

**Sommaire :**

1. *Avant-propos*
2. *Introduction*
3. *Travaux réalisés lors de ce projet*
4. *Conclusions*
5. *Annexes*

Avant-propos :

*Je voudrais tout d’abord remercier l’équipe Géoscience de Montpellier ainsi que Marianne Comte, pour leurs soutiens et leurs aides apportées lors des divers réunions, sans lesquels la réalisation du projet aurait été complexe. Je remercie tout particulièrement Olivier Maudens, pour ses précieux conseils, pour le projet, mais également pour ma future carrière d’ingénieur, et aussi Stéphane Dominguez et Sophie Peyrat pour leurs expertises dans le domaine de la Géologie.*

*Je n’oublie pas mes parents pour leurs soutiens au cours de ses cinq années d’études supérieurs.*

Introduction :

Les séismes sont responsable de pertes humaines et matérielles très importantes. La zone péripacifique est l’une des zones les plus touché, elle comprend 80% de l’activité sismique mondiale. L’un des séismes les plus dévastateurs a été le tremblement de terre de Tohoku, au Japon, qui avaient causé la mort de 15 900 personnes et les dégâts ont été estimé à des centaines de milliards de dollars. Leurs anticipations est donc un enjeu important. Une étude du comportement mécanique et de l’évolution cinématique des modèles expérimentaux grâce à l’analyse de photos numérique est déjà existante. Cependant nous pouvons compléter cette étude avec de nouvelles méthodes de prévision : nous allons donc utiliser un modèle analogique de réseaux d’accéléromètres afin de déterminer quelles sont les signes précurseurs. La mise en place de ce système est avant-gardiste : elle prévoit d’étudier les micro séismes en localisant leurs hypocentres, de déterminer la cinématique du glissement co-sismique au niveau du foyer, afin de mieux prévenir des risques. C’est le pôle Géosciences de Montpellier, situé à l’université de Montpellier qui a envisagé ce projet ambitieux, en faisant appel à Polytech Montpellier, afin de l’aider sur la partie électronique et software.

L’objectif est de créer un système que nous qualifierons de « réseau d’accéléromètres » relié à une interface graphique. Ces capteurs seront implémentés dans une maquette, présente au sous-sol du bâtiment de Géologie. Afin de permettre l’étude, ces accéléromètres doivent être précis afin de mesurer les accélérations de la plaque. Ces données seront alors transmises à l’interface à l’aide d’un microcontrôleur et du port série. Celle-ci sera alors capable d’afficher les données, présenter un aperçu de l’évolution et donner un fichier texte comprenant toutes les données de tous les capteurs. Le but ici est de créer une interface avec affichage de données et de graphiques. L’interface sera d’une grande aide pour l’équipes de géologues géosciences de Montpellier, ainsi que des futurs doctorants travaillants dans ce même domaine, mais elle ne réalisera pas les conclusions sur le plan de l’étude sismiques.

II) Travaux réalisés lors du projet :

1. Résumé du cahier des charges :

Le but du PIFE est d’obtenir un système composé de 12 accéléromètres et d’une interface capable de réceptionner les données t d’afficher leurs évolutions. Afin de simplifier la mise en place du système, nous commencerons par l’installation d’un réseau de 2 accéléromètres. Si les données reçues sont d’assez bonne qualité, nous pourrons créer un réseau de 12 accéléromètres. La première partie de ce PIFE sera de réaliser une interface simple d’utilisation, permettant d’obtenir les données des accéléromètres par un port série et de les afficher sur l’interface, tout d’abord pour un seul capteur, et par la suite pour plusieurs. L’utilisateur doit pouvoir sélectionner l’accéléromètre qui l’intéresse, afin de ne percevoir que les données de celui-ci. Nous devons également pouvoir afficher des graphiques de l’accélération des différents capteurs (un graphique général et chaque accéléromètre aura son graphique). Celle-ci sera réalisé en Python. Deux approches sont possibles : nous pouvons utiliser la librairies Pyserial, ou un hyperTerminal tel que CoolTerm, afin d’écrire les données sur un fichiers texte. L’exploitation de ce fichier sera réalisée en Python, afin d’obtenir les graphiques. l’IHM permettrait donc de simplifier grandement l’exploitation des données. Cette interface devra permettre :

* L’affichage des trois composantes X Y Z de chaque capteur

• Pouvoir choisir le capteur utilisé à l’aide d’un bouton de sélection de l’identifiant du capteur

• Des tests d’intégrité du temps et du nombre de données après appuie sur un bouton défini dans le menu : l’objectif étant de vérifier qu’il n’y est pas de données perdues.

* Modifier la plage d’accélération

La seconde partie du projet sera de permettre la prise de données de manière « *pseudo simultanée* » : Les capteurs IIS3BWD sont des capteurs capables d’utiliser la communication SPI ou I2C, cependant nous utiliserons le SPI lors de ce projet car nous aurons le chip select qui permettant de sélectionner un capteur en particulier sur le réseau et d’en extraire des données. Le problème ici est que nous ne pouvons pas sélectionner un capteur particulier parmi tout le système, à moins d’utiliser 1 microcontrôleur pour 1 capteur. Nous voulons créer un système simple où nous évitons cette dernière solution couteuse, afin de n’avoir qu’un seul microcontrôleur qui contrôle le réseau d’accéléromètre. Une des solutions serait de reliés les capteurs IIS3BWD entre eux sauf le Chip Select, qui sera relié au microcontrôleur. Tous les capteurs seront calibrés sur la même horloge car le but ici est que toutes les mesures sont synchronisées à la même heure. La fréquence de réception de données provenant des capteurs devra être aux alentours des 10 kHz.

1. Point de départ :

Ce projet avait déjà été débuté l’année précédente par un autre étudiant. Il avait alors choisi l’accéléromètre à l’aide de certaines contraintes : il doit être capable de mesurer sur les axes X, Y et Z ; les plages d’accélération doivent être faible (comprises entre g) ; la fréquence d’échantillonnage doit être supérieur à 10 kHz et il doit être pilotage en SPI pour pouvoir choisir le capteur utilisé dans le système. Le seul capteur répondant à ces caractéristiques est le IIS3DWB.

Il avait ensuite réalisé le code de récupération de données sur un microcontrôleur STM32L476RG, dans le but de pouvoir afficher les données reçus sur STM32 Cube Monitor. Ce microcontrôleur est « low power », permettant de gérer la consommation énergétique.

Malheureusement les données reçu et affichés sont obtenues à la fréquence 630 Hz, ce qui bien inférieur à la fréquence demandée dans le cahier des charges. Nous ne pouvons pas utiliser les travaux réalisés sur ce microcontrôleur… Olivier Maudens, ingénieur électronique au pôle Géosciences de Montpellier, a alors choisi de repartir sur de nouvelles bases avec le microcontrôleur Teensy 3.5.

Il a choisi ce microcontrôleur, car il fonctionne à très haute fréquence d’horloge 600 MHz et il est souvent utilisé dans le domaine des traitements des ondes sonores. De plus, il fonctionne sur l’IDE Arduino, qui est simple d’utilisation et permettant aux scientifiques du pôle de géologie de débugger le système facilement en cas de problème. M. Maudens a également effectué un premier code qui permet d’écrire les données d’un seul capteur sur une carte SD et de les envoyer sur le port série.

Nous n’utiliserons donc par l’interface proposé par STM32 Cube Monitor, mais nous allons concevoir également de zéro une interface Python, répondant au cahier des charges.

Pour résumé, nous n’utiliserons malheureusement pas le travail effectué par l’étudiant précédent, mais utiliserons le code de la Teensy pour accéder aux données et le choix du capteur en guise de point de départ.

1. Réalisation de l’IHM :

Nous commençons donc notre projet par la création de l’interface, car nous pourrons tout d’abord la tester à l’aide des données reçus sur un seuls capteur. J’ai choisi de débuter le PIFE par cette partie car je suis à l’aise avec la programmation sur Python, notamment grâce à mes divers projets réalisés via ce langage.

La librairie tkinter permet de réaliser des interface Python, assez facilement grâce à ses nombreuses fonctions. Tout d’abord, débutons par la mise en forme de celle-ci :

Elle sera constituée d’un menu (A) , en haut de la fenêtre et de 3 graphiques (B) , correspondant aux accélération sur X, Y et Z. Nous y ajouterons un menu déroulant (C) afin de permettre à l’utilisateur de choisir les données affichées par les graphes suivant le capteur sélectionné.

Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement

C

B

A

*Figure 1 : Impression d’écran de la page d’accueil de l’interface*

Pour ce faire, nous avons créé une fenêtre, puis nous l’avons séparé en trois parties : 3 canvas y sont disposés. Dans ces 3 parties, nous allons utiliser la librairie *Matplotlib* afin d’afficher des graphiques. Le système nous fournit un temps en µs et des accélérations en mg. Lors du premier test, nous avions choisis d’afficher toutes les données enregistrées depuis le début de l’expérience, mais cela ne fournissait malheureusement pas un rendu assez fluide. La solution est de n’afficher que 100 points provenant des 5000 dernières lignes du fichier data.txt. En effet, dès que nous prenons en compte 1 données, les 50 suivantes ne seront pas considérées dans notre tracé. Néanmoins, le tracé de toutes les données sera possible via la fonctionnalité « *Show Data* » dans le menu *Tools*(A). Dans le menu, l’utilisateur pourra :

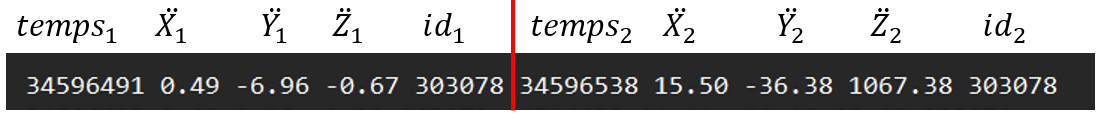
* Afficher enregistrer des fichiers provenant de la découpe du fichier des données capturés dans un dossier de son choix
* Afficher l’évolution de toutes les données d’accélération
* De choisir ou non une échelle pour les accélérations
* De faire des tests de fréquences et d’intégrité pour déterminer le nombre d’erreurs

Tout d’abord, expliquons comment fonctionne la découpe du fichier :

Tout d’abord, l’utilisateur va devoir choisir un dossier où les fichiers vont être enregistrés. Pour ce faire, nous allons utiliser la fonction *filedialog.askdirectory()*. L’endroit sélectionné est renvoyé par cette fonction. Ensuite après avoir tester si l’emplacement existante, nous allons vérifier si la taille du fichier data.txt est supérieur à la taille de découpe. Si celle-ci est inférieur alors il suffit d’enregistrer data.txt sans le découper, sinon il faut le lire ligne par ligne, calculer la taille des données à l’endroit où on se situe dans le document et s’arrêter au moment où la taille dépasse la limite fixée. Un des problèmes que j’ai rencontrés était de ne pas découper en plein milieu d’une ligne. En effet, si à la ligne n+1, la taille dépasse la limite, il faudra découper le document à la fin de cette ligne.

La seconde fonctionnalité importante présente dans le cahier des charges est de pouvoir afficher l’évolution des données sélectionnés. Comme nous travaillons sur un « réseau de capteur », l’utilisateur doit être capable de sélectionner l’accéléromètre qui l’intéresse, l’axe qu’il veut étudier et aussi si il veut afficher toutes les données ou seulement une partie. Si nous affichons toutes les données, à la fin d’une expérience de plusieurs heures, cela risquerait de prendre beaucoup de temps, et il vaudrait mieux n’afficher qu’une donnée sur 10.

Pour réaliser cette fonctionnalité, nous allons avoir besoin de créer une nouvelle fenêtre dans le but que l’utilisateur fasse les choix que nous avons précédemment présenté. Afin de créer cette fenêtre, nous utilisons la fonction *Toplevel*, qui placera la fenêtre au premier plan. Ensuite nous avons besoin d’écrire du texte pour expliquer à l’utilisateur comment faire ses choix. Nous utilisons alors *Label ( fenêtre où afficher le texte, texte à afficher)*. Ensuite pour permettre à l’utilisateur de faire le choix du capteur, nous allons utiliser une *Spinbox*, où nous demanderons l’identifiant (nombre compris entre 1 et le nombre de capteurs du système). Les *Spinbox* ne permettent que de sélectionner des nombres, ce qui est parfait pour demander un identifiant, mais inutile pour demander de sélectionner un axe. Nous allons alors utiliser une *Combobox* pour cela. De plus, nous utiliserons le même procédé pour la sélection de la fréquence. Celle-ci pourra être égale à 10 kHz, qui est la fréquence d’arrivée des données, on affiche donc toutes les données ; 5kHz, ce qui revient à n’afficher qu’une donnée sur 2, ou encore 1kHz, ce qui revient à n’afficher qu’une donnée sur 10. Cette fonctionnalité est utile dans le cas où la maquette est utilisée pour des expériences de plusieurs heures… Une fois que l’utilisateur a précisé ce qu’il voulait étudier, la fonction va lire les données de *data.txt*. Tout d’abord, il faut vérifier que la ligne étudiée n’est pas une erreur ( commence pas par « ERROR ») et que la taille est bien égale à 5 (= nombre de données envoyés par un capteur) x le nombre de capteurs du système. Ensuite il faut déterminer si la donnée doit être prise en compte, car suivant la fréquence d’affichage, nous n’affichons pas toutes les données. Pour ce faire, nous allons vérifier si le reste de la division euclidienne de l’index de ligne par la fréquence en kHz est égale à 0. Si c’est le cas, la donnée devra être affiché. Nous allons donc ajouter la donnée du temps dans la liste temporelle et la donnée du capteur dans la liste des accélérations. Nous sélectionnons l’axe grâce à l’emplacement de la donnée dans la ligne, pour rappel voici l’allure d’une ligne de donnée pour un seul capteur :



*Figure 2 : allure des données récupérées en sortie du port série*

Donc les accélérations suivant X se trace à la -ème place ; celle suivant Y à la -ème place…

Maintenant que les listes sont prêtes, il ne reste plus qu’à afficher l’évolution de l’accélération en fonction en fonction du temps. Nous utiliserons ici le *SpanSelector*, qui met à disposition des outils de zoom sur le graphique.

Une image contenant texte, diagramme, ligne, Tracé

Description générée automatiquement

A

*Figure 3 : affichage des données sur le capteur id 1, axe X*

Comme nous pouvons le remarquer, les données se situe principalement entre ± 25 mg. En réalité, même si le capteur que nous utilisons est utilisé dans l’étude de vibration et est l’un des plus précis du marché, il reste toujours un bruit blanc axé autour de 0 g et d’amplitude 25 mg. Nous remarquons également des pics, causé par des accélérations, comme des séismes… Et surtout des manques dans l’affichage (A).

L’un des principaux problèmes auxquels j’ai été confronté sur ce PIFE est l’identification et l’explication de ces manques de données. Il peut y avoir plusieurs origines possibles :

* Le microcontrôleur Teensy pourrait être responsable : plusieurs erreurs peuvent intervenir tel qu’un blocage dans les valeurs émises ce qui se manifeste par une répétition de données, un temps d’émission trop long ou encore toute les données accélérométriques émises à 0 g.
* La réception des données via le port série : cela se manifeste par une perte de donnée, que nous pourrions détecter à l’aide des identifiants. En effet, sur deux lignes n et n+1 qui se suivent, nous devrions retrouver la relation mathématique suivante :

id[n+1] = id[n] + 1

Pour déterminer si c’est le microcontrôleur qui est responsable, nous devons ajouter des phases de tests dans son code. Ainsi dans le code du Teensy, nous avons ajouté :

* Un test pour vérifier que les accélérations ne sont pas toutes nulles.
* Un test pour vérifier que les données ne se répètent pas. Pour faire cela, nous allons créer un tableau tampon, où nous stockons les 3 derniers données. Si celle-ci sont identiques, il y a une erreur, sinon on supprime la dernière ligne de stockage, on décale les données et on met à jour la première ligne avec les nouvelles données.

Tableau tampon =

Mise à jour

Est-ce que ces données sont égales ?

* Un test pour vérifier si le temps d’émission est bien de . On considère que si le temps émission entre les données d’identifiant n et celles d’identifiants n+1 est de plus de 100 us + ξ, où ξ = 50 us, alors il y a une erreur.

Si un de ces tests a été vérifié, nous allons envoyer un message d’erreur de la forme suivante :

**ERROR SENSOR 1 OFFLINE 0**

Id du capteur code d’erreur

Le code d’erreur est le suivant :

* Code 0 : toutes les données sont à 0.
* Code 1 : Il y a trop de délais entre deux données.
* Code 2 : Il y a un blocage, une répétition suspecte de données.

Ce message sera envoyé sur le port série, à la réception de ce message d’erreur, l’interface reconnaitra le code d’erreur. Maintenant nous pouvons créer la fonctionnalité de test d’intégrité. Le but de cette fonctionnalité est de déterminé s’il y a des erreurs et de déterminer leur source. Nous allons donc créer un fichier texte qui sera sous la forme suivante :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Erreur ne provenant pas du microcontrôleur

Erreur provenant du microcontrôleur

*Figure 4 : mise en forme du « error report »*

Pour créer cette fonctionnalité, nous avons besoin de parcourir le fichier contenant toutes les données : data.txt. Si nous n’avons pas de code d’erreur, nous allons déterminer si les identifiants se succèdent bien :

**Temps id**



303 078 + 1 = 303 079 OK

Si les identifiants ne se succèdent pas, nous allons faire la différence entre les deux temps afin de savoir combien de temps nous avons perdu. Puis nous allons l’ajouter dans la partie de gauche du fichier texte *error report*.

Maintenant si à la place d’avoir reçu une donnée, nous avons reçu un code d’erreur, nous allons alors l’écrire dans la partie de droite du *error report*, ainsi que quelques explications liées au code d’erreur, afin que l’utilisateur n’aie pas à connaitre les codes d’erreur.

1. Réalisation du code du microcontrôleur :
2. Gestion d’un seul capteur :

Le code de gestion d’un seul capteur était déjà réalisé. Celui-ci constitue une bonne base pour la gestion du réseau d’accéléromètre.

*Une image contenant Appareils électroniques, circuit, texte, Ingénierie électronique

Description générée automatiquement*

*Figure 5 : schéma du câblage du microcontrôleur Teensy et d’un capteur IIS3DWB*

Le capteur peut utiliser deux modes de fonctionnement : l’I2C ou le SPI. Cependant, nous voulons gérer facilement un ensemble de capteurs, donc la solution la plus adéquat est l’utilisation du SPI. De plus, c’est un protocole simple et plus rapide que l’ I2C.

Comme nous pouvons le voir sur la figure 5, il y a 4 Chip Selects, nous pourrions donc connecter 4 accéléromètres sur le Teensy. Afin de pouvoir demander les données à un capteur, on passe la pin CS à Low, et une fois que nous avons reçu nos données nous repassons la pin CS à high.

1. Gestion d’un réseau d’accéléromètre :

L’étape suivante est de pouvoir gérer un système composé de 2 accéléromètres. Ceci sera une bonne base par la suite, car c’est une manière de se confronter aux différents problèmes liés à la gestion du « réseau de capteurs », sans avoir les 12 II2S3DWBs à gérer.

Pour le choix de la communication, comme nous l’avons vu précédemment, nous nous sommes orientés vers la communication SPI.

Une image contenant Appareils électroniques, texte, circuit, Composant électronique

Description générée automatiquement

*Figure 6 : schéma du câblage de la Teensy et de 2 accéléromètres.*

Tout d’abord écrivons la fonction setup() :

La carte Teensy 3.5 possède un lecteur de carte SD intégré. Pour les expériences de plusieurs heures, il peut être important d’avoir une sauvegarde externe des données. Nous avons donc besoin d’initialiser la carte SD. Puis il faut alors configurer la liaison SPI : pour cette partie, nous n’avons besoin que de 2 Chips Selects : nous allons utiliser la pin 10 et la pin 37. Afin de ne pas avoir de problème lié au SPI, au niveau de la fréquence, nous donnons comme fréquence maximale : 10 MHz. Dans les autres paramètres, nous travaillerons également le MSBFIRST (Most Significant Bit First) et le mode du SPI = SPI\_MOD0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode | Clock Polarity (CPOL) | Clock Phase (CPHA) | Output Edge Falling | Data Capture |
| SPI\_Mode 0 | **0** | **0** | **Falling** | **Rising** |

Nous avons choisi ce mode par défaut.

Maintenant, il faut également tester si la liaison Chip Select fonctionne. Nous allons utiliser le registre « WHO I AM » :

Register adress : Défaut :

0x0F 0x7B



Nous avons donc besoin de passer le Chip Select au niveau bas, de transférer l’adresse du registre sur le Spi. Nous nous attendons a recevoir, comme réponse, la valeur 0x7B. Si c’est bien le cas, la liaison SPI est prête, sinon nous allons devoir réitérer le test afin d’attendre que la liaison soit prête. Une fois ce test validé, nous pouvons alors reset le capteur et l’initialiser.

Maintenant que la fonction setup est prête, nous pouvons préparer la fonction loop :

Nous allons avoir besoin de tableaux « tampons ». Nous allons stocker les valeurs provenant des accéléromètres. De plus nous avons besoin de connaitre le temps d’envoie. Nous allons tout d’abord écrire les données dans le fichier texte stocké dans la carte SD. Puis ensuite, nous allons envoyer sur le port série nos données. Comme mentionné dans les parties précédentes, nous désirons que les données envoyées soient sous cette forme :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Temps |  |  |  | Identifiant |

Pour calculer le délai, nous avons besoin de prendre le temps de réception des données en y soustrayant le temps de l’initialisation du système. De cette manière, en utilisant un seul microcontrôleur Teensy, nous aurons une horloge synchronisée pour tous les capteurs.

L’identifiant n’est qu’une incrémentation :

A l’instant t, la donnée à un identifiant *id[n],* alors la prochaine donnée aura l’identifiant

*id[n+1] = id[n] + 1*

Cette méthode d’incrémentation va nous être utile dans l’interface lorsque nous allons faire des tests d’intégrités. En effet, lorsque nous effectuons un test d’intégrité, l’interface va, si elle n’a pas reçu un message d’erreur du microcontrôleur, vérifier si les identifiants se suivent bien. Si ce n’est pas le cas, nous avons donc une erreur. Elle va alors calculer le temps de perdu entre les deux données. Pour les données, il faut faire différents tests pour identifier s’il n’y a pas eu une erreur au niveau de la Teensy. Il y a 3 tests possibles :

* Vérifier que ne sont pas tout les trois nulles (sinon cela signifie que le capteur est en mode erreur). Ceci sera l’erreur de code 0.
* Vérifier que ne se répètent pas trois fois d’affilés, car lors de nos nombreux tests, nous nous sommes rendus compte qu’il pouvait y avoir des blocages au niveau des valeurs renvoyé par les capteurs (erreur de code 1).
* Vérifier que le délai entre deux groupes de données émis du même capteur est cohérent : celui-ci doit être inférieur à est une marge d’erreur. Nous informerons l’interface de cette erreur via le code 2.

Maintenant que devons nous faire lorsque nous sommes dans l’un de ces trois cas de figure ?

La meilleure solution que nous avons trouvée semble être le reset du capteur qui pose problème. De plus, lorsque l’un des capteurs est en état d’erreur, il faut le signaler à l’interface pour que lorsque l’interface fait son error report, elle puisse écrire qu’il y a eu une erreur provenant de la Teensy.

1. Evolution des taux d’erreurs :

Malheureusement, lors des différents tests que nous avons effectuer avec notre système et l’interface, nous remarquons qu’il y a différentes erreurs. Lors d’expérience d’une quinzaine de minutes, on peut calculer le taux d’erreur :

=

Ce taux est d’environ 15%. C’est notre fonctionnalité de test d’intégrité qui nous donne un aperçu de ces erreurs et détermine la source de celles-ci.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

B

A

Erreurs ne provenant pas de la Teensy

Erreurs provenant de la Teensy

Résumé du *error report*

*Figure 7 : error report type provenant d’un système de 2 capteurs IIS3DWBs*

Nous remarquons que les erreurs provenant de la Teensy sont rares, ne durent pas dans le temps et ne déclenchent pas de nouvelles erreurs. Par contre les erreurs ne provenant pas du microcontrôleur sont récurrentes et durent longtemps. Les causes de celles-ci peuvent être dû à la liaison série ou encore au logiciel CoolTerm. Au début du projet, nous avons travailler avec la librairie Python Pyserial, permettant de récupérer les données provenant du port série. Mais nous nous sommes rendus compte que beaucoup de données étaient perdu, nous nous sommes donc réorientés vers le logiciel CoolTerm que Olivier Maudens m’avait conseillé. Même si ces 2 approches sont valables et devrait arriver à capturer les données correctement, CoolTerm semble, en pratique, le plus adapté.

Grâce à *l’error report*, nous pouvons remarquer plusieurs choses : les erreurs semblent aller de pairs : comme nous pouvons le voir sur la figure 7, au point A. nous retrouvons le même temps perdu deux fois d’affilés, ce qui peut signifier que lorsqu’il y a une erreur, elle se manifeste sur les deux capteurs en même temps. De plus, pour appuyer cette hypothèse, nous retrouvons le même temps perdus pour le 1er et le 2nd capteurs (B).

Même si le cahier des charges impose une fréquence de fonctionnement au minimum de 8kHz, essayons de trouver une fréquence de fonctionnement du système optimale :

Conclusion :

Nous avons redémarré ce projet de presque zéro, les travaux réalisés l’année dernière ne correspondant pas aux spécifications faites dans le cahier des charges. Nous nous sommes réorientés vers un microcontrôleur Teensy, fonctionnant sur l’IDE Arduino, avec des accéléromètres IIS3DWB. Nous avons donc créé une interface graphique permettant une étude simplifiée des données provenant des capteurs. Nous avons également créé le code du microcontrôleur Teensy, permettant de gérer le « réseau » d’accéléromètres.

Malheureusement, nous avons manqué de temps, nous n’avons pas réussi à corriger les pertes de données ni à déployer le système dans la maquette. Le projet a toujours été vu comme un travail améliorable dans le futur, c’est pour cela que nous avons mis en place des fonctionnalités tel que *l’error report* qui met nous donne des informations importantes sur les erreurs : quand ont elle eut lieu ? y a-t-il une répétition ? quelle est leur durée ? … Nous avons également pu déterminer une fréquence de fonctionnement optimale, entrainant le moins d’erreurs possibles.

L’élément important que j’ai apporté pour ce projet est la possibilité de pouvoir répertorié les erreurs directement via l’interface. Celle-ci a été programmé entièrement en Python, ce qui a permit de pouvoir la mettre en forme selon un cahier des charges qui avait été précisé, à l’avance, par l’équipe Géoscience de Montpellier.

Ce projet n’est donc pas encore fini, ni prêt à être publier dans un article. La partie la plus importante est de pouvoir augmenter la fréquence de fonctionnement sans augmenter le nombre d’erreur. Bien sûr, il va être difficile de faire fonctionner le système sans erreurs, mais il faudra trouver dans le futur une solution technique permettant de pouvoir négliger ces anomalies.

Ce projet a été bénéfique pour moi. Il m’a permis de démarrer un projet, de chercher des solutions techniques créatives et de les mettre en œuvre, d’organiser les réunions comme un chef de projet. Cela a été une expérience enrichissante car elle m’a permis de travailler avec d’autres scientifiques de divers domaines tels que la géologie ou encore des ingénieurs en électroniques. Travailler sur mon Projet de fin d’étude m’a également de d’approfondir mes connaissances dans les systèmes embarqués et de les appliquer. C’est également un travail utile pour notre société car elle permettra dans le futur de mieux étudier les séismes, de les prévoir et d’éviter beaucoup de dégâts matériels et humains. En plus des connaissances techniques, j’ai également développé des compétences transversales telles que la gestion du temps, notamment ave des diagrammes de Gantt, la communication lors de nos nombreuses réunions, essentielles pour mon développement professionnel futur.

Bibliographie :

* *Code du projet (IHM, codeTeensy) :*

<https://github.com/sebastien-doyez2812/School_projects/tree/main/End_of_Study_Project>

* *Datasheet IIS3DWB:*

<https://www.st.com/en/mems-and-sensors/iis3dwb.html>

* *Site ST pour le choix de l’accéléromètre :*

<https://www.st.com/content/st_com/en.html>

* *Site Pomad :*

<https://www.pomad.fr/PoMAD/stm32>