Projets Industriels – Elec4

Année scolaire 2014-2015

Rapport de Projet

Création d'un sismomètre pédagogique de qualité professionnelle

Version 1

Etudiants : Jonathan DAVID, Clément LE MARQUIS, Mickaël RENAULT

Encadrant : Luc DENEIRE, Enseignant Chercheur à Polytech Nice-Sophia Antipolis.

Industriel : Jean-Luc BERENGUER, Professeur de Sciences de la Vie et de la Terre, au Centre International de Valbonne

Ecole Polytechnique Universitaire de Nice Sophia-Antipolis, Département électronique
930 route des Colles, 06410 BIOT

REMERCIEMENTS

Avant de commencer ce rapport, nous tenions à remercier certaines personnes, qui nous ont permis de mener à bien ce projet, en nous apportant aide, conseils et connaissances quant à la sismologie.

Nous remercions donc en premier lieu monsieur Luc DENEIRE, enseignant chercheur à l'école polytechnique universitaire Polytech Nice-Sophia Antipolis et encadrant de ce projet, pour la confiance qu'il a en nous et l'aide qu'il nous a apportée. Nous remercions également monsieur Robert PILLET, sismologue à GeoAzur, de nous avoir accordé ses vendredis après-midi pour nous permettre de travailler sur ses sismomètres, de les comprendre, et de créer la partie électronique. Pour finir, nous remercions monsieur Jean-Luc BERENGUER, professeur de SVT au Centre International de Valbonne, qui a contribué au lancement du projet et nous a accueilli dans les locaux du CIV pour nous préciser les enjeux et organiser des rencontres avec les élèves, tant dans leurs locaux qu'à Polytech.

SOMMAIRE

Introduction

Le principe de ce projet est de concevoir un sismomètre low-cost accessible et pédagogique, puisqu'il doit pouvoir être implanté dans d'autres écoles Erasmus partenaires tout autour du globe. Quatre écoles européennes sont en concurrence autour de ce projet : le CIV, une école italienne et deux écoles anglaises.

Notre projet s'articule autour de trois objectifs. Le premier est pédagogique. Les élèves qui suivent ce projet ont des cours de sismologie, ils doivent être capables de comprendre le fonctionnement de la chaine d'acquisition complète de notre sismomètre. Le second objectif concerne les coûts. Le but du projet est de faire un sismomètre low-cost, qui ne doit pas dépasser les 200 euros. Nos recherches quant au matériel et technologies qui seront utilisées ont donc été en partie guidées par ce critère. Le dernier objectif est un objectif de portabilité. En effet, le produit final se doit d'être ni trop encombrant ni trop fragile, car il doit pouvoir être transportable.

Nous pouvons distinguer trois parties prenantes à ce projet. Tout d'abord un groupe d'une quinzaine d'élèves du CIV, en seconde pour la plupart, qui se sont intéressés au projet et ont passé toute une partie de l'année à étudier la sismologie. On retrouve ensuite toute une équipe pédagogique composée de professeurs du CIV, d'un enseignant chercheur de Polytech, et d'un sismologue de GéoAzur. Enfin, nous, trois étudiants en quatrième année d'électronique à Polytech Nice-Sophia, contribuons également en travaillant sur le développement du produit.

Nous avons choisi ce projet pour diverses raisons. Tout d'abord pour l'aspect pluridisciplinaire du sujet. Les parties mécaniques, automatiques, électroniques et informatiques sont toutes intrinsèquement liées, et notre objectif est de les faire fonctionner ensemble. Il y a ensuite une partie portant sur la pédagogie. Nous avons rencontré quelques fois les élèves du CIV qui ont suivi ce projet, dans le but de leur expliquer notre progression et nos idées, en simplifiant l'information à leur niveau de connaissances.

Au cours de ce rapport, vous découvrirez notre avancée, les choix que nous avons fait, les résultats obtenus jusqu'à présent, ainsi que quelques pistes d'améliorations possibles du sismomètre actuel.

Chapitre I : Gestion du projet

I.1. Calendrier du projet

Le calendrier réel du projet a été synthétisé ci-dessous de façon à percevoir clairement les différentes étapes de notre travail et le temps consacré.

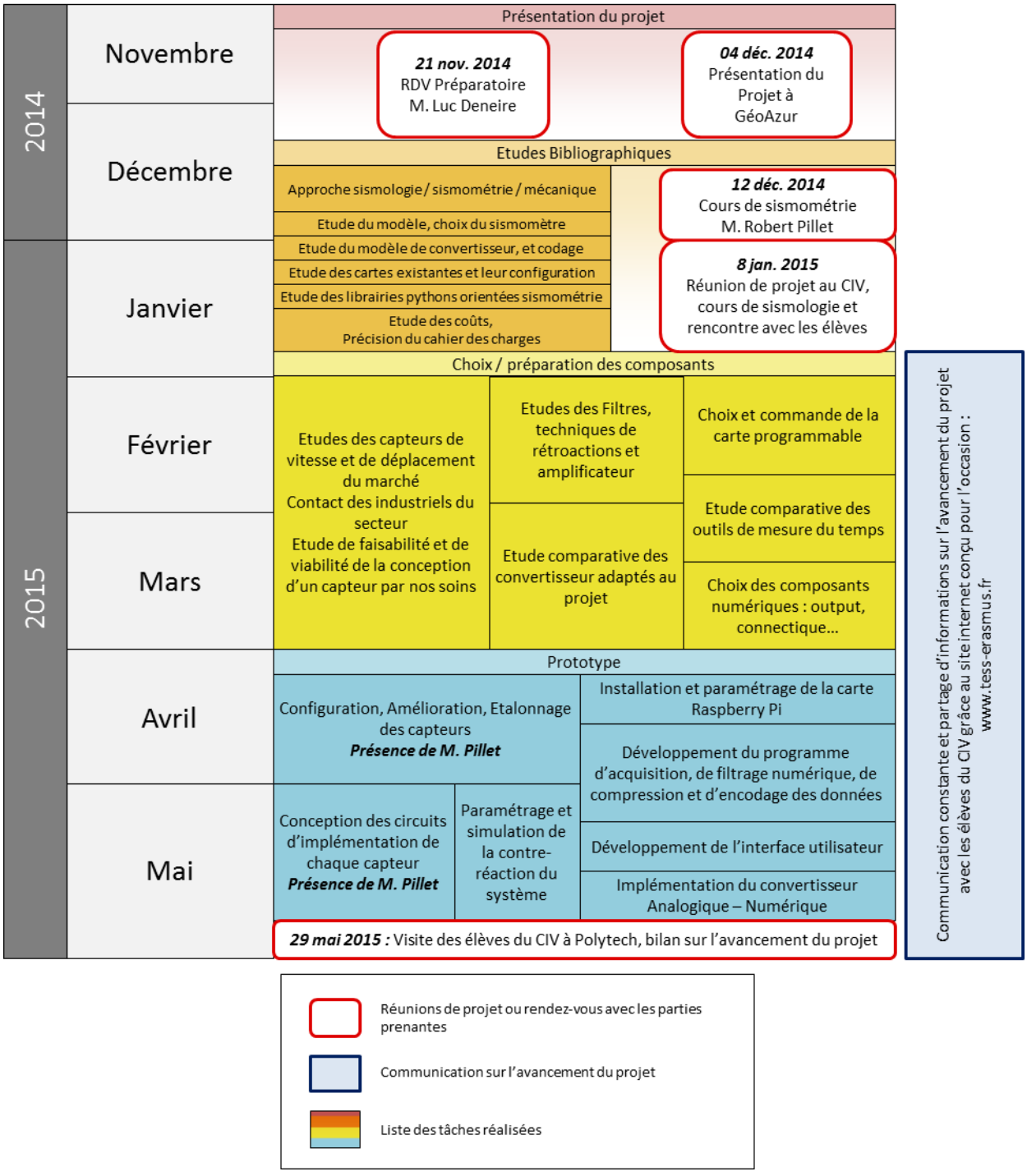


Figure I.1 *Calendrier réel du projet*

Ce calendrier diffère légèrement de celui qui a été programmé en début de projet.

En effet, nous avons consacré plus de temps que nous l'avions estimé pour l'étude, le choix, et la préparation des composants nécessaires à la bonne réalisation de notre mission. La recherche d'un capteur conforme au cahier des charges que nous nous sommes fixé, à savoir de faible coût et avec une résolution de l'ordre de 10^{-9}m, s'est révélée assez complexe. Nous avons donc souhaité prendre le temps suffisant pour explorer toutes les possibilités, les pondérer, et choisir la ou les solutions les plus adaptées. Le capteur étant le premier élément de la chaine d'acquisition électronique, nous ne pouvions pas laisser cette étape de côté car d'autres composants en dépendent directement.

Au lieu d'une organisation présenté à l'origine de manière séquentielle, c'est-à-dire étape par étape, nous nous sommes répartis en « domaine d'étude », afin de travailler en parallèle et gagner en efficacité. On retrouve bien ce fonctionnement sur le schéma ci-dessus où nos tâches sont maintenant disposées en plusieurs colonnes pour un même instant *t*. Cela nous a notamment permis d'explorer davantage de pistes dans nos travaux, en étant chacun responsable et autonome dans notre propre champ d'étude.

Le temps d'élaboration du prototype a également été prolongé de façon à apporter plusieurs solutions au projet, en présentant au final trois versions possibles au client, aux caractéristiques différentes. Nous avons apporté un soin particulier à ouvrir les champs possibles du produit, en offrant une certaine modularité, une capacité d'adaptation et d'évolution du produit pour d'éventuels successeurs. Cette attention prend du temps, mais garantit la continuité de la mission qui nous a été confiée, et le succès du projet pour les étudiants du CIV avec lesquels nous travaillons.

Ainsi, nous observons que les étapes de finalisation, de mise en situation, et d'optimisation du produit fini n'ont pas encore pu être réalisées, mais cela au profit d'un meilleur aboutissement technique dans notre travail.

I.2. Plan de projet

Dans les premiers jours suivant notre rendu bibliographique, nous nous somme consacrés à l'élaboration d'un plan de projet. Ce document public présente notre ligne directrice, la structure de notre travail, et les priorités que nous avons établies. Ce document est communiqué à nos responsables, auquel est joint le diagramme de Gantt associé, retraçant l'évolution espérée de ce plan de projet au cours du semestre. Nous détaillerons ci-dessous le plan envisagé car il présente un intérêt immédiat pour la suite de la lecture du présent rapport.

Plan de projet

Plan général des tâches à effectuer dans le cadre du projet Sismomètre Tess Erasmus+

Gestion de Projet

R1. Préparation du plan de projet
R2. Révision du calendrier
R3. Mise en forme, rédaction des protocoles, diffusion des documents

Capteur

1. Définition des besoins du capteur (résolution, fonction) et marge d'erreur
2. Etude des différents capteurs répondants aux exigences
3. Contact d'industriels pour obtenir devis et conseils quant aux capteurs potentiels

4. Classement selon leur pertinence (précision/coût)

Choix du capteur

5. Définitions des données de sortie du capteur (courant, tension)
    Analyse théorique des distorsions de sortie (ou bruit), prévision d'un filtrage possible
6. Définitions des données d'entrée du capteur (courant, tension)

Convertisseur Analogique numérique

1. Sachant le champ de mesure du capteur, déterminer le champ de numérisation du signal (soit x le nombre de bits) : *x=log(champs de mesure du capteur)/log2* à arrondir si possible à l'entier supérieur
Définition des besoins du convertisseur AN
    nb de bits (étape précédente), cadence minimale, interface de sortie (SPI ou I2C de préférence)
2. Etude des différents CAN répondants aux exigences
3. Classement selon leur pertinence

Choix du CAN

4. Vérification de l'implémentation pratique (support, exigence, alimentation, consommation, température)
Analyse théorique des distorsions (ou bruit) induites par la conversion
    Prévision d'un filtrage possible

Implémentation fonctionnelle d'une rétroaction

1. Trouver les méthodes d'implémentation d'une rétroaction numérique (montage fait-main type ASIC, processeurs, cartes Arduino, etc.)
2. Etude des modules CAN et CNA compatibles
3. Définition des exigences techniques : alimentation, courant, cadence, temps de réponse, flux de sortie
4. Classement de ces méthodes selon leur pertinence (performance, coût, temps de développement)

Choix de la méthode d'implémentation de la rétroaction

Convertisseur Numérique Analogique

1. Définition des besoins du CNA (flux en entrée, cadence, amplitude de sortie, qualité du signal de sortie)
2. Etude des différents CNA répondants aux exigences
3. Analyse théorique des distorsions (ou bruit) induites par la conversion
    Prévision d'un filtrage possible

Choix d'un CNA

Association des blocs fonctionnels

1. Comparaison des entrées et sorties des blocs précédents (amplitude, courant, bruit…)
2. Choix des améliorations à implémenter avant connexion (amplificateurs, filtres)
3. Schémas techniques des améliorations et détail des composants nécessaires

Schéma technique global de la chaîne de numérisation
Schéma technique simplifié de la chaine de numérisation

Précision du micro-contrôleur

1. Mise en équation de la rétroaction du système
2. Détermination des éléments nécessaires à l'implémentation (composants analogiques, filtres, morceaux de code, etc.)
3. Préparation d'un éventuel recalibrage externe (composants réglables, ou identification des fichiers de code à modifier). Exemple : bouton permettant l'arrêt total de toute oscillation.

Détail du protocole de programmation du micro-contrôleur, ou schéma du montage analogique

4. Détermination des paramètres (coef. PID, valeur des constantes, des composants, etc.)

Simulation

Préparation de la carte Raspberry Pi

1. Définition des besoins (alimentation, connectique)
2. Rédaction du protocole de mise en route de la carte
3. Installation des prérequis pour le traitement des données (bibliothèques python, etc.)
4. Etude des différentes méthodes de synchronisation à l'heure universelle
5. Classement de ces méthodes par pertinence (coût, précision)

Choix et installation des modules nécessaires pour la synchronisation à l'heure universelle

6. Etude des différents modules supplémentaires à implémenter (boutons démarrer, arrêter, réinitialiser, recalibrer, clé wifi)

Choix et implémentation des différents modules supplémentaires

Réception des données dans la Raspberry Pi

1. Activation des protocoles nécessaire à la réception de données (I2C, SPI, GPIO)
2. Etude des protocoles et des normes dans le traitement de l'information sismique
3. Programmation des interfaces et vérification

Décodage des données et affichage

4. Test et implémentation éventuelle de filtrage numérique
5. Etude des protocoles de stockage et choix d'un encodage préférentiel

Implémentation de l'encodage, stockage, décodage et adaptation à l'interface graphique

6. Implémentation du partage de données selon réseau (cloud éventuel, ou stockage distant)

Vérification et tests

1. Association de la chaîne complète
2. Calibrage des composants
3. Test de fonctionnement
4. Tests comparatifs avec d'autres dispositifs existants (avec élèves)
5. Amélioration du système

Contrôle de respect du cahier des charges

Clôture du budget

Propositions d'ouverture

Pistes d'amélioration

Proposition de packaging

Documentation
R1. Rédaction des protocoles de construction R2. Rédaction d'un manuel d'utilisateur R3. Rédaction du rapport de projet

Ce document construit le socle de notre travail, et permet d'expliquer aux parties prenantes du projet les détails de notre mission.

Nous produisons également un document protocolaire en interne, constitué d'un tableau à remplir à chaque étape de notre travail. Ainsi, pour chaque tâche, nous y renseignons l'action effectuée, le temps consacré, le coût à considérer, et les actions futures à prévoir. Un code couleur nous permet notamment de cibler rapidement les actions terminées, en cours, et non effectuées.

I.3. Réponse au cahier des charges initial

Cahier des Charges établit en date du 22 jan. 2015	Etat d'avancement
Capteur sismométrique	
Le LVDT que nous évoquerons par la suite est la solution apportant le meilleur compromis précision/coût, nous devrons donc implémenter et calibrer ce capteur directement sur le sismomètre du projet.	Deux capteurs sont disponibles : un LVDT et un capacitif, chacun monté sur un sismomètre, et implémenté sur une plaquette.
Numérisation du signal et rétroaction	
La numérisation doit être précise et ne générer qu'un bruit minimum. Pour cela, nous travaillerons sur la conversion de type sigma-delta. La rétroaction par PID sera implémentée numériquement pour un meilleur calibrage et une meilleure flexibilité. Le processeur utilisé pourra être celui d'un micro-contrôleur ou d'un micro-ordinateur. La partie numérique traitant le signal de la boucle de rétroaction ne sera pas forcément la même que celle qui traite les données en sortie de chaine.	La numérisation du signal est réalisée par un convertisseur Sigma-Delta de 18bits proposant jusqu'à 240 éch/s. La rétroaction a été faite analogiquement, et fonctionne avec le sismomètre muni du LVDT. Le forceur est un couple aimant/bobine, dans laquelle passe un courant.
Acquisition et affichage des données	
L'acquisition du flux de données doit être envisagée par interface I2C ou SPI, selon le type de convertisseur Analogique Numérique. Les signaux seront reçus par le port GPIO d'une carte Raspberry Pi de modèle B+, fonctionnant sous le	Le convertisseur Sigma-Delta communique parfaitement avec la carte Raspberry Pi en utilisant le protocole I2C. Les signaux sont reçus par les port GPIO 3-4-5-6 de la carte Raspberry Pi B+ fonctionnant

système d'exploitation Rasbian Wheezy. Le traitement des données sera programmé en Python sous IDLE3. Il permettra entre autre, de stocker le signal, de l'encoder au format international SEED, mais également de traiter numériquement le signal et d'afficher sous différentes formes les données du sismomètre. L'affichage des données peut être, selon le budget, soit par un mini-écran intégré au produit, soit à la charge de l'utilisateur. L'enregistrement des données se fait sur une micro-SD de classe 10 connectée à la Raspberry PI.	sous Raspian Wheezy Le traitement des données est programmé en Python2.7 (meilleure compatibilité des librairies). Les signaux sont automatiquement filtrés, encodés au format MSEED (plus optimisé) et stockés sur la carte selon un agencement clair. L'affichage reste à la charge de l'utilisateur. L'enregistrement est réalisé sur une carte micro-SD 8Go de classe 10.

I.4. Analyse des coûts

Produit	**Prix**	**Fournisseur**
Résistances, transistors, diodes	2€	Farnell
AOP270	15,93€	Farnell
Microchip NE5521N	38€	Farnell
Condensateurs	3€	Ebay Fr
Support sismomètre (profilé alu)	30€	Brico Dépôt
Pivot + plateau sismomètre	5€	Brico Dépôt
Ressort	2€	Brico Dépôt
LVDT	1€	Fait main
Convertisseur Analogique Numérique Sigma Delta 18bits	23,93£ *FPI*	ABElectronics UK
Raspberry Pi B+	35€	Farnell
Boitier transparent pour Raspberry Pi B+	3€	Amazon Fr
Carte microSD 8Go classe 10 avec adaptateur SD	5,29€ *FPI*	Amazon Fr

Hub USB (Alimente la Raspberry et permet de disposer de 3 ports USB supplémentaires), NLUSB2-222P	10,44€	ModMyPi
Carte RTC Haute précision DS3231	1,73€ *FPI*	Ebay Fr

FPI : Frais de Port Inclus

Nous sommes parvenus à diminuer les coûts de certains composants par rapports aux estimations réalisées dans notre document bibliographique. Nous avons saisi l'importance de la contrainte budgétaire du projet et nous sommes efforcés à chercher les meilleures opportunités. Finalement, nous avons un coût total d'environ 180 euros, nous respectons donc l'objectif low-cost du projet.

I.5. Communication

Notre projet portant sur la « technique » dépend d'un projet plus global, d'échelle européenne, dans lequel sont investis les étudiants du CIV : TeSS Erasmus+ (Teaching Seismology at School).

Nous avons pu rencontrer ces lycéens, échanger avec eux et comprendre notre rôle respectif dans cette aventure. Ce rendez-vous nous a fait prendre conscience de la dimension de notre mission et des attentes des élèves et de l'équipe pédagogique. Nous avons donc souhaité, par esprit d'équipe avec ces étudiants qui approfondissent chaque semaine leurs connaissances en sismologie et en sismométrie, proposer une plateforme d'échange de données pour partager notre avancement respectif.

Cette plateforme se présente sous la forme d'un site web de nom de domaine www.tess-erasmus.fr avec une base de données suffisamment grande pour accueillir tous nos travaux sur toute la durée du projet. Elle se structure de la manière d'un blog, avec plusieurs types de comptes utilisateurs (un profil Elève, un profil Equipe Pédagogique, et un profil Electronicien), et leur espace réservé sur lequel chacun peut poster le contenu qu'il juge pertinent.

Après plusieurs mois de fonctionnement, nous sommes ravis d'observer que ce site est un lieu de rassemblement de documents constructifs : vidéo et présentations réalisée par les élèves ; cours, documents bibliographiques et documents officiels postés par l'équipe pédagogique ; articles, schémas, présentations et rapports ajoutés avec notre compte.

Ce moyen de communication moderne porte notre projet et lui donne la dimension que chacun lui souhaite, grâce à la contribution de tous.

Chapitre II : Implémentation du capteur et de la chaine d'acquisition analogique

II.1. Les trois capteurs

Le choix du capteur, et donc le démarrage concret du projet, est ce qui nous a pris le plus de temps. En effet, il s'agit du premier élément de la chaine d'acquisition. Nous disposions au départ d'un capteur LVDT (expliqué dans la partie suivante) fourni par Mr. Pillet, fait à la main, disposant par conséquent d'une certaine incertitude de mesure, et n'étant initialement pas très exploitable. Nous avons alors pris la décision de nous intéresser aux capteurs industriels.

Il nous a donc fallu réfléchir à quel type de capteurs nous utiliserions, tout en restant à bas prix, ayant en tête le côté low-cost du projet. Nous avons alors contacté des entreprises spécialisées dans les capteurs de déplacement, et nous avons rapidement pris conscience que cette voie n'était pas exploitable, les capteurs les moins chers ayant une fourchette de prix entre 500 et 1000 euros.

Nous sommes donc retournés vers le capteur que nous avions alors, le LVDT, auquel s'ajouta plus tard un capteur capacitif, puis un capteur optique industriel, ce qui nous a permis de comparer les performances du low-cost fait maison avec la qualité professionnelle de l'industrie.

II.1.1. Le LVDT (Linear Variable Differential Transformer)

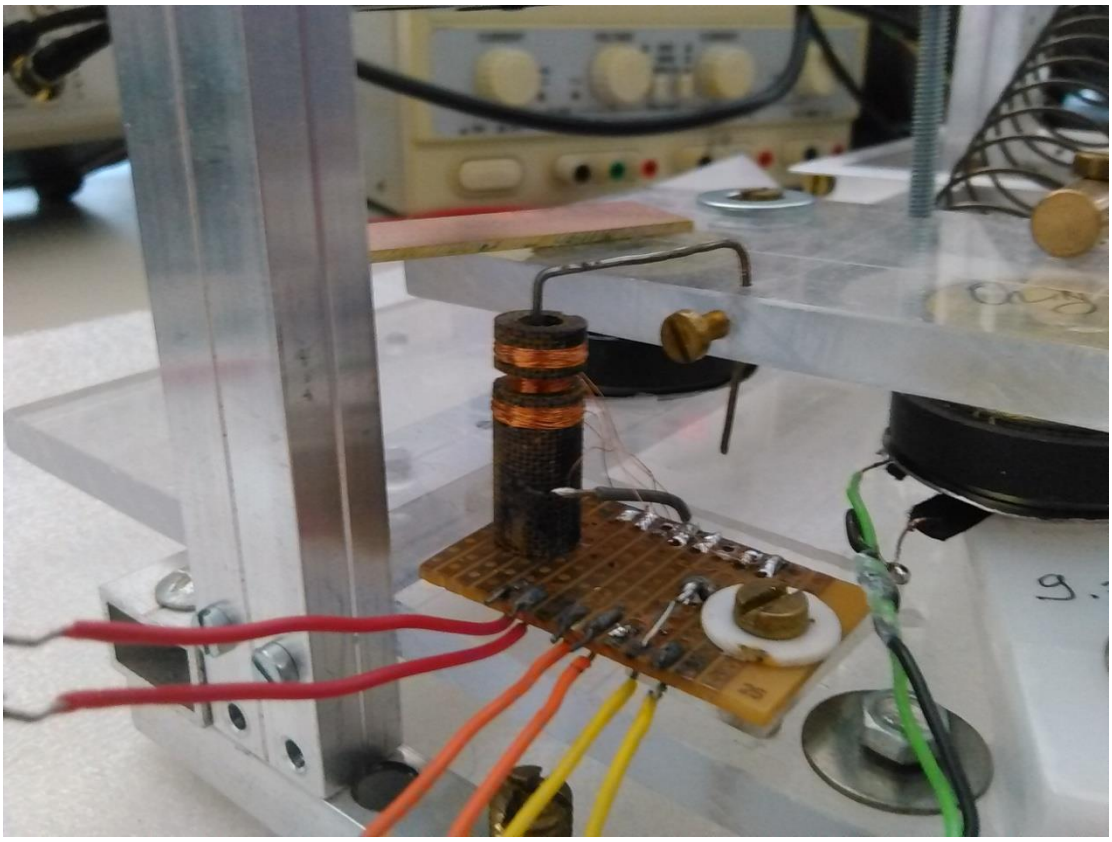


Figure II.0 *Photo du LVDT*

Il s'agit du premier capteur que nous avons implémenté. Il s'agit d'un capteur électrique passif (inductif). Le principe est d'entrainer une variation de flux magnétique grâce au déplacement. Il est composé de trois bobines (une primaire et deux secondaires) et d'une ferrite, qui se déplace dans ces bobines. On injecte un signal sinusoïdal dans le primaire, ce qui induit un signal sinusoïdal dans le secondaire. Après étalonnage (tension de sortie nulle au repos), le déplacement de la ferrite pourra se mesurer en fonction du changement observé sur la sortie du signal.

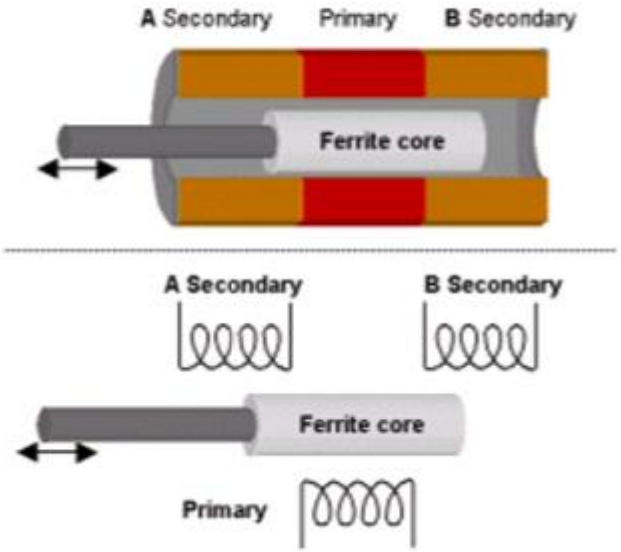


Figure II.1 *Schéma d'un capteur LVDT*

Nous avons refait à la main les trois bobinages nous-même suite à une rupture de fil et à un doute sur le primaire (le faire faire par un industriel revenant trop cher pour une opération aussi simple). Nous avons obtenu une impédance de 8 Ohms sur les secondaires, et de 11 Ohms sur le primaire.

II.1.2. Le capteur capacitif

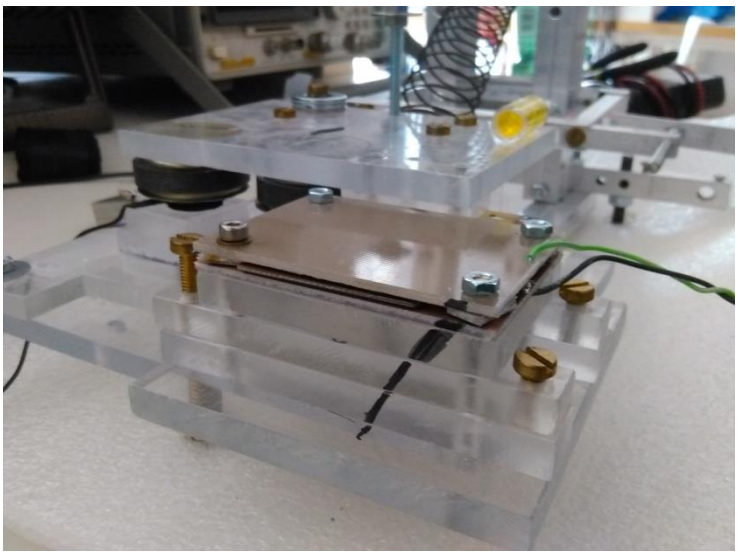


Figure II.2 *Photo du capteur capacitif*

Ce capteur a été réalisé par Robert Pillet, après que nous ayons étudié le LVDT. Il propose une solution alternative à la première, avec un principe de mesure différent. Il s'agit d'un capteur peu cher, mais difficile à réaliser, les trois plaques devant être parallèles entre elles à la position d'équilibre.

Ce type de capteur permet de mesurer des déplacements linéaires ou bien angulaires. Dans le cas du sismomètre nous avons besoin de pouvoir mesurer un déplacement linéaire, il nous faut donc utiliser un capteur capacitif plan. Avec un tel capteur, il est possible de mesurer des déplacements bien en dessous du millimètre. Le mouvement de l'armature mobile va faire évoluer le champ électrique ce qui modifie la valeur de la capacité, et par conséquent la tension de sortie. Nous pourrons alors en déduire la valeur du déplacement mesuré.

II.1.3. Le capteur optique

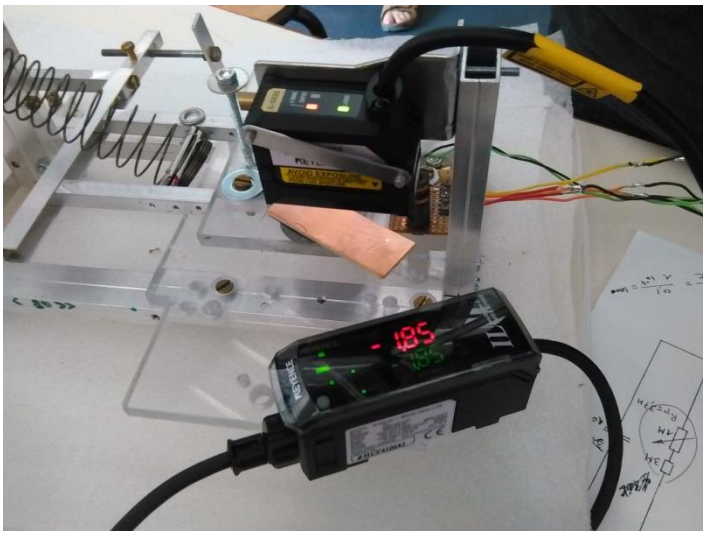


Figure II.3 *Photo du capteur optique*

Ce capteur nous a tout d'abord été prêté par la société Keyence, spécialisée dans l'électronique, avant que nous ne l'achetions pour l'étudier plus en détail. Il s'agit d'un capteur de déplacement, dont on peut expliquer le fonctionnement de la manière suivante :

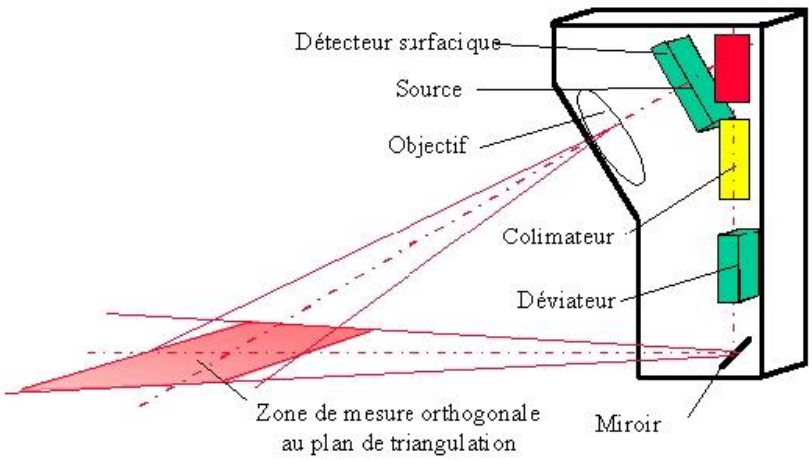


Figure II.4 *Schéma capteur optique*

La source émet de la lumière qui passe dans le collimateur et le déviateur avant d'être déviée par un miroir. Ce rayon réfléchit ensuite sur un obstacle, qui doit se trouver dans la zone de mesure, puis passe ensuite par l'objectif pour entrer dans le capteur. Le capteur mesure donc le temps mis par le laser entre son émission et sa réception pour calculer la distance le séparant de l'obstacle.

Ce capteur nous a principalement servi à faire différentes mesures de déplacement du sismomètre et ainsi effectuer des relevés de tension de sortie sur les deux autres capteurs. Nous avons également fait des mesures avec lui, et comme nous le verrons plus loin, il présente des résultats extrêmement propres, d'une grande qualité.

II.2. L'implémentation

II.2.1. Le LVDT

Il s'agit donc du premier sismomètre sur lequel nous avons travaillé :

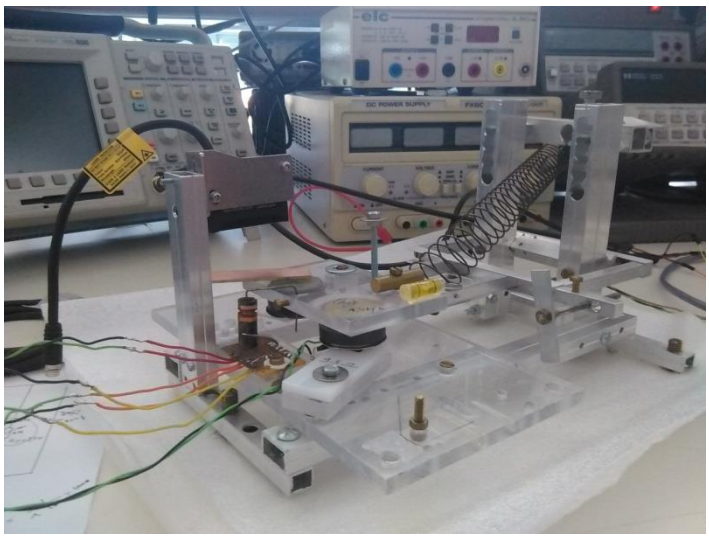


Figure II.5 *Sismomètre avec LVDT*

Nous avions à l'origine un circuit intégré permettant d'alimenter le primaire, de récupérer et analyser les sorties des secondaires et de nous fournir une sortie, mais n'ayant pas réussi à trouver de description du circuit ou de montage type pour la sismométrie, nous avons donc choisi une autre solution :

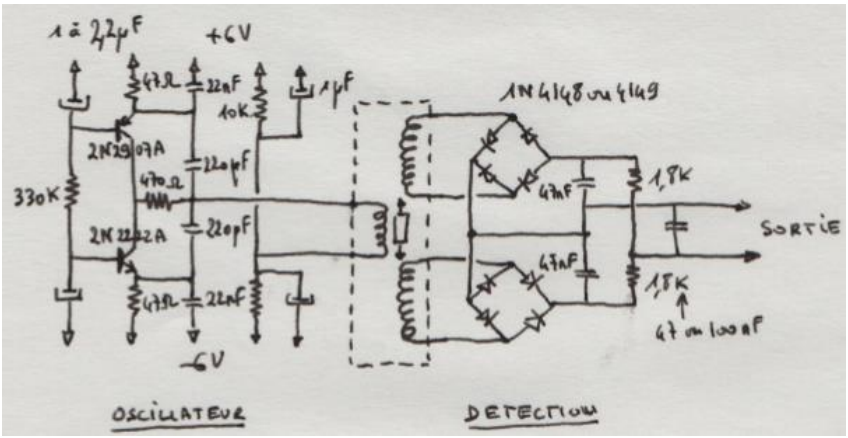


Figure II.6 *Montage du LVDT*

Ce montage a été réalisé par un collègue de Mr. Pillet. Il est composé de deux parties : une partie oscillateur, et une partie détection. La partie oscillateur nous permet de créer le signal sinusoïdal que l'on injecte dans le primaire. La partie détection, qui contient deux ponts de diodes (un pour chaque secondaire), permet de détecter la variation du champ magnétique, qui génère un courant induit, et donc une tension, que l'on récupère en différentiel sur la sortie.

Il existe aujourd'hui deux versions de ce montage : celle d'origine, que nous a fourni Mr. Pillet, et celle que nous avons réalisé nous-mêmes :

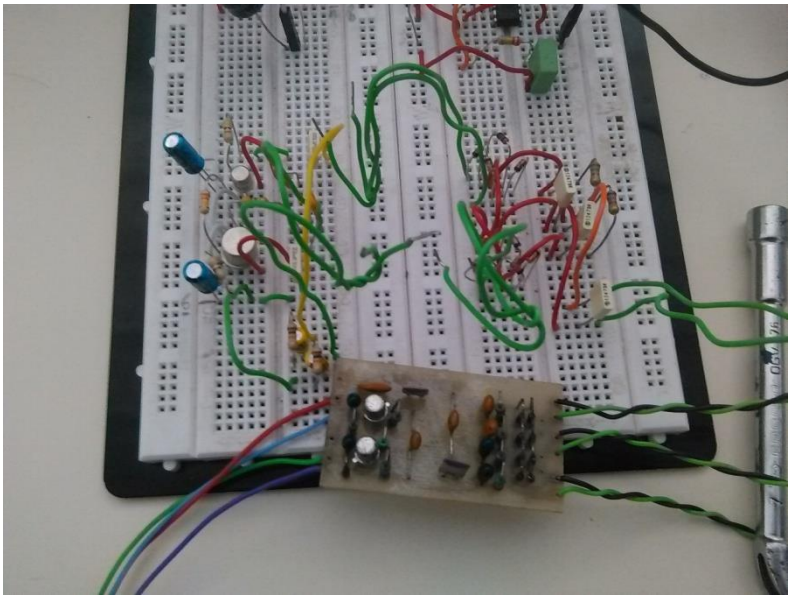


Figure II.7 *Carte et plaquette contenant le montage du LVDT*

Nous avons pu observer que la nôtre possède une fréquence d'oscillation plus élevée que celle d'origine, alors que les composants ont, aux erreurs de fabrication près, les mêmes valeurs. Suite à un problème lors d'un changement de plaquette de notre montage, nous avons continué notre projet avec la carte qui nous a été fournie (notre montage a été refait entre temps, mais nous n'avons pas eu le temps de le vérifier).

On obtient la caractéristique entrée/sortie du LVDT en modifiant la hauteur du plateau grâce à une vis, ce qui entraine une modification de la position de la ferrite du capteur. Nous pouvons alors visualiser la sortie sur un oscilloscope (en mode CC). Puis à l'aide d'un multimètre et du capteur optique, nous avons pu effectuer les relevés (tension de sortie en fonction du déplacement) suivants :

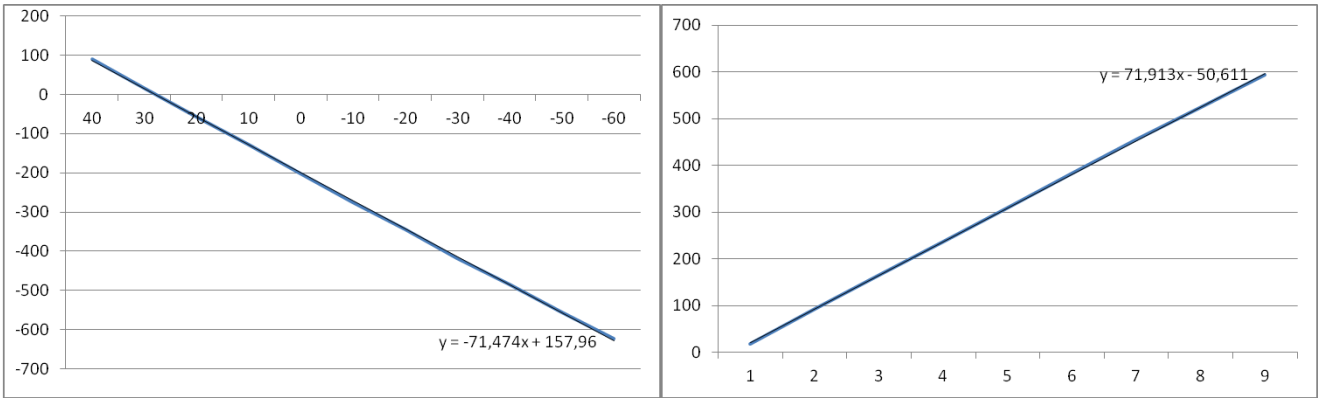


Figure II.8 *Tension de sortie (en mV) en fonction de la position (en centième de millimètre)*

Nous voyons que notre capteur est parfaitement linéaire dans le sens montant et dans le sens descendant, et possède la même pente dans les deux cas. Il nous suffira donc par la suite de mettre un gain de 100 à la sortie (amplificateur différentiel) pour travailler avec une fourchette de tension assez grande pour permettre une meilleure précision de la mesure.

II.2.2. Le capteur capacitif

Nous avons ensuite étudié le sismomètre équipé du capteur capacitif, et étudié ses performances.

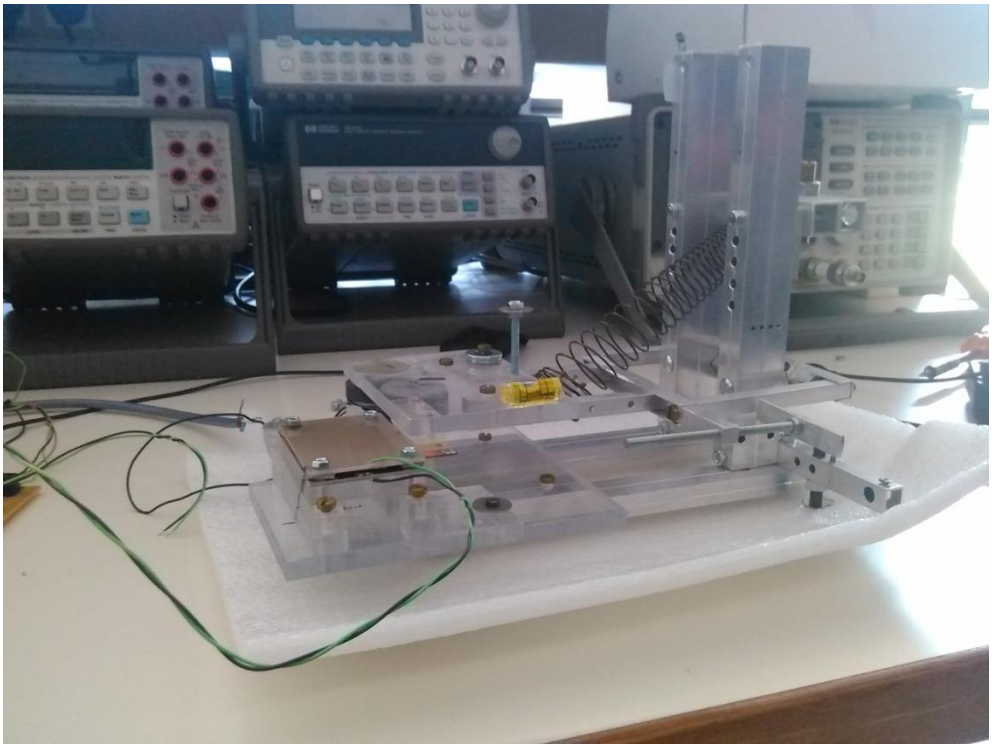


Figure II.9 *Sismomètre avec capteur capacitif*

Nous avons pour celui-ci pu utiliser un circuit intégré, le NE5521, que nous avons câblé à partir du montage ci-dessous :

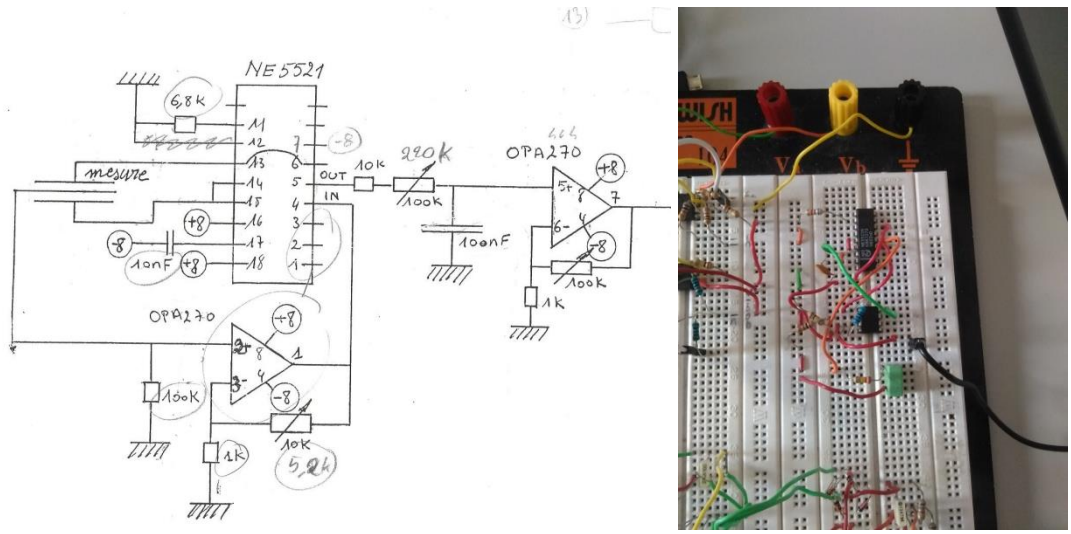


Figure II.10 *Montage du capteur capacitif*

Ce montage est composé de plusieurs parties. Le NE5521 permet de réaliser la partie oscillateur et la partie détection. Nous voyons également deux amplificateurs, permettant de générer des gains. Celui du bas permet d'amplifier la mesure, pour qu'elle soit exploitable dans le circuit intégré. Le deuxième (tout à droite), est situé en sortie du circuit, juste après un filtre passe-bas (pour ne garder que le continu). Il permet d'amplifier le signal filtré, pour améliorer la précision de la mesure de déplacement (une plus grande fourchette de tensions pour la même distance à parcourir).

Nous visualisons notre sortie sur un oscilloscope en mode CC pour trouver la plage de linéarité du capteur, comme fait précédemment avec le LVDT :

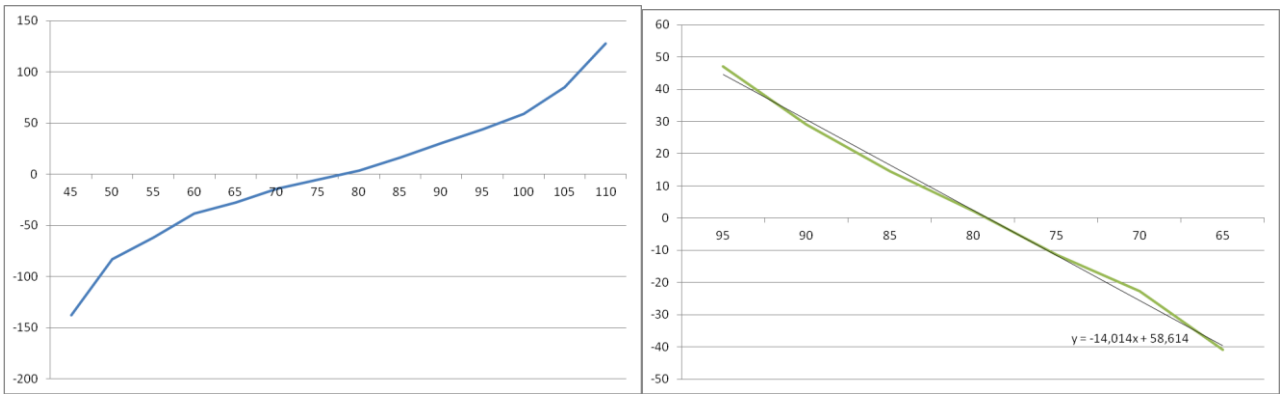


Figure II.11 *Tension de sortie (en mV) en fonction de la position (en centième de millimètre)*

Nous avons d'abord effectué la mesure sur une grande plage de déplacement (figure de gauche), en nous déplaçant tous les 0.05 millimètre. Nous pouvons alors observer trois zones sur ce graphe : une à gauche, une au centre et une à droite. Seule celle du centre est linéaire, c'est donc à cette plage que nous allons nous intéresser. Nous relevons ses bornes : entre 0.65 et 0.95 millimètre. Nous avons donc fait une nouvelle mesure (figure de droite), entre 0.65 et 0.95 mm, pour vérifier cette linéarité. Nous sommes donc centrés autour de 0, sur une plage de plus ou moins 0.15 millimètre, ce qui nous suffit pour étudier le bruit de fond sismique.

II.2.3. Capteur optique et étude comparative

Etudions le comportement du capteur optique, alimenté en 0-5V :

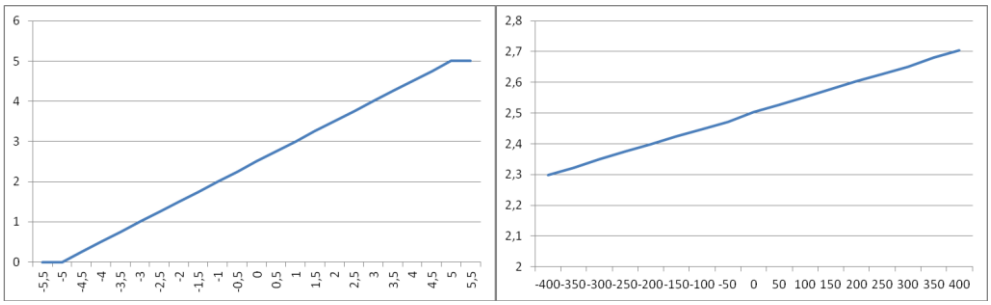


Figure II.12 *Tension de sortie (en V) en fonction de la position (en millimètre)*

L'évolution de la tension sur toute la plage de mesure, avec 2.5 V de mesure à la position 0, et 0 et 5 V aux positions extrêmes (+/- 5 mm), et saturation ensuite. Nous remarquons tout de suite la différence avec les deux autres capteurs, tant au niveau résultat que facilité de mise en œuvre. Nous pouvons également travailler en +/- 5 V avec ce capteur.

En revanche, nous ne pouvons retenir cette solution pour le sismomètre du projet, à cause de son prix, excédant de très loin les 200 euros prévus pour la totalité du sismomètre.

Nous avons alors comparé nos deux capteurs restants (capacitif et LVDT), pour choisir sur lequel nous implémenterions en premier la contre-réaction, et que nous conserverions par la suite comme solution finale (l'autre solution offrant une solution alternative).

Regardons d'abord du côté de la plage de mesure. Pour le capteur capacitif, nous avons une plage de mesure très faible, de l'ordre de 0.7 millimètre, à cause de la proximité des plaques. Pour le LVDT, la plage de mesure dépend de la taille des bobines et de la ferrite. Avec le nôtre, nous avons une plage de quelques millimètres, il est donc plus intéressant de ce point de vue, car permet l'étude de séismes de plus grande amplitude.

Côté linéarité, nous avons donc vu que le LVDT est linéaire sur une grande plage de mesure, contrairement au capteur capacitif, linéaire sur approximativement 0.3 millimètre. Nous préférerons donc utiliser pour ce critère le capteur inductif.

Côté prix, il s'agit de deux capteurs faits à la main, à partir de matériaux peu chers, nous pouvons donc considérer que les deux sont équivalents à ce niveau-là, et répondent au cahier des charges.

Nous avons donc retenu le LVDT, car il est meilleur au niveau performance (linéarité, plage de mesure), présente un coût de fabrication faible (si fait à la main et non par un industriel), et est très bien niveau pédagogique, car très visuel.

II.3. La contre-réaction

Le but de la boucle de rétroaction est de réduire le phénomène d'oscillation, on parle dans ce cas de rétroaction constamment stabilisée dans son mouvement afin de ne pas obtenir sur le sismogramme les différents effets d'oscillation de celle-ci.

Pour notre étude nous avons pris la décision d'effectuer une rétroaction analogique que nous avons implémentée sur le sismomètre équipé du capteur LVDT (celui retenu dans la partie précédente). Une rétroaction analogique nous permet de limiter les coûts (composants électroniques peu chers), et est plus intéressante d'un point de vue pédagogique qu'une rétroaction numérique.

II.3.1. Principe

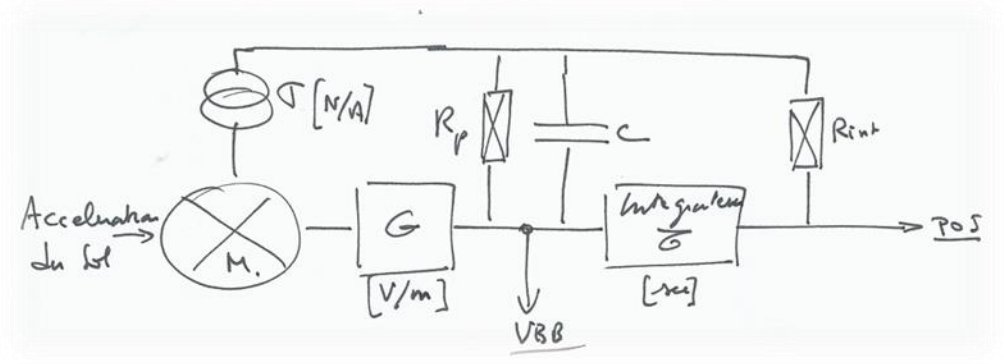


Figure II.13 *Principe de la rétroaction*

Pour réaliser la rétroaction, nous avons pris le parti d'utiliser un correcteur PID. Le correcteur PID est un mélange de trois correcteurs : proportionnel, intégral, et dérivé, réalisant des fonctions différentes. Le correcteur proportionnel permet d'augmenter le gain et tend à augmenter la rapidité du système (attention aux problèmes de convergence si gain trop grand). Le correcteur intégral amplifie les basses fréquences par rapport aux hautes fréquences, améliore la précision du système (peut entrainer un problème de stabilité). Le correcteur dérivé a tendance à réduire les perturbations du système (permet de définir les marges de stabilité).

S'il est bien réglé le PID cumule les avantages de chacune des actions proportionnelles, intégrales et dérivées. Toutefois s'il est mal réglé il peut fortement dégrader le comportement du système. L'action intégrale doit être placée à basse fréquence, bien avant la fréquence de coupure, son but est d'augmenter le gain en basse fréquence. L'action dérivée est placé où l'on mesure les marges de stabilité, le but est d'apporter de la phase pour augmenter la marge de phase. L'action proportionnelle permet de finaliser le correcteur et de fixer la rapidité du système.

Une action mécanique sur le sismomètre est alors possible par le biais d'un forceur, ici un couple aimant-bobine. Nous injectons un courant dans la bobine, calculé par le correcteur PID, ce qui engendre une modification du champ magnétique. Comme cette bobine est plongée dans un aimant, celui-ci réagit à la variation de champ électromagnétique.

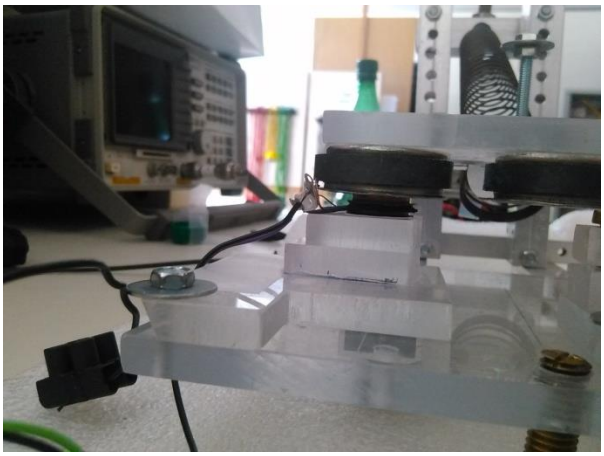


Figure II.14 *Forceur*

II.3.2. Modélisation

Nous avons décidé d'utiliser le schéma électrique suivant pour réaliser le correcteur PID :

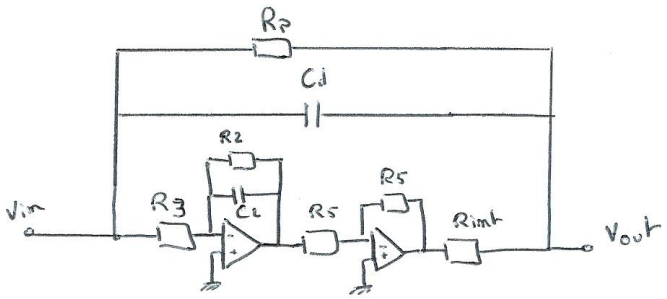


Figure II.15 *Schéma électrique du correcteur PID*

Nous avons mis en équation ce correcteur (voir Annexe).

Dans le but de simuler l'intégralité du sismomètre (partie mécanique + analogique), nous avons aussi besoin de modéliser la partie mécanique (système ressort + masse qui peut être représenté par un circuit RLC passe-bande) (voir Annexe). Nous avons implémenté les équations de chaque sous-partie avec le logiciel MATLAB (voir codes en Annexe). Par manque de temps, nous n'avons pu étudier les résultats.

Nous avons également implémenté chaque sous-partie sous CADENCE, ce qui pourrait nous permettre de simuler les résultats attendus en laboratoire.

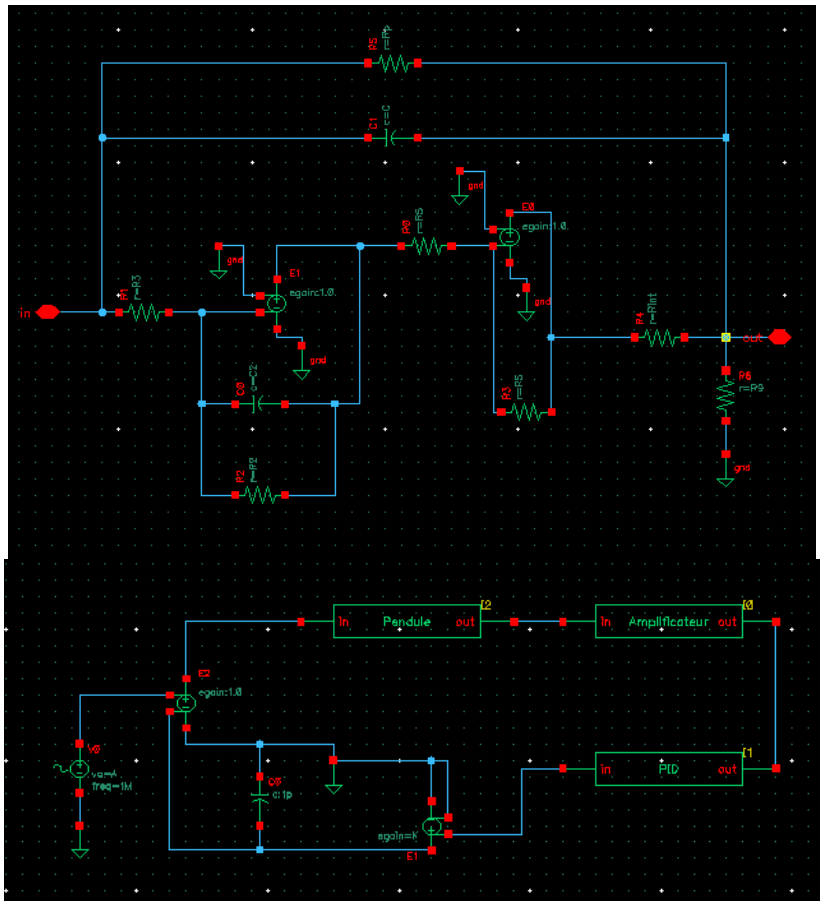


Figure II.16 *Implémentation sous CADENCE*

Chacune de nos sous-parties fonctionnent indépendamment, il reste donc à les relier entre elles et les faire fonctionner.

II.3.3. Implémentation sur le sismomètre avec LVDT

Pour la rétroaction du sismomètre équipé du LVDT, nous avons monté le modèle suivant :

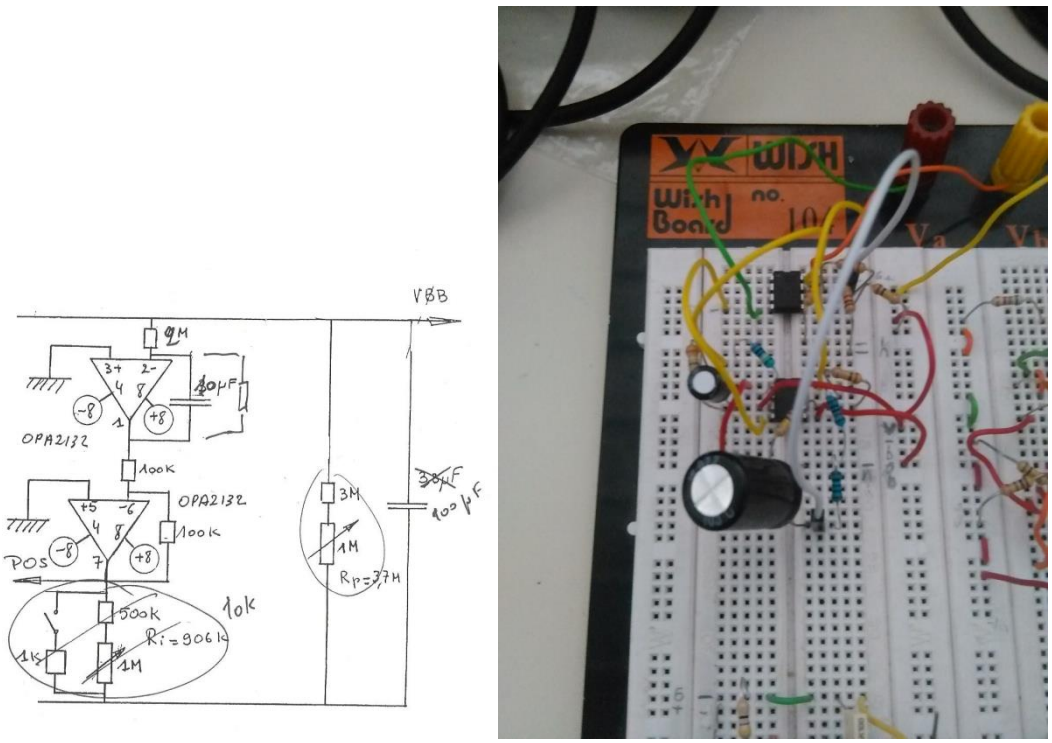


Figure II.17 *Schéma et plaquette de la contre-réaction du LVDT*

Par rapport au schéma présenté dans la sous-partie « Principe », nous a donc pris les caractéristiques suivantes pour le correcteur PID :

- Taux d'intégrateur Tint = 20 secondes
- Rint = 10 kOhms
- Rp = 3,5 MOhms
- C = 100 µF

Un taux d'intégration de 20 secondes est amplement suffisant pour une première approche. Il correspond au temps que le sismomètre est censé mettre pour retrouver sa position d'équilibre. La valeur de C nous permet de placer la valeur du plateau de la réponse fréquentielle. De plus, nous avons la relation suivante :

$$T_{CRE} = 2 * PI * \sqrt{T_{INT} * R_P * C} \qquad \text{(II.1)}$$

Nous voulons un Tcre de 20 secondes, il nous faut donc une valeur de Rp autour de 3,5 MOhms.

En entrée du correcteur, nous retrouvons l'amplificateur différentiel que nous avions mis à la suite de notre LVDT et nous envoyons la tension Vbb vers le convertisseur, correspondant à la position de la masse (mécanique). Nous récupérons en sortie de l'intégrateur la tension POS, image du courant à envoyer dans la bobine pour retrouver la position d'équilibre, et en sortie du correcteur, nous plaçons en série notre bobine (qui joue le rôle de forceur), reliée ensuite à la masse (électrique).

On détermine le gain du forceur à l'aide de la manipulation suivante. Nous mettons à la position d'équilibre la masse du sismomètre, puis nous simulons une perturbation, en ajoutant une masselotte de quelques grammes. Nous déterminons alors le courant nécessaire à injecter dans la bobine pour retrouver

la position d'équilibre, à l'aide d'une résistance variable. Dans le cas du sismomètre LVDT, nous obtenons un gain de forceur de 1,2N/A.

Nous avons finalement pu passer à la simulation, et obtenir le résultat suivant :

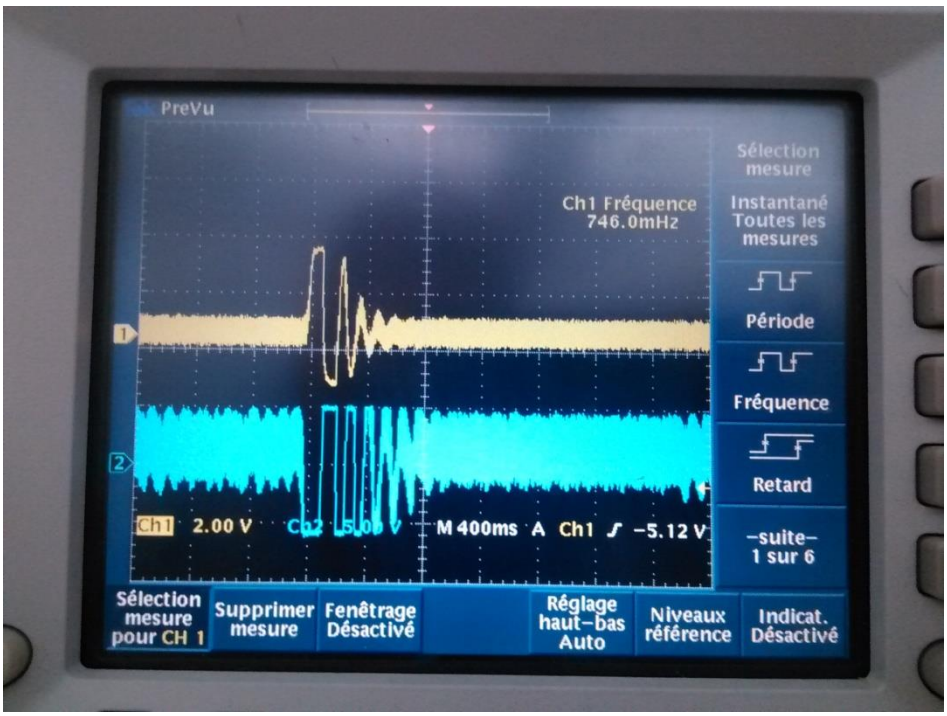


Figure II.18 *Simulation finale*

Le signal jaune correspond à la position du plateau, et le signal bleu à la tension image du courant à injecter dans la bobine (forceur). Cette réponse est celle d'un coup de poing donné sur la table à une trentaine de centimètres du sismomètre. Nous remarquons que le sismomètre retrouve sa position d'équilibre très rapidement. En revanche, un bruit d'une importance non négligeable vient parasiter la mesure, et nous empêche de mesurer une secousse plus faible (comme un coup de pied sur le sol). On peut voir également que la courbe bleue est en opposition de phase avec la courbe jaune. On retrouve la présence du bruit, avec une variation importante et rapide de la tension (ce qui induit le bruit en position). La réaction au moment de l'impact est bonne, puisque le courant injecté est négatif lorsque le plateau monte, dans le but de le faire redescendre.

Il reste maintenant à enregistrer ces données et les afficher sur un écran (autre que celui de l'oscilloscope), ce qui est l'objet de la partie suivante.

Chapitre III : Traitement des données et interaction avec l'utilisateur

Le cahier des charges mentionne l'utilisation d'une carte Raspberry Pi B+ pour le traitement des données numériques, nous devons donc étudier et comprendre toutes les potentialités de celle-ci afin de pouvoir l'exploiter au mieux, et choisir en conséquence les outils compatibles et performants adaptés à notre projet.

Nous allons donc dans un premier temps explorer les différents moyens de communications de la carte, avant de confirmer le modèle de convertisseur analogique-numérique qui transmettra les données en entrée. Nous développerons ensuite le programme de traitement des informations, puis nous terminerons par la conception d'une interface utilisateur pour la visualisation des données.

III.1. Support de traitement numérique : Raspberry Pi B+

La carte Raspberry Pi est livrée nue, sans aucune connectique ni périphérique. Nous lui connectons donc tout d'abord le nécessaire pour son démarrage et sa configuration : un adaptateur secteur micro-USB, un clavier et une souris en USB, un écran sur son port HDMI, et une clé wifi USB pour lui donner un accès à internet.

Nous rédigeons un protocole d'installation de l'OS Rasbian Wheezy et de configuration de la Raspberry Pi (document en Annexe) afin de regrouper les informations utiles au projet et donner la possibilité au client ou à d'éventuels collègues poursuivant notre travail de pouvoir refaire l'installation sur une autre carte.

Nous notons que celle-ci, équipée d'un processeur Broadcom BCM 2835 ARM11 cadencé à 700MHz et d'une mémoire RAM de 512Mo, peut parfois manquer de ressources pour l'exécution de plusieurs commandes en multitâche. Nous choisissons donc de travailler au développement du projet sur les ordinateurs portables prêtés par l'Ecole Polytech, puis d'opérer le transfert des données sur carte lorsque cela est nécessaire.

Nous configurons un accès SSH entre nos deux machines sur réseau domestique grâce à l'émulateur de terminal PuTTY. Cela nous permet de commander la carte à distance depuis notre ordinateur personnel. Nous trouvons également la possibilité d'associer à PuTTY l'émulateur d'interface graphique Xming. Nous avons alors non seulement accès à la carte par le biais d'une connexion sécurisée, mais nous pouvons également avoir un retour graphique de nos opérations. Cette installation nous permet notamment de retirer la dépendance de la carte à ses périphériques (écran, clavier, souris), et la rendre ainsi beaucoup plus « portable ».

Nous explorons dans un premier temps une des spécificités des micro-contrôleurs/micro-ordinateurs : leur port GPIO. En effet ce port aux 40 broches permet la communication avec le monde extérieur par l'émission et la réception de signaux numériques. Nous nous attachons donc à nous familiariser avec celui-ci car il sera utilisé par la suite comme interface d'échange avec le convertisseur Analogique/Numérique, le module RTC (Real Time Clock), les LEDs et boutons poussoirs. Sa configuration est programmable en Python grâce à la librairie intégrée par défaut dans l'OS : RPi.GPIO.

Le travail de préparation de la carte est terminé et mis à l'écrit sous forme de fiche protocolaire, nous allons maintenant pouvoir explorer les solutions adaptées à notre projet pour l'acquisition et le traitement de l'information en provenance du capteur.

III.2. Réception des données au format numérique

III.2.1. Convertisseur Analogique Numérique

Notre objectif d'obtenir un sismomètre de qualité professionnelle et la configuration de notre chaine d'acquisition nous impose certaines exigences techniques quant au choix du convertisseur Analogique Numérique :

- Convertisseur de type Sigma Delta
- Résolution minimale de 16 bits
- 2 Lignes d'acquisitions parallèles
- Taux d'échantillonnage de 128 éch/s
- Transmission des données par protocole I2C ou SPI

Le convertisseur que nous avons sélectionné possède toutes les caractéristiques énoncées précédemment, et plus encore, pour un coût moindre que celui estimé au début de notre projet.

C'est un convertisseur à 8 chaines, compatible avec la carte Raspberry Pi en se connectant au port GPIO de celle-ci. Le circuit repose sur la puce MCP3424 de Microchip Technology Inc. opérant la conversion Sigma-Delta avec peu de bruit et une haute précision. On observe l'intégration de deux de ces puces pour traiter la conversion des 8 chaines en entrée, ainsi qu'une interface buffer I2C permettant la connexion supplémentaire d'autres périphériques I2C sur le bus du circuit.

La puce MCP3424 contient une tension de référence interne de 2,048V avec une variation de tension en entrée de -2,048V à +2,048V (soit une amplitude maximale de 4,096V). Son taux d'échantillonnage est programmable en choisissant comme options : 240, 60, 15 ou 3,75 échantillons par seconde. Avant conversion, le circuit dispose d'un amplificateur à gain programmable que nous pouvons configurer à x1, x2, x4 ou x8.

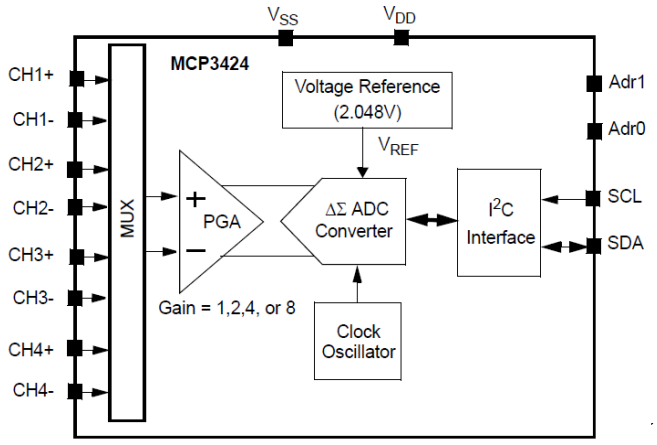


Figure III.1 *Diagramme bloc fonctionnel de la puce MP3424*

Enfin, nous avons la possibilité de choisir les adresses de liaison I2C avec la Raspberry Pi : une adresse porte 4 chaines, le convertisseur doit donc indiquer les 2 adresses qu'il réserve pour le protocole de communication. Notre choix d'adresse s'étendant entre 0x68 et 0x6F, nous choisirons simplement les deux premières adresses (0x68 et 0x69).

Après avoir installé sur la Raspberry Pi les paquets nécessaires à la bonne communication avec le convertisseur puis paramétré le système d'exploitation pour reconnaitre ce dernier dès le démarrage (détail des procédures en Annexe), nous testons la bonne détection du périphérique depuis le terminal.

La commande i2cdetect nous renvoie le tableau des périphériques connectés au bus I2C, et nous observons sur la figure ci-dessous que les adresses 0x68 et 0x69 du convertisseur sont bien correctement connectées. Le protocole de communication Convertisseur ↔ Raspberry Pi est donc validé.

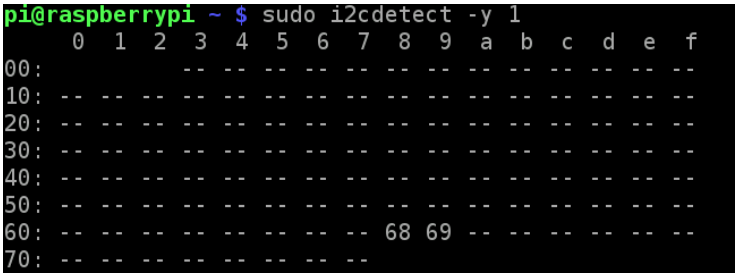


Figure III.2 *Le convertisseur Analogique Numérique a bien été détecté sur le bus I2C*

Nous testons ensuite le fonctionnement du convertisseur en injectant une tension continue de 1,5V en entrée du convertisseur et en vérifiant la valeur de sortie.

Pour cela, nous devons télécharger les librairies Python compatibles Python 2.7 depuis le Github de ABElectronics UK, et ne conserver que le répertoire DeltaSigmaPi (que nous placerons dans le dossier personnel), seul pertinent pour notre utilisation.

Nous utilisons les commandes suivantes :

```
wget https://github.com/abelectronicsuk/ABElectronics_Python_Libraries/archive/master.zip

unzip master.zip -d master

cp -R master/ABElectronics_Python_Libraries-master/DeltaSigmaPi DeltaSigmaPi

rm -r master master.zip
```

Nous rédigeons ensuite un programme Python de test du convertisseur, en incluant le chemin d'accès aux librairies précédemment téléchargées :

```
import sys

sys.path.insert(0, "./DeltaSigmaPi")

from ABE_DeltaSigmaPi import DeltaSigma

from ABE_helpers import ABEHelpers

i2c_helper = ABEHelpers()

bus = i2c_helper.get_smbus()

adc = DeltaSigma(bus, 0x68, 0x69, 18) # 4e argument : Taux d'échantillonnage (12, 14, 16, ou 18 bits)
```

```
print(adc.read_voltage(1))
```

L'interpréteur va ici successivement importer les librairies utiles, activer le bus I2C, indiquer l'adresse de notre module (le convertisseur est accessible depuis les adresses 0x68 et 0x69 du bus), puis mesurer et afficher la tension de la ligne 1. Le résultat obtenu est le suivant :

```
>> print(adc.read_voltage(1))

1.47569
```

Sur une suite d'essais, les résultats observés sont concluant, nous confirmons donc la compatibilité et la méthode d'implémentation du convertisseur sur la carte Raspberry Pi.

III.2.2. Module Real Time Clock

La Raspberry Pi a été conçue pour en réduire au maximum la taille et le coût, il a donc été choisi par le fabricant de ne pas implémenter de RTC et d'utiliser la connexion internet à chaque démarrage pour mettre à jour la date et l'heure. Pour ce faire, elle doit se connecter aux serveurs NTP (*Network Time Protocole*) qui disposent d'une grande précision grâce à une synchronisation avec l'horloge atomique. En l'absence de connexion, l'utilisateur doit manuellement mettre à jour le système.

Pour pallier à ce problème, et rendre notre produit indépendant de toute connexion internet, nous devons donc implanter un module RTC au système.

Nous réalisons une étude comparative (placée en Annexe), permettant de confirmer le choix du module haute précision DS3231, que nous placerons sur le port GPIO de la carte Raspberry Pi, afin d'apporter au signal enregistré une information de temps précise, capitale en sismologie.

III.3. Traitement les données avec le micro-ordinateur

Avant de travailler avec des signaux provenant du capteur sismométrique, nous allons simuler ces signaux de manière à travailler de manière indépendante de la partie mécanique et analogique. Pour cela, nous allons utiliser le micro de notre ordinateur comme « capteur sismique » grâce aux librairies PyAudio et Wave.

Sur ce principe, nous allons pouvoir développer et tester notre programme d'acquisition jusqu'à validation de son fonctionnement, après quoi il suffira de remplacer le signal du microphone par le signal donné par le convertisseur.

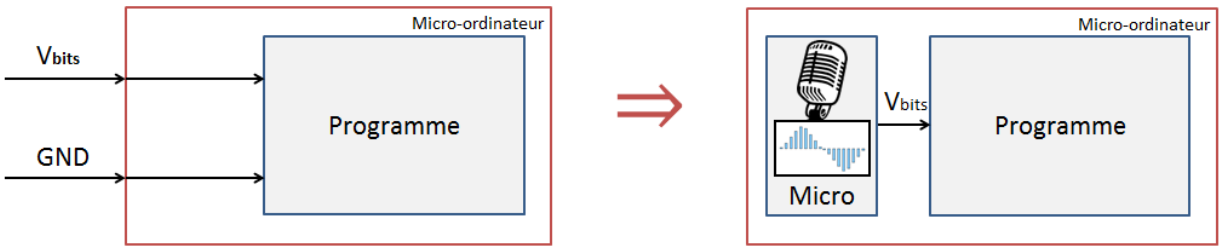


Figure III.3 *Développement indépendant du programme d'acquisition grâce à l'utilisation du microphone remplaçant le signal sismique numérique*

Objectifs : Le programme que nous allons réaliser (en Python 2.7 pour des raisons de compatibilité des librairies) doit être en mesure de recevoir 2 signaux numériques de manière permanente, les séquencer à fréquence déterminée, de leur attribuer les paramètres normalisés des courbes sismiques, les enregistrer sous format image et MiniSEED, apporter à l'utilisateur une indication sur l'état de fonctionnement du programme, et permettre l'interruption de celui-ci de manière sécurisée.

Nous suivrons les étapes de développement suivantes :

1) Enregistrer et afficher le signal

2) Attribuer les paramètres adaptés à la courbe sismologique et encoder au format MSEED

3) Ring Buffer : Enregistrer en continu

4) Gestion des interruptions

III.3.1. Enregistrement et affichage d'un signal

Comme précisé précédemment, le signal en entrée du programme provient ici du micro. La première étape de développement consiste donc, après avoir installé les librairies utiles (PyAudio et Wave), à capturer le flux d'information ou « stream ».

Pour cela, il existe 2 méthodes possibles : par copie de buffer, ou par copie valeur par valeur. Dans notre cas de traitement en temps réel, nous utiliserons la première solution, la deuxième étant plutôt destinée à la prise de mesures ponctuelles espacées dans le temps.

Le buffer du micro sera paramétré de la manière suivante : buffer en lecture seule sur la ligne 1, dont la capacité sera de 125 valeurs entières de 16 bits chacune.

```
# PyAudio : Open audio stream
p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paInt16, channels=1, input=True, frames_per_buffer=125)
stream.get_read_available()
```

Nous établissons l'enregistrement sur une trame de 5 secondes (625 valeurs) grâce à l'outil de mesure du temps proposé par Python (*time.time.()*). Chaque lecture du stream vient agrandir une variable *data* (tableau de la librairie NumPy), qui sera finalement affichée (grâce à la librairie MatplotLib).

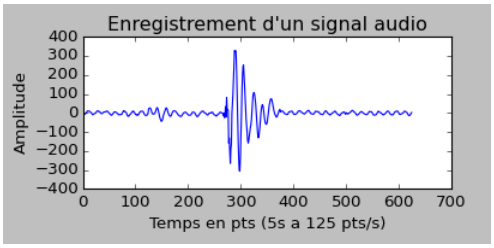


Figure III.4 *Enregistrement d'un signal sur 5s à 125 éch/s*

Ce programme sera la base de notre travail, sur lequel nous allons ajouter progressivement des fonctionnalités.

Par exemple, le premier ajout que nous opérons et l'activation d'une LED d'état indiquant à l'utilisateur que l'enregistrement des données est en cours. Cette LED commandée par le GPIO 19 s'allume au début de la boucle d'enregistrement et s'éteint à la fin. Une méthode simple mais qui apporte une indication visuelle à l'utilisateur.

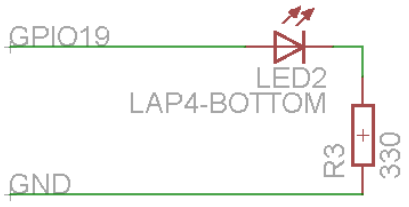


Figure III.5 *Connexion de la LED d'état au port GPIO*

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(19, GPIO.OUT)
GPIO.output(19, GPIO.LOW) #LED éteinte au début
[...]
while temps<5 :
        #LED allumée pendant l'enregistrement
        GPIO.output(19, GPIO.HIGH)
        [...]
GPIO.output(19, GPIO.LOW) #LED éteinte à la fin
[...]
```

III.3.2. Paramétrage des courbes et encodage au format MSEED

Nous allons considérer le signal capté comme un signal sismique qu'il nous faut annoter, afin de répondre aux exigences des sismogrammes numériques. Nous allons pour cela exploiter la librairie ObsPy, qui établit un lien entre les bibliothèques de traitement numérique du signal et le monde sismologique. Elle fournit notamment des analyseurs et des constructeurs pour les formats de fichiers courants et permet la manipulation de séries chronologiques sismologiques. Nous noterons que ce framework est davantage développé pour la lecture et l'analyse d'informations distantes (sismogrammes sur serveur distant par exemple) que pour la génération de sismogrammes. La raison en est sûrement que le langage Python étant un langage interprété, les opérations de gestion dynamique de ressources et de traitement en temps réel de données sont plus complexes à mettre en place. Des processus temps-réel sur un système au noyau adapté seraient sans-doute plus favorables pour un sismomètre professionnel.

Nous adaptons notre programme en intégrant le signal reçu (*data* : tableau de données) et ses métadonnées (*stats* : informations sur le signal) dans un même objet « Trace ». Nous renseignons le nom de la station, sa location, la chaine utilisée, l'heure de début, la période d'échantillonnage et le nombre de points du signal enregistré.

```
#Infos about Obspy Trace
stats=Stats()
stats.station='RASPI'
stats.channel='1'
stats.delta= 0.008      #Intervalle entre 2 échantillons à 125 ech/s
stats.npts=len(data)    #Longueur du signal (en nb d'échantillons)
#Calcul de l'heure de début d'enregistrement à partir de l'heure UTC+2 à laquelle on soustrait la durée de mesure
stats.starttime=UTCDateTime()+2*3600-stats.delta*stats.npts
```

En combinant dans un objet « Trace » le signal et les métadonnées, le tout encapsulé dans une structure « Stream » (pouvant contenir plusieurs objets « Trace ») nous donnons la possibilité au framework de construire un fichier de sismogramme numérique au format désiré.

Nous enregistrons donc l'objet « Stream » au format MiniSEED, en exécutant la commande suivante dans notre programme Python :

```
tr=Trace(data,stats)
st=Stream(tr)
st.write("sismogramme.mseed", format='MSEED', encoding=1, verbose=1)
```

Nous passerons les détails sur les configurations choisies pour l'encodage, et affichons le résultat suivant :

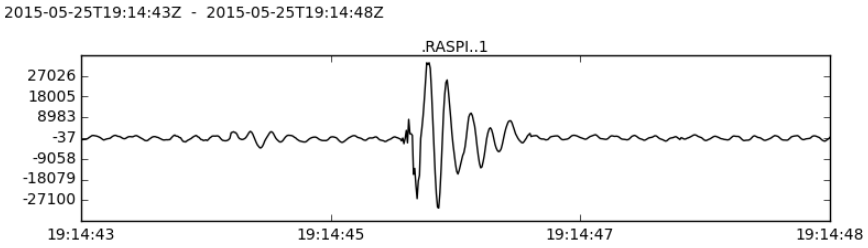


Figure III.6 *Résultat d'enregistrement au format MSEED d'un signal et ses métadonnées*

A l'ouverture de ce fichier MSEED, nous pouvons observer la durée effective de 5 secondes d'enregistrement, captée par la station RASPI sur la chaine 1.

III.3.3. Ring Buffer : Enregistrement continu du signal

Nous souhaitons pouvoir enregistrer en continu le signal sismique sans avoir à ouvrir et fermer un buffer à chaque prise de mesure, pour cela, nous optons pour la solution d'un buffer circulaire. Il permet l'enregistrement de données en continu sur un espace mémoire limité. Une fois l'enregistrement ayant rempli l'espace total alloué, les données les plus anciennes sont alors remplacées par les plus récentes.

Dans l'idéal, nous pouvons considérer un buffer circulaire de 24h, dont les données sur 18 bits sont mesurées à une fréquence de 125Hz, et qui nécessiterait ainsi l'allocation mémoire de 48,6Mo sur la carte microSD.

Nous implémentons la boucle infinie avec un « *while True :*» dont seule une instruction « *break* » pourrait en faire sortir le programme.

III.3.4. Gestion des interruptions

Alors que l'affichage d'un enregistrement de 24h (type One-Day Plot) peut présenter un intérêt certain, nous pensons qu'il est également pertinent d'établir un séquencement du buffer circulaire en enregistrant à intervalle de temps régulier la dernière trame captée. Ainsi, il sera plus simple de visualiser un évènement en affichant uniquement la trame concernée.

Nous choisissons donc de générer des interruptions toutes les 30 secondes pour opérer l'enregistrement au format MiniSEED du signal pendant cet intervalle de temps. Cela ne vient en aucun cas déstabiliser la lecture du flux de donnée, ni l'enregistrement sur le buffer circulaire.

Il est également envisagé que l'utilisateur choisisse délibérément de stopper l'enregistrement. A ce stade, la seule option existante pour cela est un *KeyboardInterrupt* (Ctrl C) dans la console. Cette méthode étant trop brutale pour le programme, nous allons proposer l'interruption manuelle par un bouton poussoir relié au GPIO 20 de la carte Raspberry.

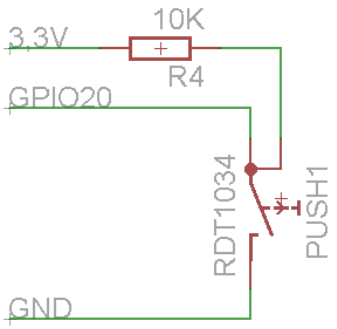


Figure III.7 *Connexion du bouton poussoir d'interruption au port GPIO*

Nous captons cette interruption dans la boucle de notre programme grâce aux instructions « *try* ... *catch* », et enregistrons l'ensemble du buffer avant de clore le programme sans qu'aucune erreur ne s'affiche. Ainsi, aucune donnée n'est perdue, l'utilisateur pourra bien accéder aux dernières informations enregistrées.

III.4. Proposition d'une interface utilisateur claire

Le cahier des charges stipule que le public auquel s'adresse le produit est en enseignement secondaire, élèves ou professeurs. Il a un but pédagogique unique : développer des connaissances en sismologie grâce à l'expérimentation. Il est donc important que les utilisateurs soient guidés de manière la plus simple possible vers les éléments qu'ils recherchent, à savoir l'étude des sismogrammes.

L'objectif est donc ici de proposer une interface intuitive qui puisse se substituer à l'utilisation de la console, et réduire les manipulations informatiques au minimum.

III.4.1. Solution ergonomique

Problème soulevé : Quel type d'interface utiliser pour remplacer la console ?

Une solution possible lorsque l'on parle d'interface est le développement d'une interface graphique sous forme de fenêtre contenant entre autres des menus, des éléments graphiques structurés, des contraintes, et un gestionnaire d'évènements pour interagir avec l'utilisateur. Tout cela est programmable à l'aide de *toolkits*, existant pour Python comme pour d'autres langages de programmation. L'application se présente finalement sous la forme d'un fichier exécutable grâce à des outils adaptés.

Dans notre cas nous nous sommes intéressés aux toolkits **Tkinter** et **wxPython**. Ces outils de conception d'interface graphique contiennent les fonctions Python nécessaire au développement de fenêtres structurées. Leur avantage est leur parfaite compatibilité avec notre programme d'enregistrement des

données précédemment développé, car tous deux écrits en Python. Le point faible est le temps de conception : tout est à faire, de l'ouverture des menus jusqu'à la réaction de la fenêtre lorsque l'utilisateur souhaite la redimensionner. De plus, nous notons que lorsqu'un gestionnaire d'évènement est en cours d'exécution (pour exécuter notre programme d'enregistrement par exemple), la boucle d'évènement du toolkit est interrompue : l'interface graphique est complètement bloquée, les boutons, les menus et la fenêtre ne répondent pas. Cela pose donc un sérieux problème, qui pourrait être corrigé par la gestion des threats (actions parallèles), mais là encore, nous nous heurtons à un problème de temps de développement considérable.

L'autre solution envisageable est l'utilisation d'un serveur web interne, dont l'interface graphique est déjà préconfigurée par le navigateur. Le point faible repose sur l'obligation de maitriser les langages de développement web. Cependant nous notons comme points positifs que le développement du contenu d'une application web est moins long que la solution précédente, qu'il existe une multitude de méthodes d'implémentations dont l'une d'elle devrait très certainement convenir à notre projet, et enfin que le public est aujourd'hui tout à fait familier avec ce genre d'interface.

Nous adoptons donc en conséquence cette solution technologique. Nous devons maintenant nous attarder sur la méthode à utiliser pour son développement.

III.4.2. Compatibilité des processus

Problème soulevé : Quel type de développement web utiliser pour une compatibilité avec notre programme d'acquisition des données ?

La majorité des applications web s'affichent à l'écran de l'utilisateur simplement avec du langage HTML, CSS, et Javascript. Cependant, cela n'est que le résultat obtenu « coté client ». En effet, la plupart des pages internet sont générées « coté serveur » par du code qui peut être bien différent, comme par exemple ASP, PHP, Python, Ajax, etc. Nous devons donc choisir le langage qui servira à la fois à générer les pages web, mais également à lancer notre programme d'acquisition.

Pour cela, nous envisageons d'utiliser une nouvelle fois le langage Python, en version 2.7, dont le framework Django peut permettre le développement d'application web. Cet outil est notamment utilisé par Instagram ou la Nasa pour leur site web respectif.

Le fonctionnement de ce Framework repose sur une architecture MVCT (Modèle Vue Contrôle Template). Concrètement, lorsque l'internaute appelle une page, l'outil Django se charge, via les règles de routage définies dans le *Contrôleur*, d'exécuter la *Vue* correspondante. Cette dernière récupère les données des *Modèles* et génère un rendu HTML à partir du *Template* et de ces données.

En nous plongeant dans l'étude de l'outil, et en le portant sur notre projet, nous somme confronté à un nouveau problème, similaire à celui évoqué précédemment : le contrôleur analyse le nom de la page qui est appelée et cible le *Modèle* à exécuter en conséquence. Nous serions donc contraint de placer l'appel de notre programme dans un *Modèle*, qui par son fonctionnement ininterrompu viendra bloquer l'exécution de la suite du *Modèle,* et donc par hiérarchie entraine l'immobilisation du *Contrôleur,* donc de l'application entière. Nous devons donc dissocier l'exécution de l'application de l'exécution du programme d'enregistrement.

Problème soulevé : Comment éviter la gestion de threats dans notre interface graphique ?

Nous devons utiliser un langage dynamique, capable d'exécuter un script Python en parallèle de l'exécution de l'interface. Les solutions sont multiples, nous opterons alors pour l'utilisation de PHP,

langage simple à prendre en main et dont nous avons déjà quelques notions. Il dispose notamment d'une fonction *exec()* qui lui permet d'exécuter une ligne de commande et même capturer et afficher la réponse du terminal. Nous allons donc pouvoir exécuter notre programme d'enregistrement de la même manière que si nous le lancions depuis la console.

Pour le contenu non-dynamique, nous coderons manuellement les pages en HTML et CSS.

Ainsi, nous disposons d'une interface dont la structure graphique (fenêtre, menus, etc.) est déjà existante, et dont l'exécution de scripts en parallèle est possible à implémenter dans nos contraintes de temps de développement.

III.4.3. Droits d'exécution

Comme cela a été mentionné précédemment, la fonction *exec()* de PHP peut exécuter une ligne de commande avec la même syntaxe que cela est permis dans un terminal. En effet, à chaque appel, cette fonction opère l'ouverture d'une console « à usage unique » sans interface graphique (donc invisible pour l'utilisateur), et exécute la commande donnée en paramètre. Le terminal se charge d'effectuer l'opération demandée, avant, le cas échéant, de retourner un résultat. La console est alors close, et le résultat donné par celle-ci est capturé par PHP qui le place dans un tableau, donné également en paramètre par l'utilisateur.

Nous observons cependant que toutes les commandes testées ne sont pas systématiquement exécutées. En effet, toute les opérations sous permission Sudo ne génèrent aucun retour, erreur ou résultat. Nous comprenons finalement que les droits d'exécution de l'application Web ne permettent pas à PHP d'invoquer Sudo, même si l'application est lancée depuis un terminal en mode Super Utilisateur.

Problème soulevé : Comment permettre à PHP d'exécuter un programme Python avec les privilèges Super Utilisateur ?

Les scripts PHP sont interprétés par Apache, le serveur HTTP permettant le fonctionnement en interne de notre application sans connexion internet. Il fonctionne en mode *Worker*, sous lequel un processus père disposant des droits d'utilisateur système, prépare les ressources pour ses threads qui gèrerons les demandes entrantes. Dans notre cas, il s'agit de "www-data" qui est responsable de la préparation des ressources, et ses droits ne sont pas suffisants pour opérer des commandes Super Utilisateur, nous devons donc corriger ce problème en modifiant le fichier /etc/sudoers, document de configuration de la commande Sudo. Nous utiliserons l'outil de vérification visudo qui, avant enregistrement des modifications apportées au fichier, vérifie son intégrité afin de ne pas autoriser une erreur irrémédiable.

A la suite des lignes existantes, nous placerons la ligne de code suivante :

```
www-data ALL=(ALL) NOPASSWD: /usr/lib/python /var/www/programme_a_executer.py
```

Cette instruction indique que l'utilisateur www-data (donc Apache), a l'autorisation d'utiliser la commande Sudo sans exigence de mot de passe pour exécuter un programme Python spécifique.

III.4.4. Développement de l'interface

Après avoir pris toutes les dispositions nécessaires au bon fonctionnement de notre interface, nous pouvons structurer et développer notre code.

L'interface donne accès à 4 rubriques : la première présente le projet, la deuxième apporte des informations sur le sismomètre et permet le lancement du programme d'acquisition, la troisième permet la visualisation des sismogrammes enregistrés, et enfin la quatrième donne accès à la documentation que nous avons rédigé.

La première et dernière partie ont des formats conventionnels, principalement composés de HTML. La seconde partie, et plus complexe puisqu'elle fonctionne autour de deux pages : un index et un lanceur. L'index et la page principale de la rubrique par laquelle l'utilisateur peut appeler le lanceur dans un nouvel onglet automatique. Enfin le lanceur se charge de l'exécution du programme Python qui va capter le signal numérique, le convertir et l'enregistrer au format MiniSEED dans un répertoire spécifique. L'application n'est alors pas arrêtée, et il est possible de continuer la navigation pour se rendre dans la rubrique « Sismogrammes ». Le programme PHP exécute ici une recherche de tous les fichiers d'extension *.mseed* dans le répertoire cité précédemment, et retranscrit la liste complète sous forme de tableau trié du plus récent sismogramme au plus ancien. Lorsqu'un fichier du tableau est sélectionné, une page spécifique s'ouvre pour donner les informations du fichier : un script Python (lancé grâce à la fonction *exec*) se charge alors de retourner le contenu des métadonnées de chaque objet « Trace » contenu dans le « Stream » MiniSEED. Le signal est également affiché sur toute la largeur de la page pour plus de clarté.

Enfin, nous plaçons sur le Bureau un lien direct vers l'adresse interne de l'Application Web grâce à une fonctionnalité du navigateur Epiphany (aperçu ci-dessous).

Figure III.8 *Icône de l'application web sur le bureau de la Raspberry Pi*

En conclusion, la combinaison des fonctions PHP et des programmes Python permet d'accéder aux données des sismogrammes et d'utiliser les outils ObsPy au sein de l'application web. La structure des échanges de données peut apparaitre complexe, mais ce travail permet au final de proposer à l'utilisateur un résultat clair et intuitif.

III.5. Synthèse et mise en production

Lors de notre travail de développement, chaque étape a été scrupuleusement notée dans nos fiches protocolaires afin de permettre la ré-implémentation sur une autre carte Raspberry Pi, en vue d'une mise en production. Ainsi, nous avons collecté les instructions de configuration, d'installation de librairies, de structure et cela en plus de notre programme et de notre interface utilisateur afin de garantir leur fonctionnement sur une autre machine.

III.5.1. Synthèse

Nous avons testé le temps nécessaire au téléchargement et à l'installation des prérequis pour l'utilisation du programme et de l'interface, en utilisant une connexion internet wifi sur un réseau domestique non encombré.

- Téléchargement de l'OS et installation sur carte SD : 20 min

- Configuration du système : 10 min

- Installation des paquets Obspy, I2C, et librairie DelatSigma : 45 min

- Installation et configuration du serveur interne (Apache, PHP, etc) : 15 min

- Installation et vérification du programme et de l'interface : 10 min

Au total, le temps de configuration de la carte Raspberry pour un opérateur s'élève à plus d'1h30 sans compter les éventuelles erreurs de manipulation obligeant à passer un temps considérable à débugger. Par ailleurs, le produit doit s'adresser à tous, et particulièrement aux élèves et aux enseignants, et ne doit pas être uniquement accessible aux utilisateurs et connaisseurs du système Linux. Nous avons donc souhaité apporter une solution à ce problème, en simplifiant les manipulations et raccourcissant le temps d'installation.

III.5.2. Partage de configuration

Le partage de configuration est la solution que nous avons privilégié. Nous nous sommes également concerté sur la possibilité de conception d'un script d'installation, mais cela nécessite un certain temps de développement dont nous ne pouvons disposer à l'heure actuelle.

Nous proposons donc de partager notre travail entier sur la plateforme d'échange www.tess-erasmus.fr en mettant à disposition du public la configuration finale de notre carte Raspberry Pi B+.

Cette configuration est un unique fichier en format image (.img), qui se télécharge comme un OS à l'adresse :

http://www.tess-erasmus.fr/raspberry_sismo.img

Tout y est déjà installé et correctement configuré, il suffit simplement d'écrire l'image sur une carte microSD d'au moins 8Go et d'insérer cette dernière dans la Raspberry Pi. Sur le bureau, un lien est proposé donnant accès à l'interface de gestion du sismomètre.

Cependant, afin de pallier à une erreur de compatibilité (installation sur un autre modèle de Raspberry Pi par exemple), nous laissons à disposition les fiches protocolaires expliquant chaque étape d'installation (en Annexe)

Chapitre IV : Perspectives du projet

IV.1. Bilan provisoire du projet

IV.1.1. Partie Mécanique et Analogique

La partie mécanique et analogique est parfaitement opérationnelle sur le sismomètre avec le LVDT. La mesure est linéaire, et la contre-réaction fonctionnelle mais rapide, elle ramène le sismomètre à la position nulle après une secousse en quelques oscillations.

IV.1.2. Partie Numérique

La chaine d'acquisition numérique, comprenant la conversion du signal, le traitement, l'encodage, l'enregistrement et la restitution des résultats, est pleinement fonctionnelle.

Les signaux analogiques en entrée sont numérisés par le convertisseur Delta Sigma en signaux 18 bits. Ils sont transmis à la carte Raspberry Pi par le bus I2C sur les adresses 0x68 et 0x69. Le signal est intercepté par un programme en Python 2.7 qui fait tourner un Ring Buffer en continu pour stocker temporairement le signal. Toutes les 30 secondes, la dernière portion du buffer est enregistrée et associée au signal de temps fourni par le module RTC pour constituer un tout, qui est encodé et enregistré au format *.mseed* et *.png*. Les résultats sont restitués par une Application Web sur Epiphany, programmée en PHP et langages web (html, css).

IV.2. Tâches restant à accomplir

Sur la partie mécanique, nous n'avons pas encore effectué la contre-réaction sur le sismomètre muni du capteur capacitif. Il s'agit plus ou moins de la même que pour le LVDT, seules les paramètres sont à préciser. Concernant la modélisation de la contre-réaction, il faut encore assembler les différentes composantes et valider le fonctionnement général.

IV.3. Pistes d'amélioration

IV.3.1. Contre-réaction

A l'origine, nous voulions partir sur une contre-réaction numérique, mais cela ne s'est encore jamais fait en sismologie (à part sur le sismomètre marsien, où une partie de la CR est numérique et l'autre analogique). A cause du manque de temps pour en faire la recherche, nous n'avons pu l'implémenter. Il s'agit donc d'une piste de recherche intéressante.

Par ailleurs, il serait possible d'améliorer la contre-réaction analogique, en la rendant modulable, avec choix d'amplification et temps d'intégration par exemple.

IV.3.2. Réduction du bruit lors de la mesure

Comme nous avons pu le voir lors de la mesure d'un déplacement, un bruit électronique important est présent, ce qui empêche de visualiser des secousses de faible amplitude. Deux solutions sont alors envisageables : filtrer le bruit, ou réduire le bruit électronique, en choisissant des composants adaptés ou en optimisant le circuit.

IV.3.3. Performance de la carte Raspberry Pi

La carte Raspberry Pi B+ fonctionne avec un processeur Broadcom BCM 2835 ARM11 cadencé à 700MHz, une carte graphique VideoCore IV, et une mémoire RAM de 512Mo.

Au moment où notre projet a été lancé, cette carte était la plus performance d'une ligne de 4 produits Raspberry Pi (A, A+, B et B+). En février 2015, la Fondation Raspberry Pi a officialisé le lancement d'un nouveau modèle, la Raspberry Pi 2. Cette dernière mouture arbore un processeur Broadcom BCM2836 quatre cœurs ARMv7 cadencé à 900 MHz, accompagné de 1 Go de RAM.

Ces nouvelles performances au même prix seraient véritablement profitable pour notre projet qui nécessite beaucoup de ressources, et une bonne fluidité en multitâche.

IV.3.4. Performance du convertisseur Analogique Numérique

Notre choix du convertisseur s'est orienté sur le Sigma Delta Pi de ABElectronics UK, répondant à notre cahier des charges. Nous observons cependant que ces performances en conversion 16 et 18 bits peuvent être insuffisantes. Il est sans doute possible de travailler sur l'optimisation du programme Python, mais il est vraisemblable que l'évolution vers un convertisseur plus puissant puisse être bénéfique. Il faudrait évaluer les coûts engendrés par cette modification, et vérifier que cela reste compatible avec le budget défini.

IV.3.5. Alimentation du sismomètre

Un des objectifs de ce projet était de construire un sismomètre facilement déplaçable. Lors de nos tests au cours de ce projet, nous avons à chaque fois utilisé une alimentation stabilisée, contraignante à déplacer, pour alimenter notre LVDT. Par ailleurs, la Raspberry Pi nécessite elle aussi une alimentation. Il faudrait donc trouver une solution pour alimenter le sismomètre intégralement, pour le rendre facilement transportable, qu'il puisse être placer dans des lieux peu facile d'accès, et que peu d'intervention soit nécessaire à son bon fonctionnement.

Conclusion

Ce projet nous a permis de découvrir le monde de la sismologie. Il s'inscrit dans le cadre du projet Erasmus Sismo des Ecoles. Le fait d'avoir pu travailler sur les différents domaines de l'électronique a rendu le projet d'autant plus intéressant.

Notre premier défi a été de trouver le capteur adéquat. Nous nous sommes intéressés aux capteurs de déplacement de type inductif, capacitif et optique. Ceci nous a amené à prendre contact avec des fabricants de capteurs, ce qui nous a permis de voir la gamme de prix de ce type de capteurs. Ne trouvant pas de prix satisfaisant le cahier des charges (inférieur à 100€) nous sommes retournés vers les capteurs faits main. Une fois le choix du capteur fait, il a fallu implémenter ce dernier. Nous avons ensuite adapté le signal en vue d'une numérisation, puis d'une contre-réaction. De plus, il a fallu trouver un moyen d'agir sur la masse dans la boucle de rétroaction.

Nous avons rencontré des difficultés avec la mise en place des systèmes, nous n'obtenions pas les résultats attendus, les valeurs des composants n'étaient pas toujours les bonnes, et la prise en main les logiciels de modélisation MATLAB et CADENCE a été délicate.

Ce projet nous a beaucoup appris sur le travail en équipe, tant au niveau répartition des tâches que communication, et sur le fait de se confronter à des problèmes inconnus. Nous avons également apprécié travailler sur la recherche de solutions toujours moins chères, dans l'optique de minimiser les coûts de production et de respecter le cahier des charges concernant la contrainte du prix, tout en ayant une qualité de mesure se rapprochant de celles des sismomètres professionnels.

Bibliographie

[Bod'14] D. Bodor, "Compatibilité des systèmes", Hackable Magazine, 2014

[Cha'09] E. Charmois, *La Conversion A /N « Single Bit » (Delta-Sigma)*, 2009

[Iri'12] I.R.I.S., *Standard for the Exchange of Earthquake Data (SEED), Reference Manual*, 2012

[NXP'14] NXP Semiconductors, *I2C-bus specification and user manual*, Rev.6, 2014

[Pil'12] R. Pillet, *De seosmometricum, Manuel de Sismométrie*, 2012

[Ras'14] Raspberry Pi Fundation, *About Raspberry Pi*, 2015

[Tav'13] C. Tavernier, *Raspberry Pi, Prise en main et premières réalisation*, 2013

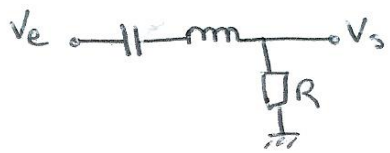
[Fel'15] J. –F. Fels, *Modèle IPG (Institut de Physique du Globe, Paris)*, 2015

Annexes

Annexe A : Modélisation des différentes parties du sismomètre

Partie mecanique : (Pendule).

Passe-Bande ⇒ circuit CLR série



$$V_S = \frac{R \cdot V_e}{\frac{1}{jC\omega} + jL\omega + R} = \frac{jRC\omega\, V_e}{1 + jRC\omega - LC\omega^2}$$

De la forme $\frac{V_s}{V_e} = \frac{2m j \frac{\omega}{\omega_0}}{1 + 2m j \frac{\omega}{\omega_0} - \left(\frac{\omega}{\omega_0}\right)^2}$

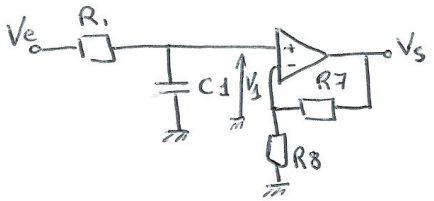
Par identification on a : $\begin{cases} \omega_0^2 = \frac{1}{LC} \\ \frac{2m}{\omega_0} = RC \end{cases} \Rightarrow \begin{cases} \omega_0 = \frac{1}{\sqrt{LC}} = \frac{2\pi}{T_0} \\ m = \frac{RC}{2} \cdot \frac{1}{\sqrt{LC}} \end{cases}$

On cherche les valeurs de R, C, L pour $\begin{cases} T_0 = [1; 3; 5; 7] \\ m = 0,4. \end{cases}$

⇒ On fixe R

$$\begin{cases} C = \frac{4m^2}{R^2} \cdot L \\ \left(\frac{2\pi}{T_0}\right)^2 = \frac{1}{LC} \end{cases} \Rightarrow \begin{cases} C = \frac{m T_0}{\pi \cdot R} \\ L = \left(\frac{T_0}{2\pi}\right)^2 \cdot \frac{1}{C} \end{cases}$$

Partie Lissage et amplification.



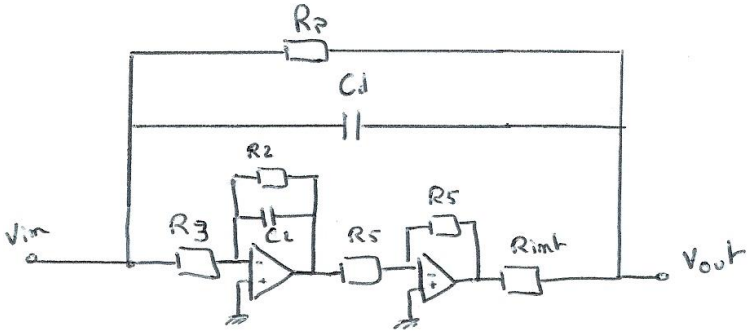
$$V_1 = \frac{V_S/R_7}{\frac{1}{R_7}+\frac{1}{R_8}} = V_S \cdot \frac{R_8}{R_7+R_8}$$

On a aussi $V_1 = \frac{1/jC_1\omega}{R_1+\frac{1}{jC_1\omega}} \cdot V_e$

D'où $V_S = \left(\frac{R_7+R_8}{R_8}\right) \times \frac{1}{1+jR_1C_1\omega} V_e$

De la forme $\frac{V_{so}}{V_e} = \frac{H_0}{1+j\frac{\omega}{\omega_0}}$ avec $\begin{cases} H_0 = (1+R_7/R_8) \\ \omega_0 = \frac{1}{R_1C_1} \end{cases}$

Partie correcteur PID



Fonction de transfert

$$\frac{V_{out}}{V_{in}} = \frac{1}{R_p} + jCd\omega + \frac{1}{R_{int}(1+j\omega\tau_{int})} = \frac{R_{int}(1+j\omega\tau_{int}) + jCd\omega + R_p}{R_p \cdot R_{int}(1+j\omega\tau_{int})}$$

$$\frac{V_{out}}{V_{in}} = \frac{R_{int}+R_p+j\omega(\tau_{int}R_{int}+Cd)}{R_pR_{int}\cdot(1+j\omega\tau_{int})}$$

où $\tau_{int} = R_3 \cdot C_2$

Annexe B : Code MATLAB

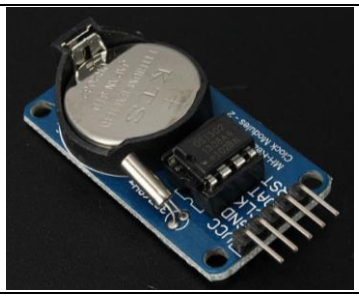
```
clear all;
close all;
clc;

% %--------------- PENDULE -------------------
% %On réalise un équivalent électrique du pendule

syms s

T1=1; % période du ressort
w01=2*pi/T1;
amort=0.4; %amortissement du ressort
R=13;
C1=T1*amort/(R*pi);
L1=(T1/(2*pi))^2/C1;

num=sym2poly(s*R*C1);
den=sym2poly(s^2*L1*C1+R*C1*s+1);

F1=tf(num,den);
figure()
hold on
bode(F1);



T3=3; % période du ressort
w03=2*pi/T3;
amort=0.4; %amortissement du ressort
R=13;
C3=T3*amort/(R*pi);
L3=(T3/(2*pi))^2/C3;

num=sym2poly(s*R*C3);
den=sym2poly(s^2*L3*C3+R*C3*s+1);

F3=tf(num,den);
hold on
```

```
bode(F3);



T5=5; % période du ressort
w05=2*pi/T5;
amort=0.4; %amortissement du ressort
R=13;
C5=T5*amort/(R*pi);
L5=(T5/(2*pi))^2/C5;

num=sym2poly(s*R*C5);
den=sym2poly(s^2*L5*C5+R*C5*s+1);

F5=tf(num,den);
hold on
bode(F5);

T7=7; % période du ressort
w07=2*pi/T7;
amort=0.4; %amortissement du ressort
R=13;
C7=T7*amort/(R*pi);
L7=(T7/(2*pi))^2/C7;

num=sym2poly(s*R*C7);
den=sym2poly(s^2*L7*C7+R*C7*s+1);

F7=tf(num,den);
hold on
bode(F7);

hold off
damp(F7)
figure()
impulse(F);
```

```
% %-------------- Lissage + Ampli -------------

syms s

R1=9000;
R7=10000;
R8=1000;
C1=1e-9;

W0=1/(R1*C1);
H0=(R7+R8)/R8;

num_LA=(H0);
den_LA=sym2poly(R1*C1*s+1);

F_LA=tf(num_LA,den_LA);
bode(F_LA);

% %-------------- PID -----------------

syms s

% Déclaration des variables :

R2=25e+6; % Résistance intégrateur réel
R3=4e+6;
R56=20e+3; % Résistance du montage ampli
Rint=906e+3;
C2=33e-6;
Tau_int=R3*C2;
Cd=3.3e-6;
Rp=3.7e+6;

num_PID=sym2poly((Rp+Rint)+s*(Tau_int*Rint+Cd));
den_PID=sym2poly(Rp*Rint(1+s*Tau_int));

F_PID=tf(num_PID,den_PID);
bode(F_PID);
```

Annexe C : Comparatif des outils de mesure du temps pour Raspberry Pi

1. Module RTC Simple DS1302 (ou DS1307)

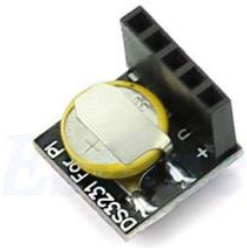


Caractéristiques
• Real time clock with the year 2100, to calculate the seconds, minutes, hours, days, weeks, months, years, the ability, as well as the ability to leap year adjustment • simple 3-wire interface • Vcc = 5V and TTL-compatible • Dual power tube for the main power and backup power supply • Wide range of operating voltage 2.0 5.5V • Current 2.0V, less than 300nA • Serial I / O port pin makes the least number of ways • Read / write clock or RAM data transmission when there are two single-byte transmit and multi-byte character sets transfer mode • 8-pin DIP package or optional 8-pin SOIC package assembly according to the surface • Size : 44mmX23mmX1.6mm • Working Temperature: 0℃~+70℃ • Operating Voltage Range:3.3v ~ 5V
Références / Coût
http://www.ebay.fr/itm/1x-RTC-DS1302-Module-Horloge-Avec-Batterie-Temps-Reel-Module-Pr-Arduino-AVR-ARM-/141412690425?pt=FR_YO_MaisonJardin_Bricolage_ElectroniqueComposants&hash=item20ecda69f9 1,23€ FPI

Avantages	Inconvénients
- Faible coût	- Déviation possible de 50ppm (2min par mois)
- Faible consommation (300nA)	- Besoin d'une connexion internet pour remettre l'heure à jour

Le produit final ne prévoit pas de connexion internet. La forte déviation des mesures semaine après semaine rend donc cette solution non conforme à nos attentes.

2. Module RTC haute précision DS3231



Caractéristiques

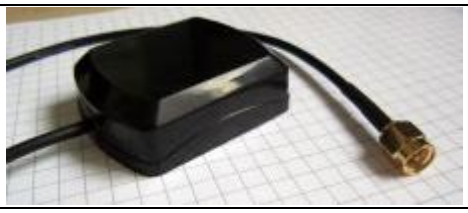
• Accuracy ±2ppm from 0°C to +40°C
• Accuracy ±3.5ppm from -40°C to +85°C
• Battery Backup Input for Continuous Timekeeping
• Operating Temperature Ranges
  ○ Commercial: 0°C to +70°C
  ○ Industrial: -40°C to +85°C
• Low-Power Consumption
• Real-Time Clock Counts Seconds, Minutes, Hours, Day, Date, Month, and Year with Leap Year Compensation Valid Up to 2100
• Two Time-of-Day Alarms
• Programmable Square-Wave Output
• Fast (400kHz) I²C Interface
• 3.3V Operation
• Digital Temp Sensor Output: ±3°C Accuracy
• Register for Aging Trim
• Active-Low RST Output/Pushbutton Reset Debounce Input

Références / Coût
http://www.ebay.fr/itm/DS3231-Module-RTC-precision-module-de-memoire-pour-Arduino-Raspberry-Pi-/171502014410?pt=FR_YO_MaisonJardin_Bricolage_ElectroniqueComposants&hash=item27ee510fca 1,73€ FPI

Avantages	Inconvénients
- Déviation max de 2ppm (2,6s par mois) - Faible coût - Faible consommation	- Besoin d'une connexion internet pour remettre l'heure à jour

La déviation maximale de la mesure est intéressante sur cette solution, et son coût est très intéressant. Le module RTC DS3231 pourrait convenir pour notre projet.

3. Module GPS avec antenne active externe



Caractéristiques
Module de positionnement Ublox MAX-7Q Récepteur : 56-channel u-blox 7 engine Fréquence de rafraichissement : up to 10 Hz Précision : < 2.5 m CEP Acquisition : Cold starts: 29s, Aided starts: 5s, Réaquisition: 1s Sensibilité : -162 dBm Crystal RTC : built-in Antennes supportées : actives et passives **Antenne** Fréquence centrale 1575.42MHz±3 MHz Bande passante ±5 MHz Impendance 50 ohm Gain maximum >3dBic par rapport à une antenne ground plane de 7x7cm Ouverture >-4dBic at –90°<0<+90° (plus de 75% en volume) **Caractéristiques du LNA/Filtre** Gain du LNA (sans le câble) 28+/-3dB Bruit 1.5dB Typ. Alimentation Tension continue 2.7V/3.0V/3.3V/5.0V – 3.0V à 5.0V Consommation 5mA /11mA/15mA Max
Références / Coût
Carte de traitement : http://ava.upuaut.net/store/index.php?route=product/product&path=59_60&product_id=95 49.51€ Antenne active : http://ava.upuaut.net/store/index.php?route=product/product&path=59_65&product_id=54 12.97€ Total : 62,48€

Avantages	Inconvénients
- Fréquence de mise à jour des données (10Hz) - Précision de la localisation - Grand nombre de canaux (56)	- Prix important - Encombrement - Consommation de l'antenne

Le choix d'un GPS apporte la précision de la mesure du temps, mais apporte également une donnée de localisation utile pour éviter au client de rentrer manuellement son fuseau horaire. Cependant le prix de ce module reste dissuasif pour notre projet.

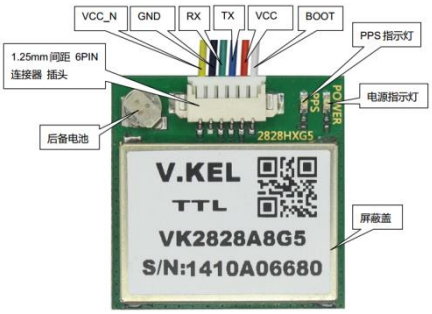
4. Module GPS USB de type « mouse »

Caractéristiques
Modèle: BU-353S4 SiRF Star IV Haute Performance GPS chipset Fréquence : L1 1575,42 MHz Canaux : 48 Sensibilité :-163 dBm Précision : <2,5m Port de connexion USB interface Livré avec logiciel CD & ventouse Dimensions: 5,2 cm x 5,2 cm x 2,0 cm Poids: 116 g
Références / Coût
http://www.dx.com/p/globalsat-bu-353s4-usb-gps-receiver-106659#.VOoZaS6Q6Ft 29,76 € FPI

Avantages	Inconvénients
- Simplicité d'utilisation et de câblage - Précision de la localisation	- Prix encore relativement important par rapport au module RTC

Cette solution « Plug&Play » présente en plus des avantages d'un GPS, celui d'être simple à utiliser et à un prix moins important. On considèrera cependant que ce prix reste encore trop important pour notre projet.

5. Module V.KEL TTL avec antenne céramique intégrée



Caractéristiques
VK2828A8G5 Specification: - C/A code: 1.023MHz, Receive frequency: L1[1575.42MHz] - Follow channel: 24 - Support DGPS[WAAS, EGNOS and MSAS] **Position performance:** - 2D dimension: < 5m - Time precision: 1µs - Coordinate system: WGS-84 **Electronic performance:** - Follow sensitivity: -161dBm - Capture sensitivity: -153dBm - Hot start: 1s, Warm start: 37s, Cold start: 39s, Recapture time: 0.1s **Operation temperature**: -30°+80° **Size:** 28*38*8.4mm **Port Electronics:** - Wide voltage range: +3.5V~+5.0V - With 25*25*4 ceramic antenna - Output baud rate: 9600bps - Output GGA, GLL, GSA, GSV, RMC data
Références
http://www.ebay.fr/itm/VK2828A8G5-G-Mouse-GPS-Module-GPS-Navigation-Board-w-TTL-Level-Antenna-/111403836517?pt=LH_DefaultDomain_71&hash=item19f02fa465 6,86€ FPI

Avantages	Inconvénients
- Localisation automatique - Réactualisation continue des mesures de temps et de localisation - Prix très avantageux	- Démarrage lent (39s) comparé aux autres systèmes - Système moins précis (5m) - Peu de canaux (24)

Cette solution GPS est la moins chère possible, et bien que la précision soit moins bonne que ses concurrents, ses performances restent suffisantes pour l'application à laquelle elle est destinée.

Classement des outils de mesure du temps

1. Module RTC haute précision DS3231
2. Module V.Kel TTL
3. Module GPS USB de type Mouse
4. Module RTC simple DS1302
5. Module GPS avec antenne externe active

Choix de l'outil de mesure du temps à utiliser

Nous choisirons le module **RTC haute précision DS3231** essentiellement pour son **prix abordable**. Nous aurions pu nous tourner pour quelques euros de plus vers une solution GPS « low-cost » mais nous pouvons faire l'économie d'une détection automatique du fuseau horaire, en donnant la possibilité à l'utilisateur de le sélectionner par lui-même.

Annexe D : Protocole de mise en route de la carte

Fiche Info : Protocole de mise en route de la carte Raspberry Pi

- Télécharger sur un ordinateur tiers (sous Windows) la dernière version Raspbian au format .zip sur le site http://www.raspberrypi.org/downloads/ et extraire le fichier « XX-raspbian-wheezy.img » de l'archive.

- Insérer la carte SD (avec son adaptateur) et noter la lettre du disque (F:/ par exemple).

- Télécharger l'utilitaire Win32 Disk Imager, permettant d'écrire l'image disque «Raspbian » sur la carte SD. Téléchargement depuis le site : http://sourceforge.net/projects/win32diskimager/

- Ouvrir l'archive, extraire l'exécutable Win32DiskImager, cliquer-droit sur celui-ci et sélectionner «Exécuter en tant qu'administrateur ».

Sélectionner le fichier « xx-raspbian-wheezy.img », puis renseigner la lettre du disque identifiant la carte SD (F:/ ici par exemple). Attention à ne pas se tromper de disque, car toutes les données présentes sur le disque sélectionné seront supprimées.

- Cliquer sur « Write » et attendre la fin de l'écriture

- Une fois l'installation terminée, vous pouvez retirer la carte SD de l'ordinateur, et insérer la microSD dans la carte Raspberry Pi, à l'emplacement prévu à cet effet (sous la carte).

- Mettre la carte Raspberry Pi sous tension, et y brancher un clavier, une souris et un écran.

- Finaliser l'installation de Rasbian en suivant les étapes à l'écran. En arrivant sur l'écran de configuration :

  ○ Vérifiez bien que la bonne langue et le bon fuseau horaire sont sélectionnés dans l'item « Internationalisation Options ».

  ○ Choisissez également « lancer le bureau au démarrage » dans l'item «Enable to boot to Desktop ».

○ Autorisez l'expansion du système à toute la carte SD en sélectionnant « Expand Filesystem».

○ Dans « Advanced Options », sélectionnez I2C afin de le rendre automatiquement actif au démarrage de la carte (permet de communiquer avec le convertisseur).

Note : Par défaut, le nom d'utilisateur est « **pi** » et le mot de passe « **raspberry** ».

Finissez la configuration en sélectionnant « Finish ».

- La Raspberry est fonctionnelle.

Attention à bien l'éteindre correctement (via l'icône « Eteindre », ou avec la ligne de commande :

```
sudo shutdown -h now
```

Ne pas la débrancher du secteur avant l'arrêt total du système.

Annexe E : Protocole d'installation des prérequis pour le traitement des données

Fiche info : Protocole d'installation des prérequis pour le traitement des données

- Connecter la carte Raspberry à internet en utilisant un câble réseau ou une clé wifi.

Pour la seconde option, il faut se rendre dans les *préférences*, puis dans *Wifi Configuration*. Appuyer sur le bouton *Scan*, puis sélectionner votre réseau (double clic). Renseigner le mot de passe du réseau dans le champs PSK de la fenêtre qui s'ouvre à vous. Cliquer enfin sur *Add*, puis sur *Connecter*. La connexion a internet devrait alors être active.

- Mettre à jour les paquets existants (l'instruction `sudo` peut nécessiter le mot de passe administrateur, soit « ***raspberry*** »):

```
sudo apt-get update
sudo apt-get upgrade
```

- Ouvrir la console (LXTerminal)

- Installer gedit :

```
sudo apt-get –y install gedit
```

- Modifier le fichier listant les sources des paquets

```
sudo gedit /etc/apt/sources.list
```

Ajouter à la suite du texte existant la ligne suivante :

```
deb http://deb.obspy.org CODENAME main
```

en remplaçant CODENAME par la version de Debian de la Raspberry (à l'heure actuelle, nous utilisons *wheezy*).

- Importer la clé GNUPG permettant de vérifier l'intégrité des packages téléchargés :

```
wget --quiet -O - https://raw.github.com/obspy/obspy/master/misc/debian/public.key | sudo apt-key add -
```

- Installer ensuite ObsPy avec toutes ses dépendances :

```
sudo apt-get update
sudo apt-get install –y python-obspy
```

La bibliothèque ObsPy est maintenant prête à être utilisée.

- Installer les module de gestion du bus I2C et outils de développement indispensables :

```
sudo apt-get install -y python-smbus build-essential libi2c-dev i2c-tools python-dev
```

- Autoriser le lancement du module I2C au démarrage :

```
sudo nano /etc/modules
```

Si elle n'y est pas, ajouter la ligne suivante. Si elle est déjà présente et précédée d'un caractère #, retirer celui-ci.

```
i2c-dev
```

Sauvegardez le fichier par un `Ctrl O` suivi de `Entrée` et quittez nano par un `Ctrl X`.

De manière inverse, si le fichier de « blacklist » contient le module I2C, il convient de commenter ou effacer la ligne :

```
sudo nano /etc/ modprobe.d/raspi-blacklist.conf
```

Editer le document, le résultat doit être le suivant :

```
#blacklist i2c-bcm2708
```

Sauvegardez le fichier par un `Ctrl O` suivi de `Entrée` et quittez nano par un `Ctrl X`.

- Télécharger, extraire, puis copier dans le dossier personnel les librairies Python permettant d'exploiter le convertisseur analogique numérique Sigma Delta de AB Electronics UK :

```
wget https://github.com/abelectronicsuk/ABElectronics_Python_Libraries/archive/master.zip
unzip master.zip -d master
cp -R master/ABElectronics_Python_Libraries-master/DeltaSigmaPi DeltaSigmaPi
rm -r master master.zip
```

Le programme Python2.7 d'enregistrement des données provenant du sismomètre est prêt à être utilisé. A ce stade, toutes les dépendances permettant son exécution ont été téléchargées, installées, et configurées.

Annexe F : Protocole d'installation de l'interface utilisateur :

Fiche Info : Protocole d'installation de l'interface utilisateur

- Ouvrir LXTerminal et installer le serveur web Apache, l'interpréteur PHP et le système de gestion des bases de données MySQL.

```
sudo apt-get install -y apache2 php5 mysql-server libapache2-mod-php5 php5-mysql
```

(demande de mot de passe : laisser vide 3 fois)

- Modifier la configuration sudo pour permettre à Apache l'exécution de programme Python avec les permissions Super-Utilisateur.

```
sudo nano /etc/sudoers
```

Sous la ligne "`root ALL=(ALL) ALL`", ajouter la ligne suivante :

```
www-data ALL=(ALL) NOPASSWD: /usr/lib/python
```

Sauvegardez le fichier par un `Ctrl O` suivi de `Entrée` et quittez nano par un `Ctrl X`.

- Télécharger l'Application Web depuis la plateforme tess-erasmus.fr :

```
wget http://www.tess-erasmus.fr/tess.zip
sudo chmod 777 -R /var/
unzip tess.zip -d tess
sudo chmod 777 -R tess
sudo cp -R tess  /var/www/
sudo chmod 777 -R /var/www/tess
rm -r tess tess.zip
```

- Ouvrir le navigateur Web et lancer l'application :

```
epiphany-browser http://localhost/tess
```

- Dans la barre d'outil en haut à droite, sélectionner « Enregistrer en tant qu'application Web » et disposer l'icône sur le Bureau pour un accès rapide.