



Rapport de Projet Tutoré

Étude et utilisation de l'accéléro-compas

Présenté par :
Sandric BRETECHER
&
Louis MUSIAL

Soutenue le 14/04/2023



Sommaire

I.	Figures	3
II.	Introduction.....	4
III.	Les composants	5
1.	Le capteur LSM303DLHC.....	6
2.	L'horloge temps réel.....	7
3.	Le stockage SD	7
4.	La carte Arduino UNO	8
5.	La carte d'extension Arduino	8
IV.	Mise en place du protocole de communication.....	9
V.	Le programme Arduino	12
VI.	Tests et résultats obtenus	15
1.	Test du magnétomètre.....	16
2.	Relevé des trames du modules LMS303.....	16
3.	Test pour la création d'un fichier.....	17
4.	Test pour la déconnection de la carte SD	18
VII.	Difficultés rencontrées et améliorations.....	19
VIII.	Conclusion.....	20
IX.	Bibliographie	21
X.	Annexes	22

I. Figures

- Figure n°1 : schéma d'implantation du capteur..... 5
- Figure n°2 : Orientation géographique du capteurs..... 5
- Figure n°3 : Module Compass Click..... 6
- Figure n°4 : Module d'horloge RTC2 Click..... 7
- Figure n°5 : Module MicroSD Click..... 7
- Figure n°6 : Carte Arduino Uno..... 8
- Figure n°7 : Carte Arduino Uno Click SHIELD..... 8
- Figure n°8 : schéma de câblage du capteur sur Arduino..... 12
- Figure n°9 : Diagramme fonctionnel..... 13
- Figure n°10 : Calibration accéléromètre..... 15
- Figure n°11 : Calibration magnétomètre..... 15
- Figure n°12 : Calibration horloge..... 15
- Figure n°13 : Affichage du cap..... 16
- Figure n°14 : Relevé de SDA et SCL à l'oscilloscope..... 16
- Figure n°15 : Relevé du champ magnétique sur l'axe X à l'oscilloscope..... 16
- Figure n°16 : Test fonction fichier - Fichier '.txt' 17
- Figure n°17 : Test fonction fichier - Fichier '.csv' 17
- Figure n°18 : Test fonction fichier - Moniteur série..... 17
- Figure n°19 : Carte SD déconnectée..... 18

II. Introduction

Lors de ce projet, qui consiste à intégrer sur un drone (déjà créer par d'autres licences lors des années précédentes) de nouveaux systèmes et capteurs permettant d'avoir plus d'informations en vol.

Notre licence doit donc intégrer sur ce drone :

- Un capteur de pression statique et de température
- Un GPS
- Un capteur de pression total
- Une centrale inertielle
- Un module LoRa
- Un accéléromètre et compas

Notre choix s'est donc porté sur l'accéléromètre et le compas qui permettra de communiquer au pilote du drone les informations concernant le cap suivi ainsi que les accélérations subies. Grâce à une horloge et à une carte SD il pourra également visionner les paramètres une fois le vol fini.

Nous avons donc à notre disposition pour mener à bien ce projet, une carte Arduino Uno, un module LSM303DLHC, qui comporte un accéléromètre et un magnétomètre, un module microSD click pour le stockage des données sur carte SD, une horloge RTC2 click pour réaliser un horodatée en temps réel, ainsi qu'un module Arduino Uno click SHIELD qui va permettre de connecter les modules à la carte Arduino.

III. Les composants

Avant tout il est important de noter que les capteurs que nous utilisons n'ont pas besoin d'être en contact avec le vent relatif comme un capteur de pression doit l'être. Nous proposerons donc de positionner le capteur au plus proche de l'avant du drone comme sur le schéma ci-dessous (s'il existe de la place pour y intégrer le capteur).

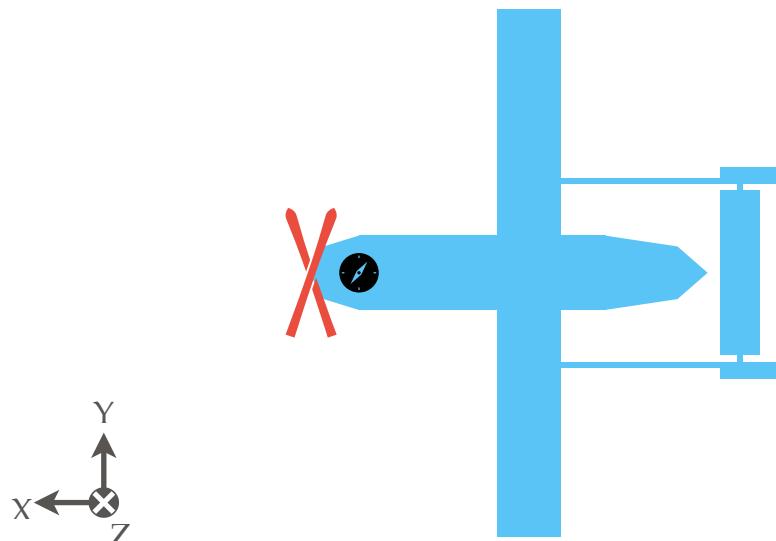


Figure n°1 : schéma d'implantation du capteur

Nous conseillons également d'éloigner le plus possible le capteur de toute source magnétique et électromagnétique si cela est possible.

La programmation a été effectuée dans un environnement magnifiquement calme si l'environnement dans le drone ne l'est pas il faudra envisager d'effectuer des tests de calibrations et modifier le programme pour limiter les erreurs dues à la déclinaison magnétique.

De plus pour la véracité des mesures il serait préférable de fixer le capteur au centre de l'axe transversal et dans le sens du drone de façon à ce que l'axe X (visible sur le capteur LSM303DLHC) soit en parallèle avec l'axe longitudinal du drone comme sur la figure ci-dessous.

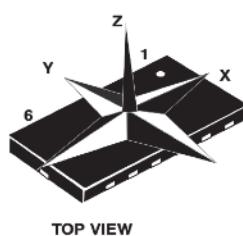


Figure n°2 : Orientation géographique du capteurs

Dans un premier temps nous allons étudier les différents modules afin de comprendre leurs fonctionnements et pouvoir correctement les utiliser par la suite.

Nous allons donc détailler le LSM303DLHC (capteur), le RTC2 click (horloge), le MicroSD click (microSD) et pour finir l'Arduino UNO et son click SHIELD.

1. Le capteur LSM303DLHC

Le LSM303DLHC est un capteur de mouvement 3 axes, qui combine un accéléromètre linéaire et un magnétomètre dans le même module. Il a été conçu pour être utilisé avec des microcontrôleurs tels que l'Arduino.

L'accéléromètre linéaire mesure les forces d'accélération sur trois axes : X, Y et Z (en m/s²), tandis que le magnétomètre mesure la direction du champ magnétique de la Terre dans les trois dimensions (en Tesla T).

Le capteur est équipé d'une interface numérique I2C ou SPI, ce qui le rend facile à intégrer dans une carte Arduino. Il peut être alimenté avec une tension allant jusqu'à 3,6 V et a une consommation de courant très faible. Dans le protocole I2C, la broche SDA (Serial Data) est utilisée pour la transmission des données bidirectionnelles entre les périphériques, tandis que la broche SCL (Serial Clock) est utilisée pour transmettre l'horloge qui synchronise la communication entre les périphériques.

En termes de spécifications, le LSM303DLHC a une plage de mesure de $\pm 2g/\pm 4g/\pm 8g/\pm 16g$ pour l'accéléromètre linéaire et de $\pm 1.3/\pm 1.9/\pm 2.5/\pm 4.0\mu T$ pour le magnétomètre. Il a également une résolution de 12 bits pour l'accéléromètre et 16 bits pour le magnétomètre. La précision des mesures est de $\pm 2\%$ pour les deux mesures. Son utilisation nécessite d'ajouter dans le code les librairies suivantes :

- Adafruit_LSM303DLH_Mag.h
- Adafruit_LSM303_Accel.h
- Adafruit_Sensor.h

Son alimentation doit être comprise entre 2,16V et 3,6V pour une consommation de 330 μA en mode normal. Il peut résister à des températures comprises entre -40°C et +85°C ce qui lui permet de fonctionner sur le drone lors de toutes les conditions météorologiques.

Concernant l'adresse du capteur, sa valeur dépend de la configuration des broches SDO/SA0. Par défaut, l'adresse est 0x19 si la broche SDO/SA0 est connectée à la masse (GND). Si la broche SDO/SA0 est connectée à une alimentation (VDD), l'adresse du capteur devient 0x1E. Ces adresses correspondent à l'adresse I2C 7 bits du capteur. Lors



Figure n°3 : Module Compass Click

de la communication avec le capteur via l'interface I2C, l'adresse doit être envoyée sous forme de 8 bits, avec le bit LSB (le bit de poids faible) indiquant si la communication est une lecture ou une écriture. Par exemple, pour lire des données à partir du capteur avec l'adresse 0x19, la valeur 0x32 (0x19 << 1 | 0x01) doit être envoyée en tant qu'adresse de lecture I2C. Le capteur a donc la possibilité d'être appelé sur 48 adresses différentes (annexe n°2) en fonction de ce qu'on lui demande (lecture seule ou écriture + lecture).

2. L'horloge temps réel



Le RTC2 Click est un module d'horloge en temps réel. Il communique via l'I2C. Son horloge interne a une précision de 1 minute par an, qui peut être utilisée pour garder une trace du temps. Le module peut être alimenté par une tension de 3,3 V ou 5 V et a une faible consommation de courant, ce qui le rend adapté aux applications alimentées par batterie. Il dispose d'un cristal de quartz de 32,768 kHz pour augmenter sa précision (± 1 ppm), ainsi que de fonctions de contrôle d'accès à l'écriture pour protéger les données stockées dans la mémoire du module. Son utilisation nécessite d'ajouter dans le code la librairie RTCLib.h.

Figure n°4 : Module d'horloge RTC2 Click

Grâce à ce composant, on va pouvoir réaliser des tâches temps réel basée sur une clock, car en effet, de base en étant sur Arduino et plus précisément sur Windows, il est très compliqué de faire du temps réel en programmation car il ne s'agit pas d'un RTOS comme peut l'être Linux.

3. Le stockage SD

Le MicroSD Click est un module d'extension pour les cartes microSD qui peut être connecté à la carte Arduino à l'aide de broches mikroBUS. Le module utilise une interface SPI pour communiquer avec la carte Arduino. Voici les caractéristiques techniques principales du module MicroSD Click :

- Alimentation : 3,3 V ou 5 V
- Interface : SPI
- Vitesse maximale de lecture : 25 Mo/s
- Vitesse maximale d'écriture : 18 Mo/s
- Tension de fonctionnement de la carte microSD : 2,7 V à 3,6 V

Son utilisation nécessite d'ajouter dans le code la librairie SD.h



Figure n°5 : Module MicroSD Click

4. La carte Arduino UNO



Figure n°6 : Carte Arduino Uno

L'Arduino Uno est une carte électronique basée sur un microcontrôleur ATMega328P. L'Arduino Uno dispose également de 32 ko de mémoire flash. La carte fonctionne à une tension d'alimentation de 5 V et dispose de 14 entrées/sorties numériques, dont 6 peuvent être utilisées comme sorties PWM, ainsi que de 6 entrées analogiques.

L'Arduino Uno est également équipé de plusieurs interfaces de communication, notamment UART, SPI et I2C, qui permettent une connexion facile à une large gamme de dispositifs externes. Les broches de sortie peuvent fournir jusqu'à 20 mA de courant chacune, et le courant maximal pour toutes les broches est de 200 mA.

5. La carte d'extension Arduino

L'Arduino Uno click SHIELD est une carte d'extension de la même taille que l'Arduino Uno et se connecte directement sur les broches d'entrée/sortie de la carte Uno. Elle est équipée de deux connecteurs Click, qui permettent de brancher des modules Click pour étendre les capacités de la carte Arduino Uno.

La carte d'extension est alimentée par la carte Arduino Uno elle-même, via les broches d'alimentation 5 V et GND. Elle dispose également de broches d'entrée/sortie supplémentaires, ainsi que d'un interrupteur pour sélectionner la source d'alimentation (soit l'Arduino Uno, soit une source d'alimentation externe).

Nous ajouterons également la librairie Wire.h qui permet la communication en I2C pour permettre à l'horloge de communiquer.

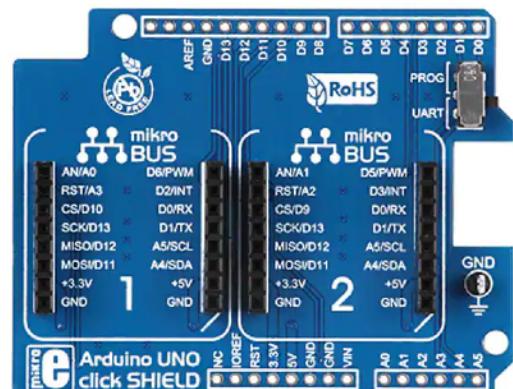


Figure n°7 : Carte Arduino Uno Click SHIELD

IV. Mise en place du protocole de communication

En parallèle de la programmation nous avons dû définir la trame qui sera utilisé par les autres groupes pour pouvoir uniformiser nos échanges à travers la liaison série. Cette trame doit être un moyen simple, rapide et compréhensible de transférer des informations.

En concertation avec les autres groupes nous avons décidé d'envoyer une trame par type d'informations et non par capteurs pour éviter les confusions.

Selon les datas de chacun, nous avons défini que 12 octets de data suffisent pour chaque type. La trame comportera donc une entête de 20 octets et une data de 12 pour une taille de trame totale de 32 octets.

Pour cette trame, nous sommes donc partis de l'entête : \$23EAS. Ainsi, nous avons créé le protocole suivant :

Entête	Label & Version	Secondes	Minutes	Heures	Jours	Mois	Année	Data 1	Data 2	Data N
\$23EAS	0x 00	00 à 59	00 à 59	00 à 23	00 à 31	00 à 12	≥ 2023			
String	Char	Int	Int	Int	Int	Int	Int	-	-	-
6 octets	2 octet	2 octet	2 octet	2 octet	2 octet	2 octet	2 octets	12 octets au total		

La section label et version va nous permettre de déterminer le type d'informations que contient la taille, ce type dépend des données envoyées. Pour la version nous mettons 0 de base puis, par la suite, si des modifications sont apportées à la trame ou aux datas envoyés nous passerons à N+1 ce qui permettra de connaître la version de la trame.

Ci dessous le tableau des labels utilisées :

Label	Type de data	Groupe
1	Pression statique	Kylian Damien
2	Température	
3	Altitude	
4	Latitude	Kevin Theo
5	Longitude	
6	Pression dynamique	Axel Carlos
7	Pression statique	
8	Vitesse (m/s)	
9	Cap magnétique	Sandric Louis
A	Accélération	
B	Incidence	Aliya Romain
C	Pente	
D	Assiete	
E	-	-
F	-	-

Si nous prenons l'exemple de notre donnée du cap magnétique le label et la version sera donc 0x 90.

Pour la section horodatage l'on envoie les secondes, les minutes, les heures puis le jour, le mois et enfin l'année. Il convient donc que les horloges de chaque carte Arduino soient le plus possible similaires pour ne pas créer de confusions dans l'envoi des trames.

Pour finir, nous passons a la partie data, elle est de 16 octets ce qui permet a tout les groupes d'envoyer les différentes informations sans problème. La définition des données sera définie par le type de donnée (dans le label) et permettra donc le déchiffrage de la data. Si la data envoyé dans la trame la data prend moins de 16 octets alors on enverras des 0.

Nous avons donc les trames suivante :

- 1ère trame :

Label	Numéro de Data	Nom data	Taille octet	Type data
90	1	Cap (Deg°)	4 octets	Float

- 2nde trame :

Label	Numéro de Data	Nom data	Taille octet	Type data
A0	1	X (m / s) ²	4 octets	Float
	2	Y (m / s) ²	4 octets	Float
	3	Z (m / s) ²	4 octets	Float

V. Le programme Arduino

Avant d'amorcer la phase de programmation, nous avons procédé à la mise en place du câblage de la carte Arduino avec les autres modules requis. Veuillez noter que la carte Click Shields n'a pas été représentée sur le schéma électrique car elle permet simplement la distribution ou la combinaison directe des broches. Voici donc le schéma correspondant :

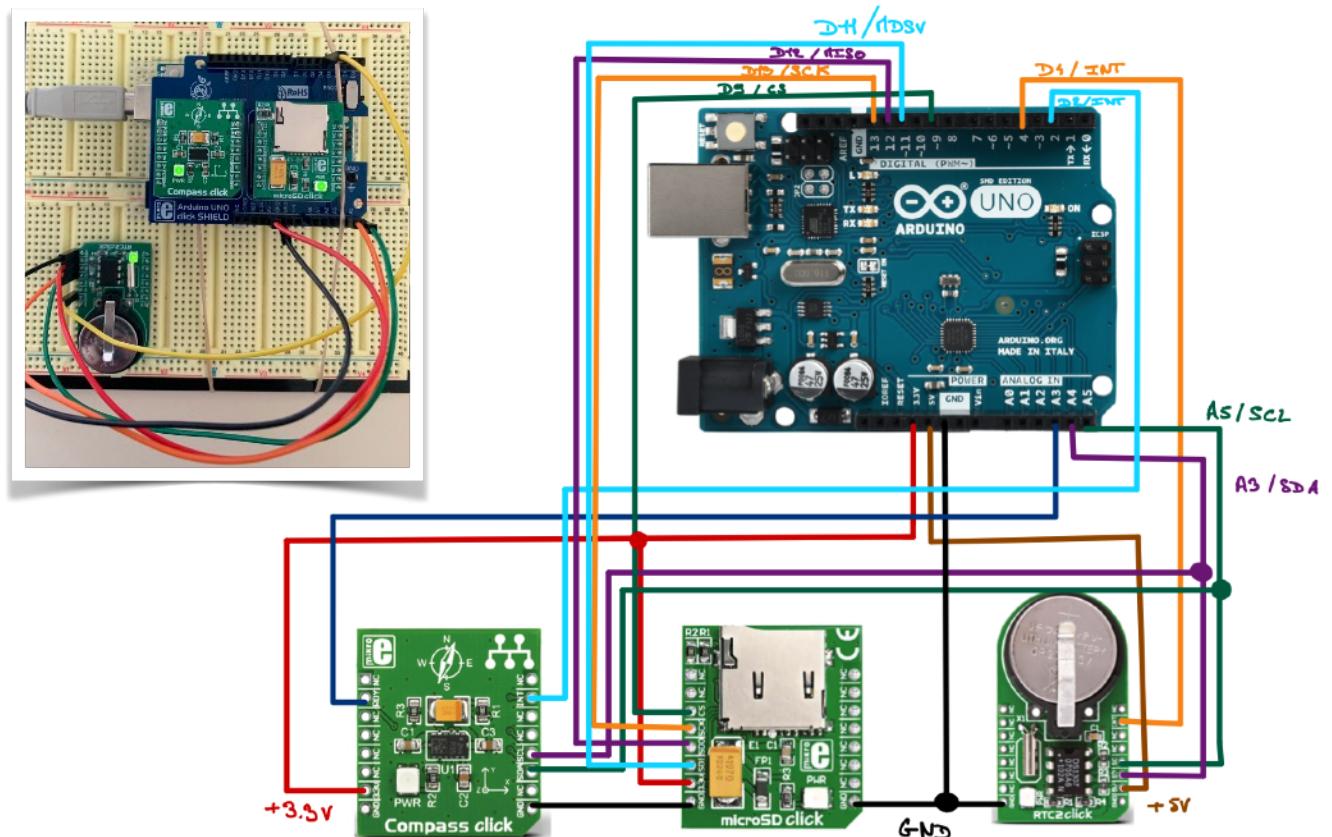


Figure n°8 : schéma de câblage du capteur sur Arduino

Ensuite, avant même de commencer à coder le programme pour la carte Arduino, nous avons pris soin de bien comprendre les besoins et les attentes du projet en nous basant sur le cahier des charges. Cela nous a permis d'avoir une vision claire des fonctionnalités à implémenter et des interactions entre les différents composants du système. Nous avons ensuite réalisé un diagramme fonctionnel (figure n°9) qui nous a permis de visualiser de manière claire et concise l'ensemble des tâches que le programme devait accomplir. Ce diagramme nous a également permis d'identifier les différentes fonctions et sous-fonctions à implémenter et de définir une hiérarchie entre celles-ci. Cette étape est cruciale pour garantir que le programme soit efficace et réponde aux exigences du cahier des charges.

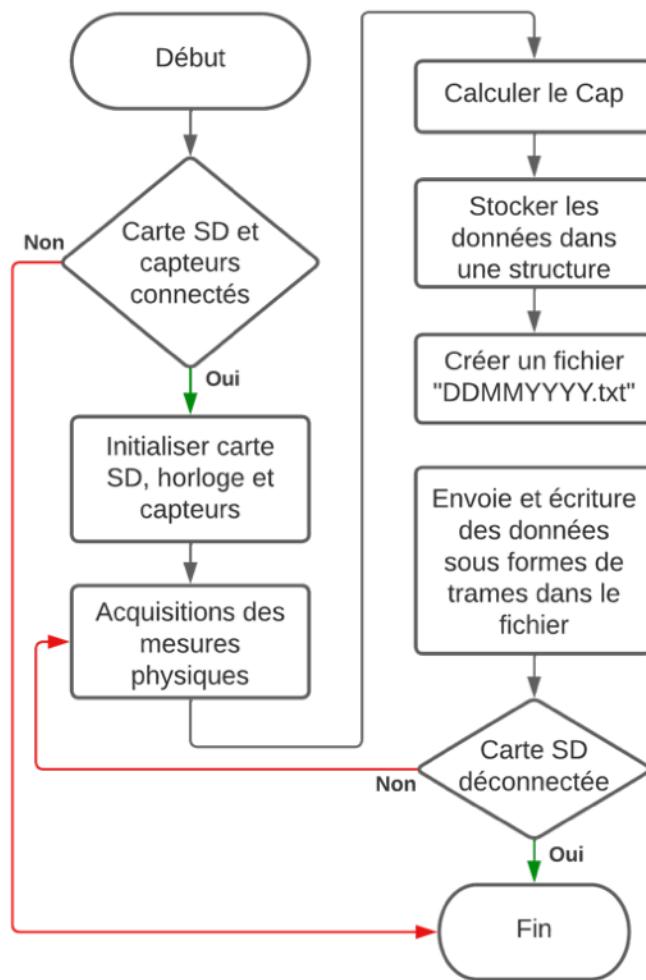


Figure n°9 : Diagramme fonctionnel

Ensuite, pour décrire rapidement ce diagramme, on vérifie tout d'abord si la carte SD et les capteurs sont connectés. Si c'est le cas, on initialise la carte SD, l'horloge et les capteurs. Ensuite, on effectue les mesures physiques, pour l'accéléromètre on va donc mesurer les accélérations linéaires sur les 3 axes (x, y et z) en m/s², quant au magnétomètre, lui va mesurer le champ magnétique sur les 3 axes en Tesla (T). On va donc ensuite calculer la valeur du cap grâce et stocke les données dans une structure. Il crée ensuite un fichier portant la date du jour, puis envoie et écrit les données sous forme de trames dans le fichier. Si la carte SD est déconnectée, le programme se termine. Sinon, il repart sur une boucle à partir des acquisitions, afin de continuer d'envoyer des mesures sur le fichier présent dans la carte SD.

Concernant le programme ([annexe n°1](#)), tout d'abord, il est important de noter que celui-ci utilise plusieurs bibliothèques pour lire différents capteurs, ainsi que la bibliothèque standard d'Arduino pour certaines fonctions. Il s'agit notamment de la bibliothèque pour la communication I2C, la bibliothèque pour l'utilisation de la carte SD, la bibliothèque pour la lecture de l'horloge RTC, qui va permettre notamment de vérifier l'acquisition en temps réel, la bibliothèque pour la lecture du capteur magnétique LSM303DLH et la bibliothèque pour la lecture de l'accéléromètre LSM303.

Le programme commence par définir des constantes, des objets et des structures pour stocker les données qui seront enregistrées sur la carte SD. Ensuite, il définit des fonctions `setup()` pour chaque équipement, comme l'accéléromètre, le capteur magnétique et l'horloge RTC. Ces fonctions `setup()` sont appelées une seule fois au début du programme pour configurer correctement chaque équipement.

Le programme utilise également une fonction `NomFichier()` pour générer un nom de fichier unique pour chaque enregistrement. Ce nom de fichier est basé sur la date actuelle, et pourraient être incrémenter de "`-i++`", avec `i` un entier réel, si un fichier portant le même nom existe déjà. Cependant, cette fonction pour l'incrémentation du nom de fichier n'a pas abouti à temps.

Enfin, le programme utilise des structures de données pour stocker les informations de chaque capteur, puis écrit ces informations dans un fichier sur la carte SD. Chaque enregistrement commence par une entête pour identifier le début de la trame, suivie d'un label pour identifier le type de données stockées (accéléromètre ou capteur magnétique), suivi de la version du logiciel et des informations de date et d'heure de l'enregistrement. Enfin, les données réelles de chaque capteur sont stockées, suivies d'un "padding" pour remplir la trame jusqu'à 32 octets si nécessaire.

En résumé, ce programme utilise plusieurs modules ; les 2 capteurs présent sur le module LSM303DLHC pour collecter des données accéléromérique et magnétique ; le module SD pour stocker ces données sur une carte SD. Les données sont stockées dans un format spécifique pour faciliter leur traitement ultérieur. Le programme est conçu pour fonctionner en continu, en enregistrant périodiquement les données sur la carte SD.

VI. Tests et résultats obtenus

Lors de ce projet, nous avons effectué une série de test, afin de vérifier le bon fonctionnement de chaque équipement, mais aussi pour vérifier la cohérence des résultats obtenus par rapport aux attentes.

Dans un premier temps, nous avons utilisé les programmes fournies avec les bibliothèques pour calibrer nos capteurs. Ils permettent de reconfigurer les capteurs pour avoir une nouvelle calibration et réduire la marge d'erreur. Nous pouvons visualiser sur le moniteur série :

The image contains two side-by-side screenshots of a terminal window titled "Max!Accelerometer Test" and "Magnetometer Test".

Max!Accelerometer Test Data:

Sensor	Driver Ver	Unique ID	Max Value	Min Value	Resolution
LSM303	1	54321	0.00 m/s^2	0.00 m/s^2	0.00 m/s^2

Range set to: +/- 4G
Mode set to: Normal

X	Y	Z	Unit
-0.31	0.08	9.20	m/s^2
-0.23	0.08	8.97	m/s^2
-0.31	0.08	9.05	m/s^2
-0.31	0.15	9.13	m/s^2
-0.31	0.15	8.97	m/s^2
-0.31	0.15	8.97	m/s^2
-0.31	0.00	9.05	m/s^2
-0.23	0.15	8.97	m/s^2

Magnetometer Test Data:

Sensor	Driver Ver	Unique ID	Max Value	Min Value	Resolution
LSM303	1	12345	0.00 uT	0.00 uT	0.00 uT

X	Y	Z	Unit
-13.91	20.73	-6.12	uT
-14.00	20.82	-5.92	uT
-14.00	20.55	-5.92	uT
-13.55	20.45	-5.92	uT
-14.00	20.55	-5.82	uT
-14.00	20.45	-5.92	uT
-14.00	20.73	-6.02	uT

Figure n°10 : Calibration accéléromètre

Figure n°11 : Calibration compas

Pour vérifier le bon fonctionnement de la Calibration, nous avons orienté le capteur vers le nord en se basant sur le compas magnétique de notre téléphone.

La calibration s'est donc effectuée et le compas affichait les champs magnétiques sur les 3 dimensions proche du Nord.

Ensuite, nous avons programmé la clock, celle-ci a dû être programmée lors de sa première activation et devra être reprogrammée si la pile en lithium présente s'est entièrement déchargée. Pour cela nous utilisons le programme d'exemple de la bibliothèque Arduino.

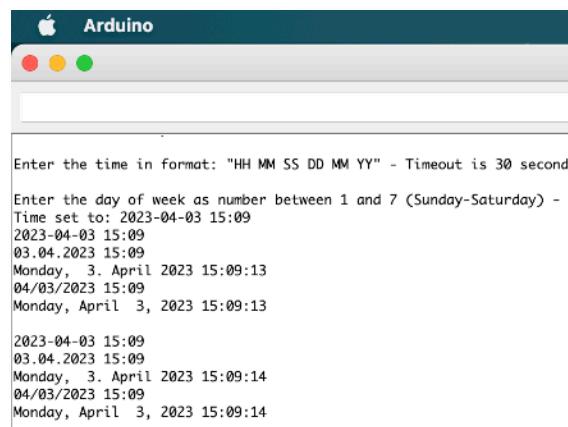


Figure n°12 : Calibration horloge

Après avoir effectué la phase de calibration et de réglages, nous avons procédé à la mise en œuvre du programme principal pour les phases de vol du drone. On a donc réalisé différents tests dans différentes configurations pour vérifier la cohérence des mesures, indépendamment de la configuration du drone.

1. Test du magnétomètre

Dans un premier temps, nous avons affiché les valeurs de cap et d'accélération sur le moniteur série pour valider la première partie du programme (figure n°13).

```
Compass Heading: 107.43
Compass Heading: 107.74
Compass Heading: 107.91
Compass Heading: 108.12
Compass Heading: 107.69
Compass Heading: 107.85
Compass Heading: 107.23
Compass Heading: 107.24
Compass Heading: 107.80
Compass Heading: 107.54
Compass Heading: 107.70
Compass Heading: 105.87
Compass Heading: 107.75
Compass Heading: 107.70
Compass Heading: 108.04
Compass Heading: 106.49
Compass Heading: 106.56
Compass Heading: 105.75
Compass Heading: 97.01
```

Figure n°13 : Affichage du cap

2. Relevé des trames du module LMS303

Afin de comprendre le fonctionnement du capteur en I₂C, nous avons observé les trames envoyées par le module sur les broches SDA et SCL. Nous avons donc relevé le signal suivant :

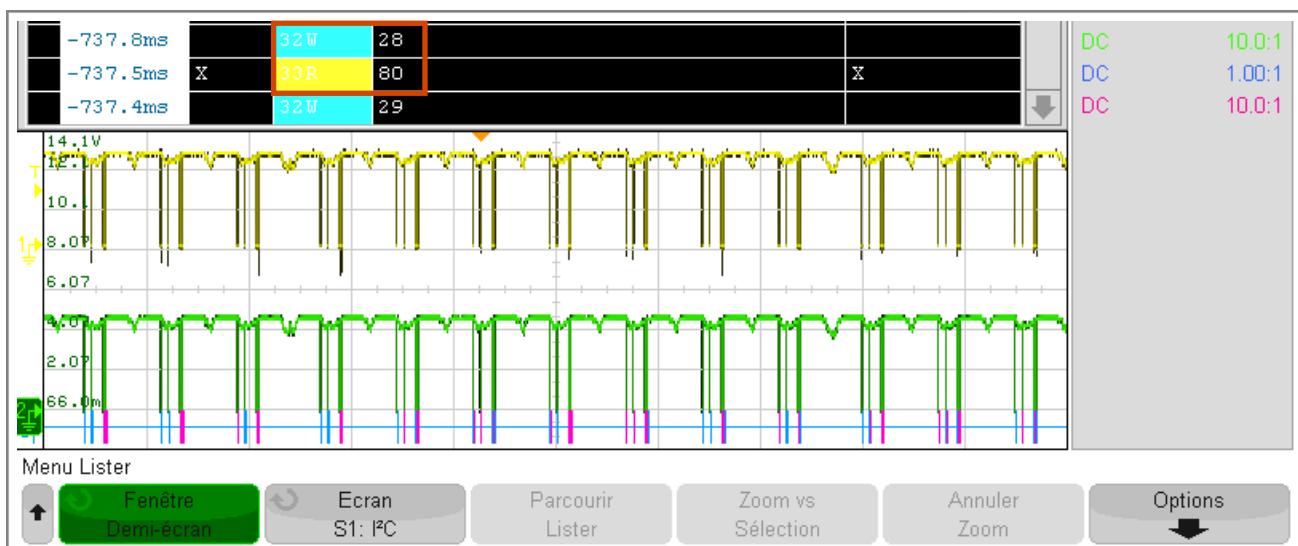


Figure n°14 : Relevé de SDA et SCL à l'oscilloscope

Sur ce relevé, on peut voir en bleu les commandes d'écriture dans les registres du capteur et en jaune les commandes de lecture dans les registres du capteur. Sur cette figure, on peut notamment voir une commande d'écriture "32W" sur le registre 0x32, qui est le registre servant à définir le seuil d'accélération minimal de l'accéléromètre et qui à la valeur par défaut 0x00. Or ici, on peut voir que le seuil est à 0x28. Pour continuer cette fois-ci sur la commande de lecture, nous sommes ici sur le registre 0x33, qui lui sert à définir la durée minimale de l'événement de l'accélération que l'accéléromètre soit déclenché.

Nous avons aussi réussi à relever une trame dans le registre 0x04, correspondant au registre de lecture pour le magnétomètre sur l'axe X. La valeur donnée était 0x51 ce qui correspondant à 81 en décimal. On avait donc une mesure, ici, de $81 \mu\text{T}$ avec le magnétomètre. Sur l'annexe n°3, on peut retrouver les 2 commandes on complet en écriture et écriture sur les registres 0D et 1D.

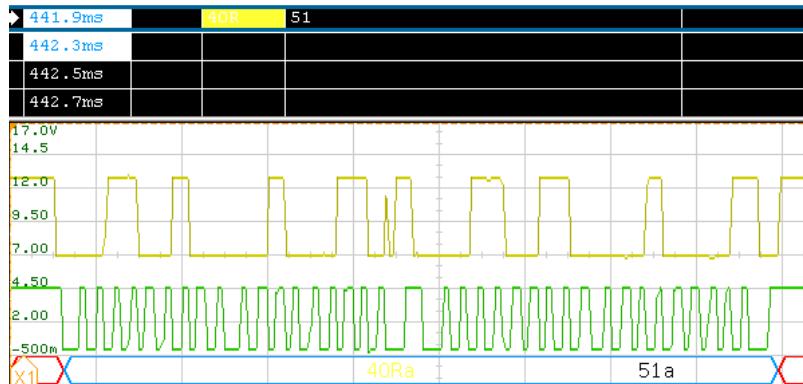


Figure n°15: relevé du champ magnétique sur l'axe X à l'oscilloscope

3. Test pour la création d'un fichier

Comme vu précédemment dans le cahier des charges, après avoir acquis les données via les capteurs, on souhaite les enregistrer sous formes de trames dans un fichier sur une carte SD. Pour se faire, grâce aux fonctions 'NomFichier()', qui pour rappel permet de créer un nom de fichier unique basé sur la date actuelle grâce à l'horloge, et 'OpenFile()', qui elle permet de créer et d'ouvrir un fichier en écriture avec le nom retourner par la fonction 'NomFichier()'. Nous avons donc créée un fichier ".txt" (figure n°16) dans un premier temps, puis un fichier ".csv" (figure n°17), qui lui nous permets de directement séparer nos données via un séparateur pour les ressortir sous formes de tableau. Ensuite, nous avons affiché le nom du fichier dans le moniteur série afin de vérifier si celui-ci se créer bien (figure n°18), tout en écrivant les données mesurées dedans en le vérifiant par un message présent en fin de boucle.

```
20230407.TXT — Partagé
$23EAS90$41227420231_360000
$23EAS90$4122742023-6_318_159.05
$23EAS90$41227420230_170000
$23EAS90$4122742023-6_688_239.13
$23EAS90$41227420230_700000
$23EAS90$4122742023-6_158_009.13
$23EAS90$41227420230_690000
$23EAS90$4122742023-6_158_089.05
$23EAS90$41227420230_350000
$23EAS90$4122742023-6_688_159.13
$23EAS90$41227420230_860000
$23EAS90$4123742023-6_158_158.97
$23EAS90$41237420230_860000
$23EAS90$4123742023-6_158_089.13
$23EAS90$41237420230_860000
$23EAS90$4123742023-6_238_089.13
$23EAS90$41237420230_700000
$23EAS90$4123742023-6_158_089.13
$23EAS90$41237420230_160000
$23EAS90$4123742023-6_688_088.90
$23EAS90$41237420230_520000
$23EAS90$4123742023-6_688_089.13
$23EAS90$41237420230_650000
$23EAS90$4123742023-6_158_239.05
$23EAS90$41247420230_530000
$23EAS90$4124742023-6_688_088.97
$23EAS90$41247420230_000000
$23EAS90$4124742023-6_230_239.05
$23EAS90$41247420230_600000
```

Figure n°16 : Test fonction fichier - Fichier '.txt'

20230407 -						
	A0	A1	A2	A3	A4	A5
Fichier 1						
1	\$23EAS	90	13:51:23	7-4-2023	359.11	0 0 0 0
2	\$23EAS	A0	13:51:23	7-4-2023	-0.23	0.08 8.97
3	\$23EAS	90	13:51:23	7-4-2023	0.18	0 0 0 0
4	\$23EAS	A0	13:51:23	7-4-2023	-0.15	0.08 9.20
5	\$23EAS	90	13:51:23	7-4-2023	359.64	0 0 0 0
6	\$23EAS	A0	13:51:23	7-4-2023	-0.08	0.23 8.97
7	\$23EAS	90	13:51:24	7-4-2023	359.64	0 0 0 0
8	\$23EAS	A0	13:51:24	7-4-2023	-0.23	0.08 8.97
9	\$23EAS	90	13:51:24	7-4-2023	358.93	0 0 0 0
10	\$23EAS	A0	13:51:24	7-4-2023	-0.15	0.08 9.05
11	\$23EAS	90	13:51:24	7-4-2023	358.73	0 0 0 0
12	\$23EAS	A0	13:51:24	7-4-2023	-0.15	0.08 9.05
13	\$23EAS	90	13:51:24	7-4-2023	358.91	0 0 0 0
14	\$23EAS	A0	13:51:24	7-4-2023	-0.23	0.08 9.05
15	\$23EAS	90	13:51:24	7-4-2023	359.64	0 0 0 0
16	\$23EAS	A0	13:51:24	7-4-2023	-0.08	0.15 9.05
17	\$23EAS	90	13:51:24	7-4-2023	359.11	0 0 0 0

Figure n°17 : Test fonction fichier - Fichier '.csv'

La carte SD est initialisée.

Le fichier 20230407.txt a été créé avec succès !

Envoie des trame sur la carte SD...

4. Test pour la déconnection de la carte SD

Enfin, pour la version finale du programme, nous avons ajouté une fonction permettant de vérifier si à chaque boucle de mesure, la carte SD était toujours connectée et si cela n'était pas le cas la dernière trame était envoyée et le programme attendait que la carte SD soit reconnectée. Cette fonction permet de ne pas perdre de trame en cas de déconnexion de la carte SD et permet aussi de faire face au problème de connexion intermittente qui pourrait survenir en cas de turbulence, de vibration, etc. On peut donc voir sur la figure (n°19) ci-dessous, l'affichage sur le moniteur série avec le programme final. On peut donc voir un horodatage à chaque fois qu'un fonction est exécutée, ainsi que les trames envoyées dans le fichier sur la carte SD et enfin l'arrêt du programme après la déconnexion de la carte.

```
14:00:03.044 -> Envoie des trames sur la carte SD...
14:00:03.148 -> $23EAS901359577420230.520000
14:00:03.186 -> $23EASA0135957742023-1.070.239.13
14:00:03.223 -> Envoie des trames sur la carte SD...
14:00:03.294 -> $23EAS901359577420230.690000
14:00:03.328 -> $23EASA0135957742023-0.150.088.90
14:00:03.362 -> Envoie des trames sur la carte SD...
14:00:03.465 -> $23EAS901359577420231.210000
14:00:03.498 -> $23EASA0135957742023-0.080.159.13
14:00:03.535 -> Envoie des trames sur la carte SD...
14:00:03.605 -> $23EAS901359577420231.380000
14:00:03.643 -> $23EASA01359577420230.080.469.05
14:00:03.680 -> Envoie des trames sur la carte SD...
14:00:05.728 -> Carte SD déconnectée
```

Figure n°19 : Carte SD déconnectée

VII. Difficultés rencontrées et améliorations

Au cours de l'avancement du projet, nous avons rencontré différentes difficultés.

Dans un premier temps, nous avons eu de petites difficultés sur la compréhension des différents registre du capteurs. Cela a vite été résolue en lisant bien la datasheet et en faisant posant des questions au profs.

La seconde difficultés que nous avons rencontrées, est la définition de la trame. Car la difficulté a été de centraliser les données de chacun pour définir la taille de la trame ainsi que l'entête. Puis au moment de la programmation, nous nous sommes confrontés au choix du format de chacun des octets. Mais après plusieurs tests, nous nous sommes penchés sur le format 'int' pour l'horodatage de la trame, le label en 'char' et un 'string' pour l'entête et des 'float' pour nos datas, ce qui nous a menée à une trames de 32 octets maximum.

Ensuite, nous avons rencontré une dernière difficulté, plus axée sur le perfectionnisme. En effet, pour le moment le programme ne permet pas d'incrémenter le nom du fichier s'il y en a déjà un existant à la date de création. Cependant, nous nous sommes fortement penché sur le sujet qui peut mettre une grosse problématique par exemple pour la réalisation de plusieurs vol dans une même journée. Effectivement, en cas de plusieurs vols dans une même journée, le fichier où seront écrites les trames sera celui créé lors du premier vol de la journée. Cependant, les données vont simplement s'écrire à la suite des données précédente. Le problème de l'incrémantion vient sûrement du fait que nous utilisons des fonction différente pour créer un nom unique et pour ouvrir le fichier avec ce nom unique.

Enfin, nous pourrions apporter différente modifications aux projet ou a programme. En effet, on pourrait tout d'abord passer sous un réel système temps réel en passant par un RTOS (Real Time Operating System) type Linux ou bien VxWorks, qui un est un RTOS performant pour l'aéronautique et spatial. Ensuite, concernant le code, nous pourrions apporter des modifications pour ajouter un entête à nos fichier qui nous servirait a voir directement l'ordre des données en fonction du type de capteur. Pour finir, on pourrait aussi améliorer le systèmes un ajoutant un gyromètre au systèmes, qui permettrait d'obtenir la position et la direction exact du drone en alliant l'accéléromètre, le magnétomètre et le gyromètre.

VIII. Conclusion

En conclusion, ce projet nous a permis de mettre en application les aspects théoriques et pratiques de la programmation étudié cette année, ainsi que mettre en avant les différents notions de réseaux vue en cours (protocol, registre, etc.) Quant aux difficultés rencontrées, elles nous ont permis d'en apprendre plus sur notre sujet et sur le langage de programmation.

A travers l'objectif d'ajouter de nouveaux systèmes de mesure à un drone nous avons mis en place un déroulé méthodologique pour arriver au bout de ce projet. Tout d'abord nous avons étudié chaque composant à notre disposition afin de comprendre leurs fonctionnement, par la suite nous avons câblé puis programmé la carte Arduino et réaliser un diagramme fonctionnel pour mettre au clair les besoins attendu.

La prochaine étape de ce projet consistera à intégrer pleinement notre capteur au drone ainsi que de le faire communiquer avec le LoRa. Nous pourrions donc alors rajouter un programme, qui prend en compte des requêtes ou bien un programme, qui interprète les données du capteur pour envoyer les messages d'alerte (déviation par rapport à un cap pré définit)

Au terme de ce projet, nous avons un programme fonctionnel, qui envoie les information du magnétomètre et de l'accéléromètre sur une carte SD qui horodate les mesures, ainsi que sur une trame pour pouvoir transmettre en direct les information de vol.

IX. Bibliographie

- Datasheet du capteur LSM303DLHC
- Datasheet du module mikroBUSTM
- Librairie Arduino :
 - *Wire.h* pour la communication I2C
 - *SD.h* pour l'utilisation de la carte SD
 - *Adafruit_LSM303DLH_Mag.h* pour l'utilisation du magnétomètre
 - *Adafruit_LSM303_Accel.h* pour l'utilisation de l'accéléromètre
 - *Adafruit_Sensor.h* pour l'utilisation du module LSM303DLHC
 - *RTClib.h* pour l'utilisation de l'horloge RTC

X. Annexes

- Annexe n°1 : Programme Arduino 23
- Annexe n°2 : Registre du capteur LSM303DLHC..... 27
- Annexe n°3 : Relevé d'une trame complète à l'oscilloscope 28

- Annexe n°1 : Programme Arduino

```

ProjetLPEAS_Compas_Accelero
1 #include <Adafruit_LSM303DLH_Mag.h> // Bibliothèque pour la lecture du capteur magnétique LSM303DLH
2 #include <Adafruit_LSM303_Accel.h> // Bibliothèque pour la lecture de l'accéléromètre LSM303
3 #include <Adafruit_Sensor.h> // Bibliothèque pour la lecture des capteurs Adafruit
4 #include <Wire.h> // Bibliothèque pour la communication I2C
5 #include <SD.h> // Bibliothèque pour l'utilisation de la carte SD
6 #include "Arduino.h" // Bibliothèque standard d'Arduino
7 #include "RTCLib.h" // Bibliothèque pour la lecture de l'horloge RTC
8
9 #define ENTETE "$23EAS" // Entête pour identifier le début de la trame
10#define LABEL_CAP '9' // Label pour le CAP en "
11#define LABEL_ACCELERO 'A' // Label pour l'accélero avec X/Y/Z en m/s
12#define LABEL_V1 '0' // Label pour identifier la version V1 de la trame
13#define PADDING 0 // PADDING pour remplir les trames jusqu'à 32 octets si la trame est inférieur à 32 octets
14
15 File myFile; // Fichier sur la carte SD
16 RTC_DS1307 rtc; // Horloge RTC
17
18 const int pinSD = 9; // Définition de la broche CS pour la carte SD
19
20 Adafruit_LSM303DLH_Mag_Unified mag = Adafruit_LSM303DLH_Mag_Unified(12345); // Objet capteur magnétique
21 Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321); // Objet accéléromètre
22
23 /*
24 Définition des structures pour remplir les trames avec une taille de 32 octets
25
26 Taille de la trame en octets = 32 octets
27 Taille int : 2 octets par int
28 Taille Char : 1 octet par char
29 Taille String : 1 octet par char présent dans le String, soit 6 octets pour l'entête
30 Taille float : 1 octet par float
31
32 */
33 struct MagnetoStruct { // structure pour la trame concernant le cap
34     String entete;
35     char label, version;
36     int second, minute, hour, jour, month, annee;
37     float heading ;
38     int padding1, padding2, padding3, padding4 ;
39 }
40 struct AcceleroStruct { // structure pour la trame concernant l'accélero
41     String entete;
42     char label, version;
43     int second, minute, hour, jour, month, annee;
44     float accelX, accely, accelZ;
45 };
46
47 /* Fonctions setup pour les différents équipements*/
48 void Accelero_setup() {
49     Serial.println("Accelerometer Test"); // Affiche un message sur la console série
50     if (!accel.begin()) { // Si l'initialisation de l'objet accel a échoué
51         Serial.println("Erreur ! LSM303 non détecté !"); // Affiche un message d'erreur sur la console série
52         while (1); // Boucle infinie pour bloquer le programme
53     }
54     accel.setRange(LSM303_RANGE_4G); // Configure la plage de mesure de l'accéléromètre à ±4g
55     accel.setMode(LSM303_MODE_NORMAL); // Configure le mode de mesure de l'accéléromètre en mode normal
56 }
57
58 void Compas_setup() {
59     Serial.println("Magnetometer Test"); // Affiche le message "Magnetometer Test" sur la console série
60     if (!mag.begin()) { // Si la communication avec le capteur de champ magnétique n'est pas établie
61         Serial.println("Erreur ! LSM303 non détecté !"); // Affiche un message d'erreur sur la console série
62         while (1); // Boucle infinie pour bloquer le programme
63     }
64 }
65
66 void Clock_setup() {
67     // Vérifier si le module RTC fonctionne
68     if (!rtc.begin()) {
69         while (1) delay(10);
70     }
71     // Si le module RTC ne fonctionne pas, le programme reste bloqué dans cette boucle while
72     // jusqu'à ce que l'Arduino soit redémarré manuellement
73     if (!rtc.isrunning()) {
74         // Si l'horloge RTC ne fonctionne pas, on la règle avec la date et l'heure du moment où
75         // le programme est téléchargé sur l'Arduino (grâce aux macros __DATE__ et __TIME__)
76         rtc.adjust(DateTime(F(__DATE__)), F(__TIME__));
77     }
78 }

```

```

80 String NomFichier() {
81     //On obtient la date actuelle
82     DateTime now = rtc.now();
83
84     char namefile[14];
85
86     // Crée un nom de fichier unique en fonction de l'heure et de la date actuelles
87     sprintf(namefile, "%d%d%d%d.txt", now.year(), now.month(), now.day());
88     return namefile;
89
90     /*Code pour vérifié si le nom de fichier exste déjà et si oui, on créer un nom avec une extension "-i++". Par exemple : 20230403-1, 20230403-2, 20230403-(i++) */
91     // if (!SD.exists(namefile)) {
92     //     return nameFile;
93     // }
94     // else {
95     //     int i = 0;
96     //     while (true) {
97     //         char namefile2[17];
98     //         String nom = String(now.year()) + String(now.month(),DEC) + String(now.day(),DEC);
99     //         String extension = ".txt";
100    //         String nomExtension = nom + "-" + String(i) + extension ;
101    //         sprintf(namefile2,nomExtension.c_str());
102    //         sprintf(namefile2,"%d%d%d%d-%d.%txt", now.year(), now.month(), now.day(), i);
103    //         if (!SD.exists(namefile2)) {
104    //             return namefile2;
105    //         }
106    //         i++;
107    //     }
108    // }
109 }
110 void OpenFile(String nom) {
111     // Ouvrir le fichier en écriture
112     myFile = SD.open(nom, FILE_WRITE);
113
114     // Vérifier si le fichier a bien été créé
115     if (!myFile) {
116         Serial.println("Erreur : impossible d'ouvrir le fichier !");
117         return;
118     }
119 }
120
121 void CarteSD_setup() {
122     Serial.begin(9600); // Initialisation de la communication série
123     while (!Serial); // Attente de la connexion de l'ordinateur
124
125     if (!SD.begin(pinSD)) { // Initialisation de la carte SD
126         Serial.println("Carte SD déconnectée");
127         while (1); // Boucle infinie en cas d'erreur
128     }
129     Serial.println("La carte SD est initialisée.");
130 }
131
132 void setup(void) {
133     Clock_setup(); // Initialisation de l'horloge RTC
134     delay(50);
135     Accelero_setup(); // Initialisation de l'accéléromètre
136     delay(50);
137     Compas_setup(); // Initialisation du magnétomètre
138     delay(50);
139     CarteSD_setup(); // Initialisation de la carte SD
140     delay(50);
141     Serial.println();
142     OpenFile(NomFichier()); // On ouvre le fichier pour écrire les trames ultérieurement.
143     if (myFile) { // si le fichier a été créé, on affiche son nom dans le moniteur série
144         Serial.print("Le fichier ");
145         Serial.print(NomFichier());
146         Serial.println(" a été créé avec succès !");
147         Serial.println();
148     }
149     delay(500);
150 }
151 }
```

```

154 void loop() {
155 // Obtenir la date et l'heure actuelles
156 DateTime now = rtc.now();
157
158 // On vérifie si la carte SD est présente à chaque boucle
159 if (!SD.begin(pinSD)) {
160   Serial.println("Carte SD déconnectée");
161   while (1);
162 }
163
164 /*Ouverture du fichier d'écriture*/
165 Openfile(Nomfichier());// Appel de la fonction pour écrire dans le fichier créer précédemment
166
167 /*Accéléromètre*/
168 // Acquisition des données de l'accéléromètre
169 sensors_event_t event;
170 accel.getEvent(&event);
171 // On stocke les composantes d'accélération dans des variables flottantes pour avoir une bonne précision
172 float X = event.acceleration.x;
173 float Y = event.acceleration.y;
174 float Z = event.acceleration.z;
175
176 /*Magnétomètre*/
177 // Acquisition des données du magnétomètre (CAP)
178 mag.getEvent(&event); // récupère les données du magnétomètre dans la variable event
179 // Calcul de la direction du nord magnétique (heading) à partir des données du magnétomètre
180 float heading = atan2(event.magnetic.y, event.magnetic.x) * 180 / PI; // On calcule l'angle en radians à partir des données du magnétomètre, puis on le convertit en degrés
181 heading += (heading < 0) ? 360 : 0; // On s'assure que l'angle est compris entre 0 et 360 degrés
182
183 /*Ecriture des données acquises
184
185     Pour écrire dans la trame en hexadécimal les données recues, on pourrait écrire :
186     - myFile.print(String((int)trameMagneto.heading, HEX));
187     - myFile.print(String((int)trameAccelero.accelX, HEX));
188     - myFile.print(String((int)trameAccelero.accelY, HEX));
189     - myFile.print(String((int)trameAccelero.accelZ, HEX));
190 */
191
192 //Stockées les données d'acquisition dans 2 structures différentes permet de réaliser 2 trames avec un format d'en-tête identique
193 struct MagnetoStruct trameMagneto = {ENTETE, LABEL_CAP, LABEL_V1, now.second(), now.minute(), now.hour(), now.day(), now.month(), now.year(), heading, PADDING, PADDING, PADDING, PADDING}; // Tableau pour stocker les échantillons du CAP
194 //On vérifie que la taille de la trame en octets est =>32/
195 // Serial.print("La taille de la structure trameCap est : ");
196 //Serial.println(sizeof(trameMagneto));
197
198 struct AcceleroStruct trameAccelero = {ENTETE, LABEL_ACCELEROMETER, LABEL_V1, now.second(), now.minute(), now.hour(), now.day(), now.month(), now.year(), X, Y, Z}; // Tableau pour stocker les échantillons de l'ACCELEROMETER
199 //On vérifie que la taille de la trame en octets est =>32/
200 //Envoie de des trames sur le moniteur série
201 Serial.print(trameMagneto.entete);
202 Serial.print(trameMagneto.label);
203 Serial.print(trameMagneto.version);
204 Serial.print(trameMagneto.hour);
205 Serial.print(trameMagneto.minute);
206 Serial.print(trameMagneto.second);
207 Serial.print(trameMagneto.jour);
208 Serial.print(trameMagneto.month);
209 Serial.print(trameMagneto.annee);
210 Serial.print(trameMagneto.heading);
211 Serial.print(trameMagneto.padding1);
212 Serial.print(trameMagneto.padding2);
213 Serial.print(trameMagneto.padding3);
214 Serial.println(trameMagneto.padding4);
215
216 Serial.print(trameAccelero.entete);
217 Serial.print(trameAccelero.label);
218 Serial.print(trameAccelero.version);
219 Serial.print(trameAccelero.hour);
220 Serial.print(trameAccelero.minute);
221 Serial.print(trameAccelero.second);
222 Serial.print(trameAccelero.jour);
223 Serial.print(trameAccelero.month);
224 Serial.print(trameAccelero.annee);
225 Serial.print(trameAccelero.accelX);
226 Serial.print(trameAccelero.accelY);
227 Serial.println(trameAccelero.accelZ);
228

```

```

229 // Stockage des données dans la carte SD/
230 // On ouvre le fichier pour écrire les données stockées dans les structures
231 if (myFile) {
232     Serial.println("Envoie des trames sur la carte SD...");
233
234     // Envoie de la trame Cap sur le fichier de la carte SD
235     myFile.print(trameMagneto.entete);
236     // myFile.print(''); // le ';' sert à séparer les mots dans la trame
237     myFile.print(trameMagneto.label);
238     myFile.print(trameMagneto.version);
239     // myFile.print(''); 
240     myFile.print(trameMagneto.hour);
241     // myFile.print(':'); // ':' permet de séparer HH:MM:SS
242     myFile.print(trameMagneto.minute);
243     // myFile.print(':');
244     myFile.print(trameMagneto.second);
245     // myFile.print(''); 
246     myFile.print(trameMagneto.jour);
247     // myFile.print('-'); // '-' permet de séparer JJ-MM-YYYY
248     myFile.print(trameMagneto.month);
249     // myFile.print('-'); 
250     myFile.print(trameMagneto.annee);
251     // myFile.print(''); 
252     myFile.print(trameMagneto.heading);
253     // myFile.print(''); 
254     myFile.print(trameMagneto.padding1);
255     // myFile.print(''); 
256     myFile.print(trameMagneto.padding2);
257     // myFile.print(''); 
258     myFile.print(trameMagneto.padding3);
259     // myFile.print(''); 
260     myFile.println(trameMagneto.padding4);
261
262     // Envoie de la trame Accelero sur le fichier de la carte SD
263     myFile.print(trameAccelero.entete);
264     // myFile.print(''); 
265     myFile.print(trameAccelero.label);
266     myFile.print(trameAccelero.version);
267     // myFile.print(''); 
268     myFile.print(trameAccelero.hour);
269     // myFile.print(''); 
270     myFile.print(trameAccelero.minute);
271     // myFile.print(':'); 
272     myFile.print(trameAccelero.second);
273     // myFile.print(''); 
274     myFile.print(trameAccelero.jour);
275     // myFile.print('-'); 
276     myFile.print(trameAccelero.month);
277     // myFile.print('-'); 
278     myFile.print(trameAccelero.annee);
279     // myFile.print(''); 
280     myFile.print(trameAccelero.accelX);
281     // myFile.print(''); 
282     myFile.print(trameAccelero.accelY);
283     // myFile.print(''); 
284     myFile.println(trameAccelero.accelZ);
285
286     myFile.close(); // Fermer le fichier
287 } else {
288     // Afficher un message d'erreur si le fichier ne peut pas être ouvert
289     Serial.println("Erreur : impossible d'écrire dans le fichier");
290 }
291 Serial.println();
292 delay(50); // Delais de 50us entre chaque loop d'acquisition. Plus le delay est petit, plus il y aura de précision en temps réel. Cependant, nous ne sommes pas réellement sur une équipement temps réel.
293 }

```

- Annexe n°2 : Registre du capteur LSM303DLHC

Name	Slave address	Type	Register address		Default	Comment
			Hex	Binary		
Reserved (do not modify)	Table 14		00 - 1F	--	--	Reserved
CTRL_REG1_A	Table 14	rw	20	010 0000	00000111	
CTRL_REG2_A	Table 14	rw	21	010 0001	00000000	
CTRL_REG3_A	Table 14	rw	22	010 0010	00000000	
CTRL_REG4_A	Table 14	rw	23	010 0011	00000000	
CTRL_REG5_A	Table 14	rw	24	010 0100	00000000	
CTRL_REG6_A	Table 14	rw	25	010 0101	00000000	
REFERENCE_A	Table 14	rw	26	010 0110	00000000	
STATUS_REG_A	Table 14	r	27	010 0111	00000000	
OUT_X_L_A	Table 14	r	28	010 1000	output	
OUT_X_H_A	Table 14	r	29	010 1001	output	
OUT_Y_L_A	Table 14	r	2A	010 1010	output	
OUT_Y_H_A	Table 14	r	2B	010 1011	output	
OUT_Z_L_A	Table 14	r	2C	010 1100	output	
OUT_Z_H_A	Table 14	r	2D	010 1101	output	
FIFO_CTRL_REG_A	Table 14	rw	2E	010 1110	00000000	
FIFO_SRC_REG_A	Table 14	r	2F	010 1111		
INT1_CFG_A	Table 14	rw	30	011 0000	00000000	
INT1_SOURCE_A	Table 14	r	31	011 0001	00000000	
INT1_THS_A	Table 14	rw	32	011 0010	00000000	
INT1_DURATION_A	Table 14	rw	33	011 0011	00000000	
INT2_CFG_A	Table 14	rw	34	011 0100	00000000	
INT2_SOURCE_A	Table 14	r	35	011 0101	00000000	
INT2_THS_A	Table 14	rw	36	011 0110	00000000	
INT2_DURATION_A	Table 14	rw	37	011 0111	00000000	
CLICK_CFG_A	Table 14	rw	38	011 1000	00000000	
CLICK_SRC_A	Table 14	rw	39	011 1001	00000000	
CLICK_THS_A	Table 14	rw	3A	011 1010	00000000	
TIME_LIMIT_A	Table 14	rw	3B	011 1011	00000000	
TIME_LATENCY_A	Table 14	rw	3C	011 1100	00000000	
TIME_WINDOW_A	Table 14	rw	3D	011 1101	00000000	
Reserved (do not modify)	Table 14		3E-3F	--	--	Reserved
CRA_REG_M	Table 16	rw	00	00000000	0001000	
CRB_REG_M	Table 16	rw	01	00000001	0010000	
MR_REG_M	Table 16	rw	02	00000010	00000011	
OUT_X_H_M	Table 16	r	03	00000011	output	
OUT_X_L_M	Table 16	r	04	00000100	output	
OUT_Z_H_M	Table 16	r	05	00000101	output	
OUT_Z_L_M	Table 16	r	06	00000110	output	
OUT_Y_H_M	Table 16	r	07	00000111	output	
OUT_Y_L_M	Table 16	r	08	00001000	output	
SR_REG_Mg	Table 16	r	09	00001001	00000000	
IRA_REG_M	Table 16	r	0A	00001010	01001000	
IRB_REG_M	Table 16	r	0B	00001011	00110100	
IRC_REG_M	Table 16	r	0C	00001100	00110011	
Reserved (do not modify)	Table 16		0D-30	--	--	Reserved
TEMP_OUT_H_M	Table 16		31	00000000	output	
TEMP_OUT_L_M	Table 16		32	00000000	output	
Reserved (do not modify)	Table 16		33-3A	--	--	Reserved

- Annexe n°3 : Relevé d'une trame complète à l'oscilloscope

