Mini-projet : réalisation d'un podomètre

Alexandre FAYE-BEDRIN

Résumé

Ce mini-projet guidé a pour finalité la réalisation d'un système embarqué, plus précisément un podomètre.

1 Introduction

Afin de mesurer l'activité physique de personnes, pour estimer leur dépense énergétique et ainsi constituer des bases de données utiles à la recherche, on souhaite réaliser un podomètre (système embarqué qui compte les pas de la personne qui le porte).

En premier lieu, il serait nécessaire de concevoir, fabriquer et programmer un prototype, démontrant la faisabilité du projet.

On dispose pour cela de matériel de prototypage, à savoir :

— une carte Arduino Nano
— un accéléromètre ADXL335
— un adaptateur de carte microSD

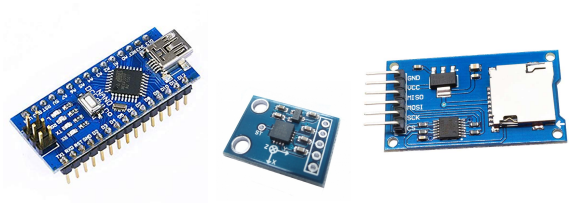

FIGURE 1 – Matériel

2 Arduino et liaison série

L'Arduino Nano est une carte de développement au format réduit, qui dispose d'un microcontrôleur ATmega328. Elle est programmable, comme toutes les cartes signalées compatibles Arduino, au moyen de l'environnement de développement consacré (*IDE Arduino*).

Un programme Arduino (écrit en C++) est constitué d'au moins 2 procédures, `setup` et `loop`. La première est appelée une fois lors de l'initialisation de la carte, la seconde est ensuite exécutée en boucle.

2.1 Utilisation basique

Le port série de la carte doit être initialisé avant toute utilisation, avec l'instruction `Serial.begin` informée de son taux de transfert en Bauds. On peut par la suite lire des données à l'aide de `Serial.read` et en envoyer au moyen de `Serial.print` et `Serial.println`. Un exemple d'utilisation de ces différentes méthodes est présenté dans le listing suivant :

```
1  int incomingByte = 0; // pour stocker les données reçues
2
3  void setup() {
4    Serial.begin(9600); // initialisation du port à 9600 Bauds
5  }
6
```

```
7  void loop() {
8    if (Serial.available() > 0) { // vérification qu'on a des données à lire
9      incomingByte = Serial.read(); // lecture d'un octet de données
10
11     Serial.print("J'ai reçu : ");
12     Serial.println(incomingByte, DEC); // renvoi des données reçues, on précise la
       base (DEC) pour le format des nombres
13   }
14 }
```

Listing 1 – Exemple d'utilisation du port série

On pourra recopier ce listing (ou le récupérer dans le répertoire de partage de l'établissement), et le modifier, pour comprendre l'utilisation de ces méthodes. Des exemples plus complexes sont également disponibles dans l'*IDE Arduino* (menu *Fichier→Exemples*).

Pour utiliser le port série avec l'ordinateur, on pourra se servir du moniteur série présent dans l'IDE (menu *Outils→Moniteur série* ou bouton en haut à droite de la fenêtre).

2.2 Évènements

Les cartes Arduino peuvent réagir à tout évènement arrivant sur le port série. Pour cela, il existe plusieurs méthodes, et l'une d'entre elles est de définit la méthode serialEvent comme dans le listing suivant :

```
1  String inputString = ""; // pour stocker les données reçues
2  bool stringComplete = false;  // pour indiquer que la ligne est complète
3
4  void serialEvent() {
5    while (Serial.available()) { // on vérifie qu'on a bien reçu des données
6      char inChar = (char)Serial.read();
7      inputString += inChar; // concaténation du caractère reçu à la ligne stockée
8      if (inChar == '\n') {
9        stringComplete = true;
10     }
11   }
12 }
13
14 void setup() {
15   Serial.begin(9600); // initialisation du port à 9600 Bauds
16
17   inputString.reserve(200);
18 }
19
20 void loop() {
21   if (stringComplete) {
22     Serial.println("Chaine reçue :");
23     Serial.print(inputString);
24     inputString = ""; // réinitialisation de la chaine
25     stringComplete = false;
26   }
27 }
```

Listing 2 – Exemple d'utilisation des évènements série

De même que précédemment, on pourra se familiariser avec la méthode en modifiant le code proposé.

2.3 Traitement de chaînes de caractères

Il est possible de réaliser quelques opérations sur les chaînes de caractères au-delà de la concaténation : on peut, par exemple, les découper, les comparer, regarder seulement leur début, *etc.*

Voici un exemple de traitement de chaîne et de transmission par port série :

```
1  void setup() {
2      Serial.begin(9600);
3      String chaine = "Ceci est une phrase.";
4      Serial.println(chaine);
5      String chaine2 = chaine;
6      Serial.println(chaine2);
7      chaine.toLowerCase();
```

```
8       Serial.println(chaine);
9       Serial.println(chaine2);
10      chaine.replace("ceci","cela");
11      if(chaine.startsWith("cela")) {
12          Serial.println(chaine.substring(0,10));
13      } else {
14          Serial.println(chaine.substring(10));
15      }
16      String a_enlever = "une";
17      chaine2.remove(chaine2.indexOf(a_enlever), a_enlever.length());
18      chaine2.toUpperCase();
19      Serial.println(chaine2);
20  }
```

Listing 3 – Exemple de traitements sur des chaînes de caractères

2.4 Réalisation : un mini-shell

Un shell est l'interface s'un système d'exploitation. Bien qu'une carte Arduino n'en possède pas réellement, on peut développer un programme capable de reconnaître des actions demandées par l'utilisateur au moyen de commandes, et d'y répondre.

Une façon pour le shell de recevoir des instructions est de le faire sous la forme de commandes textuelles, qui peuvent accepter des arguments (texte séparé par des espaces et situé après).

Voici un exemple d'utilisation du shell présent dans Windows (appelé Invite de commande) :

```
1  C:\>dir
2   Répertoire de C:\
3  10/08/2019  10:35    <DIR>          Program Files
4  18/10/2018  14:10    <DIR>          Users
5  27/09/2017  14:53    <DIR>          Windows
6  C:\>cd users
7  C:\Users>dir
8   Répertoire de C:\Users
9  22/06/2019  15:36    <DIR>          .
10 22/06/2019  15:36    <DIR>          ..
11 10/10/2018  10:00    <DIR>          u_1
```

Listing 4 – Session d'invite de commande Windows

On peut ici voir l'utilisation de 2 commandes, `dir` qui permet d'afficher la liste des fichiers présents dans un dossier et `cd` qui permet de se déplacer en prenant en argument le nom du prochain emplacement.

On propose ici la réalisation d'un shell capable de réaliser des traitements sur les chaînes de caractères à l'aide de commandes sur le port série de l'Arduino.

Les commandes demandées sont les suivantes (mais on pourra en rajouter) :

- `lower` : prend une chaîne en argument, la met en minuscule et l'affiche
- `hello` : affiche «Hello World ! »
- `help` : affiche cette liste de commandes, sans expliquer en détail le fonctionnement
- `sto` : si un argument est donné, le stocke dans la mémoire, sinon affiche la dernière chaîne stockée.
- `sto_rem` : prend une chaîne en argument, et efface les occurrences de cette chaîne présentes dans ce qui est enregistré par `sto`
- `upper` : prend une chaîne en argument, la met en majuscule et l'affiche

3 Accéléromètre

L'accéléromètre ADXL335 est un modèle analogique assez répandu dans l'industrie. On peut le trouver monté sur une carte à circuit imprimé destinée aux loisirs et au prototypage, telle que nous avons à disposition.

La carte dispose de 5 ports annotés :

1. Alimentation (5V)
2. Masse
3. Sortie axe x

4. Sortie axe y
5. Sortie axe z

On reliera donc les broches 5V et masse de la carte à leur homologues présentes sur l'Arduino.

Les sorties d'axe sont une tension image de l'accélération subie par l'accéléromètre. Elle est comprise entre $0V$ et $5V$, et peut représenter des accélérations comprises entre $-3g$ et $3g$.

Pour observer des variations d'accélération en temps réel, on peut observer les valeurs d'un axe *via* le port série. On relie donc la sortie de l'axe x à l'entrée analogique A0 de l'Arduino, et on peut par exemple utiliser ce code :

```
void setup() {
    Serial.begin(9600); // initialisation du port série
    Serial.println("Liaison série initialisée");
}

void loop() {
    Serial.print("Valeur : ");
    Serial.println(analogRead(A0), 10); // affichage de la valeur récupérée en base 10
    delay(100); // attente de 100ms avant le prochain affichage
}
```

Listing 5 – Affichage sur port série des valeurs issues de l'accéléromètre

En poursuivant, on pourra brancher les 2 autres sorties d'axes sur des entrées analogiques et ainsi connaître les 3 valeurs d'accélération simultanées. On réalisera le programme capable d'afficher ces 3 valeurs en même temps.

En faisant pivoter l'accéléromètre, on devrait voir varier les valeurs des sorties d'axe à cause de la gravité.

4 Lecteur de carte microSD

Ce lecteur permet à une carte Arduino d'effectuer une gestion des fichiers sur une carte microSD formatée en FAT ou FAT32. Il possède quelques limitations, et ne sait s'occuper que des fichiers ayant un nom au format DOS 8.3 (c'est-à-dire possédant un nom de 8 caractères au maximum, et une extension de nom de 3 caractères au plus, par exemple : patate26.txt).

Il est relié à la carte Arduino par un bus SPI, dont il n'est pas utile d'avoir les détails puisque la bibliothèque fournie dans l'*IDE* en fait abstraction.

On câblera entre les deux cartes comme suit :

Port lecteur	Port Arduino
VCC	$3.3V$
GND	GND
CS	4
SCK	13
MOSI	11
MISO	12

On pourra essayer différentes opérations sur les fichiers en s'inspirant de ce programme :

```
#include <SD.h>

File myFile;

void setup() {
  Serial.begin(9600);
  Serial.print("Initialisation de la carte SD...");
  if (!SD.begin(4)) { // 4 correspond à CS (Chip Select)
    Serial.println("échec.");
    while (1);
  }
  Serial.println("réussi !");
  if(SD.exists("test.txt") {
    Serial.println("le fichier existe déjà");
    // si on voulait le supprimer (pour le vider), on utiliserait SD.remove
```

```
16    } else {
17      Serial.println("le fichier n'existe pas encore");
18    }
19    // ouverture du fichier en mode écriture
20    // note : on ne peut ouvrir qu'un seul fichier à la fois
21    myFile = SD.open("test.txt", FILE_WRITE);
22    if (myFile) {
23      Serial.print("Écriture de test.txt...");
24      myFile.println("test 1, 2, 3.");
25      // fermeture du fichier, essentiel pour enregistrer ses modifications
26      // si on veut l'enregistrer sans le fermer, on pourra utiliser myFile.flush()
27      myFile.close();
28      Serial.println("fait");
29    } else {
30      Serial.println("erreur lors de l'ouverture de test.txt");
31    }
32    // réouverture du fichier pour lecture:
33    myFile = SD.open("test.txt");
34    if (myFile) {
35      // on lit le fichier jusqu'à la fin:
36      while (myFile.available()) {
37        Serial.write(myFile.read());
38      }
39      myFile.close();
40    } else {
41      Serial.println("erreur lors de l'ouverture de test.txt");
42    }
43  }
```

Listing 6 – Manipulation d'un fichier

On réalisera un programme capable d'enregistrer une fois des valeurs issues de l'accéléromètre, au format x,y,z, dans un fichier data.csv

5 Podomètre

On dispose à présent de tous les éléments nécessaires à la réalisation du podomètre.

5.1 Partie Arduino

Au moyen des différentes possibilités explorées dans les parties précédentes, on écrira un programme Arduino qui :

- toutes les 30ms, lit les données issues de l'accéléromètre et les ajoute dans un fichier de données sous la forme d'une ligne au format x,y,z
- reconnaît au moins deux commandes, nommées selon vos choix :
  - lecture du fichier de données
  - suppression du fichier de données
- lorsque l'utilisateur entre une commande sur le port série, interrompt l'enregistrement des données

5.2 Récupération des données

Python est un langage qui dispose d'une extensive bibliothèque standard et communautaire. Son module pyserial permet d'utiliser les ports série de l'ordinateur, et en particulier pour communiquer avec une carte Arduino (un peu comme le moniteur série).

Voici un script d'exemple :

```
1  import serial
2  import serial.tools.list_ports
3  print("Liste des ports COM :")
4  l = list((serial.tools.list_ports.comports()))
5  print(*l, sep="\n")
6  p = input("Entrez le nom du port COM à utiliser : ")
7  f = open("fichier.txt","wb") # ouverture de fichier.txt en mode écriture binaire
8  with serial.Serial(p, timeout=1) as ser: # un temps d'attente maximal d'une seconde
9      print("Port initialisé")
10     ser.write(bytes(s+"\n",'utf-8'));
```

```
11      l=ser.readline()
12      while l:
13          print(l)
14          f.write(l) # enregistrement de la ligne reçue
15          l=ser.readline()
```

Listing 7 – Communication série en Python

Il devrait être possible de l'utiliser avec le mini-shell réalisé plus tôt, ainsi qu'avec le programme du podomètre.

On réalisera un programme qui récupérera automatiquement les données du podomètre sur l'ordinateur, puis videra la carte SD du podomètre afin de faire de la place pour de futurs enregistrements.

5.3 Traitement des données

À ce point d'avancement, on peut récupérer les données d'accélération sur ordinateur, mais on ne sait toujours pas combien de pas ont été faits par le porteur.

On va devoir utiliser des techniques de traitement de signal pour arriver à ces fins.

La première étape est de charger les données en mémoire. Voici un script Python capable de charger correctement des données au bon format pour les traiter ensuite :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  f = "data.csv"
5  data = np.loadtxt(f,dtype=int,delimiter=",").transpose()
6  # la variable data contient à présent les données
7  plt.plot(data[0]) # affichage d'une courbe, l'indice pourra varier de 0 à 2 (pour
       les 3 axes)
8  plt.show()
```

Listing 8 – Chargement et affichage de données

On voit que le signal est irrégulier, et on ne sait pas forcément à quel axe de l'accéléromètre la courbe correspond.

Une manière de le rendre un peu plus exploitable est de :

1. le recentrer
2. calculer sa norme à chaque instant

Pour cela, une bibliothèque est fournie. On pourra ajouter au script précédent :

```
1  from bibliotheque import *
2  ma=np.zeros(data.shape)
3  for i in range(3):
4      ma[i]=moving_average(data[i],150) # on pourra changer
5  # ma contient la moyenne glissante, soit la moyenne sur un petit intervalle
6  v=f-ma # v contient à présent le signal recentré
7  plt.plot(data[0])
8  plt.plot(v[0])
9  plt.plot(ma[0])
10 plt.show()
11 norm=norm_acc(v)
12 plt.plot(norm)
13 plt.show()
```

On dispose à présent d'un signal qui ne dépend presque que des mouvements du porteur du podomètre. On peut remarquer qu'un pas de l'utilisateur correspond à un pic du signal, il suffirait donc de détecter ces pics pour connaître l'instant et le nombre des pas.

Une méthode a été conçue spécifiquement pour connaître l'emplacement de pics d'un signal : il s'agit de la dilatation.

La dilatation est une transformation du signal qui consiste à prendre, en chaque point, le maximum du signal sur l'intervalle environnant.

Une fonction de dilatation est fournie dans la bibliothèque sous le nom de `dilate`.

On peut l'utiliser comme suit :

```
norm_dil=dilate(norm, 5) # 5 est le nombre de points pris en compte pour le maximum
# à changer pour un meilleur résultat
plt.plot(norm,label="norme")
plt.plot(norm_dil,label="dilatation")
plt.show()
pas=[]
for i in range(len(norm)):
    if norm[i]==norm_dil[i]:
        pas.append(1)
    else:
        pas.append(0)
plt.plot(pas)
plt.show()
```

À présent, la liste pas contient les instants où le porteur a effectué un pas. Pour éviter de compter des pas alors que l'utilisateur ne bouge pas, on pourrait définir un seuil au-dessus duquel le signal doit être pour qu'un pas soit détecté. On peut calculer le nombre de pas total à partir de cette liste, et l'afficher.

Le système est maintenant fonctionnel, il faut effectuer des essais pour s'assurer que tous les paramètres sont bien choisis.