsons/mkwil3ghhtc5v?format=jsonp HTTP/1.1
> Host: api.tom.tools
> User-Agent: curl/7.50.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Sat, 05 Nov 2016 17:52:58 GMT
< Server: WSGIServer/0.1 Python/2.7.9
< Vary: Accept, Cookie
```

< X-Frame-Options: SAMEORIGIN
< Content-Type: application/javascript; charset=utf-8
< Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
<
*** Closing connection 1**
callback({"name":"mkwil3ghhtc5v","latitude":"48.829459","longitude":"2.381946","position":26, "last_activity":"2016-11-05T16:28:55.009682Z"});

on doit pouvoir voir notre nelson sur cette page : http://api.tom.tools/nelsons/mkwil3ghhtc5v/ , donc le nelson que vous avez créé est sur la page : http://api.tom.tools/nelsons/<mon_nelson>/ .

Si cette commande ne marche pas il faut s'assurer que le paquet «**curl**» , est bien installé sur le système utilisé , pour Ubuntu, ou une distribution Debian il doit être installé via la commande :

apt-get install curl

qui doit être lancée avec les droits root,

Si elle ne marche toujours pas essayer de remplacer « **jsonp** » , dans la commande par « **json** » . En faisant :

$ curl -v GET "http://api.tom.tools/nelsons/mkwil3ghhtc5v?format=json"

Car j'ai constaté que suivant les dates c'est l'une ou l'autre de ces deux syntaxe qui fonctionne.

Le programme suivant permet de récupérer des données au format JSON concernant notre montage sur le serveur **tom** .
On peut à la fois visualiser dans le moniteur série les données concernant notre montage, qu'il récupère via son module wifi, sur le serveur tom et aussi faire bouger son bras via les données de position que nous avons entrés sur le serveur.
Le programme s'articule de la sorte :

```
/**
 * BasicHTTPClient.ino
 *
 *  Created on: 24.05.2015
 *  *Modifié le 06 Novembre 2016 par Jacques TOLA
 *
 */

#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <Servo.h>
#include <ESP8266HTTPClient.h>

#define Serial Serial

Servo myservo;  // créé un objet servo pour contrôler le servomoteur

ESP8266WiFiMulti WiFiMulti;
```

```
void setup() {

    Serial.begin(115200);
   // Serial.setDebugOutput(true);

    Serial.println();
    Serial.println();
    Serial.println();

    for(uint8_t t = 4; t > 0; t--) {
        Serial.printf("[SETUP] WAIT %d...\n", t);
        Serial.flush();
        delay(1000);
       }

    pinMode(16, OUTPUT);
    myservo.attach(D1);// le servo est branché sur la broche D1
   // WiFiMulti.addAP("SSID", "PASSWORD");
   //WiFiMulti.addAP("perso_15", "32A3FD519736FA323E1F92EE671142");
   WiFiMulti.addAP("<ssid_de_votre_wifi>", "<mot_de_passe_de_votre_wifi>");
}

void loop() {
    // wait for WiFi connection
    if((WiFiMulti.run() == WL_CONNECTED)) {

        HTTPClient http;

        Serial.print("[HTTP] begin...\n");
        // configure traged server and url
        //http.begin("https://192.168.1.12/test.html", "7a 9c f4 db 40 d3 62 5a
6e 21 bc 5c cc 66 c8 3e a1 45 59 38"); //HTTPS
       // http.begin("http://192.168.1.12/test.html"); //HTTP
       //http.begin("http://api.tom.tools", 80,"/nelsons/mkwil3ghhtc5v/");
//modif perso
       //http.begin("http://api.tom.tools/nelsons/mkwil3ghhtc5v/");//donne du
html
        //http.begin("api.tom.tools/nelsons/mkwil3ghhtc5v?format=jsonp");
        http.begin("api.tom.tools", 80, "/nelsons/mkwil3ghhtc5v?format=jsonp");
// donne du JSON


        Serial.print("[HTTP] GET...\n");
        // start connection and send HTTP header
        int httpCode = http.GET();
        int pos = 0 ; //perso
        // httpCode will be negative on error
        if(httpCode > 0) {
            // HTTP header has been send and Server response header has been
handled
            Serial.printf("[HTTP] GET... code: %d\n", httpCode);

            // file found at server
            if(httpCode == HTTP_CODE_OK) {
                String payload = http.getString();
                Serial.println(payload);

              //ajout perso
              int start = payload.indexOf("position\":");
              int end = payload.indexOf(',',start+1);
              String positionString = "";
              for ( int i = start+10;i<end;i++)
                  //positionString += payload.CharAt(i);
                  positionString += payload.charAt(i);
```

```
              pos = positionString.toInt();

              Serial.println(pos);

               for (int i = 0; i <= pos; i += 1) { // je clignote pos fois avant
de passer à la suite
                digitalWrite(16, HIGH);   // turn the LED off (HIGH is the
voltage level)
                delay(50);              // wait for a second
                  digitalWrite(16, LOW);    // turn the LED on by making the
voltage LOW
                  delay(50);              // wait for a second
                   digitalWrite(16, HIGH);    // turn the LED off by making the
voltage HIGH
                  }

              myservo.write(pos); // dis au servomoteur va à la position donné
par la variable variable 'pos'

            }
        } else {
            Serial.printf("[HTTP] GET... failed, error: %s\n",
http.errorToString(httpCode).c_str());
        }

        http.end();
    }

    delay(10000);
}
```

La aussi les identifiants qui sont utilisés pour la connexion au wifi sont les miens. L'utilisateur doit utiliser les siens pour faire marcher son montage.

Dans le cadre des programmes qui utilisent le serveur tom, nous avons celui-ci qui lit une valeur, celle de la position du servo sur le serveur tom. Si cette valeur est différente de la valeur de la positon de la tête sur le servomoteur. Ce qui sera le cas si l'utilisateur déplace le bras du nelson. Le programme de renvoie vers le servo moteur la valeur lut sur tom pour qu'il reprenne sa position de départ.

Le programme dont il est question que j'ai appelé «**Nelson_et_Tom_FabOCS3_marche.ino**» , s'articule de la sorte :

```
/*
  MOOC Fabriquer un objet connecté : semaine 3

 Lit la valeur analogique du servo moteur sur la broche A0 et envoie
 cette valeur sur la plateforme TOM en utilisant une requête PATCH
 Toutes les 3 secondes, nous récupérons la valeur sur TOM afin de voir si le
 servo a bougé

 Le montage :
 * Un servo moteur branché sur les broches 3V3, GND et D1
 * La patte retour d'information du servo est branchée sur la broche A0

 créé le 30 Mars 2016
 par Laurent Mattlé et Baptiste Gaultier
 modifié le 7 Novembre 2016 par Jacques TOLA

 Ce code est en CC0 1.0 Universal
```

```
 */
#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

#include <ESP8266HTTPClient.h>

ESP8266WiFiMulti WiFiMulti;

#include <Servo.h>

// création de l'objet servo issu du moule Servo
Servo myservo;

// déclaration d'une variable payload dont on va se servir pour
// l'envoie et la réception de nos messages
String payload;

// initialisation des constantes spécifiques à votre installation
//const String nelsonName = "";
//const char* ssid = "";
//const char* password = "";

const String nelsonName = "mkwil3ghhtc5v";

const char* ssid     = "<ssid_de_votre_wifi>";
const char* password = "<mot_de_passe_de_votre_wifi>";

// initialise deux variables entières
// pos contient la valeur actuelle de notre servo
int pos = 0;
// lastPos contient la valeur de notre servo lors du dernier tour de boucle
int lastPos = 0;

// tolerance permet d'avoir une approximation du feeback entre deux tours
// de boucle
int tolerance = 2;

void setup() {
  // on souhaite communiquer avec l'ordinateur
  Serial.begin(9600);
  Serial.setDebugOutput(true);

  Serial.println("Nelson v0.3 PATCH starting...");

  for(uint8_t t = 4; t > 0; t--) {
      Serial.printf("[SETUP] WAIT %d...\n", t);
      Serial.flush();
      delay(1000);
  }

  // on se connecte au Wifi
  WiFiMulti.addAP(ssid, password);

}

void loop() {
  // on regarde si on est bien connecté au point d'accès wifi
  if((WiFiMulti.run() == WL_CONNECTED)) {
    // création d'un objet appelé http réutilisable et sorti du moule HTTPClient
    HTTPClient http;
```

```
    Serial.println("[HTTP] begin...");
    //String uri = "/nelsons/" + nelsonName + "/";
    //String uri = "/nelsons/" + nelsonName + "/?format=jsonp" + "/"; //perso
    String uri = "/nelsons/" + nelsonName; //perso
    uri += "/?format=jsonp"; //perso ,où on spécifie que le format est du json

    http.begin("api.tom.tools", 80, uri);

    Serial.println("[HTTP] GET...");
    // démarrer la connexion et envoyer les entêtes HTTP
    //http.addHeader("Content-type", "application/json");  //non commenté à
l'origine
    //http.addHeader("Content-type", "application/jsonp"); //test perso

    int httpCode = http.GET();

    // httpCode sera négatif si on rencontre une erreur
    if(httpCode > 0) {
      // les entêtes HTTP ont été envoyés et
      Serial.printf("[HTTP] GET... code: %d\n", httpCode);

      // si le serveur TOM répond par OK
      if(httpCode == HTTP_CODE_OK) {
        // alors on récupère la réponse sous forme de chaîne de caractères
        String payload = http.getString();
        Serial.println("[HTTP] GET... payload:");
        Serial.println(payload);

        int start = payload.indexOf("position\":");
        int end = payload.indexOf(',', start + 1 );
        String positionString = "";
        for (int i= start + 10; i < end; i++)
          positionString += payload.charAt(i);

        // récupère la valeur de tom et la stocke dans pos
        pos = positionString.toInt();

        // nous écrivons ensuite cette valeur sur le servo
        Serial.println("Ne pas toucher au servo pour l'instant"); //perso doute
sur l'emplacement dans le programme
        myservo.attach(D1);
        myservo.write(pos);
        delay(500);
        myservo.detach();
        Serial.println("Vous pouvez modifier la position du servo, vous avez 5
secondes"); //perso doute sur la position dans le programme.
        delay(5000); //perso
        Serial.println("Relâchez le servo maintenant");
        delay(3000);
      }
    }
    else
      Serial.printf("[HTTP] GET... failed, error: %s\n",
http.errorToString(httpCode).c_str());

    http.end();

    // et nous n'oublions pas de relâcher le servo afin de pouvoir le manipuler
    int pos = map(analogRead(A0), 0, 1023, 0, 179);

    // on attend trois secondes avant la prochaine requête
    delay(3000);

    // on regarde à nouveau si le servo n'a pas bougé
```

```
    int lastPos = map(analogRead(A0), 0, 1023, 0, 179);

    Serial.print("pos:");
    Serial.println(pos);
    Serial.print("last:");
    Serial.println(lastPos);

    if(abs(pos - lastPos) > tolerance) {
      Serial.println("[HTTP] PATCH...");

      String payload = "{\"position\":";
      payload += String(lastPos);
      payload += "}";

      Serial.print("[HTTP] PATCH... payload:");
      Serial.println(payload);

      // démarrer la connexion et envoyer les entêtes HTTP
      //http.addHeader("Content-type", "application/json"); //non commenté à
l'origine
      //http.addHeader("Content-type", "application/jsonp"); //test perso

      httpCode = http.sendRequest("PATCH", (uint8_t *) payload.c_str(),
payload.length());

      // httpCode sera négatif si on rencontre une erreur
      if(httpCode > 0) {
        // les entêtes HTTP ont été envoyés et
        Serial.printf("[HTTP] PATCH... code: %d\n", httpCode);

        // si le serveur TOM répond par OK
        if(httpCode == HTTP_CODE_OK)
          Serial.println("[HTTP] PATCH... OK!");
      }
      else
        Serial.printf("[HTTP] PATCH... failed, error: %s\n",
http.errorToString(httpCode).c_str());

      http.end();
    }
  }
}
```

Le dispositif reçoit comme précédemment des données, via le wifi, qui sont visibles sur le moniteur série, les données issues de tom, sous forme d'un fichier json . Le moniteur série donne aussi le moment ou l'utilisateur peut tourner le moteur, car pendant ce laps de temps le moteur est débrayé pour que l'utilisateur puissent le tourner, ou plus exactement pour qu'il puisse déplacer le bras du dispositif. L'utilisateur, reçoit aussi, toujours via le moniteur série, une indication pour qu'il ne touche plus au servo moteur. Afin que le programme puisse reprendre le contrôle de ce dernier.

Ici un autre programme qui, cette fois, va envoyer une valeur sur le serveur tom afin de changer une des valeurs contenu dans le json, la valeur position. Cette manip est faite via la commande « **PATCH** », contenu dans le programme. J'ai appelée celui-ci « **nelson_interrupteur_TP3_perso.ino** » et que voici :

```
/*
  Correction TP 3 : le Nelson interrupteur

 Lit la valeur analogique du servo moteur sur la broche A0 et si la
 valeur est supérieure à 90° alors envoie une requête PATCH à Tom

 Le montage :
 * Un servo moteur branché sur les broches 3V3, GND et D1
 * La patte retour d'information du servo est branchée sur la broche A0

 créé le 30 Mars 2016
 par Laurent Mattlé et Baptiste Gaultier

 Ce code est en CC0 1.0 Universal
 modifié le 14 Novembre 2016 par Jacques TOLA

 */

#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

#include <ESP8266HTTPClient.h>

ESP8266WiFiMulti WiFiMulti;

#include <Servo.h>

// création de l'objet servo issu du moule Servo
Servo myservo;

// déclaration d'une variable payload dont on va se servir pour
// l'envoie et la réception de nos messages
String payload;

// initialisation des constantes spécifiques à votre installation
const String nelsonName = "mkwil3ghhtc5v";
//const char* ssid     = "<ssid_de_votre_wifi>";
//const char* password = "<mot_de_passe_de_votre_wifi>";

const char* ssid     = "perso_15";
const char* password = "32A3FD519736FA323E1F92EE671142";

// initialise deux variables entières
/*  la variable pos contient la valeur actuelle de notre servo
 *  pour des problèmes de tolérance je me positionne à 40
 *  au lieu de 0. Au moins pour être sur que
 *  cela fonctionne L'utilisateur pourra faire ses tests
 *  avec 0° .
 */
//int pos = 0;
int pos = 40; //perso

void setup() {
  // on souhaite communiquer avec l'ordinateur
  Serial.begin(9600);
  Serial.setDebugOutput(true);

  Serial.println("Nelson v0.3 PATCH starting...");

  for(uint8_t t = 4; t > 0; t--) {
      Serial.printf("[SETUP] WAIT %d...\n", t);
      Serial.flush();
```

```
      delay(1000);
  }

  // on se connecte au Wifi
  WiFiMulti.addAP(ssid, password);

  // on positionne le bras du Nelson à 0° _(40° pour l'instant)
  myservo.attach(D1);
  myservo.write(pos);
  delay(500);
  myservo.detach();
}

void loop() {
  // on regarde si on est bien connecté au point d'accès wifi
  if((WiFiMulti.run() == WL_CONNECTED)) {
    // création d'un objet appelé http réutilisable et sorti du moule HTTPClient
    HTTPClient http;

    // construction de l'uri à partir du nom du Nelson
    Serial.println("[HTTP] begin...");
    String uri = "/nelsons/" + nelsonName + "/";
    http.begin("api.tom.tools", 80, uri);

    // on regarde si on a dépassé les 90°
    int pos = map(analogRead(A0), 0, 1023, 0, 179);

    // si on a dépassé les 90° :
    if(pos > 90) {
      Serial.println("[HTTP] PATCH...");

      // on prépare un json contenant la valeur 180 pour" position".
      String payload = "{\"position\":180}";

      Serial.print("[HTTP] PATCH... payload:");
      Serial.println(payload);

      // démarrer la connexion et envoyer les entêtes HTTP
      //envoyer à tom la valeur 180, pour "position".
      http.addHeader("Content-type", "application/json");
      int httpCode = http.sendRequest("PATCH", (uint8_t *) payload.c_str(),
payload.length());  //envoi de la requete au serveur tom.

      // httpCode sera négatif si on rencontre une erreur
      if(httpCode > 0) {
        // les entêtes HTTP ont été envoyés
        Serial.printf("[HTTP] PATCH... code: %d\n", httpCode);

        // si le serveur TOM répond par OK
        if(httpCode == HTTP_CODE_OK) {
          Serial.println("[HTTP] PATCH... OK!");


          // ici nous envoyons une requête PATCH après avoir attendu 1 secondes
           // puis nous traitons le retour de tom en position 0 ( non 40)
          //modif perso
           delay(1000);
          //payload = "{\"position\":0}";
          String payload = "{\"position\":40}";
          Serial.print("[HTTP] PATCH... payload:");
          Serial.println(payload);

          // On envoie à tom la valeur 40 (valeur de la position du bras).
          http.addHeader("Content-type", "application/json");
```

```
          int httpCode = http.sendRequest("PATCH", (uint8_t *) payload.c_str(),
payload.length());

          myservo.attach(D1);
          //myservo.write(0);

          // Utilisé aussi pour l'envoie de la valeur 0 (donc 40, pour nous),à
tom.
          http.sendRequest("PATCH", (uint8_t *) payload.c_str(),
payload.length());
           // puis on replace le servo en position 0 (ici 40, la valeur que nous
avons prise)
          myservo.write(40); //perso

          //on attend une demi-seconde.
          delay(500);

          //on debraye le bras pour que l'utilisateur puisse le bouger
          myservo.detach();
        }
      }
      else
        Serial.printf("[HTTP] PATCH... failed, error: %s\n",
http.errorToString(httpCode).c_str());

      http.end();
    }
    // on attend trois secondes avant la prochaine requête
    delay(3000);
  }
}
```

V] Installation du logiciel «Processing»

j'ai téléchargé à partir du site https://processing.org/download/ le fichier **processing-3.3.3-linux64.tgz** que j'ai décompressé sur le bureau via l'instruction :

$ tar -xzf ~/Téléchargements/processing-3.3.3-linux64.tgz -C ~/Bureau/

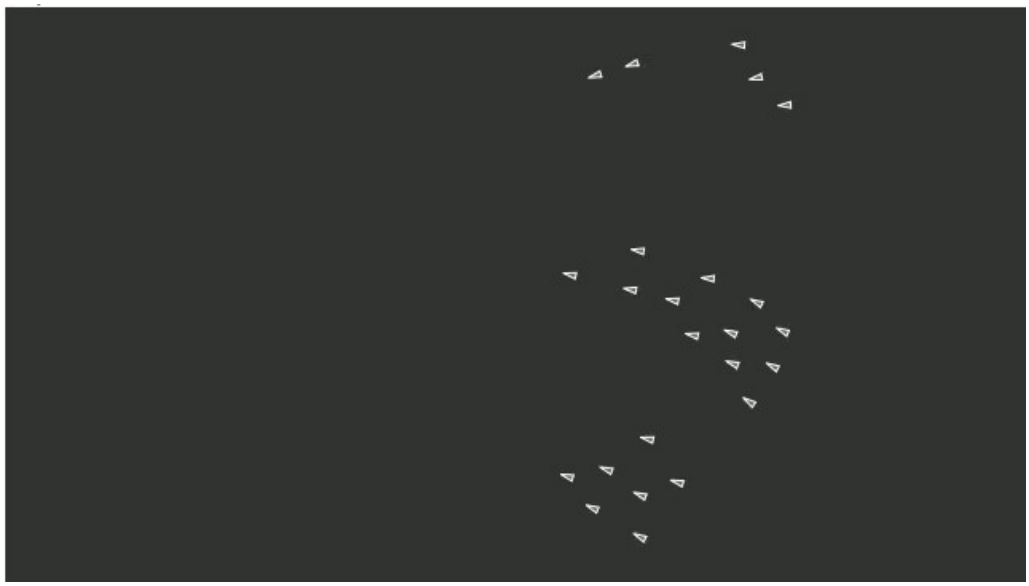
Cela a créé sur le bureau le dossier : processing-3.2.3 dans lequel je me rends via l'instruction :

$ cd ~/Bureau/processing-3.3.3/

Une fois dans le dossier je lance le logiciel via l'instruction :

$ bash processing

Cela lance le logiciel , qui présente une interface du même type que celui du logiciel Arduino. On lance une des fichiers d'exemple, le fichier « **Flocking** » , en faisant comme dans le cas d'Arduino **Fichier → Exemples → Topics → Simulate → Flocking** qui, lorsqu'on le lance nous donne la fenêtre suivante :



Ce fichier nous servira de base pour le prochain programme. Qui à pour fonction de faire apparaître sur un terminal le nombre d'appui qui a été réalisé sur le bras du nelson.

Ce genre de fonctionnalité peut-être utilisé, par exemple, pour savoir combien de personnes sont présentes dans le fablab sachant que chaque nouvel arrivant, appui sur un le bras du nelson et que sur un terminal apparaît ainsi le nombre de personne présentent dans le lieu. Par l'incrémentation d'un compteur qui se fait chaque fois que le bras est appuyé . Et par l'augmentation du nombre des petits triangles sur l'écran graphique représentant aussi les personnes présentent dans le Fablab . Cette incrémentation se fait aussi si on clique avec la souris, ou si on appuie sur une touche du clavier.

VI] Programmation sur le logiciel « Processing » .

Le logiciel Processing fonctionne d'une manière similaire au logiciel Arduino. Avec ce dernier nous avons utilisé à chaque fois deux parties, « **setup** » , et « **loop** », la première partie contenant les opérations d'initialisation qu'il n'est nécessaires d'entrer qu'une seule fois dans notre

programme. La deuxième partie contenant les instructions qui se répètent tout au long de notre programme.

Il en est de même dans le principe pour le logiciel Processing, qui contient aussi deux parties, « **setup** » , et « **draw** » qui fonctionnent de façon similaires, ce sont comme le dit la documentation sur openclasseroom , des «**fonctions pré-implémentées et prêtes à l'emploi**».

L'avantage du logiciel Processing par rapport au logiciel Arduino, est qu'il permet d'ajouter à notre programme une interface graphique, il comporte aussi un terminal intégré, et envoi aussi des messages sur un des émulateur de terminal Gnome d'Ubuntu . Ce que nous n'avions pas avec le logiciel Arduino. Qui comporte seulement un terminal et un traceur série . Processing permet de faire des applications très élaborées au niveau graphique et notamment de la 3D .

Comme spécifié précédemment nous allons faire un programme qui nous affiche le nombre de personnes présentent dans une salle, dans notre cas le fablab du script. Où nous avons placé à l'entrée notre nelson. Les nouveaux entrants dans le fablab appuient sur le bras du nelson. Et celui-ci va incrémenter un compteur sur un ordinateur, situé dans le lieu où se trouve la personne qui vérifie le nombre de personnes présentent dans le Fablab. Il faut que cette machine dispose du logiciel processing et soit raccordée au réseau.

Le programme que nous avons réalisé, s'appelle « **FlockingMOOC_perso4.pde** » , il se présente de la sorte :

```
/**

 * FabLab counter
 * créé par Baptiste Gaultier
 *
 * Une reprise du programme Boids de Craig Reynold (simulant le
 * comportement d'une nuée d'oiseaux en vol). Ce programme a été
 * modifié pour aller chercher toutes les dix secondes la position
 * du servomoteur d'un nelson sur la plateforme api.tom.tools.
 * Si cette valeur est supérieure à 100, cela signifie qu'une
 * personne est entrée dans le lab et que l'on doit incrémenter
 * le compteur de 1.
 *
 * Le compteur peut également être incrémenté en appuyant sur une
 * touche du clavier ou de la souris.
 * Programme modifié par Cédric DEVILLERS et Jacques TOLA
 *
 */

import processing.net.*;
Flock flock;
String data;

// déclaration des variables nécessaires au programme

JSONObject json;
PImage logo;

int folks = 0;

// constante spécifique à votre nelson
//String nelsonName = "mkwil3ghhtc5v";
```

```
void setup() {
  // passer l'interface en plein écran
  //fullScreen();

  size(720, 480);
  // dé-commenter la ligne ci-dessous si vous avez un écran Retina
  //pixelDensity(2);

  // charger une image qyue nous afficherons plus tard
  logo = loadImage("logo.png");

  // création du système
  flock = new Flock();

  // ajouter un élément initial
  flock.addBoid(new Boid(width/2,height/2));

  // on passe le nombre de personnes de 0 à 1
  folks ++;
  loadData();
}
void loadData() {
String url = "http://api.tom.tools/nelsons/mkwil3ghhtc5v/?format=jsonp";
String[] lines = loadStrings(url);
String html = join(lines, "");
/*On parse le programme pour récupérer les infos entre la parenthèse ouvrante
et la parenthèse fermante, du renvoi jsonp
*/
 int start = html.indexOf("callback(");
        int end = html.indexOf(')', start + 1 );
        String positionString = "";
        for (int i= start + 9; i < end; i++)
          positionString += html.charAt(i);
    json = JSONObject.parse(positionString) ;
    println(positionString);

// on regarde si la position de notre nelson est supérieur à 100 (j'ai pris 90,
mais cela marche avec 100).
  //JSONObject json = loadJSONObject(url);

  int position = json.getInt("position");
  if(position > 90) {
    // si >100 : appuyé, on ajoute un nouvel élément
    int posX = int(random(width-70));
    int posY = int(random(height-80));
    flock.addBoid(new Boid(posX, posY));
    folks ++;
  }

}

// le bloc draw est l'équivalent de loop dans Arduino
void draw() {
  background(255);

  // on ajoute un logo du lab en haut à gauche
  image(logo, 12, 12, logo.width/1.5, logo.height/1.5);

  // on s'assure que le système va bien être mis à jour
  flock.run();

  // on affiche le nombre de personnes en bas à droite
```

```
  textSize(16);
  fill(71, 113, 181);
  text(folks + " personnes", width-132, height-32);

  // on envoie une requête à tom toutes les ~10s
  //if (frameCount % 600 * frameRate == 0)
    if (frameCount % 300 * frameRate == 0)
    loadData();
}


// on ajoute une nouvelle personne si on clique
void mousePressed() {
  flock.addBoid(new Boid(mouseX,mouseY));
  folks ++;
}

// on ajoute une nouvelle personne si on appuie sur une touche clavier
void keyPressed() {
  int posX = int(random(width-70));
  int posY = int(random(height-80));
  flock.addBoid(new Boid(posX, posY));
  folks ++;
}
```

le programme, suivant la structure utilisée par «**Processing**», se trouve dans un répertoire de même nom sans l'extension avec deux autres programmes auxquelles il fait appel, que l'on voit dans le corps du programme «**Flocking_MOOC_perso4.pde**», ce sont les programmes « **Boid.pde** » et « **Flock.pde** ».

Le programme «**Boid.pde** » se compose de la sorte :

```
// The Boid class


class Boid {

  PVector location;
  PVector velocity;
  PVector acceleration;
  float r;
  float maxforce;    // Maximum steering force
  float maxspeed;    // Maximum speed

    Boid(float x, float y) {
    acceleration = new PVector(0, 0);

    // This is a new PVector method not yet implemented in JS
    // velocity = PVector.random2D();

    // Leaving the code temporarily this way so that this example runs in JS
    float angle = random(TWO_PI);
    velocity = new PVector(cos(angle), sin(angle));

    location = new PVector(x, y);
    r = 2.0;
    maxspeed = 2;
    maxforce = 0.03;
  }
```

```
  void run(ArrayList<Boid> boids) {
    flock(boids);
    update();
    borders();
    render();
  }

  void applyForce(PVector force) {
    // We could add mass here if we want A = F / M
    acceleration.add(force);
  }

  // We accumulate a new acceleration each time based on three rules
  void flock(ArrayList<Boid> boids) {
    PVector sep = separate(boids);   // Separation
    PVector ali = align(boids);      // Alignment
    PVector coh = cohesion(boids);   // Cohesion
    // Arbitrarily weight these forces
    sep.mult(1.5);
    ali.mult(1.0);
    coh.mult(1.0);
    // Add the force vectors to acceleration
    applyForce(sep);
    applyForce(ali);
    applyForce(coh);
  }

  // Method to update location
  void update() {
    // Update velocity
    velocity.add(acceleration);
    // Limit speed
    velocity.limit(maxspeed);
    location.add(velocity);
    // Reset accelertion to 0 each cycle
    acceleration.mult(0);
  }

  // A method that calculates and applies a steering force towards a target
  // STEER = DESIRED MINUS VELOCITY
  PVector seek(PVector target) {
    PVector desired = PVector.sub(target, location);  // A vector pointing from
the location to the target
    // Scale to maximum speed
    desired.normalize();
    desired.mult(maxspeed);

    // Above two lines of code below could be condensed with new PVector
setMag() method
    // Not using this method until Processing.js catches up
    // desired.setMag(maxspeed);

    // Steering = Desired minus Velocity
    PVector steer = PVector.sub(desired, velocity);
    steer.limit(maxforce);  // Limit to maximum steering force
    return steer;
  }

  void render() {
    // Draw a triangle rotated in the direction of velocity
    float theta = velocity.heading2D() + radians(90);
    // heading2D() above is now heading() but leaving old syntax until
Processing.js catches up
```

```
    fill(255);
    strokeWeight(2);
    stroke(71, 113, 181);
    pushMatrix();
    translate(location.x, location.y);
    rotate(theta);
    beginShape(TRIANGLES);
    vertex(0, -r*2);
    vertex(-r, r*2);
    vertex(r, r*2);
    endShape();
    popMatrix();
  }

  // Wraparound
  void borders() {
    if (location.x < -r) location.x = width+r;
    if (location.y < -r) location.y = height+r;
    if (location.x > width+r) location.x = -r;
    if (location.y > height+r) location.y = -r;
  }

  // Separation
  // Method checks for nearby boids and steers away
  PVector separate (ArrayList<Boid> boids) {
    float desiredseparation = 25.0f;
    PVector steer = new PVector(0, 0, 0);
    int count = 0;
    // For every boid in the system, check if it's too close
    for (Boid other : boids) {
      float d = PVector.dist(location, other.location);
      // If the distance is greater than 0 and less than an arbitrary amount (0
when you are yourself)
      if ((d > 0) && (d < desiredseparation)) {
        // Calculate vector pointing away from neighbor
        PVector diff = PVector.sub(location, other.location);
        diff.normalize();
        diff.div(d);        // Weight by distance
        steer.add(diff);
        count++;            // Keep track of how many
      }
    }
    // Average -- divide by how many
    if (count > 0) {
      steer.div((float)count);
    }

    // As long as the vector is greater than 0
    if (steer.mag() > 0) {
      // First two lines of code below could be condensed with new PVector
setMag() method
      // Not using this method until Processing.js catches up
      // steer.setMag(maxspeed);

      // Implement Reynolds: Steering = Desired - Velocity
      steer.normalize();
      steer.mult(maxspeed);
      steer.sub(velocity);
      steer.limit(maxforce);
    }
    return steer;
  }

  // Alignment
```

```
  // For every nearby boid in the system, calculate the average velocity
  PVector align (ArrayList<Boid> boids) {
    float neighbordist = 50;
    PVector sum = new PVector(0, 0);
    int count = 0;
    for (Boid other : boids) {
      float d = PVector.dist(location, other.location);
      if ((d > 0) && (d < neighbordist)) {
        sum.add(other.velocity);
        count++;
      }
    }
    if (count > 0) {
      sum.div((float)count);
      // First two lines of code below could be condensed with new PVector
setMag() method
      // Not using this method until Processing.js catches up
      // sum.setMag(maxspeed);

      // Implement Reynolds: Steering = Desired - Velocity
      sum.normalize();
      sum.mult(maxspeed);
      PVector steer = PVector.sub(sum, velocity);
      steer.limit(maxforce);
      return steer;
    }
    else {
      return new PVector(0, 0);
    }
  }

  // Cohesion
  // For the average location (i.e. center) of all nearby boids, calculate
steering vector towards that location
  PVector cohesion (ArrayList<Boid> boids) {
    float neighbordist = 50;
    PVector sum = new PVector(0, 0);   // Start with empty vector to accumulate
all locations
    int count = 0;
    for (Boid other : boids) {
      float d = PVector.dist(location, other.location);
      if ((d > 0) && (d < neighbordist)) {
        sum.add(other.location); // Add location
        count++;
      }
    }
    if (count > 0) {
      sum.div(count);
      return seek(sum);  // Steer towards the location
    }
    else {
      return new PVector(0, 0);
    }
  }
}
```

Le programme «**Flock.pde**» se compose de la sorte :

```
// The Flock (a list of Boid objects)


class Flock {
```

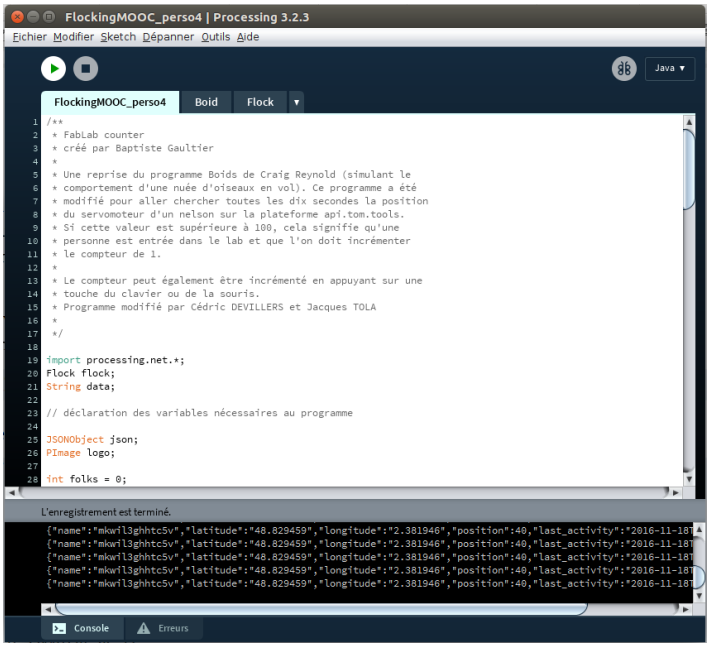
```
  ArrayList<Boid> boids; // An ArrayList for all the boids

  Flock() {
    boids = new ArrayList<Boid>(); // Initialize the ArrayList
  }

  void run() {
    for (Boid b : boids) {
      b.run(boids);  // Passing the entire list of boids to each boid
individually
    }
  }

  void addBoid(Boid b) {
    boids.add(b);
  }

}
```

j'ai fait également sur la base du programme cité en référence dans le mooc, un logo simple nommé « **logo.png** » , auquel le programme «**FlockingMOOC_perso4.pde**» fait aussi appel, il doit se trouver dans le même répertoire que les 3 programmes cités. il s'affichera dans l'interface graphique :

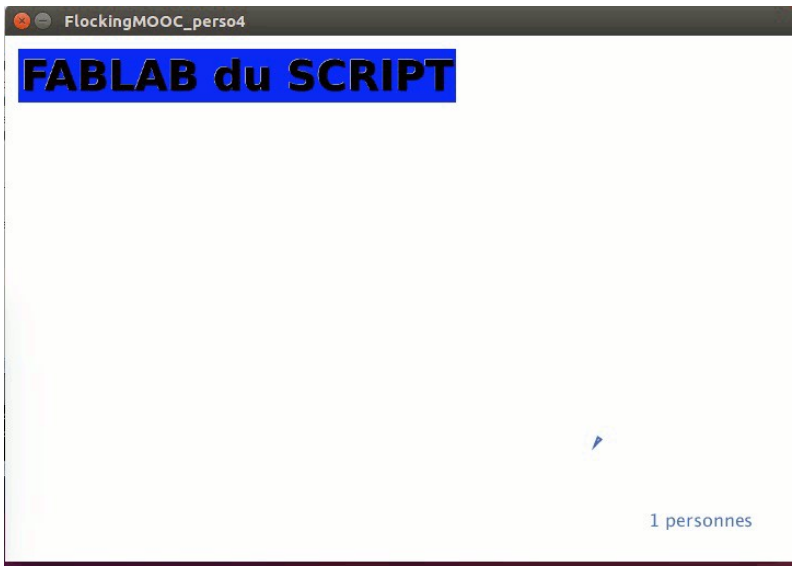
FABLAB du SCRIPT

Le programme nécessite que le programme que nous avons créé sur le logiciel Arduino «**nelson_interrupteur_TP3_perso.ino**» , soit téléversé dans notre nelson au préalable. Puis ensuite nous lançons, le programme , **FlockingMOOC_perso4.pde** , sur processing en appuyant sur la flèche en haut à gauche :

On doit avoir au départ la fenêtre suivante :



La fenêtre graphique qui a été ouverte a l'allure suivante :



Le bras de notre nelson a donc été poussé plusieurs fois, le résultat aurait été le même si on avait appuyé plusieurs fois sur une touche du clavier où tourne le logiciel **processing**. On considère qu'il y a deux personnes présentes dans le Fablab.

Ce document a été fait, à partir du cours proposé en 2016 par la plateforme fun : « **Fabriquer un objets connecté**» . Il a été mis en forme par moi-même, afin de permettre aux visiteurs, et utilisateurs de notre Fablab, et pourquoi-pas aux membres intéressés du service, de rapidement fabriquer un objet, qui bien facilement réalisable à mon sens, permet en même temps de s'initier voire de se perfectionner à plusieurs domaines : l'électronique, la programmation en C, la programmation en Java, la prise en main et l'utilisation des outils du Fablab, comme la paillasse électronique, avec le fer à souder, mais aussi le multimètre, et la connaissance d'au moins deux appareils électronique, comme l'**ESP8266** et le servomoteur. Comme aussi **l'imprimante 3D**, pour fabriquer le boîtier et les 2 pièces mobiles.

Il permettra aux utilisateurs de mieux comprendre le fonctionnement du réseau au niveau de l'interaction entre un serveur, que l'on ne voit pas, mais qui est là et qui envoi des informations, à notre montage, et un ordinateur, car c'est quand même un petit ordinateur que nous avons réalisé, ou du moins assemblé. Il montre aussi qu'il n'est pas nécessaire d'être un experts en électronique pour faire des montages, relativement avancés, avec la panoplie des circuits Arduino.

Concernant le temps à investir pour la réalisation d'un seul nelson Il faut tabler sur 9 heures. Avec les réglages que j'ai utilisé sur l'imprimante 3D , l'impression des pièces à pris un peu plus de 8 heures. Donc le mieux me semble t-il est déjà dès le départ de lancer l'impression des pièces, et pendant que celles-ci sont imprimés, procéder aux autre tâches comme l'installation du logiciel sur l'ordinateur, la préparation du moteur pour avoir le retour d'information, la préparation des différents fils que l'ont va utiliser pour les connexions.

Je suis à l'écoute de tout suggestion pour apporter des améliorations à ce document qui en est à sa deuxième version .