Déctecteur de Chute

AUTHORS

BARCELLA Léo - EIS/ESE1
E SILVA NETO José de Deus - EIS/ESE1

2024-12-06

Table des figures

Liste des tableaux

Acronymes

I2C Inter-integrated circuit. 4, 6

LSB Bit de Poids Faible. 10

MEMS Microsystème Electromecanique. 1

PCB Circuit Imprimé. 1, 7, 12

Remerciements

Nous souhaitons tout d'abord exprimer notre gratitude envers toutes les personnes qui nous ont aidées à mener à bien ce projet.

En premier lieu, nos sincères remerciements vont à M. Allane, notre professeur encadrant. Vos recommandations pour orienter notre projet et vos conseils dans les moments critiques ont été d'une aide précieuse. Votre disponibilité et votre expertise nous ont guidés avec efficacité, et vos encouragements ont été une source d'inspiration tout au long de ce travail.

Nous souhaitons également remercier M. Cantelli pour son soutien dans la réalisation des deux PCBs. Son expertise dans ce domaine nous a permis d'échanger à plusieurs reprises pour apporter des améliorations aux conceptions initiales. Ses conseils judicieux nous ont aidés à finaliser ce projet avec succès.

Enfin, nous adressons nos remerciements à M. Drevet, qui nous a guidés dans le choix des composants lors du prototypage de la carte. Sa connaissance approfondie des différents éléments disponibles au magasin nous a permis de réaliser ce projet en utilisant un grand nombre d'objets de récupération, contribuant ainsi à une démarche économique et durable.

Nous tenons également à exprimer notre reconnaissance à nos collègues pour leur soutien lors du projet, leurs échanges constructifs et leur compréhension dans les moments cruciaux. Leur collaboration a été essentielle pour atteindre les objectifs fixés.

Ce rapport est le fruit d'un travail collectif et ne serait pas complet sans l'apport de toutes ces contributions. Nous remercions chaleureusement chacun pour leur implication et leur bienveillance.

Table des matières

1 Introduction

Ce projet s'inscrit dans le cadre des cours EE410 et EE470 [1], axés sur l'intégration de systèmes et l'instrumentation autour des capteurs. L'objectif de notre projet est de concevoir un système capable de détecter les chutes, en utilisant un accéléromètre compatible Arduino et un autre accéléromètre nécessitant une intégration spécifique. L'idée finale est d'intégrer les deux capteurs et de comparer leurs performances.

Pour y parvenir, nous avons d'abord utilisé une machine de Mealy pour organiser et gérer les interactions entre le capteur Arduino et le système. Nous avons ensuite choisi un capteur non compatible Arduino, que nous devions intégrer.

Pour intégrer ce nouveau capteur, nous avons réalisé deux Circuit Imprimé (PCB)s, une expérience totalement nouvelle pour nous. Ces circuits ont été conçus à l'aide du logiciel KiCad, ce qui nous a permis de développer nos compétences en conception et design électronique.

Cette expérience a été l'occasion d'apprendre de nouvelles techniques en électronique et en programmation, tout en réalisant un projet concret ayant une application pratique significative.

2 L'accéléromètre

2.1 La technologie MEMS

Les 2 accéléromètres que nous utilisons : l'ADXL345 (Arduino) [2] et ADXL359 [3] (non Arduino) utilisent la même technologie, la technologie Microsystème Electromecanique (MEMS).

La technologie MEMS repose sur l'intégration de structures mécaniques et de circuits électroniques à l'échelle microscopique, principalement à partir de silicium.
Dans le cas d'un capteur MEMS, comme un accéléromètre, le fonctionnement repose sur une petite masse suspendue à l'intérieur du capteur, reliée par des microstructures flexibles.

Lorsqu'une accélération est appliquée, cette masse se déplace légèrement en raison de l'inertie. Ce déplacement est détecté à l'aide de capteurs capacitifs, qui mesurent les variations de distance entre la masse et des électrodes fixes. Ces variations sont ensuite converties en un signal électrique proportionnel à l'accélération appliquée.

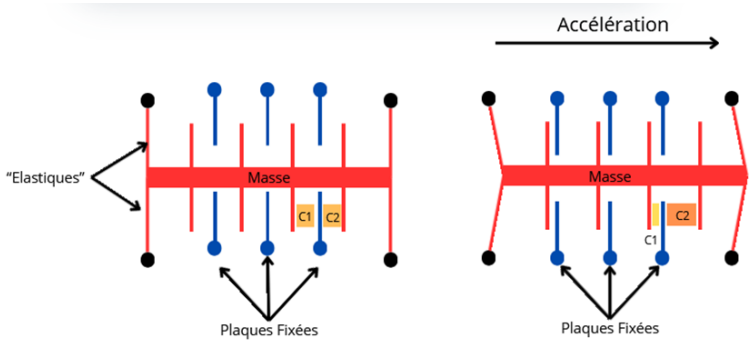


FIGURE 1 – Schéma simplifié de la technologie MEMS.

2.2 Différences entre l'ADXL345 et l'ADXL359

Nous avons consigné les différences données par les datasheets des 2 capteurs dans le Tableau 1.

	ADXL345 (Arduino)	ADXL359 (non Arduino)
Interface	SPI, I²C	SPI, I²C
Technologie utilisée	MEMS	MEMS
Plage de mesure	±2g, ±4g, ±8g, ±16g	±10g, ±20g, ±40g
Résolution	13 bits	20 bits
Bruit	$380\mu g/\sqrt{(}Hz)$	$80\mu g/\sqrt{(}Hz)$
0g Offset	±150 mg	±125 mg
Taille du boîtier	3 mm × 5 mm × 1 mm	4 mm × 4 mm × 1 mm

TABLE 1 – Tableau comparatif des 2 capteurs.

Nous avons choisis l'ADXL359, car il restait dans la même série que le capteur Arduino ; il n'était simplement pas intégré. De plus, nous souhaitions nous baser sur la librairie de l'ADXL345 [4] pour développer le code de l'ADXL359.

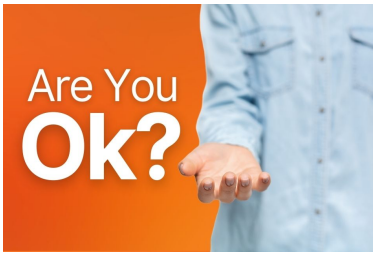
3 L'application du capteur

3.1 Principe

Le détecteur de chute va fonctionner comme un système communicant auquel l'Arduino envoi des messages à un serveur, ici, c'est un ordinateur qui s'occupe de prendre la décision d'appeler les urgences en cas de chute.

Pour cela, nous avons choisi d'utiliser le protocole Bluetooth pour envoyer l'état actuel. La gestion des messages reçus par le PC sont gérés par le code Python en section 10. Nous avons 3 étapes, la première est évaluée comme normal, la deuxième est la vérification de chute, c'est-à-dire si on a toutes les caractéristiques d'une chute, et finalement un état d'appel des urgences ou un retour à l'état normal. On représente les situations initiales en les affichant sur l'écran du PC. Ce sont les images de la Fig. 2.

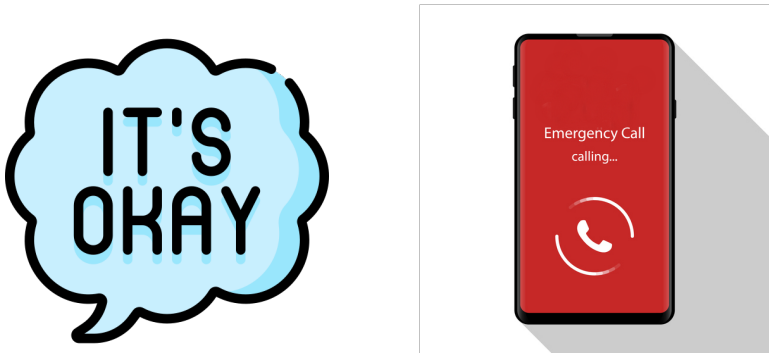
(a) IDLE → StartFall.



(b) StartFall → WaitFall.

FIGURE 2 – Les états de Vérification

De façon similaire, on a différents débouchés possibles. Soit la situation est une fausse alerte (pas de vraie chute), soit l'utilisateur est en situation de danger. Les images sont illutrées à la Fig. 3.

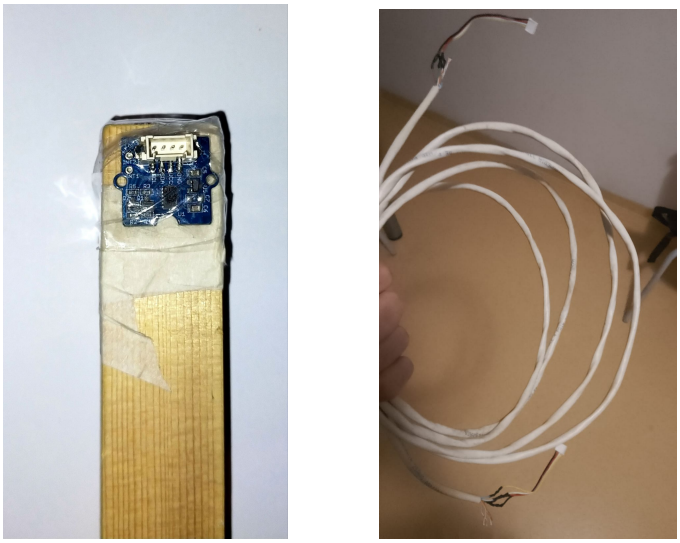


(a) WaitFall → IDLE. (b) WaitFall → FallDetected.

FIGURE 3 – Différents scénarios possibles

3.2 Modélisation d'une chute

Afin de générer les premiers résultats et bien comprendre comment sont caractérisées les chutes. On s'est servi d'un bâton auquel on a branché le capteur sur la pointe et fixé avec un scotch. En suite, nous avons coupé un câble Ethernet pour lier les connecteurs Inter-integrated circuit (I2C) au capteur et à l'Arduino. Le matériel utilisé est illustré à la Fig. 4.



(a) Bâton. (b) Câble du capteur.

FIGURE 4 – Dispositif des premiers tests

Pour modéliser une chute, nous avons simplement placé le bâton à la verticale, puis l'avons poussé pour initier sa chute.
Lors d'une chute, le module de la gravité est compensé par l'effet de la chute : le module diminue par rapport à sa valeur statique.
À la fin de la chute, lorsque le capteur heurte le sol, la gravité et l'accélération due à la chute s'additionnent, ce qui crée un pic sur le module de l'accélération. Le graphique à la Fig. 5 montre l'évolution de l'accélération sur les différents axes ainsi que du module de l'accélération lors d'une chute.
Notre code se base sur ce phénomène pour détecter les chutes. Nous définissons donc les seuils HAUT et BAS de manière arbitraire pour la suite de notre code.

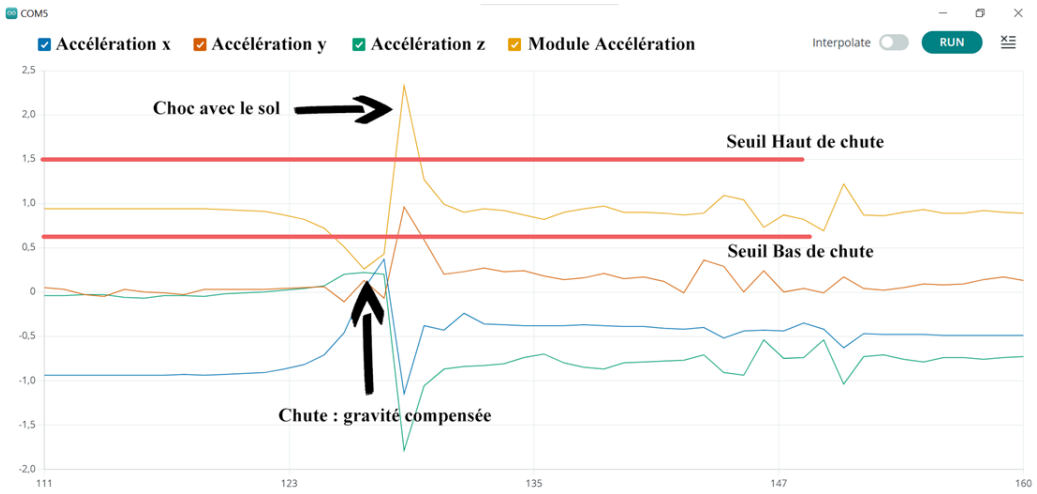


FIGURE 5 – Valeurs de l'accélération lors d'une chute.

3.3 Le code (machine de Mealy)

Pour bien modéliser notre système, on a choisi utiliser une machine de Mealy pour essayer d'optimiser le code et avoir les moins d'états possibles. Avec cet objectif, on reprend l'explication en 3.1 et on trouve le diagramme de la Fig. 6.

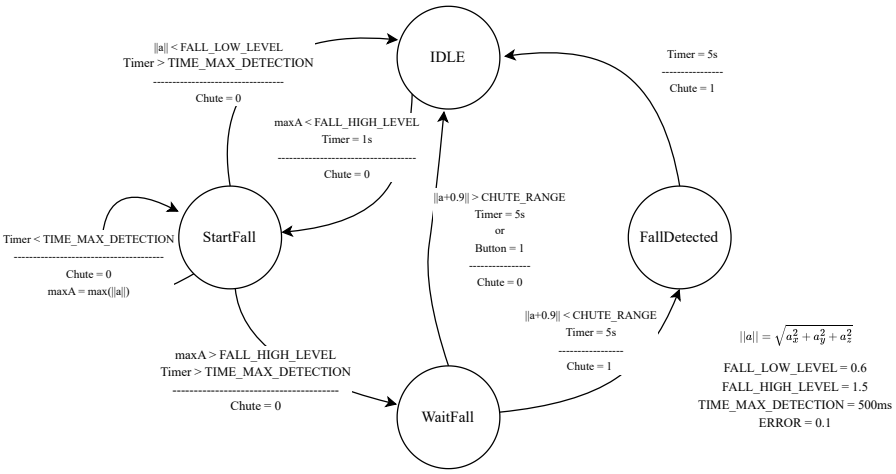


FIGURE 6 – Machine de Mealy.

A la Fig. 6 on remarque les détails de ce qu'il passe dans le code Arduino. D'abord si jamais on détecte une accélération au-dessus de 0.6, dans ce cas, l'hypothèse, c'est qu'on a eu une variation contraire à la gravité, donc on espère une valeur plus petite que 1 (la valeur standard étalonné). En suite, on enregistre l'accélération maximale pendant 500 ms. Pendant ce temps-là, pour qu'une chute arrive, il faut que l'utilisateur tombe au sol pour qu'on puisse avoir une valeur d'accélération ajoutée à la gravité : plus grande que 1. Si ça n'arrive pas, on retourne à l'état initial.

Au contraire, si on retrouve les caractéristiques d'une chute, on passe à l'état de vérification, Fig. 2b, dans lequel on attend une réponse du côté utilisateur. En ce cas, le button de non-chute sert à dire que tout se passe bien, sinon une attente plus grande que 5 secondes fait passer le système passer à l'état de danger, et les urgences sont notifiés. De plus, l'utilisateur est notifié par le signal sonore par un buzzer pour représenter l'état de danger [5].

3.4 Récuperation des données de l'ADXL359

Pour récupérer les données de l'ADXL, nous utilisons le protocole I2C. Nous avons basé notre code sur la librairie de l'ADXL345 pour nous familiariser avec le fonctionnement du capteur, ainsi que l'utilisation des différentes fonctions.

Dans le cas de l'ADXL359, il n'est pas nécessaire de passer par plusieurs états différents pour l'utiliser. On doit seulement être en mode standby pour changer ses paramètres.

- Mettre le bit 0 de POWER_CTL à 0 : Mode standby
- Régler d'autres paramètres au besoin...
- Mettre le bit 0 de POWER_CTL à 1 : Mode mesure

- Mesure, en lisant XDATA, YDATA e ZDATA

4 Conception du prototype

4.1 Schématique et routage

La Fig. 7 présente le schéma utilisé pour l'élaboration du prototype.

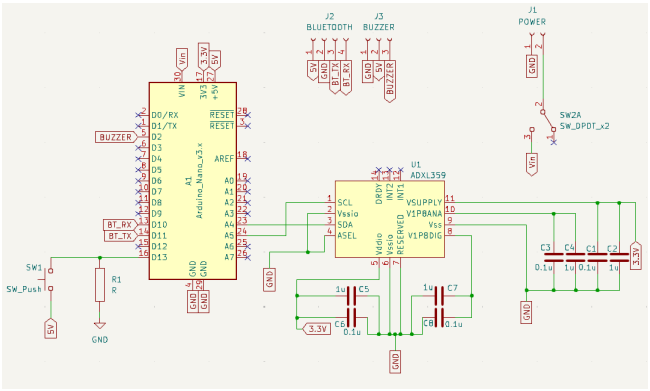


FIGURE 7 – Dessin schématique.

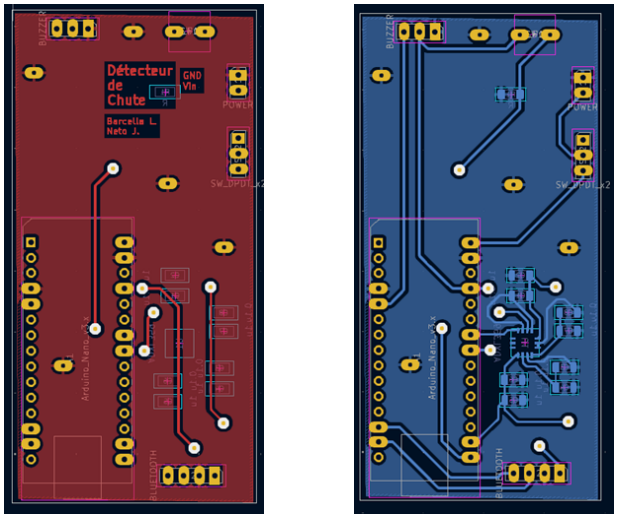
4.2 Réalisation du PCB

Le PCB et le dessin d'empreinte, sont présentés à la Fig. 8.

Le routage a été fait sur deux couches (ce qui est proposé à l'esisar avec une finesse de gravure minimum de 0.3 mm.

Nous avons choisi de grossir la taille de pads utilisés pour faciliter la soudure, recommandé par M. CANTELLI. Pour ce qui est des capacités et des résistances nous avons choisi le type 1206, la taille la plus grosse disponnible au magasin de composants de l'école. Dans l'optique ausi de faciliter les soudures.

Pour la soudure de l'ADXL359, nous avons eu recours au four à réfusion : la structure du capteur avec des pads, petits (0.35 mm de large) et sous le capteur ; rendait la soudure impossible au fer à souder. Nous avons donc pu voir son foctionnement (crème à braser, courbe de température à respecter...).



(a) Couche Top. (b) Couche Bottom.

FIGURE 8 – Routage de la carte finale.

4.3 Problématique lors de la conception

La Fig. 9 zoome sur le routage autour du capteur.

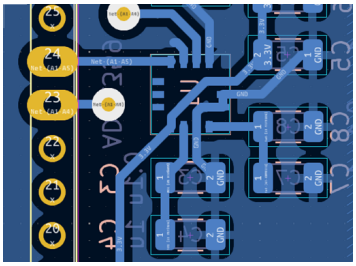


FIGURE 9 – Routage du capteur problématique.

Malheureusement, après avoir soudé tous les composants, le capteur ne fonctionnait pas : il ne répondait pas lorsque nous essayions de lire ses données. Après inspection, tous les signaux (SDA, SCL pour l'I²C, 3.3 V et GND) arrivaient correctement jusqu'au bout des pistes, et aucun court-circuit ne perturbait notre carte.
Après avoir demandé conseil à M. CANTELLI et M. ALLANE, nous avons décidé de refaire le routage de la carte autour du capteur (Fig. 10). Nous avons veillé à ce qu'aucune piste ne passe sous le capteur et avons également allongé la taille des pads, de sorte que lorsqu'il y a trop de crème à braser sur les pads, celle-ci se répartisse sur leur longueur, améliorant ainsi la qualité de notre soudure.

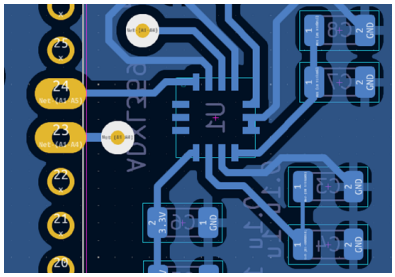


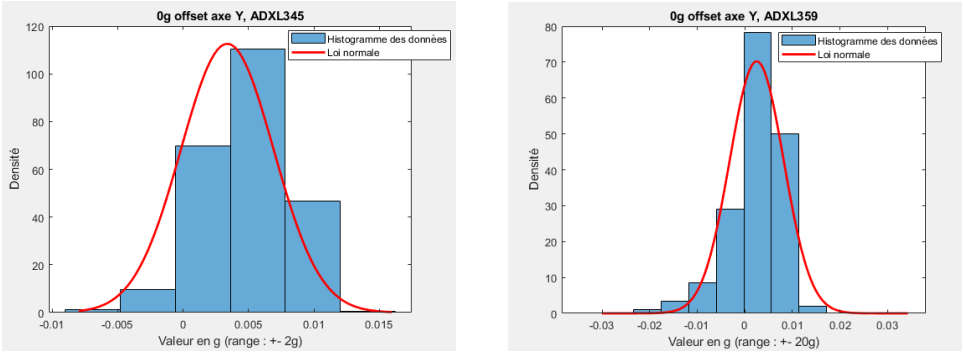
FIGURE 10 – Capteur avec le bon routage.

Après avoir réalisé notre deuxième PCB, le capteur nous renvoyait enfin les résultats liés à l'accélération. Les modifications que nous avons approtés nous a donc permis de réduire l'aléa lié produit par le four à réfusion.

5 Différences expérimentales entre les 2 capteurs

5.1 Estimation du 0g Offset

Nous essayons de caractériser le 0g Offset du capteur, c'est a dire la valeur qu'il lit lorsqu'il est censé voir 0g sur cet axe. Pour cela, on le place de sorte à ce que la gravité n'agisse que sur un seul axe, celui-ci voit 1g, tandis que les autres axes ne voient que 0g. On répète ensuite l'expérience sur les 2 autres axes pour obtenir 3 lots de données. On peut voir ces résultats à la Fig. 13 pour l'ADXL345 et à la Fig. 14.



(a) ADXL345. (b) ADXL359.

FIGURE 11 – Comparaison du 0g Offset sur l'axe Y.

On peut voir sur la Figure 11 que le 0g offset suit bien un loi normale centrée en 0. Sur le Tableau 2, on peut voir que nos résultats sont bien dans les critères données pas la datasheet.

	ADXL345 (Arduino, ±2g)	ADXL359 (non Arduino ±20g)
Moyenne 0g offset X expérimental	3.39 mg	6.45 mg
Moyenne 0g offset Y expérimental	3.30 mg	2.69 mg
Moyenne 0f offset Z expérimental	3.45 mg	-12.3 mg
0g offset (range)	±20 mg	±40 mg
0f offset datasheet	±150 mg	±125 mg

TABLE 2 – Tableau comparatif des valeurs du 0g Offset.

5.2 Estimation du gain

Pour estimer le gain : la valeur que nous devons multiplier à la valeur donnée par le capteur, cette donnée est en g/Bit de Poids Faible (LSB). Nous réutilisons les données que nous avions récupérées pour estimer le 0g Offset.

La formule que nous utilisions est la suivante :

$$\sqrt{(G_X \times (X_{raw} - X_{offset}))^2 + (G_Y \times (Y_{raw} - Y_{offset}))^2 + (G_Z \times (Z_{raw} - Z_{offset}))^2} = 1. \quad (1)$$

Cette équation s'applique sur les 3 lots de données que nous avions. Nous avons donc 3 équations (avec différents paramètres) et 3 inconnues (G_X, G_Y et G_Z). Les résultats sont résumés dans le Tableau 3.

	ADXL345 (Arduino, ±2g)	ADXL359 (non Arduino ±20g)
Gain axe X expérimental	4.00 mg/LSB	39.7 µg/LSB
Gain axe Y expérimental	3.99 mg/LSB	36.3µg/LSB
Gain axe Z expérimental	3.87 mg/LSB	40.8 µg/LSB
Gain datasheet	3.90 mg/LSB	39.0 µg/LSB

TABLE 3 – Tableau comparatif des valeurs de gain.

5.3 Différence des capteurs lors d'une chute

Nos réalisons une chute aucours de laquelle nous venons mesurer le module de l'accélération sur les deux capteurs (cf : Fig. 12). On peut voir que les capteurs se suivent lorsqu'il y a des variation. Cependant, lorsque l'accélération dépasse 1.5 g, la valeur devient faussée pour l'ADXL359.

Nous avons quelques idée sur les causes de cette erreur :

- Problème de rafraîchissement des valeurs : le code utilisé possède un temps de rafraîchissement de 10 ms du code Arduino, ce qui nous empêche peut-être de récupérer les bonnes valeurs
- Effet de "clipping" : lorsque l'accélération atteint un certain seuil, il est possible que les réponses soit faussées
- Problème d'électronique (le plus probable) : il se peut qu'il y ait un faux contact lorsque le capteur chute sur le sol ce qui vient fausser les résultats.

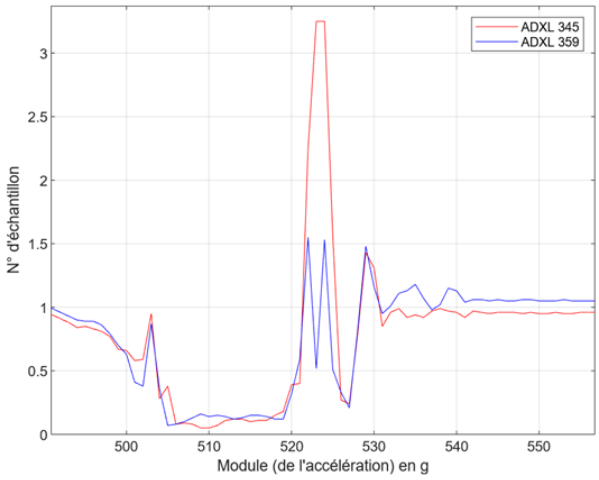


FIGURE 12 – Comparaison du module de l'accélération lors d'une chute.

6 Conclusion

Ce projet nous a permis de combiner théorie et pratique en concevant un système de détection de chutes basé sur l'intégration de deux capteurs : un compatible Arduino et un autre nécessitant une intégration spécifique. L'utilisation d'une machine de Mealy pour organiser les interactions entre les capteurs et le système a renforcé notre compréhension des machines à états.

Une des grandes leçons tirées de ce projet est l'importance de la sélection des composants en fonction des moyens disponibles. En effet, les contraintes matérielles rencontrées nous ont appris à adapter nos choix et à tirer parti des ressources limitées.

La réalisation des deux PCBs, conçus avec le logiciel KiCad, a été un véritable défi. N'ayant aucune expérience préalable dans ce domaine, nous avons beaucoup appris, que ce soit sur le processus de conception ou sur les étapes nécessaires à leur fabrication et leur utilisation.

En résumé, ce projet nous a permis de développer des compétences variées en électronique et en programmation tout en répondant à une problématique concrète et pratique. Cette expérience, riche en apprentissages, restera un point marquant dans notre parcours académique. [3].

Références

[1] B. V., "III Synthèse RTL." https://chamilo.grenoble-inp.fr/courses/ESISAR4AMCE419/document/CM/CE410_CE419_III_Synthese_RTL_1_2.pdf?cidReq=ESISAR4AMCE419&id_session=0&gidReq=0&gradebook=0&origin=, 2024. Accessed : 06-12-2024.

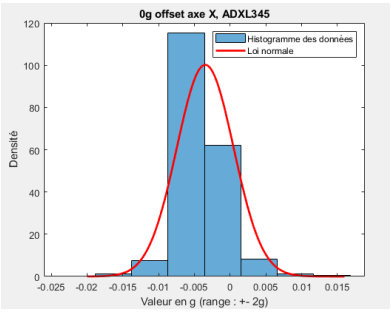
[2] Analog Devices, *3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer*, 05 2022. Rev. G.

[3] Analog Devices, *Low Noise, Low Drift, Low Power 3-Axis MEMS Accelerometer*, 06 2023. Rev. 0.

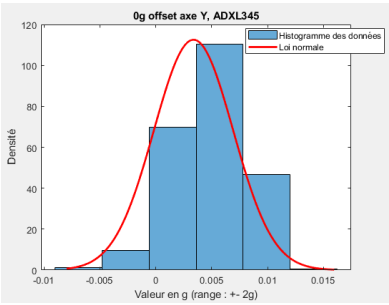
[4] Speed-Studio, "Accelerometer_ADXL345." https://github.com/Seeed-Studio/Accelerometer_ADXL345, 2024.

[5] B. A. Nassim, "Star-Wars-theme-Song-Arduino-." https://github.com/NassimBouyacoub/Star-Wars-theme-Song-Arduino-/tree/master, 2013.

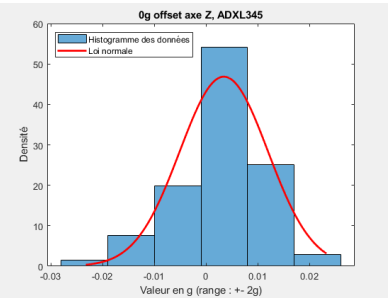
7 Appendix A - Caractérisation des paramètres des capteurs



(a) Axe X

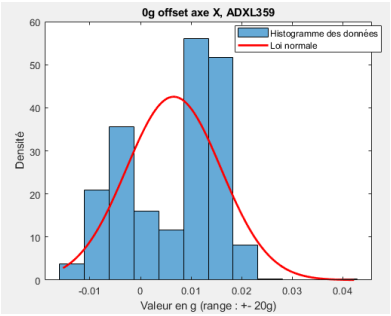


(b) Axe Y

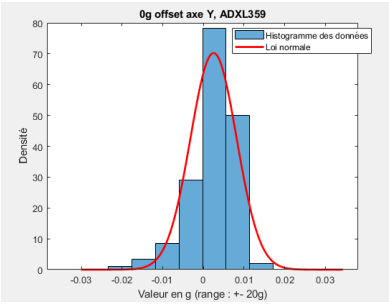


(c) Axe Z

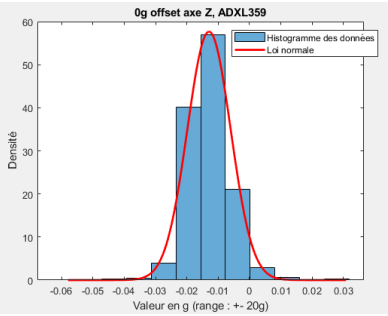
FIGURE 13 – Caractérisation du 0g Offset pour l'ADXL345



(a) Axe X



(b) Axe Y



(c) Axe Z

FIGURE 14 – Caractérisation du 0g Offset pour l'ADXL359

8 Appendix B - Code Arduino

```
//IMPERIAL MARCH
const int c = 261;
const int d = 294;
const int e = 329;
const int f = 349;
const int g = 391;
const int gS = 415;
const int a = 440;
const int aS = 455;
const int b = 466;
const int cH = 523;
const int cSH = 554;
const int dH = 587;
const int dSH = 622;
const int eH = 659;
const int fH = 698;
const int fSH = 740;
const int gH = 784;
const int gSH = 830;
const int aH = 880;

const int buzzerPin = 2; // Digital Pin 8
const int ledPin1 = 12;  // Digital Pin 12
const int ledPin2 = 13;  // Digital Pin 13 Built In Led can Change it if you want

int counter = 0;


//#include <Wire.h>
#include <ADXL345.h>
#include <SoftwareSerial.h>

#define BUTTON_PIN 13
#define BUZZER 2

#define FALL_LOW_LEVEL 0.6
#define FALL_HIGH_LEVEL 1.5
// Time during which we look to see if there has been a peak in acceleration (in millisec
```

```
#define TIME_FALL_MAX_DETECTION 500
#define CHUTE_ERROR 0.4  //Margin d'erreur sur laquel une personne peut buger sans que le
//false chute

#include <Wire.h>

//double xyz[3];
//// Les valeurs de l'accélération suivant les différents axes
//double ax, ay, az;
//double ;
double maxModuleValue = 1;
int buttonValue;
bool chuteLabel = false;  //when in waiting state, this label is resposible for passing t


double xyz359[3];
// Les valeurs de l'accélération suivant les différents axes
double ax359, ay359, az359;
double module359;

// ========================Valeurs pour la librairie ========================

byte _buff359[9] ;    //6 bytes buffer for saving data read from the device
double gains359[3];         // counts to Gs
bool status359;            // set when error occurs
// see error code for details
byte error_code359;        // Initial state

#define ADXL359_DEVICE (0x1D)    // ADXL359 device address for I2C
#define ADXL359_TO_READ (9)      // num of bytes we are going to read each time (two byte

#define ADXL359_OK    1 // no error
#define ADXL359_ERROR 0 // indicates error is present

#define ADXL359_NO_ERROR   0 // initial state
#define ADXL359_READ_ERROR 1 // problem reading accel
#define ADXL359_BAD_ARG    2 // bad method argument


#define ADXL359_STATUS 0x04
```

```
#define ADXL359_THRESH_ACT_H 0x25 // Registre du theresold activation
#define ADXL359_THRESH_ACT_L 0x26 // Registre du theresold activation
#define ADXL359_ACT_COUNT 0x27 // Number of consecutive events above threshold (from ACT_
#define ADXL359_DATAX3 0x08
#define ADXL359_POWER_CTL 0x2d
#define ADXL359_RANGE 0x2C // Setting the range
#define ADXL359_FILTER 0x28 // Filter settings
#define ADXL359_OFFSET_X_H 0x1E

SoftwareSerial mySerial(11, 10); // RX, TX

// Enumeration of the differents states
enum State {
  Idle,
  StartFall,
  WaitFall,
  FallDetected
};

// The current State
State CS = Idle;

long startTime;
long startWaitTime;
long startFallTime;
long startOkTime;


void setup() {
  mySerial.begin(9600);
  mySerial.println("Starting Bluetooth Connection");

  pinMode(BUTTON_PIN, INPUT);
  pinMode(BUZZER, OUTPUT);

  //digitalWrite(BUZZER, LOW);

  // put your setup code here, to run once:
  Serial.begin(9600);
  //for 20g
```

```
  gains359[0] = 0.000039; gains359[1] = 0.000039; gains359[2] = 0.000039;
  // for 10g
  // gains359[0] = 0.0000195;   gains359[1] = 0.0000195;   gains359[2] = 0.0000195;

  // Start communication
  Wire.begin();

  //Turning on the ADXL359
  setRegisterBit359(ADXL359_POWER_CTL, 0, 1); // Go to Standby Mode, must go to measurmen
  delay(100);

  // Set range to +- 20g
  setRegisterBit359(ADXL359_RANGE, 0, 0);
  setRegisterBit359(ADXL359_RANGE, 1, 1);


  setRegisterBit359(ADXL359_POWER_CTL, 0, 0); // Go to measurment mode

}

void loop() {
  getAcceleration359(xyz359);
  getAccelerometerValues359();
  module359 = sqrt(ax359 * ax359 + ay359 * ay359 + az359 * az359); // We calculate de mag
  buttonValue = digitalRead(BUTTON_PIN);

  // The running of the fallDetector => Following an ASM
  runASM();

  //  printAccelerationAndModule359();
  delay(200);

}

void writeTo359 (byte address, byte val) {
  Wire.beginTransmission(ADXL359_DEVICE); // start transmission to device
  Wire.write(address);             // send register address
  Wire.write(val);                 // send value to write
  Wire.endTransmission();         // end transmission
}
```

```
bool getRegisterBit359(byte regAdress, int bitPos) {
  byte _b;
  readFrom359(regAdress, 1, &_b);
  return ((_b >> bitPos) & 1);
}

void setRegisterBit359(byte regAdress, int bitPos, bool state) {
  byte _b;
  readFrom359(regAdress, 1, &_b);
  if (state) {
    _b |= (1 << bitPos);  // forces nth bit of _b to be 1.  all other bits left alone.
  } else {
    _b &= ~(1 << bitPos); // forces nth bit of _b to be 0.  all other bits left alone.
  }
  writeTo359(regAdress, _b);
}


void getAcceleration359(double* xyz) {
  int i;
  int xyz_int[3];
  readAccel359(xyz_int);
  for (i = 0; i < 3; i++) {
    xyz359[i] = xyz_int[i] * gains359[i];
  }
}

// Reads the acceleration into three variable x, y and z
void readAccel359(int* xyz) {
  readXYZ359(xyz, xyz + 1, xyz + 2);
}
void readXYZ359(int* x, int* y, int* z) {
  uint8_t buffer[9]; // Buffer pour lire 3 axes (3 octets par axe)
  readFrom359(ADXL359_DATAX3, 9, buffer); // Lire 9 octets à partir du registre DATAX0

  // Extraction des données brutes sur 20 bits pour chaque axe
  *x = (int)((((uint32_t)(buffer[0])) << 12) |
             (((uint32_t)(buffer[1])) << 4) |
             (((uint32_t)(buffer[2])) >> 4));
```

```
  *y = (int)((((uint32_t)(buffer[3])) << 12) |
             (((uint32_t)(buffer[4])) << 4) |
             (((uint32_t)(buffer[5])) >> 4));

  *z = (int)((((uint32_t)(buffer[6])) << 12) |
             (((uint32_t)(buffer[7])) << 4) |
             (((uint32_t)(buffer[8])) >> 4));

  // Sign extension pour convertir les valeurs 20 bits en entiers signés
  if (*x & 0x80000) *x |= 0xFFF00000; // Si bit 20 = 1, étendre le signe
  if (*y & 0x80000) *y |= 0xFFF00000; // Idem pour Y
  if (*z & 0x80000) *z |= 0xFFF00000; // Idem pour Z
}


// Reads num bytes starting from address register on device in to _buff array
void readFrom359(byte address, int num, byte _buff[]) {
  Wire.beginTransmission(ADXL359_DEVICE); // start transmission to device
  Wire.write(address);             // sends address to read from
  Wire.endTransmission(false);         // end transmission

  Wire.beginTransmission(ADXL359_DEVICE); // start transmission to device
  Wire.requestFrom(ADXL359_DEVICE, num);    // request 6 bytes from device

  int i = 0;
  while (Wire.available()) {      // device may send less than requested (abnormal)
    _buff[i] = Wire.read();    // receive a byte
    i++;
  }
  if (i != num) {
    status359 = ADXL359_ERROR;
    error_code359 = ADXL359_READ_ERROR;
  }
  Wire.endTransmission();         // end transmission
}


void getAccelerometerValues359() {
  getAcceleration359(xyz359);
```

```
  ax359 = xyz359[0]; ay359 = xyz359[1]; az359 = xyz359[2];
}

void printAccelerationAndModule359() {
  Serial.print(ax359);
  Serial.print(",");
  Serial.print(ay359);
  Serial.print(",");
  Serial.print(az359);
  Serial.print(",");
  Serial.println(module359);
}

void runASM() {
  // Notre diagramme d'Etat
  switch (CS) {
    case Idle:
      // Current working
      //digitalWrite(BUZZER, LOW);
      Serial.print(module359);
      Serial.println(",idle");
      mySerial.println("idle");

      if (module359 < FALL_LOW_LEVEL) { // Maybe there is a fall
        CS = StartFall;
        //maxModuleValue = 1; // Valeur neutre
        //startTime = millis(); // Start Measuring time
      }

      break;

    case StartFall:
      startTime = millis();

      while ((millis() - startTime) < TIME_FALL_MAX_DETECTION) { // Si on a dépassé le te
        getAccelerometerValues359();
        module359 = sqrt(ax359 * ax359 + ay359 * ay359 + az359 * az359);
        maxModuleValue = max(maxModuleValue, module359);

        if (module359 > FALL_HIGH_LEVEL) {
```

```
        Serial.print(module359);
        Serial.println(",analyse");
        delay(100);
      }
    }

    if (maxModuleValue > FALL_HIGH_LEVEL) { // Il y a bien eu une chute
      CS = WaitFall;
    }
    else {
      CS = Idle;
      //Serial.print("Pas de chute détectée finalent, max Module : ");
      //Serial.println(maxModuleValue);
      Serial.print(module359);
      Serial.print(",ok");
      mySerial.println("ok");
    }

    break;

  case WaitFall:
    startWaitTime = millis();

    while ((millis() - startWaitTime) < 5000) { // On attend 5 seconds pour voir si la
      getAccelerometerValues359();
      buttonValue = digitalRead(BUTTON_PIN);

      module359 = sqrt(ax359 * ax359 + ay359 * ay359 + az359 * az359);
      //Serial.println(module);

      if ((0.9 - CHUTE_ERROR < module359) and (module359 < 0.9 + CHUTE_ERROR) and (digi
        CS = WaitFall;
        Serial.print(module359);
        Serial.println(",waiting");
        mySerial.println("waiting");
        //digitalWrite(BUZZER, HIGH);
        delay(100);
        chuteLabel = true;

      } else {
```

```
        startOkTime = millis();
        CS = Idle;
        chuteLabel = false;


        while ((millis() - startOkTime) < 5000) {
          //digitalWrite(BUZZER, LOW);
          Serial.print(module359);
          Serial.println(",ok");
          mySerial.println("ok");
          delay(100);
        }

        break;
      }
    }

    if (chuteLabel == true) {
      CS = FallDetected;
    } else {
      CS = Idle;
    }
    break;

  case FallDetected:
    //digitalWrite(BUZZER, LOW);
    startFallTime = millis();
    CS = Idle;

    while ((millis() - startFallTime) < 10) {
      Serial.print(module359);
      Serial.println(",danger");
      mySerial.println("danger");
      IMPERIALMARCH();
      delay(100);
    }

    //Serial.print("Chute détectée !!!!!!!!!!!!!!!!!!!! max :");
    //Serial.println(maxModuleValue);
    //delay(5000);
```

```
      //Serial.println("Retour etat initial");

      break;

    default :
      Serial.println("Not a State, Error");
      delay(1000);
      break;
  }
}

void beep(int note, int duration)
{
  //Play tone on buzzerPin
  tone(buzzerPin, note, duration);

  //Play different LED depending on value of 'counter'
  if(counter % 2 == 0)
  {
    digitalWrite(ledPin1, HIGH);
    delay(duration);
    digitalWrite(ledPin1, LOW);
  }else
  {
    digitalWrite(ledPin2, HIGH);
    delay(duration);
    digitalWrite(ledPin2, LOW);
  }

  //Stop tone on buzzerPin
  noTone(buzzerPin);

  delay(50);

  //Increment counter
  counter++;
}

void firstSection()
{
```

```
  beep(a, 500);
  beep(a, 500);
  beep(a, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 650);

  delay(500);

  beep(eH, 500);
  beep(eH, 500);
  beep(eH, 500);
  beep(fH, 350);
  beep(cH, 150);
  beep(gS, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 650);

  delay(500);
}

void secondSection()
{
  beep(aH, 500);
  beep(a, 300);
  beep(a, 150);
  beep(aH, 500);
  beep(gSH, 325);
  beep(gH, 175);
  beep(fSH, 125);
  beep(fH, 125);
  beep(fSH, 250);

  delay(325);

  beep(aS, 250);
```

```
  beep(dSH, 500);
  beep(dH, 325);
  beep(cSH, 175);
  beep(cH, 125);
  beep(b, 125);
  beep(cH, 250);

  delay(350);
}

void IMPERIALMARCH()
{

  //Play first section
  firstSection();

  //Play second section
  secondSection();

  //Variant 1
  beep(f, 250);
  beep(gS, 500);
  beep(f, 350);
  beep(a, 125);
  beep(cH, 500);
  beep(a, 375);
  beep(cH, 125);
  beep(eH, 650);

  delay(500);

  //Repeat second section
  secondSection();

  //Variant 2
  beep(f, 250);
  beep(gS, 500);
  beep(f, 375);
  beep(cH, 125);
  beep(a, 500);
```

```
  beep(f, 375);
  beep(cH, 125);
  beep(a, 650);

  delay(650);
}
```

9 Appendix C - Code appel des urgences

```
import tkinter as tk

import serial
from PIL import Image, ImageTk

# Set up serial communication
interface = "rfcomm0"
ser = serial.Serial(f"/dev/{interface}", 9600, timeout=1)

# Initialize Tkinter window
root = tk.Tk()
root.title("Serial Image Display")

# Load images once and store in a dictionary
image_paths = {
    "idle": "Images/walking.jpg",
    "waiting": "Images/Blog-Image-Are-You-Ok.jpg",
    "ok": "Images/its_ok.png",
    "danger": "Images/RAZ_Emeregency_Service.png",
}

images = {
    key: ImageTk.PhotoImage(Image.open(path)) for key, path in image_paths.items()
}

# Create a label to display images
image_label = tk.Label(root)
image_label.pack()


# Function to update the displayed image
def update_image(status):
    if status in images:
        image_label.config(image=images[status])
        image_label.image = images[status]  # Keep reference to avoid garbage collection
    else:
        print(f"Unknown status: {status}")
```

```
# Main loop to read from the serial port and update the GUI
def read_serial():
    if ser.in_waiting > 0:
        line = ser.readline().decode("utf-8").strip()
        update_image(line)  # Update image based on received status
    root.after(100, read_serial)  # Schedule next read


# Start reading serial data
root.after(100, read_serial)

# Run the Tkinter event loop
try:
    root.mainloop()
except KeyboardInterrupt:
    print("Exiting program.")
    ser.close()
```

10 Appendix D - Code Matlab pour la caractérisation des paramètres

```
%% ========== ESTIMATION DES PARAMETRE DE L'ADXL 345 ==========
close all;
DataX = readmatrix("AxeX345.txt_formate.txt");
DataY = readmatrix("AxeY345.txt_formate.txt");
DataZ = readmatrix("AxeZ345.txt_formate.txt");

%% ========== Offsets ==========
% === Calcul de l'offset sur X
dataY_x = DataY(:, 1); dataZ_x = DataZ(:,1);
rawX_offset = [dataY_x; dataZ_x];
tracerLoiNormaleHistogram(rawX_offset, 12, "0g offset axe X, ADXL345");
x_offset = mean(rawX_offset)

% === Calcul de l'offset sur Y
rawY_offset = [DataX(:,2); DataZ(:,2)];
tracerLoiNormaleHistogram(rawY_offset, 12, "0g offset axe Y, ADXL345");
y_offset = mean(rawY_offset)

% === Calcul de l'offset sur Z
rawZ_offset = [DataX(:,3); DataY(:,3)];
tracerLoiNormaleHistogram(rawZ_offset, 12, "0g offset axe Z, ADXL345");
z_offset = mean(rawZ_offset)

% ====== ADXL345
% x_offset =   -0.8703
% y_offset =    0.8459
% z_offset =    0.8846

% ====== ADXL359
% x_offset =  165.5092
% y_offset =   69.0781
% z_offset = -316.8318
```

```
%% ========== Gains ==========
raw_x1 = mean(DataX(:,1));
raw_y1 = mean(DataX(:,2));
raw_z1 = mean(DataX(:,3));
raw_x2 = mean(DataY(:,1));
raw_y2 = mean(DataY(:,2));
raw_z2 = mean(DataY(:,3));
raw_x3 = mean(DataZ(:,1));
raw_y3 = mean(DataZ(:,2));
raw_z3 = mean(DataZ(:,3));

% Paramètres donnés pour résoudre l'équation
% sqrt( (ax)²+(by)² + (cz²)) = 1
% sqrt (dx)² + (ey)² + (fz²)) = 1
% sqrt( (gx)² + (hy)² + (iz)²) = 1

a = raw_x1-x_offset; b = raw_y1-y_offset; c = raw_z1-z_offset; % Exemple de valeurs
d = raw_x2-x_offset; e = raw_y2-y_offset; f = raw_z2-z_offset; % Exemple de valeurs
g = raw_x3-x_offset; h = raw_y3-y_offset; i = raw_z3-z_offset; % Exemple de valeurs

% Définir les équations sous forme de fonctions anonymes
equations = @(vars) [
    sqrt((a * vars(1))^2 + (b * vars(2))^2 + (c * vars(3))^2) - 1;
    sqrt((d * vars(1))^2 + (e * vars(2))^2 + (f * vars(3))^2) - 1;
    sqrt((g * vars(1))^2 + (h * vars(2))^2 + (i * vars(3))^2) - 1
];

% Valeurs initiales pour [x, y, z]
initial_guess = [0.5, 0.5, 0.5];

% Résolution avec fsolve
solution = fsolve(equations, initial_guess);

% Résultat
x = solution(1);
y = solution(2);
z = solution(3);

% Afficher la solution
disp(['Solution trouvée : x = ', num2str(x), ', y = ', num2str(y), ', z = ', num2str(z)])
```

```
% ====== ADXL345
% Solution trouvée : x = 0.0040005, y = 0.0039862, z = 0.0038718
% ====== ADXL359
% Solution trouvée : x = 3.9673e-05, y = 3.6318e-05, z = 4.0804e-05



%% Functions


function tracerLoiNormaleHistogram(data, nbHistogram, titre)
    % Calculer les paramètres de la loi normale
    mu = mean(data);          % Moyenne des données
    sigma = std(data);        % Ecart-type des données

    % Créer l'histogramme des données
    figure;
    histogram(data, nbHistogram, 'Normalization', 'pdf');  % Normalisation en PDF

    hold on;  % Maintenir l'histogramme pour superposer la loi normale

    % Tracer la courbe de la loi normale manuellement
    x = linspace(min(data), max(data), 1000);  % Créer des points pour l'axe X
    y = (1 / (sigma * sqrt(2 * pi))) * exp(-0.5 * ((x - mu) / sigma).^2);  % Densité de l

    plot(x, y, 'r', 'LineWidth', 2);  % Tracer la loi normale (en rouge)

    % Ajouter des labels et un titre
    title(titre);
    xlabel('Valeur (en g +- 20g)');
    ylabel('Densité');
    legend('Histogramme des données', 'Loi normale');
    hold off;  % Fin du maintien de la figure

end
```