Mesures d'accélérations et système d'acquisition par microcontrôleur Arduino Uno

Solène Kojtych

Note technique

MESURES D'ACCÉLÉRATIONS ET SYSTÈME D'ACQUISITION PAR MICROCONTRÔLEUR ARDUINO UNO

Solène Kojtych
Laboratoire d'Analyse Vibratoire et Acoustique (LAVA)
Département de Génie Mécanique, École Polytechnique de Montréal

Table des matières

Liste des figures

CHAPITRE 1 Objectif

Comment concevoir un système d'acquisition simple et peu coûteux permettant de mesurer des accélérations ?. Cette question s'est posée lors de mon projet de maîtrise, à l'École Polytechnique de Montréal. Le travail effectué pour répondre à cette question est présenté dans cette note technique.

L'outil proposé repose sur un microcontrôleur Arduino UNO qui permet d'acquérir les mesures faites par un accéléromètre. Le microcontrôleur est relié à un ordinateur qui permet de traiter et d'afficher des graphiques de résultats à partir de scripts en langage Python 2.

Plus précisément, la méthodologie utilisée permet d'atteindre les critères suivants :

- utiliser une **fréquence d'échantillonnage stable jusqu'à 1000 Hz au moins**,
- réaliser des mesures d'une **durée d'acquisition d'au moins 20 secondes**,
- mesurer des **accélérations comprises entre -16 g et 16 g**,
- sauvegarder les mesures dans un fichier texte,
- représenter graphiquement les accélérations mesurées.

Le public visé par cette note est supposé connaître des bases de programmation informatique, bien que la démarche proposée requiert peu de modifications au sein des scripts. Une connaissance du langage Python 2 et/ou C est également bienvenue.

La première partie de cette note permet de se familiariser avec les différents composants du système d'acquisition et donne les étapes de mise en place basanas. Pour chaque étape, le montage à réaliser est présenté, le protocole de mesure, les scripts pour l'accéléromètre et l'interfaçage en Python sont fournis ainsi que des feuilles de mesures le cas échéant. Les protocoles sont mis en relief dans des encarts jaunes et les scripts sont illustrés à partir d'un exemple à chaque étape, dans des encarts bleus.

La deuxième partie de la note donne des informations théoriques supplémentaires ou plus techniques pour le lecteur intéressé, notamment concernant les choix d'implémentation et la technique d'échantillonnage utilisée.

Bonne lecture !

CHAPITRE 2 Démarche

2.1 Prise en main et préparation du matériel

Le matériel nécessaire *a minima* pour suivre les indications de cette note est le suivant :

- ✓ accéléromètre ADXL326 de la marque Analog Devices,
- ✓ microcontrôleur Arduino UNO révision 3 et son câble USB,
- ✓ environnement de développement de l'Arduino (IDE),
- ✓ interpréteur Python 2 et module *pySerial*,
- ✓ matériel électronique de base pour les connexions (fils, platine de prototypage).

Les modules Python *time*, *numpy*, *os* et *matplolib* sont également nécessaires mais sont généralement installés directement avec Python.

Plusieurs composants non nécessaires mais fortement recommandés ont été utilisés dans cette note et sont décrits dans les paragraphes suivants :

- ✓ accéléromètre ADXL326 monté sur la carte électronique du fournisseur Adafruit,
- ✓ kit de démarrage pour l'Arduino UNO,
- ✓ IDE Spyder pour le développement en langage Python,
- ✓ fer à souder,
- ✓ plusieurs plaquettes de prototypage.

Le lecteur familier avec le matériel énoncé est invité à consulter directement la prochaine section. La présentation du matériel et les recommandations d'achat données ci-après ont pour seul but de guider le lecteur inexpérimenté. Il est à noter que les scripts de cette note ont été testés uniquement sur un environnement d'exploitation Linux. Le coût du système d'acquisition se réduit principalement au coût de l'accéléromètre et du microcontrôleur Arduino.

Accéléromètre ADXL326 $\pm 16g$

L'accéléromètre analogique utilisé est le modèle ADXL326 [1] $\pm 16g$ de la marque Analog Devices. Il possède trois axes et permet donc de mesurer des accélérations selon trois directions (X,Y et Z sur la photo 2.1). Il peut être acheté directement monté sur une petite carte

1. https://www.analog.com/en/products/adxl326.html

électronique, pour faciliter les connexions, par exemple par le fournisseur Adafruit [2]. Seuls les broches de connexion (en anglais *pin*) sont alors à souder. Au moment de la rédaction de cette note, son prix était d'environ 25 $CAD. Pour plus d'informations, se référer à la notice de l'appareil [1]

Microcontrôleur Arduino UNO

Le microcontrôleur Arduino, modèle UNO révision 3 [3] a été utilisé (figure 2.2). Il s'agit d'une carte électronique comportant plusieurs entrées et sorties et permettant d'acquérir les données issues de capteurs, de les traiter et de les transmettre vers un ordinateur via un câble USB. Le kit de démarrage [4] est un bon point de départ pour les nouveaux utilisateurs : il contient la carte électronique avec le câble USB, plusieurs composants électroniques de base (fils de connexion, condensateurs...) et des capteurs (température, lumière...) ainsi qu'un tutoriel avec plusieurs projets.

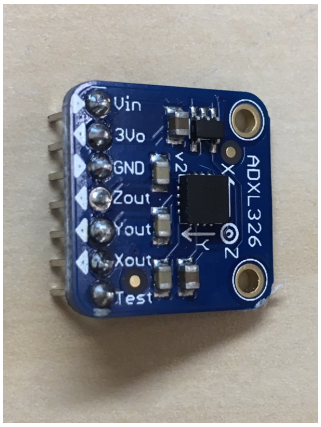


Figure 2.1 accéléromètre ADXL326 de la marque Analog Device monté sur plaquette de prototypage Adafruit

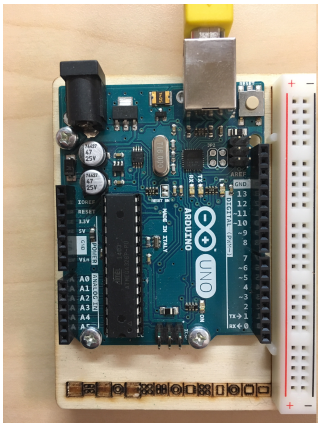


Figure 2.2 microcontrôleur Arduino UNO révision 3 avec la carte de prototypage du kit de démarrage

Environnement de développement pour l'Arduino

La carte électronique est contrôlée pas des scripts en langage C qui sont directement chargés sur la carte à partir de l'ordinateur via un environnement de développement gratuit et téléchargeable sur le site Arduino [5] (figure 2.3). Une version entièrement en ligne de cet environnement est également disponible. L'installation y est détaillée, ainsi que la procédure

2. https://www.adafruit.com/product/1018
3. https://store.arduino.cc/usa/arduino-uno-rev3
4. https://store.arduino.cc/usa/arduino-starter-kit
5. https://www.arduino.cc/en/Main/Software

de téléversement des scripts sur la carte Arduino. De nombreux tutoriels et codes ouverts y sont également disponibles.

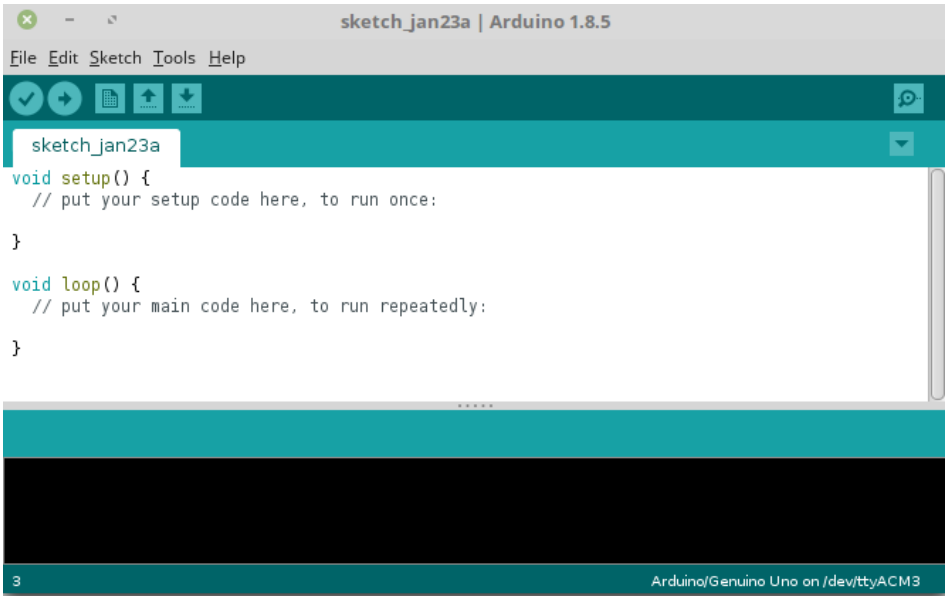


Figure 2.3 IDE de l'Arduino pour le langage C

Interpréteur Python

L'environnement de développement pour l'Arduino est minimaliste et ne permet pas de représenter graphiquement les résultats. Pour cette raison, des scripts en langage Python ont été réalisés pour faciliter le traitement des résultats. Certains paramètres de ces scripts doivent être modifiés dans les étapes suivantes, la démarche suivie sera alors mentionnée explicitement.

Pour l'exécution des scripts il est nécessaire d'installer un interpréteur pour Python 2 sur l'ordinateur. Les étapes ne seront pas détaillées ici mais toute l'information peut être trouvée sur le site de Python [6]. L'installation du module *pyserial* [7] est également nécessaire. L'utilisation d'un environnement de développement intégré (IDE), tel que l'outil librement accessible Spyder [8] qui a été utilisé pour ce projet (figure 2.4), permet avantageusement de modifier les paramètres des scripts ainsi que d'afficher et de post-traiter les résultats obtenus.

6. https://www.python.org/
7. https://pythonhosted.org/pyserial/
8. https://docs.spyder-ide.org/

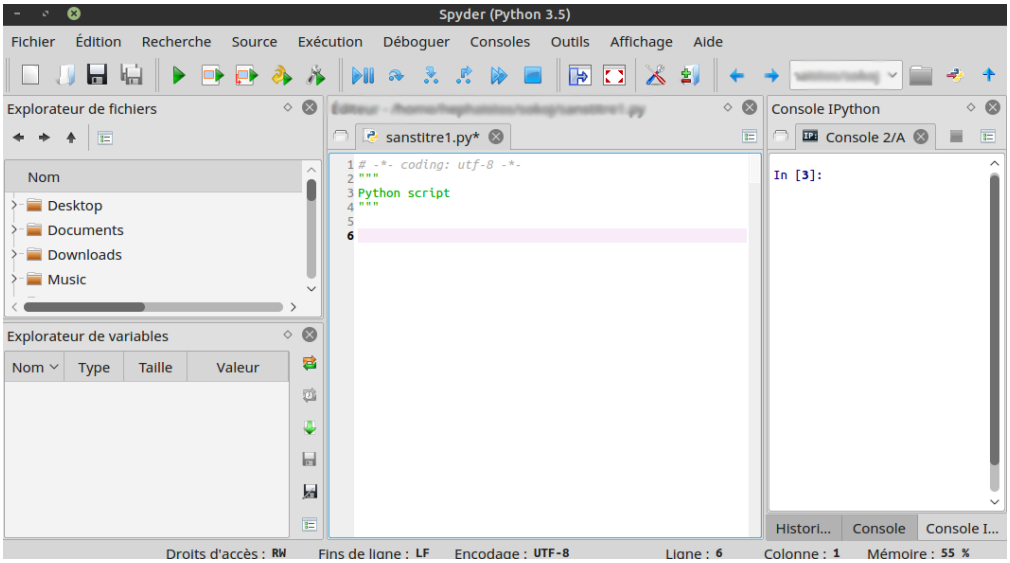


Figure 2.4 IDE Spyder pour le langage Python 2

Matériel électronique supplémentaire

Pour connecter l'accéléromètre au microcontrôleur, des fils de connexion sont nécessaires ainsi qu'au moins une platine de prototypage (figure 2.5). Ces éléments sont contenus dans le kit de démarrage mentionné précédemment.

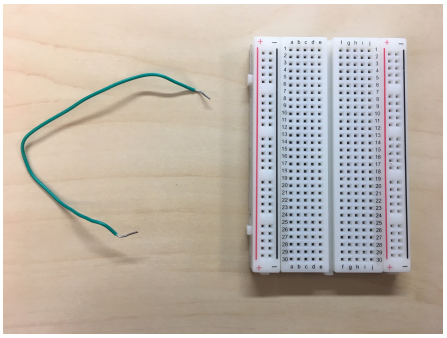


Figure 2.5 fil de connexion et platine de prototypage

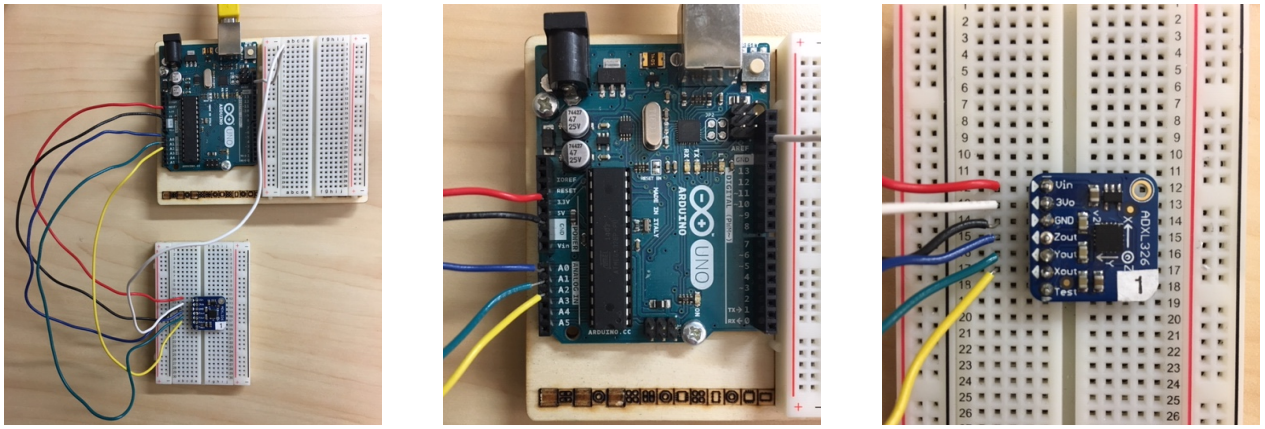
2.2 Calibration de l'accéléromètre

Objectif : obtenir le biais et la sensibilité de l'accéléromètre

La calibration permet de déterminer certains paramètres (le biais et la sensibilité), propres à l'accéléromètre utilisé et nécessaires par la suite pour mesurer correctement des accélérations. Plus d'informations théoriques sur ce sujet sont données dans la troisième partie de la note.

Attention

Avant toute modification de montage électrique, vérifiez que l'Arduino est hors tension.



(a) Montage global (b) Branchements côté Arduino (c) Branchements côté accéléromètre

Figure 2.6 montage de calibration

Préparation

Protocole : préparation du montage de calibration

1. Connecter l'accéléromètre à la carte Arduino selon la configuration de la figure 2.7.
2. Ouvrir le script *calibration_ADXL326.ino* (annexe A.1) dans l'IDE Arduino.

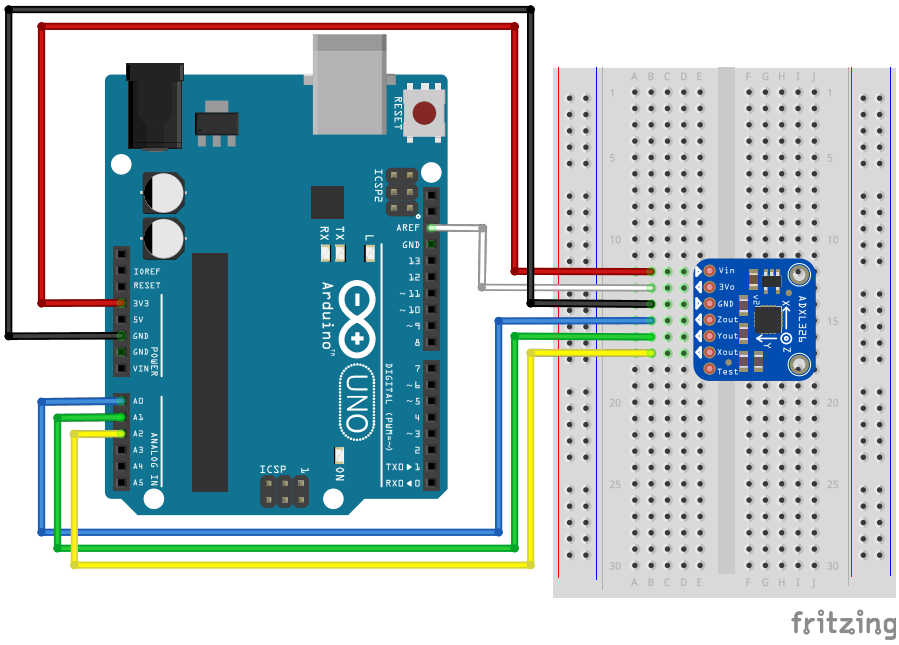
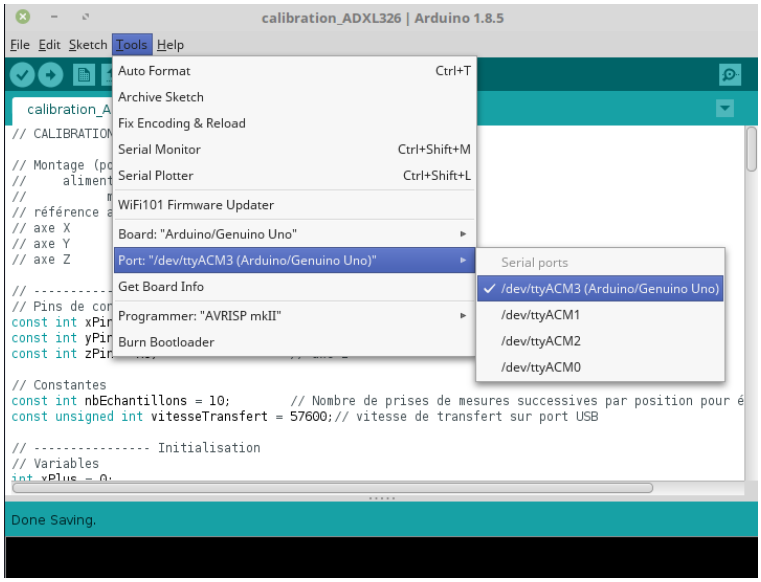


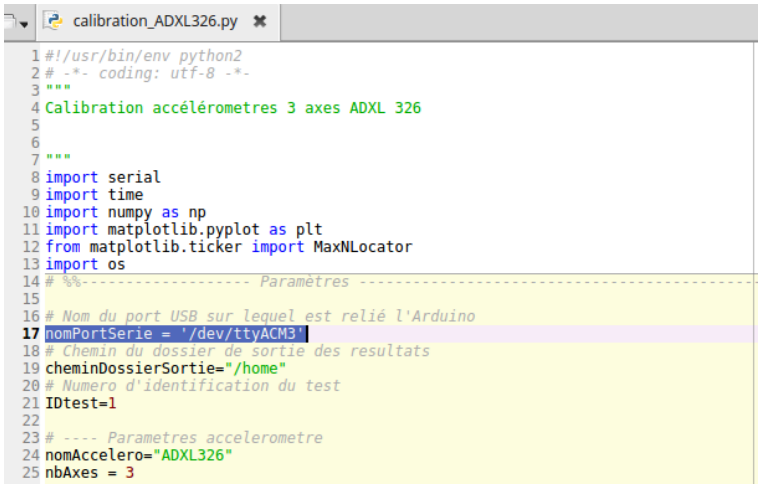
Figure 2.7 schéma des branchements pour le montage de calibration

3. Brancher l'Arduino à l'ordinateur avec le câble USB et téléverser le script. Vérifier qu'il n'y a pas d'erreurs au moment du téléversement en contrôlant les informations qui apparaissent dans l'IDE.
4. Ouvrir le fichier *calibration_ADXL326.py* (annexe A.2) avec un éditeur de code (Spyder par exemple). Les modifications à apporter au script sont situées dans la partie « paramètres » :
   - le nom du port USB sur lequel est relié l'Arduino doit être modifié ; pour le connaître, il suffit de lancer l'IDE Arduino, puis sous le menu *Tools* > *Port* le port utilisé par l'Arduino est coché (voir figure 2.8). Reporter le nom du port dans le script *calibration.py* ;
   - le chemin du fichier de résultat peut également être modifié ainsi que le numéro d'identification du test. Le script va créer, s'il n'existe pas déjà, un dossier *resultats_calibration* en suivant le chemin indiqué contenant un fichier de résultat au format *.txt* ;
5. Préparer une zone de calibration avec une surface plane utilisable des deux côtés (une table par exemple), et une surface verticale perpendiculaire à la surface plane (voir

figure 2.9).



(a) visualisation du nom du port dans l'IDE Arduino



(b) paramètre correspondant dans le script *calibration_ADXL326.py*

Figure 2.8 exemples de paramètres du script de calibration

Paramètres du script de calibration

Les valeurs des paramètres donnés pour notre exemple sont visibles sur la figure 2.8.

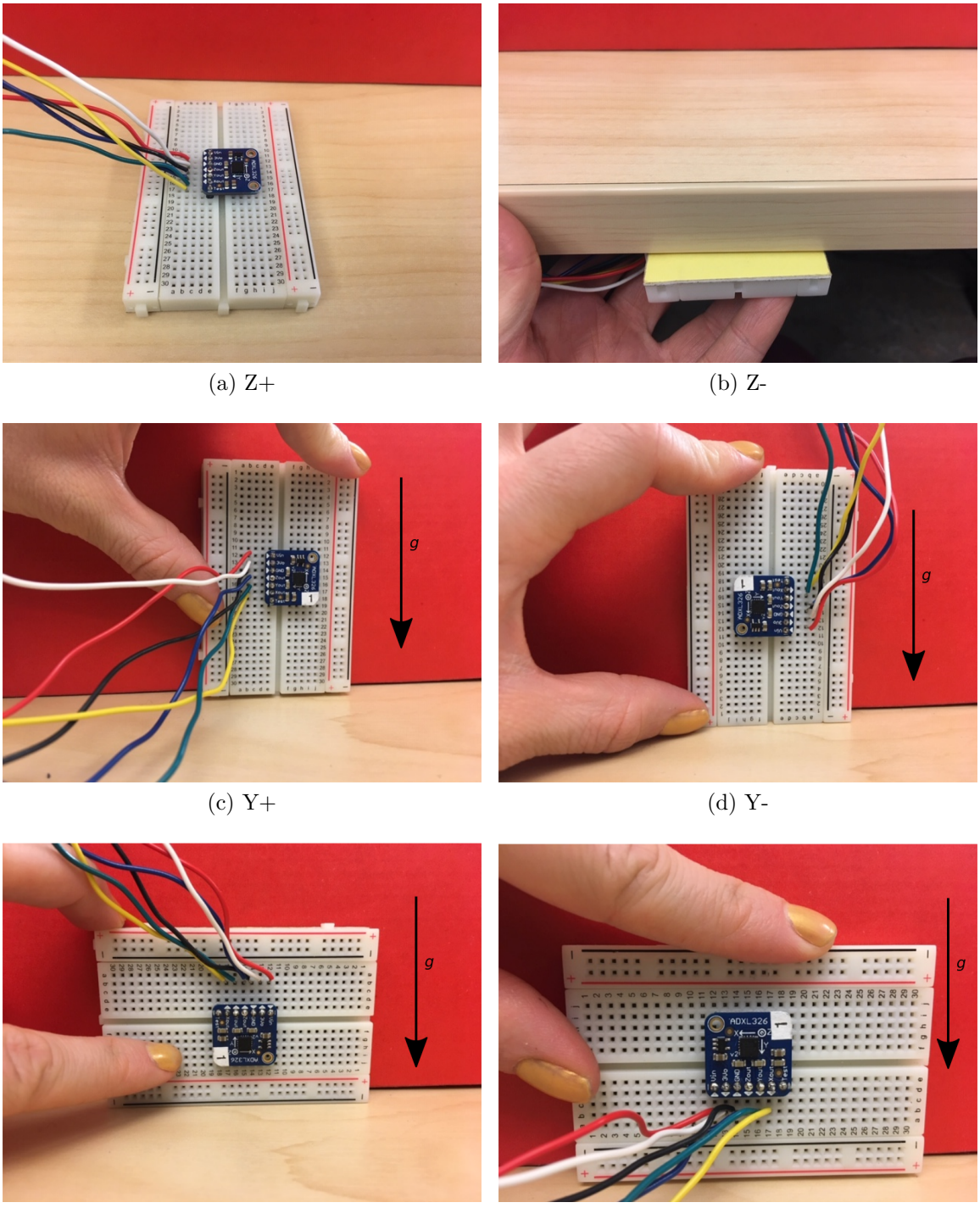
Acquisition

Une feuille de mesure telle que celle présentée en annexe A.3 peut être utilisée pour relever les valeurs issues de la calibration et vérifier le bon déroulement du test.

Protocole : calibration

1. Brancher l'Arduino à l'ordinateur via le câble USB.
2. Exécuter le script Python *calibration_ADXL326.py* et suivre les indications :
   - pour le positionnement de l'accéléromètre selon les différents axes, utiliser un niveau ou une référence verticale et horizontale (voir les différents positionnements figure 2.9). La consigne "Axe X+ vers le plafond" signifie que l'axe X positif doit être orientée selon le vecteur $-g$;
   - après mise en place de l'accéléromètre, attendre que le prompteur » s'affiche puis taper « 1 » et « entrer » pour valider ;
   - à la fin de la procédure de calibration de calibration, vérifier l'allure des courbes à l'écran ; elles doivent être linéaires (voir le graphique figure 2.10a).
3. Après exécution du script, le script Python affiche les valeurs mesurées pour chaque axe pour +1g, 0g et 1g. Ces valeurs sont à reporter dans la fiche de mesures et peuvent être retrouvées dans le fichier de résultats (figure 2.10b).
4. Débrancher l'Arduino du PC.

Pour calibrer le plus précisément possible l'accéléromètre, il est conseillée de répéter trois fois cette procédure, en débranchant le câble USB et les fils de connexion du montage entre deux procédures. La feuille de mesures permet de calculer automatiquement les résultats issus de ces trois tests.



(a) Z+ (b) Z-

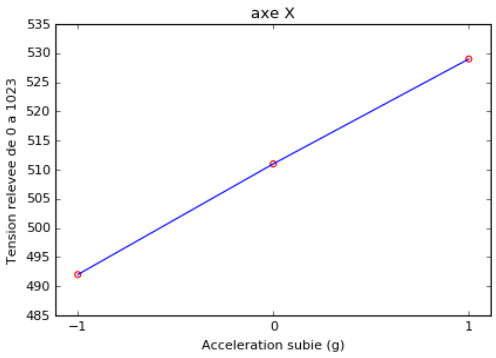
(c) Y+ (d) Y-

(e) X+ (f) X-

Figure 2.9 positionnement de l'accéléromètre suivant la procédure de calibration

Exécution de la procédure de calibration

Une façon simple de positionner l'accéléromètre lors de la calibration est présentée sur la figure 2.9. Un exemple de graphique obtenu à la fin de la procédure est donné sur la figure 2.10a : la valeur numérique fourni à l'ordinateur par le système d'acquisition est représenté en fonction de l'accélération subie par l'accéléromètre (voir détails dans la dernière section). On constate que la courbe est bien linéaire. Après 3 exécutions du protocole de calibration, un exemple de feuille de mesures complétée est présenté sur la figure 2.11 : les données remplies par l'utilisateur sont les cases en jaune.



(a) Graphique obtenu pour l'axe X

```
# Calibration ADXL326 run 1
# ---- MESURES ----
Axe          -1g          0g          1g
axe X         492         511         530
axe Y         488         509         526
axe Z         503         521         541
```

(b) Fichier de résultat obtenu

Figure 2.10 résultats obtenus après exécution de la procédure de calibration

	A	B	C	D	E	F	G	H	I	J
1	**Calibration d'accéléromètre**									
2										
3	**Identification du test**									
4	**Modèle accéléromètre**	ADXL 326								
5	**Nombre d'axes**	3								
6	**Date**									
7										
8	**Valeurs relevées de 0 à 1023**									
9	**Axe**	Axe X			Axe Y			Axe Z		
10	**N° test\Accélération (g)**	**-1**	**0**	**1**	**-1**	**0**	**1**	**-1**	**0**	**1**
11	**1**	492	511	530	488	509	526	503	521	541
12	**2**	492	510	529	489	509	526	503	522	541
13	**3**	492	510	530	489	508	527	503	522	541
14										
15										
16	**Détermination des caractéristiques**									
17	**Valeur moyenne μ**	492,00	510,33	529,67	488,67	508,67	526,33	503,00	521,67	541,00
18	**Médiane**	492,00	510,00	530,00	489,00	509,00	526,00	503,00	522,00	541,00
19	**Écart type σ**	0,00	0,58	0,58	0,58	0,58	0,58	0,00	0,58	0,00
20	**Sensibilité moyenne**	18,8			18,8			19,0		
21	**Biais moyen à 0 g**	510,7			507,9			521,9		
22	**Biais moyen à 1 g**	529,5			526,7			540,9		
23	**Biais moyen à -1g**	491,8			489,1			502,9		
24	**Coefficient de détermination R²**	1,00			1,00			1,00		

Figure 2.11 feuille de mesures complétée après trois exécutions du protocole de calibration. Mesures relevées sur fond jaune et valeurs de sensibilité et biais à utiliser par la suite sur fond vert.

Résultats

Les résultats obtenus sont les valeurs situées dans le deuxième tableau de la feuille de calibration, sur fond vert, aux lignes intitulés *Sensibilité* et *Biais* (voir figure 2.11). Pour chaque axe, les sensibilités et biais doivent être relevés pour la prochaine étape. La procédure de calibration n'a pas à être répétée pour chaque nouvelle acquisition dès lors que les valeurs de sensibilité et biais sont connues.

Résultats de la procédure de calibration

Pour notre exemple, les valeurs retenues sont résumées dans le tableau 2.1.

Tableau 2.1 résultats issus de la procédure de calibration

paramètres - axe	**axe X**	**axe Y**	**axe Z**
sensibilité	18,8	18,8	19,0
biais à 0 g	510,7	507,9	521,9
biais à 1 g	529,5	526,7	540,9
biais à -1 g	491,8	489,1	502,9

2.3 Prise de mesure

Objectif : mesurer des accélérations sur une durée T et représenter graphiquement le signal obtenu

Préparation

Le montage à réaliser est le même que le montage de calibration précédent (figure 2.6a).

Protocole : préparation du montage de prise de mesures

1. Fixer l'accéléromètre sur l'objet dont on souhaite mesurer l'accélération ; il est préférable d'un des axes de l'accéléromètre soit orienté selon la direction de la gravité.
2. Connecter l'accéléromètre à la carte Arduino selon la configuration de la figure 2.7.
3. Ouvrir le script *priseMesures_ADXL326.ino* (annexe B.1) dans l'IDE Arduino, modifier si besoin la durée d'acquisition et la fréquence d'échantillonnage souhaitée dans la section « paramètres » en début de script (voir figure 2.13).
4. Brancher l'Arduino à l'ordinateur avec le câble USB et téléverser le script. Vérifier qu'il n'y a pas d'erreurs au moment du téléversement en contrôlant les informations qui apparaissent dans l'IDE.
5. Ouvrir le fichier *priseMesures_ADXL326.py*. Les modifications à apporter au script sont situées dans la partie « paramètres » (voir figure 2.14) :
   - le port sur lequel est branché l'Arduino doit être renseigné comme pour le script de calibration ;
   - le chemin du dossier de résultats peut être modifié ainsi que le numéro du test et le nombre de chiffres après la virgule pour l'écriture des résultats ;
   - les valeurs de sensibilité pour chaque axe doivent directement être complétées à partir des résultats de la calibration ;
   - les valeurs de biais pour chaque axe sont à choisir en fonction de l'orientation de l'accéléromètre, selon les indications de la figure 2.12 ;
   - le nombre de bips sonores précédant le démarrage de l'acquisition peut être modifié.

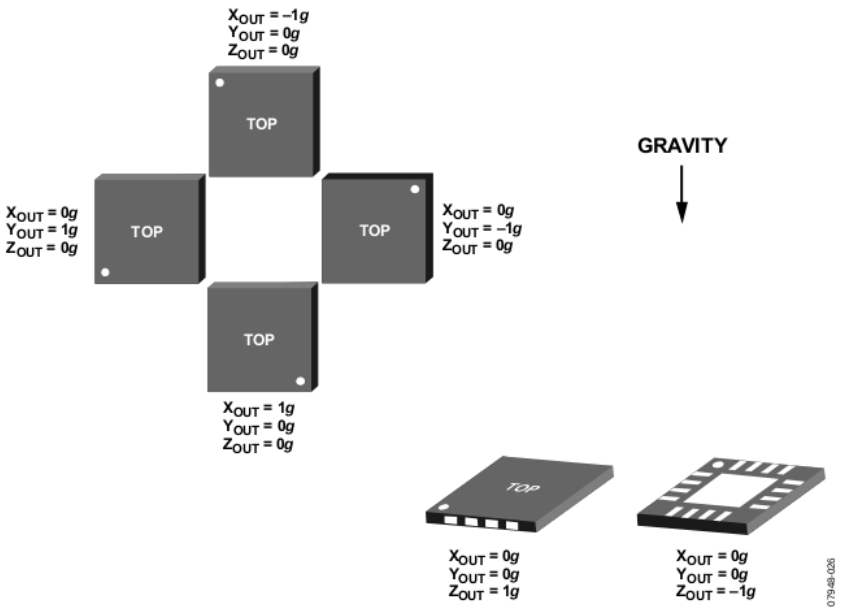


Figure 2.12 biais à utiliser pour chaque axe en fonction de l'orientation de l'accéléromètre. Issue de [1].

Paramètres du script de prise de mesures

Par exemple, pour une prise de mesure d'une durée de 20 secondes et une fréquence d'échantillonnage de 1000 Hz, les paramètres dans le script *priseMesures_ADXL326.ino* sont modifiés comme sur la figure 2.13.

Ensuite, pour une prise de mesure avec l'accéléromètre à plat(axe Z+ selon $-g$), les axes X et Y correspondront à une sortie de 0 g et l'axe Z à une sortie de 1 g d'après la figure 2.12. À partir des résultats de la calibration (tableau 2.1), les paramètres de biais adéquats sont choisis pour l'acquisition et renseignés dans les paramètres du script *priseMesures_ADXL326.py*, comme le montre la figure 2.14.

priseMesures_ADXL326

```
// DATA-ACQUISITION (3-AXIS ACCELEROMETER ADXL326)

//Wiring (accelerometer pin -------- Arduino pin)
// power                      :     Vin     --------   power 3.3V
// ground                     :     GND     --------   GND
//  voltage reference  :     3Vo     --------   AREF
// X axis                     :      Xout  --------   A0
// Y axis                     :      Yout  --------   A1
// Z axis                     :       Zout --------   A2

//--------------- PARAMETERS ----------------------
const float fs = 1000;       // Sampling frequency in Hz
const int T = 20;                  // Sampling duration in seconds
//-------------------------------------------------------------
```

Figure 2.13 modification de la durée d'acquisition dans le fichier *priseMesures_ADXL326.ino*

priseMesures_ADXL326.py

```
# -*- coding: utf-8 -*-
"""
Prise de mesure générique pour accéléromètre
"""

import serial
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# %%------------------- Paramètres ----------------------------------------------
# Nom du port USB sur lequel est relié l'Arduino
nomPortSerie = '/dev/ttyACM3'

# ---------- Fichier de résultat
# Chemin du dossier de sortie des resultats
chemin_fichier = "/home/sokoj/Desktop/codes_Arduino/Materiel_lacher/Scripts"
# Nombre de chiffres après la virgule pour l'écriture des résultats
nbChiffres = 3

# ---- Parametres accelero
nomAccelero="ADXL326_1"
nbAxes = 3
sensibilite = [18.8,18.8,19]   #  (entre 0 et 1023) pour les axes X,Y,Z
biais = [510.7,507.9,540.9]            # (entre 0 et 1023) pour les axes X,Y,Z

# ---- Parametres acquisition
nbBips = 4                     # nombre de bips avant le debut de l'acquisition
```

Figure 2.14 exemples de paramètres du script de prise de mesures Python avec report des valeurs issues de la calibration

Acquisition et résultats

Protocole : prise de mesures

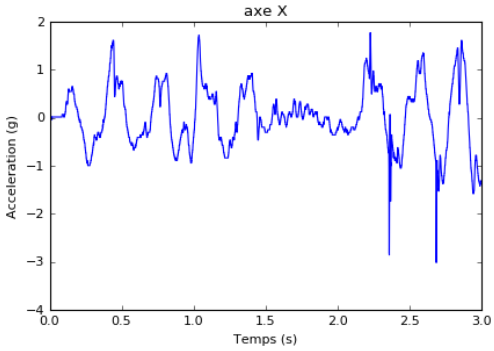
1. Brancher l'Arduino à l'ordinateur via le câble USB.

2. Exécuter le script Python *priseMesures_ADXL326.py* et suivez les instructions :
   - quand la consigne apparaît, appuyer sur la touche « 1 » puis sur « entrée » pour lancer l'acquisition ; les mesures défilent à l'écran ;
   - après le nombre de bips sonores courts définis dans les paramètres, un bip long se produit et la prise de mesures débute ;
   - la représentation graphique du signal obtenu sur les différents axes de l'accéléromètre apparaît à la fin de l'acquisition (voir extrait figure 2.15a) ;
   - au chemin indiqué pour les résultats, un dossier *resultats_acquisition* est créé contenant un dossier *nomAccelerometre_IDtest* à partir des paramètres fournis. Pour chaque acquisition, un fichier *mesures_numeroAcquision.txt* est produit, contenant 4 colonnes séparées par le symbole « ; » : le temps en micro secondes, et les mesures brutes relevées sur les 3 axes (voir explications dans la troisième section de cette note). Un fichier *accelerations_numeroAcquisition.txt* est également produit avec la même structure, mais les données pour chaque axe sont cette fois les accélérations en *g*. La quantité *numeroAcquisition* est incrémentée automatiquement ;
   - le script peut être exécuté plusieurs fois pour réaliser différentes acquisitions.
3. Débrancher l'Arduino du PC.

Exécution de la procédure de prise de mesures

Un exemple de graphique issu de la prise de mesures est visible sur la figure 2.15 ainsi que le fichier de résultat associé en accélérations figure 2.15b.

Le bruit est inévitable sur les sorties de l'accéléromètre. Pour estimer sa valeur, il est possible de réaliser une acquisition avec l'accéléromètre immobile. Les fluctuations du signal informent sur l'ordre de grandeur du bruit (voir figure 2.16).



(a) Accélérations mesurées sur l'axe X en g

Temps;axe X;axe Y;axe Z
992;0.016;-0.048;0.058
1992;0.016;-0.048;0.058
2992;0.016;0.005;0.058
3992;0.016;-0.048;0.058
4992;0.016;-0.048;0.058
5992;0.016;-0.048;0.058

(b) fichier de résultats

Figure 2.15 représentation graphique et textuelle des accélérations mesurées

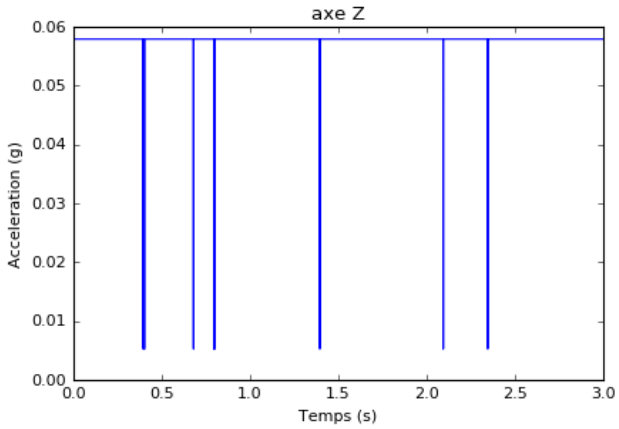


Figure 2.16 estimation de l'ordre de grandeur du bruit sur la sortie de l'axe Z de l'accéléromètre après une acquisition en position immobile

CHAPITRE 3 Pour aller plus loin

3.1 Accéléromètre et sa carte électronique

Un accéléromètre sert à mesurer des accélérations linéaires, dans notre cas selon 3 axes orthogonaux : X, Y et Z. L'accéléromètre ADXL326 utilisé comporte un capteur dont le rôle est de fournir une tension de sortie pour chaque axe proportionelle à l'accélération subie. Il intègre également d'autres composants qui permettent d'obtenir un signal analogique bien conditionné en sortie.

L'accéléromètre utilisé est monté sur une carte électronique de la marque Adafruit qui a plusieurs rôles : **(1)** fournir la bonne tension d'alimentation à l'accéléromètre, **(2)** renvoyer cette tension vers l'ordinateur via une broche spécifique et **(3)** découpler le signal de sortie du bruit issu de l'alimentation. Ces trois aspects sont expliques ci-dessous :

1. d'après la notice de l'accéléromètre ADXL326, sa tension d'alimentation doit être comprise entre 1,8 V et 3,6 V. La tension d'alimentation fournie par la carte Arduino est de 5V ou de 3,3 V selon la broche utilisée. La carte électronique Adafruit intègre un composant qui permet d'alimenter l'accéléromètre à exactement 3,3 V, en le reliant à l'une des deux broches de l'Arduino ;
2. la carte électronique Adafruit retourne également sur la broche *3V0* la tension d'alimentation exactement fournie, de 3,3 V normalement. Cette information sera utile pour le microcontrôleur Arduino ;
3. la carte intègre également des condensateurs de 10 μ F proches de l'alimentation pour découpler l'accéléromètre du bruit de l'alimentation, comme indiqué dans la notice.

Pour chaque axe de mesure (X,Y,Z), l'accéléromètre fournit une tension de sortie analogique sur les broches *xOut*, *yOut*, *zOut*. Cette tension est proportionnelle à l'accélération subie et comprise entre 0 V et la tension d'alimentation de 3,3 V . Par exemple, les tensions 0 V et 3,3 V correspondent respectivement à une accélération subie de -16 g et de +16 g environ.

3.2 Convertisseur analogique/numérique

Le microcontrôleur Arduino comporte un convertisseur analogique/numérique qui permet de relever les valeurs de tensions fournies sur les entrées analogiques, à une fréquence donnée par l'utilisateur et sur une durée fixe : cette opération s'appelle l'échantillonnage.

Plus précisément, le convertisseur permet de transformer les tensions fournies par l'accéléromètre (signal analogique) en une série de valeurs discrètes (signal numérique) prises à intervalles de temps réguliers, selon la fréquence d'échantillonnage. Les valeurs de tensions admissibles sur une entrée analogique de l'Arduino (entre 0 et 5 V) sont automatiquement converties en valeurs entières entre 0 et 1023 compris, 0 correspondant environ à 0 V et 1023 environ à 5 V. Ainsi, une variation de 1 en sortie du convertisseur correspond à une variation de 4,8 mV en entrée environ ; cette quantité est la résolution du convertisseur. Une bonne résolution permet de distinguer de petites variations de la tension en entrée, et donc de petites variations d'accélération subies par l'accéléromètre.

Étant donnée que l'accéléromètre fournit au plus une tension de 3,3 V en sortie, la plage admissible du convertisseur analogique/numérique n'est pas complètement exploitée et donc la résolution n'est pas optimale compte tenu du matériel utilisé. Pour remédier à ce problème, la broche de connexion *AREF* du microcontrôleur peut être utilisée pour modifier la tension maximale de référence pour le convertisseur analogique/numérique (l'entrée qui sera convertie en la valeur 1023). Dans le montage de cette note, l'entrée *AREF* du microcontrôleur est reliée à la sortie spéciale de l'accéléromètre *3V0* évoquée plus haut qui donne exactement la tension de sortie maximale de l'accéléromètre (correspondant à sa tension d'alimentation). Dans les scripts C pour l'Arduino en annexe, la commande *analogReference*[1] est utilisée pour indiquer qu'on utilise la tension de la broche *AREF* comme tension de référence. Ainsi, la résolution du convertisseur analogique/numérique est adaptée au montage et vaut désormais 3,2 mV environ.

3.3 Calculs de sensibilité et biais de l'accéléromètre

Bien que des valeurs types de sensibilité et biais soient mentionnées dans la notice de l'accéléromètre, la phase de calibration est indispensable pour déterminer précisément ces valeurs, qui peuvent varier légèrement d'un accéléromètre à l'autre.

Sensibilité

La sensibilité de l'accéléromètre selon un axe donné correspond à la variation de tension en sortie de l'accéléromètre sur cet axe lorsqu'il subit une accélération de 1 g. Dans cette note, la sensibilité est donnée en valeurs numériques entre 0 et 1023, fournies par le convertisseur analogique/numérique. Dans l'exemple présenté, on a une sensibilité selon l'axe X de 19 environ. Cela signifie qu'une variation d'accélération de 1 g correspond à une variation de

1. https://www.arduino.cc/reference/en/language/functions/analog-io/analogreference/

3,2 *19 = 60 mV, où 3,2 est la résolution du convertisseur analogique/numérique en mV évoquée dans la section précédente. La sensibilité de cet axe est donc de 60 mV/g environ, ce qui est bien inclus dans la plage de valeurs types mentionnée dans la notice de l'accéléromètre : de 51 à 63 mV/g.

Biais

Le biais pour un axe donnée est la tension en sortie d'accéléromètre pour cet axe lorsque le système est immobile (accélération subie de 0 g). Théoriquement le biais en valeurs numériques en sortie de convertisseur est de 511 puisque la plage du convertisseur, de 0 à 1023, correspond à la plage d'accélérations de l'accéléromètre, de -16 g à 16 g. Une fois encore, la phase de calibration est indispensable pour déterminer précisément cette valeur.

Selon l'orientation de l'accéléromètre lorsqu'il est immobile, la valeur numérique retournée sur l'axe ne correspond pas exactement à la définition précédente du biais, c'est-à-dire à une accélération de $0g$. La valeur au repos peut correspondre à une accélération de 0 g, -1 g ou 1 g, selon les informations de la figure 2.12 issue de la notice de l'accéléromètre. Il est donc important de bien réfléchir à l'orientation de l'accéléromètre pour choisir la valeur de biais appropriée.

3.4 Filtrage des accélérations

D'après la notice de l'accéléromètre ADXL326, la plage de fréquences acceptables en entrée est de 0,5 Hz à 1600 Hz pour les axes X et Y et de 0,5 Hz à 550 Hz pour l'axe Z. Cela signifie que la fréquence des accélérations subie par l'accéléromètre dans cette plage doit être située dans cet intervalle. Pour ce faire, il est possible de mettre en place des filtres analogiques, directement au niveau du montage électrique à l'aide de condensateurs, ou des filtres numériques, qui agissent sur les signaux numériques obtenus en sortie de convertisseur.

Il est nécessaire de limiter les hautes fréquences pour deux principales raisons :

– éviter le phénomène de repliement typique en traitement du signal. Pour ce faire, il faut s'assurer que $f_\mathrm{s} \geq 2f_\mathrm{max}$ où f_s est la fréquence d'échantillonnage et f_max la fréquence maximale en entrée de l'accéléromètre ;

– améliorer la résolution de l'accéléromètre. En effet, plus la plage de fréquence admissible est élevée, plus le bruit est important sur la sortie de l'accéléromètre. Or la résolution de l'accéléromètre est la plus petite variation d'accélération pouvant être détectée, elle doit donc être plus importante que le bruit. Donc si le bruit est élevée, la fréquence de résolution de l'accéléromètre est plus faible.

Pour éviter ces problèmes, un filtre passe-bas (qui laisse passer les basses fréquences) analogique peut être mis en place à l'aide d'un condensateur, sur la sortie associée à chaque axe. La figure 3.1 issue de la notice indique quelle condensateur utiliser selon la fréquence maximale acceptable souhaitée. La carte électronique Adafruit intègre nativement un condensateur de 0,1 μF sur la sortie de chaque axe, limitant ainsi la fréquence admissible à 50 Hz environ. Cette fréquence de coupure peut être modifiée en ajoutant un condensateur en série sur la sortie souhaitée, et en calculant la capacité du condensateur équivalent obtenu.

Pour interdire les fréquences trop basse, le choix a été fait d'implémenter un filtre passe-haut numérique autorisant seulement les fréquences au delà de 0,5 Hz. La portion de code correspondant à ce signal est identifiable et commentée dans le script Python *priseMesures_ADXL326.py*.

Table 4. Filter Capacitor Selection, C_X, C_Y, and C_Z

Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01

Figure 3.1 choix du condensateur de filtrage en fonction de la bande de fréquence souhaitée. Issue de [1]

3.5 Résolution du système d'acquisition

Plusieurs résolutions ont été évoquées dans les sections précédentes :

- la résolution de l'accéléromètre, qui est la plus petite variation d'accélération pouvant être détectée, *i.e.* produisant un signal d'amplitude significativement plus grande que le bruit de l'accéléromètre,
- la résolution du microcontrôleur, qui est la plus petite variation de tension requise sur une des entrée analogique pour entraîner une variation de 1 en sortie du convertisseur analogique/numérique.

La résolution du système d'acquisition au complet est la plus petite variation d'accélération pouvant être détectée par la chaîne d'acquisition, autrement dit, il s'agit de répondre à la question : quelle est la variation d'accélération nécessaire en entrée pour entraîner une modification de la valeur en sortie du convertisseur analogique/numérique de 1 ?. Cette résolution globale est limitée par la résolution du microcontrôleur. Il est mentionné plus haut que pour varier la sortie du convertisseur de 1, une variation de 3,2 mV est nécessaire. Considérant

que la sensibilité de l'accéléromètre est de 60 mV/g sur l'axe X, une variation de 3,2 mV est engendré par une variation d'accélération de 0,053 g environ. En conclusion, la résolution du système d'acquisition au complet est donc d'environ 0,05 g.

3.6 Détails techniques sur l'échantillonnage

Cette section concerne l'implémentation de la procédure d'échantillonnage au niveau du script C permettant le contrôle de l'Arduino. Différentes commandes sont discutées et les les choix effectués sont justifiés afin de guider un éventuel lecteur désireux de se lancer dans des développements liés à l'échantillonnage. La solution retenue dans cette note consiste à effectuer l'échantillonnage en utilisant des interruptions de l'horloge interne du microcontrôleur, à partir de la librairie dédiée *TimerOne.h*. À noter que les procédures d'échantillonnages discutées dans cette section ne sont pas spécifiques à l'accéléromètre et peuvent être utilisées pour d'autres capteurs analogiques.

Commandes *micros* et *millis*

Ces deux commandes permettent de relever le temps depuis le début d'exécution du programme. *millis*[2] retourne cette durée en millisecondes tandis que *micros*[3] renvoie cette durée en microsecondes. La commande *micros* a été retenue pour cette note car elle permet un échantillonnage plus précis.

Commande *delay*

Une possibilité pour échantillonner le signal analogique est d'utiliser la commande *delay*[4]. Cette commande permet de faire une pause dans le programme, d'une durée fixée en millisecondes. Insérée dans une boucle, il est donc possible de relever les valeurs sur une sortie à intervalle donné.

L'inconvénient de cette commande est qu'elle ne permet pas de garantir une fréquence d'échantillonnage stable. En effet, la commande *delay* stoppe toutes les autres opérations pour la durée prescrite. Le temps de lecture de l'entrée analogique (commande *analogRead*) et d'écriture dans le moniteur série (commande *Serial.print*) ne sont donc pas pris en compte dans cette durée. Comme le temps d'exécution de ces deux commandes peut varier, la durée entre deux échantillons varie fortement, ce qui n'est pas acceptable pour la prise de mesures

2. https://www.arduino.cc/reference/en/language/functions/time/millis/
3. https://www.arduino.cc/reference/en/language/functions/time/micros/
4. https://www.arduino.cc/reference/en/language/functions/time/delay/

souhaitée.

Interruptions d'horloge

Une autre possibilité pour échantillonner le signal analogique est d'utiliser la fréquence d'horloge du microcontrôleur comme base pour calculer la fréquence d'échantillonnage. Plus précisément, la fréquence d'horloge du microcontrôleur est de 16 MHz, ce qui signifie qu'on a un *tic* d'horloge toutes les 1/16000000 sec. Trois compteurs (en anglais *timer*) sont construits à partir de ce *tic* et opérationnels sur le microcontrôleur :

- le compteur 0 pouvant stocker des données sur 8 bits, qui contrôle, entre autres, les fonctions *micros* et *millis* ;
- les compteurs 1 et 2 à 16 bits qui contrôlent des opérations non nécessaires pour notre application.

Il a donc été choisi d'utiliser le compteur 1 comme base pour obtenir manuellement un compteur spécifique dont la fréquence d'échantillonnage est de 1000 Hz. Le *tic* de ce compteur peut commander ce qu'on appelle une interruption d'horloge, c'est-à-dire une ou plusieurs actions devant se produire à chaque *tic*. En règle général, le programme C qui contrôle l'Arduino est exécuté de manière séquentielle en répétant indéfiniment le contenu de la boucle *loop*. Grâce aux interruptions d'horloge, il est possible de programmer une action se produisant à intervalle de temps précis, peu importe ce qui se passe dans la boucle loop au même instant.

Dans un premier temps, une interruption d'horloge a été programmée manuellement ; cependant, les résultats étaient instables lorsque la fréquence d'échantillonnage était modifiée. Par la suite, la librairie *TimerOne.h* a été utilisée. Celle-ci permet de programmer facilement des interruptions d'horloge à la fréquence d'échantillonnage souhaitée. Dans le script C *prise-Mesures_ADXL326.ino*, la fonction *interruption* définit les actions à réaliser à chaque *tic* du compteur : la lecture des signaux sur les 3 broches de sortie de l'accéléromètre et le transfert des résultats à l'ordinateur via le port USB (série). Dans la fonction *setup()* d'initialisation du programme (voir annexe B.1) , le compteur 1 est initialisé pour produire la fréquence d'échantillonnage f_s souhaitée et la fonction *interruption* est « attachée » à ce compteur par les commandes suivantes :

```
// timer initialization at sampling frequency fs
 Timer1.initialize(fs);
 // links timer1 with interruption function
 Timer1.attachInterrupt(interruption);
```

Enfin, pour contrôler la durée d'acquisition, il est nécessaire de « détacher » la fonction *interruption* du compteur 1 après la durée souhaitée par l'utilisateur. Dans notre script, la boucle *loop* a donc pour rôle de compter les interruptions d'horloge effectuées et de les stopper une fois que la durée d'acquisition souhaitée est atteinte.

Il est important de mentionner que la quantité d'opérations réalisable à chaque interruption d'horloge est limitée. En effet, il ne faut pas que la durée nécessaire à une exécution de la fonction *interruption* soit supérieure à la durée entre deux exécutions de cette fonction. Pour cette raison, étant donnée que l'opération la plus coûteuse en termes de temps de calcul dans cette fonction est l'envoi des données sur le port USB, l'opération n'est réalisée qu'une fois, en envoyant les données des trois axes dans une même chaîne de caractères.

Fréquence d'échantillonnage et durée d'acquisition maximales

Le dispositif d'acquisition proposée dans cette note a été testé pour une fréquence d'échantillonnage de 1000 Hz, et pour une durée maximale d'acquisition de 1 minute avec des résultats satisfaisant en terme de stabilité de la fréquence d'échantillonnage et de précision.

Compte tenu de l'implémentation proposée, la durée d'acquisition théorique maximale est d'environ 70 minutes. En effet, l'utilisation de la fonction *micros* utilise un compteur de microsecondes qui atteint sa capacité maximale après environ 70 minutes. Après cette durée le compteur est remis à zéro (principe d'*overflow* en anglais).

3.7 Précisions sur l'interfaçage Python/arduino

Choix de l'interface en Python

Le comportement du microcontrôleur est régi par des scripts en langage C, qui peuvent être directement écrit sur l'ordinateur et transféré à l'Arduino via l'IDE dédié accessible gratuitement sur le site d'Arduino (voir figure 2.1). La prise de mesures ne nécessite donc a priori que cet IDE. Cependant, compte tenu des spécificités relatives à l'échantillonnage, peu de données peuvent être transférées à l'ordinateur pour chaque échqntillon mesuré. Il n'est pas exemple pas recommmandé de transférer les données avec un affichage plus complet, incluant par exemple une conversion des valeurs numériques du convertisseur en accélérations en g, car cela pourrait compromettre la stabilité de la fréquence d'échantillonnage. De plus, l'IDE Arduino ne permet pas de représenter graphiquement les résultats obtenus. Pour ces raisons, le choix a donc été fait de réaliser une interface de pré- et post-traitement en langage Python 2. Ce langage libre a été choisi pour sa facilité d'implémentation ; il est de plus très

utilisé dans le domaine du calcul scientifique et présente des similarités avec certains logiciels commerciaux tels que Matlab.

Vitesse de transfert

Les données issues de la prise de mesure sont transmises du microcontrôleur à l'ordinateur via le port USB. La vitesse de transfert du microcontrôleur peut être modifiée si besoin au début des scripts en langage C, dans les paramètres. La vitesse de transfert doit être rentrée à la même valeur dans le script Python correspondant, sans quoi l'interfaçage ne pourra pas fonctionner.

Une vitesse de transmission élevée est nécessaire pour garantir la transmission de l'intégrité des données obtenues par interruptions d'horloge. Cependant, il a été observé qu'une vitesse trop élevée semblait augmenter les erreurs de transmission. Par conséquent, la vitesse de transmission a été fixée à 57600 bits/sec pour le script de calibration et à 200000 bits/sec pour le script de prise de mesure. En effet, il n'est pas nécessaire d'avoir une vitesse de transmission très élevée pour le script de calibration car seules 10 mesures sont réalisées pour chaque axe, ce qui correspond à une faible quantité de données à transmettre. En revanche, pour le script de mesures, la durée d'acquisition a été testée jusqu'à une minute ce qui correspond donc à un grand nombre d'échantillons à transmettre en peu de temps.

Remise à zéro du script du microcontrôleur

Les compteurs lues par les commandes *micros* et *millis* sont s'incrémentent automatiquement dès la mise sous tension du microcontrôleur, et non au moment du démarrage de l'acquisition en tant que telle. Pour s'assurer que les conditions d'échantillonnage sont les mêmes pour chaque acquisition, le script Python réinitialise le programme C chargé sur le microcontrôleur avant chaque acquisition, en envoyant directement les commandes suivantes :

```
1 portSerie.setDTR(False) # indicates that USB port isn't ready
2 time.sleep(0.022)       # sufficient delay to reboot Arduino script
3 portSerie.setDTR(True)  # USB port is now ready
```

Ces commandes ne sont pas des commandes en langage C mais des commande issues du module Python *PySerial*. Elles permettent un contrôle du transfert des données par le port USB qui induit une réinitialisation du script sur le microcontrôleur.

Liste de références

[1] *ADXL326 Data-sheet*, Analog Device, 2018. [En ligne]. Disponible : https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL326.pdf

ANNEXE A Scripts pour la calibration

A.1 Script *calibration_ADXL326.ino*

```
// CALIBRATION (3-AXIS ACCELEROMETER ADXL326)
// Author : Solène Kojtych
// solene.kojtych@polymtl.ca
// 2018


//Wiring (accelerometer pin -------- Arduino pin)
// power                          :     Vin     --------   power 3.3V
// ground                         :     GND     --------   GND
// voltage reference              :     3Vo     --------   AREF
// X axis                             :      Xout  --------   A2
// Y axis                             :      Yout  --------   A1
// Z axis                             :       Zout --------   A0

// ------------------ Parameters
// Connexion pins
const int xPin = A2;                  // X axis
const int yPin = A1;                  // Y axis
const int zPin = A0;                  // Z axis

// Constants
   // Nb of measures for each position
const int samplingsNb = 10;
  //  Data rate in bits/s for serial data transmission
const unsigned int serialDataRate = 57600;

// ---------------- initialization
// Variables
int xPlus = 0;
int xMinus = 0;

int yPlus = 0;
int yMinus = 0;

int zPlus = 0;
int zMinus = 0;

int xZero = 0;
int yZero = 0;
int zZero = 0;

char pythonMessage ='0';

void setup()
{// sets voltage reference for  the analog-to-digital converter
  analogReference(EXTERNAL);
// starts serial communication with computer
  Serial.begin(serialDataRate);
}

void loop()
{// Wait for start trigger
Serial.println("*Appuyer sur 1 pour débuter la calibration");
while(pythonMessage!='1'){
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
pythonMessage ='0';

//------------ Procedure de calibration ---------
  // Z+ axis
Serial.println("*Positionner l'axe Z+ vers le plafond et appuyer sur 1");
while(pythonMessage!='1'){
```

```
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
      // Samples reading
pythonMessage ='0';
xZero = ReadAxis(xPin);
yZero = ReadAxis(yPin);
zPlus = ReadAxis(zPin);

  // Z- axis
Serial.println("*Positionner l'axe Z- vers le plafond et appuyer sur 1");
while(pythonMessage!='1'){
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
      //  Samples reading
pythonMessage ='0';
zMinus = ReadAxis(zPin);

  // Axe X +
Serial.println("*Positionner l'axe X+ vers le plafond et appuyer sur 1");
while(pythonMessage!='1'){
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
      // Samples reading
pythonMessage ='0';
xPlus = ReadAxis(xPin);
zZero = ReadAxis(zPin);

  // X- axis
Serial.println("*Positionner l'axe X- vers le plafond et appuyer sur 1");
while(pythonMessage!='1'){
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
      // Samples reading
pythonMessage ='0';
xMinus = ReadAxis(xPin);

  // Y+ axis
Serial.println("*Positionner l'axe Y+ vers le plafond et appuyer sur 1");
while(pythonMessage!='1'){
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
      // Samples reading
pythonMessage ='0';
yPlus = ReadAxis(yPin);

  //Y- axis
Serial.println("*Positionner l'axe Y- vers le plafond et appuyer sur 1");
while(pythonMessage!='1'){
    if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
}
      // Samples readings
pythonMessage ='0';
```

```
yMinus = ReadAxis(yPin);


//-------------Communication with python script ----------------------------
pythonMessage = 0;
Serial.println("Transmission des informations au script Python");
delay(1000);

Serial.println("DEBUT");
// Sending power values issued by DAC
Serial.print(xMinus);
Serial.print(";");
Serial.print(xZero);
Serial.print(";");
Serial.println(xPlus);
Serial.print(yMinus);
Serial.print(";");
Serial.print(yZero);
Serial.print(";");
Serial.println(yPlus);
Serial.print(zMinus);
Serial.print(";");
Serial.print(zZero);
Serial.print(";");
Serial.println(zPlus);
Serial.println("FIN");

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

delay(1000);
}

// ------------ Functions--------------------------------

//MEAN :  average of samplingsNb samples on given axisPin
int ReadAxis(int axisPin){
  long reading = 0;
  analogRead(axisPin);
  delay(1);
  for (int i = 0; i < samplingsNb; i++)
  {
    reading += analogRead(axisPin);
  }
  return reading/samplingsNb;
}
```

A.2 Script *calibration_ADXL326.py*

```
# -*- coding: utf-8 -*-
"""
Calibration script for 3-axis accelerometer ADXL326

Python 2
02/2018
Author : Solène Kojtych
solene.kojtych@polymtl.ca
"""
import serial
import time
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import os

# %%------------------ Parameters ----------------------------------------

# Name of USB port used for Arduino
serialPortName = '/dev/ttyACM0'
# Path of the result folder
resultFolderPath="/home"
# test identification number
IDtest=1

# Accelerometer's parameters
accelerometerName="ADXL326"
axisNb = 3

# Connexion parameters
dataRate = 57600  # data rate in bits per second (same as Arduino script)

# %%  ----------------- Initialization ----------------------------------

# Connexion via USB port
    # opens connexion
serialPort = serial.Serial(serialPortName, dataRate)

    # erase remaining data in buffer
if serialPort.in_waiting>0:
    serialPort.reset_input_buffer
while serialPort.in_waiting>0:
    pass

    # reset script on Arduino
serialPort.setDTR(False)
time.sleep(0.022)
serialPort.setDTR(True)

# %%------------------- Acquisition -----------------------------------------
time.sleep(1)
continueAcquisition = True

# Calibration
while continueAcquisition  is True :
    time.sleep(1)
    if serialPort.in_waiting>0:
            line = serialPort.readline()
            textLine = line.rstrip()
            print(line)
            if textLine =='DEBUT':   # write 'BEGIN'
                continueAcquisition=False
    else:
        response = str(input('>> '))
```

```
        serialPort.write(response)

# Reading of calibration results from serial port
calibrationValues = np.zeros([axisNb,3])
for axis in range(axisNb):
    line = serialPort.readline()
    textLine = line.rstrip()
    print("Axe "+str(axis)+" : "+textLine)
    # get values for -1, 0 and 1 g
    calibrationValues[axis,:] = textLine.split(';')

# Connexion closure
serialPort.flushInput()        # erase buffer

print('Donnes restantes')      #'Remaining data'
print(serialPort.in_waiting)

serialPort.close()
print('Connexion fermee')     #'End of connexion'


# %%---------------------- Calculations and graphs -----------------------------------------

#----- Graphs
accelerationAbscissa = [-1,0,1]
graphicsTitles = ["axe X","axe Y","axe Z"]

for axis in range(axisNb):
    ax=plt.figure().gca()
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    plt.xticks(np.arange(-1,2))
    plt.plot(accelerationAbscissa,calibrationValues[axis,:])
    plt.scatter(accelerationAbscissa,calibrationValues[axis,:],
                facecolors='none', edgecolors='r')
    plt.ylabel("Tension relevee de 0 a 1023") # 'voltage between 0 and 1023
    plt.xlabel("Acceleration subie (g)")  #'acceleration (g)
    plt.title(graphicsTitles[axis])
    plt.show()

#---- Sensibility and bias calculation (values between 0 and 1023)

# Array with one line per axis (sensibility- bias)
axisResults =  np.zeros([axisNb,2])
for axis in range(axisNb):
    # linear regression (sensitivity = slope, biais = y-intercept)
    axisResults[axis] = np.polyfit(accelerationAbscissa,
                calibrationValues[axis], 1)

# %% ----------------- Results printing ----------------------------
completeResultFolderPath=resultFolderPath+'/'+"resultats_calibration"

# Creation of a results folder if it doesn't exist
if not os.path.exists(completeResultFolderPath):
    os.makedirs(completeResultFolderPath)

# Results file creation
resultsFilePath = completeResultFolderPath+'/calibration_'
resultsFilePath=resultsFilePath+accelerometerName+'_'+str(IDtest)+'.txt'
resultsFile = open(resultsFilePath, "w+")
ordinateAxis=' '
for axis in range(axisNb):
    ordinateAxis =ordinateAxis+'   '+graphicsTitles[axis]
resultsFile.write('# Calibration '+accelerometerName+' run '+str(IDtest)+'\n')
resultsFile.write('# ---- MESURES ----\n')    # 'Measures
```

```
resultsFile.write("Axe         -1g           0g         1g \n")
for axis in range(axisNb):
    resultsFile.write("{}        {:.0f}         {:.0f}       {:.0f} \n"
                      .format(graphicsTitles[axis],calibrationValues[axis,0],
                       calibrationValues[axis,1],calibrationValues[axis,2]))

# Result file closure
resultsFile.close()

# %%------------------ Calibration summary ----------------------

print('----------  Récapitulatif  ------------------') # 'Summary'
    # 'Accelerometer reference
print("Référence accéléro : {0}".format(accelerometerName))
    # results file's name
print("Nom du fichier de calibration produit : {0}".format(resultsFilePath))
    # 'Axis order'
print("Axes dans l'ordre : {}".format(ordinateAxis))
print("Valeurs relevées")   # 'Values'
    # Axis -1g 0g 1g'
print("Axe         -1g           0g         1g")
for axis in range(axisNb):
    print("{}        {:.0f}         {:.0f}       {:.0f}"
          .format(graphicsTitles[axis],calibrationValues[axis,0],
                  calibrationValues[axis,1],calibrationValues[axis,2]))
print('-----------------------------------------------')
```

A.3 Exemple de feuille de mesure vierge pour la calibration

	A	B	C	D	E	F	G	H	I	J
1	**Calibration d'accéléromètre**									
2										
3	**Identification du test**									
4	**Modèle accéléromètre**	ADXL 326								
5	**Nombre d'axes**	3								
6	**Date**									
7										
8	**Valeurs relevées de 0 à 1023**									
9	**Axe**	Axe X			Axe Y			Axe Z		
10	**N° test\Accélération (g)**	**-1**	**0**	**1**	**-1**	**0**	**1**	**-1**	**0**	**1**
11	**1**									
12	**2**									
13	**3**									
14										
15										
16	**Détermination des caractéristiques**									
17	**Valeur moyenne μ**	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
18	**Médiane**	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
19	**Écart type σ**	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
20	**Sensibilité moyenne**	#DIV/0!			#DIV/0!			#DIV/0!		
21	**Biais moyen à 0 g**	#DIV/0!			#DIV/0!			#DIV/0!		
22	**Biais moyen à 1 g**	#DIV/0!			#DIV/0!			#DIV/0!		
23	**Biais moyen à -1g**	#DIV/0!			#DIV/0!			#DIV/0!		
24	**Coefficient de détermination R^2**	#DIV/0!			#DIV/0!			#DIV/0!		
25										

Figure A.1 extrait de feuille de mesures vierge pour la phase de calibration (cases à compléter en jaune)

ANNEXE B Scripts pour la prise de mesure

B.1 Script *priseMesures_ADXL326.ino*

```
// DATA-ACQUISITION (3-AXIS ACCELEROMETER ADXL326)
// Author : Solène Kojtych
// solene.kojtych@polymtl.ca
// 2018

//Wiring (accelerometer pin -------- Arduino pin)
// power                        :     Vin     --------   power 3.3V
// ground                       :     GND   --------   GND
//  voltage reference  :     3Vo     --------   AREF
// X axis                       :      Xout  --------   A0
// Y axis                       :      Yout  --------   A1
// Z axis                       :       Zout --------   A2

//--------------- PARAMETERS ----------------------
const float fs = 1000;       // Sampling frequency in Hz
const int T = 3;                  // Sampling duration in seconds
//-------------------------------------------------------------

// Constants
const int xPin = A2;
const int yPin = A1;
const int zPin = A0;
const int long serialDataRate =2000000;
const unsigned long int samplingsNb = fs*T;

// Variables
char pythonMessage = 0;
int xValue = 0;
int yValue = 0;
int zValue = 0;
unsigned long startTime;
unsigned long int counter = 0;
unsigned long int copyCounter =0;
boolean endMessage = true;
String stringValues;

// Librairy necessary for accurate sampling
#include <TimerOne.h>             // enables timer interrupts

void setup(){
  analogReference(EXTERNAL);
  Serial.begin(serialDataRate);
  Serial.println("DEBUT");

  // Data acquisition trigger
  while(pythonMessage!='1'){
  if (Serial.available() > 0){
      pythonMessage = Serial.read();
    }
  }
  startTime = micros();

  // Timer1 I initialization at frequency fs
  Timer1.initialize(fs);
  Timer1.attachInterrupt(interruption);
}

// Tasks attached to Timer 1 interruption
void interruption(void){
   Serial.print(micros()-startTime);       //write time
   xValue = analogRead(xPin);               //record values
   yValue = analogRead(yPin);
   zValue = analogRead(zPin);
```

```
   stringValues = ";";                    //assemble results
   stringValues += xValue ;
   stringValues += ";";
   stringValues += yValue ;
   stringValues += ";";
   stringValues += zValue ;

   Serial.println(stringValues);     //data sending
   counter = counter+1;                  // number of samples taken
}

void loop(){
  // Reading counter's value
  noInterrupts();
  copyCounter = counter;
  interrupts();

  // Stops sampling procedure after samplingsNb + 1 samples
  // to ignore first incomplete sample
  if(copyCounter >samplingsNb && endMessage==true) {
    Timer1.detachInterrupt();         //stops timer interruption
    endMessage=false;
    Serial.println("FIN");            //final trigger for python script
  }
}
```

B.2 Script *prisesMesures_ADXL326.py*

```
# -*- coding: utf-8 -*-
"""
Data acquisition for accelerometer ADXL326
(and possibly various accelerometer)

Python 2
02/2018
Author : Solène Kojtych
solene.kojtych@polymtl.ca
"""

import serial
import time

import numpy as np
import matplotlib.pyplot as plt
import os

# %%------------------ Parameters -----------------

# Name of USB port used for Arduino
serialPortName = '/dev/ttyACM0'
# Path of the result folder
resultFolderPath = "/home"
# test identification number
IDtest=1
# number of figures after comma for results
nbOfFigures = 3

# Accelerometer's parameters
accelerometerName="ADXL326"
axisNb = 3
sensitivity = [18.8,18.8,19] # (betweeen 0 and 1023) for axis X,Y,Z
bias = [510.7,507.9,540.9]  # (betweeen 0 and 1023) for axis X,Y,Z

# Acquisition parameter
beepsNb = 0              # number of beeps before the acquisition starts

# Connexion parameters
dataRate  = 2000000  # data rate in bits per second (same as Arduino script)

# %%  ----------------- Initialization ------------------------------------

# Creation of a folder for results if it doesn't exist
completeResultFolderPath = resultFolderPath+'/resultats_acquisition'
if not os.path.exists(completeResultFolderPath ):
    os.makedirs(completeResultFolderPath )

# Creation of a folder for results of this particuliar test
completeResultFolderPathAccelero =completeResultFolderPath+'/'
completeResultFolderPathAccelero+=accelerometerName+'_'+str(IDtest)
if not os.path.exists(completeResultFolderPathAccelero ):
    os.makedirs(completeResultFolderPathAccelero)

# Text file opening
    # automatic increment of testNumber
testNumber = 1

folderName=completeResultFolderPathAccelero+"/mesures_%s" % testNumber+'.txt'
while os.path.exists(folderName):
    testNumber += 1
    folderName=completeResultFolderPathAccelero+"/mesures_%s"% testNumber+'.txt'

resultsFilePath = completeResultFolderPathAccelero +'/mesures_'
```

```
resultsFilePath+=str(testNumber)+'.txt'
resultsFile = open(resultsFilePath , "w+")

# Connexion initialization
    # Opening
serialPort = serial.Serial(serialPortName, dataRate)

    # erase remaining data in buffer
if serialPort.in_waiting>0:
    serialPort.reset_input_buffer
print('Donnes restantes')       #'Remaining data'
print(serialPort.in_waiting)

    # reset script on Arduino
serialPort.setDTR(False)
time.sleep(0.022)
serialPort.setDTR(True)

# Reading of the word "DEBUT" ('BEGIN')
line = serialPort.readline()
line = line.decode("utf-8")
print(line)
serialPort.reset_input_buffer
time.sleep(1)

#%%  -------------------- Sampling----------------------------------------
response = '0'
print('Appuyez sur 1 pour commencer puis entree') #'Push 1 and enter
while response !='1':
    response = str(input('>> '))

# Delay
if beepsNb >0:
    # Short beep
    order ="aplay "+resultFolderPath+'/bip0.wav'
    for i in range(0,beepsNb):
        os.system(order)
        time.sleep(0.9)
    # Long beep => acquisition starts
    order ="aplay "+resultFolderPath+'/bip1.wav'
    os.system(order)

serialPort.write(response)    # sends response to Arduino

# Read data on serial port uniil word "FIN" appear ('END')
continueReading = True
while continueReading  is True :
    line = serialPort.readline()
    textLine = line.rstrip()
    if textLine == 'FIN':    #'END'
        continueReading=False
    else:
        print(textLine)
        resultsFile.write(line)   # writes data

# %%--------------------Closure of serial connexion -------------------------
# File closure
resultsFile.close()

# erase remaining data in buffer
serialPort.flushInput()
print('Donnes restantes')   #'Remaining data'
print(serialPort.in_waiting)
```

```
# Serial port closure
serialPort.close()
print('Connexion fermee')   #'End of connexion'
# %%-------------------- Post-processing ------------------------------------
graphsTitles = ["axe X","axe Y","axe Z"] # 'Axis X, axis Y, axis Z'

# File reading (time + accelerations in numerical values betwwen 0 and 1023)
data=np.genfromtxt(resultsFilePath, delimiter=';')
data = data[1:,:]                       # removes first line (numerical error)


# Acceleration calculations (in g)
physicalQuantity = np.copy(data)
for axis in range(axisNb):
    physicalQuantity[:,axis+1] = (data[:,axis+1]-bias[axis])/sensitivity[axis]

# Writing of accelerations in a new file (in g)
writingFormat = '%.'+str(nbOfFigures)+'f'
postProcessingFileName = completeResultFolderPathAccelero+'/accelerations_'
postProcessingFileName+=str(testNumber)+'.txt'
headerCSV = ['Temps'] +graphsTitles[:axisNb]  # 'Time'
np.savetxt(postProcessingFileName, physicalQuantity, delimiter=";",
           header=' '.join(headerCSV))

# %%------------------- Graphs  --------------------------
# Initialization
time= physicalQuantity[:,0]/1000000      # time in seconds
samplingsNb = time.size
accelerations=np.zeros([samplingsNb,3])

# Graphs
for axis in range(axisNb):
    # Accelerations
    accelerationsAxis = physicalQuantity[:,axis+1]
    accelerations[:,axis] = accelerationsAxis

    # Graph for this axis
    plt.figure()
    plt.plot(time, accelerationsAxis)
    plt.title(graphsTitles[axis])
    plt.ylabel('Acceleration (g)')
    plt.xlabel('Temps (s)')
```