École Centrale de Lyon

Projet d'Études

26 juin 2013

PE 103 : Création d'un outil d'aide à la mesure des champs magnétiques de basse fréquence

---

Olivier CHURLAUD, Sylvain HEMETTE, Yuechen LIU, Cédric OGER, Xiaoyi YANG

Ampère

ÉCOLE CENTRALE LYON

Résumé

Ce projet a été réalisé sous la tutelle et à la demande de MM. Laurent Morel et Riccardo Scorretti, du laboratoire Ampère, afin qu'ils puissent par la suite étudier l'action d'un champ magnétique basse fréquence sur l'organisme.

Le but du projet est de concevoir un outil d'aide à la mesure des champs magnétiques de basse fréquence dans un espace défini, telle une pièce. Cet outil nous permet de cartographier le champ magnétique dans un tel environnment.

Nous avons réalisé un support mobile permettant de déplacer le capteur ainsi qu'un programme MATLAB, qui réalise l'ensemble des acquisitons et des traitements de données. Le programme est réalisé pour travailler avec quatres entrées analogiques : les trois premières étant les magnétomètres mobiles permettant d'obtenir les composantes selon x,y et z du champ, la quatrième servant de référence pour le calcul de la phase du signal.

La réelle valeur ajoutée de l'outil est sa capacité de détecter la position du capteur, ce qui accélère énormément les mesures.

Mots-clés : champ magnétique, basse fréquence, ELF, mesure, outil, programme, aide, expérimentation

Table des matières

0 Introduction **4**
0.1 Contexte . . . 4
0.2 Projet . . . 4
0.3 Étapes et ordres de présentations . . . 4

1 Équipe du Projet d'Étude 103 **5**

2 État de l'art **6**

3 Cahier des Charges **8**
3.1 Présentation . . . 8
3.2 Définition des objectifs et contraintes . . . 8
3.2.1 Contexte . . . 8
3.2.2 Définition fonctionnelle . . . 8
3.2.3 Contraintes . . . 8
3.2.4 Environnement . . . 8
3.2.5 Livrables . . . 8

4 Conception de l'outil **10**
4.1 Présentation du fonctionnement attendu . . . 10
4.1.1 Traduction des besoins en un outil fonctionnel . . . 10
4.1.1.1 Prérequis . . . 10
4.1.1.2 Acquisition des données . . . 11
4.1.1.3 Acquisition des coordonnées du capteur . . . 11
4.1.1.4 Traitement des données . . . 13
4.1.1.5 Sauvegarde des données . . . 13
4.1.2 Scénario d'utilisation de l'outil . . . 13
4.1.2.1 Préparation . . . 13
4.1.2.2 Mesures . . . 14
4.1.2.3 Visualisation . . . 14
4.1.2.4 Erreurs . . . 14
4.1.3 Bilan . . . 15
4.2 Programmation de l'outil . . . 15
4.2.1 Réflexions et décisions . . . 15
4.2.1.1 Choix du language de programmation . . . 15
4.2.1.2 Choix de l'architecture . . . 15
4.2.2 Programme : présentation par fonctionnalité . . . 16
4.2.2.1 Initialisations . . . 16
4.2.2.2 Acquisition et traitement des données . . . 16

4.2.2.3 Acquisition des coordonnées du capteur . . . 17
4.2.2.4 Sauvegarder les données . . . 17
4.2.2.5 Showme : un GUI pour afficher la cartographe 3D de la pièce . . . 18
4.2.3 Maintenabilité et documentation . . . 18
4.2.3.1 Maintenabilité . . . 18
4.2.3.2 Robustesse . . . 19
4.2.3.3 Documentation . . . 19
4.2.4 Améliorations possibles . . . 20
4.2.5 Bilan . . . 20
4.3 Conception de la carte électronique et choix des magnétomètres . . . 20
4.3.1 Étapes suivies lors de la conception du choix du capteur et de la conception du circuit électrique . . . 20
4.3.2 Choix des capteurs . . . 21
4.3.3 Carte de conditionnement du capteur . . . 21
4.4 Conception du support . . . 22
4.4.1 Cahier des charges . . . 22
4.4.2 Réflexion . . . 23
4.4.3 Construction . . . 23
4.4.4 Améliorations . . . 25
4.4.5 Bilan . . . 25
4.5 Validations de l'outil . . . 25
4.5.1 Programmation . . . 25
4.5.1.1 Rectification de la perspective . . . 25
4.5.1.2 Coordonnées du capteur dans l'espace . . . 26
4.5.1.3 Acquisition des données . . . 26
4.5.1.4 Traitement et visualisation des données . . . 26
4.5.2 Électronique . . . 28
4.5.2.1 Test du capteur . . . 28
4.5.2.2 Test du circuit d'initialisation . . . 28

5 Perspectives d'améliorations 29
5.1 Programme . . . 29
5.2 Support . . . 29
5.3 Outil général . . . 29

6 Conclusion 30

A Gestion de projet 31

B Documentation 35

Table des figures

1.1 Organigramme du groupe-projet 5

4.1 Image acquise déformée par la perspective 12
4.2 Image reconstruite à partir de H 12
4.3 Scénario d'utilisation du programme 14
4.4 GUI pour cartographier le champ d'une pièce 19
4.5 Avec trois capteurs unidimensionnels 21
4.6 Avec un unique capteur tridimensionnels 21
4.7 Photo et concept de fonctionnement du HMC2003 21
4.8 Schéma de branchement du HMC2003 21
4.9 Forme des signaux dans le circuit 22
4.10 Circuit d'initialisation du capteur 22
4.11 Esquisse du support (réflexion) 23
4.12 Photographie du support terminé 24
4.13 Image acquise déformée par la perspective 26
4.14 Image reconstruite à partir de H 26
4.15 Amplitude du champ créé par un fil en fonction de l'éloignement 27
4.16 Affichage du champ créé par un fil 27
4.17 résultat de l'interpolation d'une hyperbole 28
4.18 schéma de principe du MOFSET 28

A.1 Diagramme de décomposition des tâches 32
A.2 Diagramme Pert 33
A.3 Diagramme Gantt 34

0. Introduction

0.1 Contexte

L'homme côtoie de nombreux types de champs électromagnétiques. Certains sont d'origine naturelle comme le champ magnétique terrestre ou la lumière solaire, d'autres ont été créees par l'homme pour de nombreuses applications (transport, communication, industrie). Les conséquences à long terme de ces types de champs sur la santé humaine sont encore mal connues. Comme le nombre de sources de champs électromagnétiques est en forte croissance depuis les années 80, les institutions et centres de recherches s'interrogent sur leurs effets sur la santé. L'Union Européenne travaille par exemple à la définition de niveaux admissibles pour limiter l'exposition humaine aux champs électromagnétiques.

Si les champs de moyenne et haute fréquence sont assez documentés (quoi qu'un grand nombre d'études restent à effectuer), les champs de basse et très basse fréquence (moins de 500 Hz) ont été peu étudiés. C'est pourtant une gamme de champs générés par de nombreux systèmes : le courant secteur (50 Hz) et les machines électriques ou moteurs dans les usines (20 à 200 Hz) notamment.

0.2 Projet

Notre commanditaire, le laboratoire Ampère, travaille sur un projet ayant pour but de simuler et quantifier l'effet de ce type de champs sur l'organisme. Pour cela, il est nécessaire de caractériser précisément l'environnement électromagnétique, dans le cadre d'une activité productive ou dans des lieux de vie. Une telle caractérisation nécessite de nombreuses mesures qui devront être traitées par la suite. Cette partie expérimentale est très fastidieuse et il n'existe actuellement pas d'outil référencé pour aider à la réalisation de ces mesures. Notre groupe de Projet d'Etude a alors commencé la conception d'un tel outil. L'acquisition de données doit en être accélérée afin que le projet de recherche de nos commanditaires soit plutôt concentré sur le traitement des données plutôt que son recueillement.

0.3 Étapes et ordres de présentations

Notre projet doit donc suivre une procédure de *documentation* et *état de l'art* afin de définir un *cahier des charges* avec les commanditaires. Cela aboutira à la *conception* d'un programme-outil répondant aux objectifs fixés. Enfin afin de valider le travail accompli, il est nécessaire de *calibrer* le capteur et de le confronter à la réalité d'un problème connu afin de *quantifier ses erreurs*. Pour assurer une maintenabilité et un futur au projet, il devra être *documenté* et un *manuel d'utilisation* l'accompagnera.

Nous expliciterons notre projet en suivant les trois grands domaines de notre organisation : le développement du programme (section 4.2) permettant l'acquisition et le traitement des données, puis la réalisation du circuit électrique (section 4.3) et enfin la conception du support (section 4.4). Chaque partie s'accompagnera de perspectives d'améliorations qui seront résumées dans une dernière partie (section 5).

1. Équipe du Projet d'Étude 103

L'équipe s'est organisée en pôles de compétences :

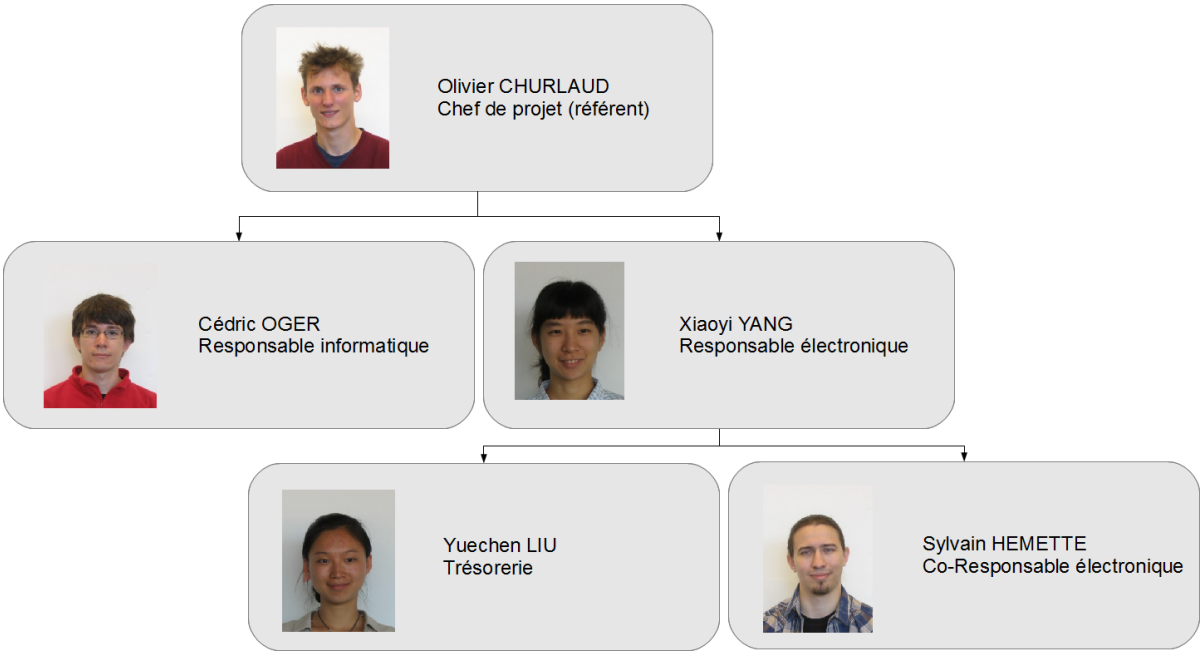

FIGURE 1.1 – Organigramme du groupe-projet

2. État de l'art

Des conjonctures imputent aux champs magnétique de basse fréquence d'accroitre le nombre de leucémies ou cancers. En conséquence, la norme ICNIRP a fixé un seuil critique à 10^{-4} Tesla. Mais les études sur les champs de basses fréquences sont moins nombreuses et variées que celle sur les champs de haute fréquence. Nous avons étudié la littérature en début d'année, il en ressort qu'il y a finalement peu d'études dans le domaine sur lequel on se cantonne. Voici un échantillon représentatif des documents que nous avons étudiés.

Mesure du champ magnétique autour d'une plaque à induction [2]. Un mur de mousse percé tous les 90cm permet de placer un capteur autour de la plaque d'induction. Le champ est ensuite interpolé entre les différents points avant d'en faire l'étude. Cette étude est intéressante car :

- La méthode de mesure est très proche de celle que l'on veut appliquer : on acquière de nombreuses mesures puis on interpole le champ dans le reste du volume.

Mais les points suivants font qu'elle n'et pas applicable dans notre cas :

- Les mesures se font en hautes fréquences (20kHz)
- Le placement des capteurs se fait selon une grille préétablie, dans un volume très petit. Or on veut prendre des mesures dans une pièce de plus grande taille, et sans contraintes de positionnement pour l'utilisateur.

Modélisation du champ produit par des lignes hautes tension dans un magasin [8]. Afin de vérifier si les normes de sécurité sont suivies, un logiciel de modélisation 3D à été utilisé pour calculer le champ magnétique dû aux lignes HT dans le magasin. Cette étude est intéressante car :

- L'étude se fait dans la bonne gamme de fréquence
- Elle permet d'avoir une bonne idée de la répartition des champs dans l'espace

Mais les points suivants font qu'elle n'est pas applicable dans notre cas :

- C'est une modélisation analytique du champ, pas des mesures de celui-ci

Mesure des champs auxquels sont soumis les travailleurs [9], [5]. Des capteurs fixes, repartis à une certaines distance de la machine, ou alors des capteurs sur la poitrine des utilisateurs permettent de mesurer le champ produit par des machines à forts courants électriques. Cette étude est intéressante car :

- On mesure le champ dans la pièce, ou bien sur l'utilisateur

Mais les points suivants font qu'elle n'est pas applicable dans notre cas :

- Les mesures effectuées sont des valeur efficaces du champ, on ne peut donc pas reconstituer le signal dans la pièce.
- Dans le cas où les mesures sont prises dans la pièce, le nombre de relevés est trop faible pour avoir une bonne cartographie. Dans l'autre cas, une cartographie n'est pas possible.

Modélisation du champ crée par une machine de soudure et les courants induits dans le corps humain [4]. Modélisation du corps humain comme un réseau de

résistance et vérification du modèle sur le champ magnétique créé par une machine de soudure. Cette étude est intéressante car :

- Elle montre que les normes ne sont pas assez précises pour protéger les utilisateurs

Mais les points suivants font qu'elle n'et pas applicable dans notre cas :

- Un signal pur 50 Hz est utilisé en entrée de la machine à souder, sans prendre en compte les harmoniques.

Bilan Si les articles comme le dernier ne nous permettent pas d'avancer le projet, ils montrent qu'il y a un réel besoin de développer des outils pour mesurer les basses fréquences de manière efficace. Ceci pour améliorer les normes, et surtout leur mise en place. Le premier texte nous a donné un protocole de mesure qui a pu être discuté dans notre projet. Les autres textes nous ont permis de nous familiariser avec le sujet.

3. Cahier des Charges

3.1 Présentation

Réalisation d'un outil permettant de mesurer et caractériser un champ magnétique très basse fréquence industrielle (avec une attention particulière pour le 50Hz, pas de fréquence Maximale définie) dans un espace délimité (exemple : local domestique ou industriel). L'outil sera simple d'utilisation et permettra une acquisition rapide des données

3.2 Définition des objectifs et contraintes

3.2.1 Contexte

Problème Actuellement, le manipulateur doit réaliser la mesure et ajouter dans un tableur les données de position, l'amplitude et la phase, à la main. C'est une manipulation longue et pénible qui pourrait être améliorée.

But Permettre la mesure du champ dans le but de vérifier le respect des normes de santé.

3.2.2 Définition fonctionnelle

- Acquisition des données de position transparente pour l'utilisateur : une webcam permet la détection des coordonnées.
- Mesure de l'amplitude et de la phase du signal des champs dans un domaine de 10Hz à 50kHz, avec une amplitude de 0,1μT à 1mT
- Interpolation de la fonction champ en une fonction continue.
- Sortie : fonction MATLAB f(x,y,z,f) = [Bx,By,Bz]
- Si avancement suffisant, demande de points supplémentaires : intégration du logiciel de calcul de champs électromagnétiques dans la « boucle de mesure », estimation de l'erreur de mesure, et de l'impact de ce dernier sur les résultats du calcul.

3.2.3 Contraintes

- Le logiciel de traitement est une application MATLAB
- Erreur de moins de 10 %

3.2.4 Environnement

- On suppose les signaux périodiques décomposables en séries de Fourier
- On teste l'outil dans une pièce quasi-vide

3.2.5 Livrables

- Rapport pédagogique
- Logiciel

- Prototype
- Estimation de l'erreur commise
- Manuel d'utilisation
- Test de validation de l'outil

4. Conception de l'outil

Nous avons décomposé la conception de l'outil en grands domaines. Tout d'abord, une traduction du cahier des charges en fonctions et utilités recherchées. Quelques points théoriques y seront aussi abordés. Nous viendrons ensuite à la programmation de l'outil. Ces deux premières parties sont relativement proches et se complètent. Ainsi, la première suffira au lecteur souhaitant uniquement comprendre le fonctionnement global du programme final, quand la deuxième décrira plus précisemment comment le code est pensé et implémenté. En revanche, aucune ligne de code ne sera détaillée ici : la documentation B regroupe l'intégralité du programme.

Nous explicitons ensuite l'électronique du système, à savoir la carte électronique et la manière dont les magnétomètres ont été sélectionnés, puis la manière dont a été pensé le support mobile, qui porte la capteur principal.

Enfin la validation de différentes parties du projet sera traitée, ainsi que des améliorations auxquelles nous avons pensées.

4.1 Présentation du fonctionnement attendu

Le fonctionnement de l'outil réalisé est étroitement lié au protocole expérimental souhaité. Notre premier besoin était donc la mise en place d'un protocole général, suffisament large pour des applications diverses. Il semble par exemple raisonnable de définir que l'utilisateur placera le capteur, lancera la mesure puis, une fois la mesure effectuée, déplacera à nouveau le capteur pour une nouvelle mesure. Finalement un tel recueil de mesure est ce qu'il y a de plus simple. Appliquer cette apparente évidence n'est pas forcément la simplicité même.

4.1.1 Traduction des besoins en un outil fonctionnel

4.1.1.1 Prérequis

On supposera que l'utilisateur a les élément suivants :

- **quatre magnétomètres** dont un fixe servant de référence pour le calcul de la phase et trois mobiles (ou un tridimensionnel) permettant d'obtenir les coordonnées du champ dans les trois dimensions
- une **carte d'acquisition** pour faire communiquer l'ordinateur avec les magnétomètres
- une **webcam**
- un ordinateur avec MATLAB et les pilotes pour la carte d'acquisition et la webcam
- ce que l'on appelera à partir de maintenant le **gabarit** c'est à dire une feuille A3 permettant de calibrer la webcam
- ce que l'on appelera à partir de maintenant le **disque de couleur** ou **marqueur** qui permettra de repérer le capteur
- le support du capteur

4.1.1.2 Acquisition des données

L'acquisition des données se fait relativement simplement : une carte d'acquisition contrôlée par une fonction MATLAB transmet les données depuis un capteur vers l'ordinateur par connexion USB. Il n'y a qu'à trouver et implémenter cette fontion.

4.1.1.3 Acquisition des coordonnées du capteur

Lors de la conception du cahier des charges (voir 3), nous avons proposé l'utilisation d'une webcam pour simplifier l'acquisition de la position du capteur dans l'espace. Le problème auquel avons donc été confronté est l'extraction, depuis une photo en 2D, d'une position dans l'espace. Avec une seule photo il est impossible d'obtenir trois dimensions. Nous avons donc considéré qu'il ne serait pas trop pénible d'indiquer la hauteur du capteur à chaque acquisition, si l'on pouvait en contrepartie s'affranchir de l'indication des coordonnées dans le plan.

Nous avons continué de simplifier le programme en décidant que le programme ne chercherait pas le capteur dans la photo prise par la webcam mais uniquement un objet, par exemple circulaire et de couleur vive, posé au sol sous le capteur. Cela nous permet de nous concentrer sur la détection d'objet sur un plan. Évidemment la webcam a un angle de vue et la perspective modifie donc les coordonnées de l'objet sur l'image. Il est donc nécessaire ici de comprendre la transformation qui nous fait passer d'une photographie de l'espace, comportant une déformation due à la perspective, à une image sans perspective.

Démonstration de cette transformation [3] Notons le point $x = \begin{pmatrix} x \\ y \end{pmatrix}$.

Passage en coordonnées homogènes. Les coordonnées homogènes sont une classe de vecteurs, dans la laquelle les vecteurs $k.X$ et X sont équivalents. Ainsi, dans un plan, une ligne est représentée par une équation du type : $ax + by + c = 0$, qui est équivalent à $k.(ax + by + c) = 0$. On peut donc représenter la ligne par les coordonnées $l = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ ou $k.\begin{pmatrix} a \\ b \\ c \end{pmatrix}$. Les coordonnées sont homogènes.

En faisant varier a, b et c, on peut obtenir différentes lignes.

Réécrivons l'équation de la ligne $ax + by + c = (x, y, 1) * l = 0 = k.(x, y, 1) * l$. Un vecteur de $\mathbb{R}^2$ peut donc être représenté par les coordonnées homogènes $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$. En appelant $P^2 = \mathbb{R}^3 - 0, 0, 0$ l'espace de projection, on peut généraliser : le vecteur homogène $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ dans P^2 permet de caractériser le point $\begin{pmatrix} \frac{x_1}{x_3} \\ \frac{x_2}{x_3} \end{pmatrix}$ dans $\mathbb{R}^2$.

Une autre propriété évidente mais utile d'un point est qu'il est à l'intersection de deux ligne, et qu'on peut écrire : $x = l \wedge l'$ car $< x|l >= 0$ et $< x|l' >= 0$ et x est équivalent à $k \times x$. de même $l = x \wedge x'$.

Dans P^2, la classe de transformations qui nous intéresse est la classe des applications projectives, ou homographies. Ce sont des applications linéaires bijectives entre coordon-

nées homogènes, qui peuvent s'écrire sous la forme d'une matrice 3×3 :

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{H} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rectification de la perspective Une des application des homographies est la rectification de la perspective, et c'est ce qui nous intéresse ici. Dans l'image 4.1, les points

FIGURE 4.1 – Image acquise déformée par la perspective

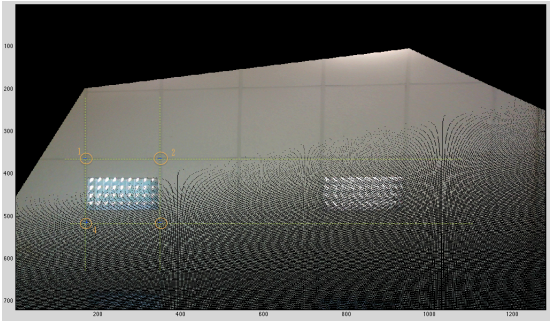

FIGURE 4.2 – Image reconstruite à partir de H

ont des coordonnées (x, y), soit $(x, y, 1)$ en coordonnées homogènes. Dans l'image 4.2, les coordonnées sont (x', y'), soit $(x1, x2, x3)$ en coordonnées homogènes . On a alors les équations :

$$\begin{cases} x_1 = h_{11} \cdot x + h_{12} \cdot y + h_{13} \\ x_2 = h_{21} \cdot x + h_{22} \cdot y + h_{23} \\ x_3 = h_{31} \cdot x + h_{32} \cdot y + h_{33} \end{cases}$$

et

$$\begin{cases} x' = \frac{x_1}{x_3} \\ y' = \frac{x_2}{x_3} \end{cases}$$

Obtenir H En fixant 4 points dans la première image et leur coordonnées après transformation dans la deuxième image, on obtient 8 équations indépendantes (si l'aire du quadrilatère n'est pas dégénérée) qui nous permettent d'obtenir tous les coefficients de H. On pose pour le premier point la matrice :

$$S_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x'_i & y_i x'_i & x'_i \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i x'_i & y_i x'_i & x'_i \end{bmatrix}$$

On concatène les 4 matrices pour obtenir une matrice 8×9, puis on cherche le noyau de cette matrice.

En effet si on pose :

$$h = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}$$

alors pour $1 \geq p \geq 4$

$$(S * h)_{2p-1} = \underbrace{-x.h_{11} - y.h_{12} - h_{13}}_{-x_1} + x'(\underbrace{x.h_{31} + y.h_{32} - h_{33}}_{x_3}) = -x_1 + \frac{x_1}{x_3}x_3 = 0$$

$$(S * H)_{2p} = \underbrace{-x.h_{21} - y.h_{22} - h_{23}}_{-x_2} + x'(\underbrace{x.h_{31} + y.h_{32} - h_{33}}_{x_3}) = -x_2 + \frac{x_2}{x_3}x_3 = 0$$

Donc le vecteur h cherché est bien vecteur directeur du noyau de S, il reste juste ensuite à le transformer en une matrice 3×3 pour obtenir H.

4.1.1.4 Traitement des données

Lors de l'acquisition des données, nous obtenons un signal temporel. Afin qu'il soit exploitable, il est nécessaire d'en calculer sa transformée de fourier et sa phase, afin d'obtenir pour chaque fréquence une amplitude complexe.

Comme nous utilisons un capteur basse fréquence, il nous faut supprimer les composantes moyenne et basse fréquence qui ne reflètent pas le champ mesuré. Après l'avoir fenêtré (fenêtre de Hanning), le signal subit donc une transformée de Fourier discrète (via l'algorithme FFT) puis est idéalement filtré par suppression des composantes supérieures à la fréquence de coupure.

Enfin, à l'aide du deuxième capteur, nous pouvons récupérer la phase du signal. Une fonction de nos commanditaires permettant déjà de réaliser cette opération, nous n'aurons qu'à l'ajouter au programme final.

4.1.1.5 Sauvegarde des données

Le but de l'outil étant d'obtenir les données traitées, il nous a fallu choisir la manière dont nous allions sauvegarder les données. Afin d'augmenter la robustesse de l'enregistrement et ne pas tout perdre en cas de problème logiciel, à chaque acquisition nous écrivons à la fin d'un fichier `data.mat`.

4.1.2 Scénario d'utilisation de l'outil

4.1.2.1 Préparation

L'utilisateur place l'ordinateur, branche la carte d'acquisition et attend qu'elle soit reconnue et fixe une webcam en hauteur.

Il lance MATLAB et notre application. Il entre la commande :

```
>> init
```

qui démarre l'initalisation du programme. Au cours de cette initialisation, un certain nombre de questions seront posées à l'utilisateur lui permettant d'initialiser la carte d'acquisitioon, la webcam et les différentes variables du programme.

4.1.2.2 Mesures

L'utilisateur règle la hauteur du pied de support pour une série de mesures et le pose à l'endroit où la mesure doit-être faite. Sur l'ordinateur, il saisit :

```
>> acquisition(1.05)
```

où 1.05 est la valeur, en mètres, de la hauteur du support.

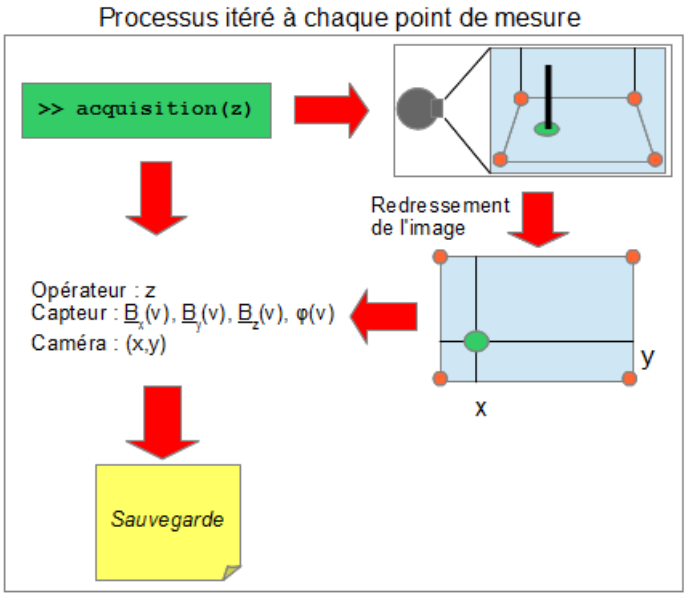

FIGURE 4.3 – Scénario d'utilisation du programme

Il peut ensuite déplacer le capteur et recommencer la démarche jusqu'à ce qu'il juge le nombre de mesures suffisant. A la fin de la commande, les données de obtenues sont automatiquement enregistrées sous forme de structure (cf. documentation MATLAB) dans le dossier courant de l'utilisateur. Tout cela est schématiquement résumé dans la figure 4.3.

4.1.2.3 Visualisation

L'utilisateur pourra de plus visualiser une cartographie du champ magnétique dans l'espace étudié en trois dimensions en entrant la commande :

```
>> showme
```

4.1.2.4 Erreurs

Si l'utilisateur s'aperçoit que les mesures de coordonnées ne sont pas assez précises ou qu'il s'est trompé lors d'une des initialisations, il a la possibilité de relancer toute l'initialisation comme lors du démarrage de l'expérience. Il peut aussi ne lancer que la partie à corriger (`init_variables`, `init_daq`, `init_webcam` ou `init_matrix_H`).

4.1.3 Bilan

Nous avons ici traduit le cahier des charges en un concept de programme, comprenant d'une part des outils théoriques et un plan de travail, d'autre part d'un scénario, qui nous permettront d'ailleurs de rédiger rapidement une première documentation pour l'utilisateur (voir la section 4.2.3 et l'annexe B) et d'avoir les idées claires pour le développement du code, explicité dans la section 4.2

4.2 Programmation de l'outil

L'outil à créer repose essentiellement sur l'informatique pour l'acquisition et le traitement des données, quelles soient spatiales ou temporelles. Le programme doit donc être pensé et préparé au niveau de sa structure et de sa logique d'une manière la plus complète possible afin de ne pas être gêné lors de l'assemblage des diverses parties de code.

Il doit non seulement répondre aux besoins des commanditaires (respect du cahier des charges) mais aussi être modifiable et améliorable par de prochaines équipes ou le commanditaire. La maintenabilité est donc le deuxième point crucial. Dans la mesure où ce projet est une preuve de concept, la précision n'arrive qu'en troisème position dans la liste des priorités.

4.2.1 Réflexions et décisions

4.2.1.1 Choix du language de programmation

Dans la mesure où nos commanditaires utilisent régulièrement MATLAB dans leurs travaux, que c'est un outil très répandu en recherche et que le language de programmation est relativement commun, nous avons décidé de programmer entièrement la partie logicielle de notre outil sur MATLAB. De plus, ne pas avoir à compiler le programme et le fait qu'il soit multi-plateforme permet un gain de temps non négligeable dans la programmation à plusieurs développeurs.

4.2.1.2 Choix de l'architecture

Nous souhaitions tout d'abord réaliser une interface GUI (c'est à dire avec des menus, icones cliquables etc.). Après réflexion, nous nous sommes arrêtés sur une solution en ligne de commande. En effet, il n'y a que 3 à 6 commandes nécessaires à l'utilisation complète de notre programme. Afin de simplifier l'utilisation, il nous suffit d'ajouter une commande `helpme` qui rappellera l'ensemble des commandes définies.

Pour alléger les codes de ces commandes principales et permettre la réutilisation de certains codes récurrents, nous avons découpé chaque script ou fonction possible en sous-fonctions. La maintenabilité est alors accrue, puisqu'il n'est pas necessaire de relire quelques milliers de lignes de code pour comprendre le fonctionnement du programme : le simple enchaînement des appels fonctions et scripts permet d'appréhender comment l'outil a été conçu.

Enfin, pour une meilleure gestion de la mémoire et de l'organisation des données, nous utilisons 4 structures définies de manière globale comprennant l'ensemble des données utiles et réutilisables au fil du programme :

- `data` : ensemble des données nécessaires à l'exploitation des acquistions (coordonnées et champs aux différents points).

- `filter_parameters` : paramètres du filtre passe-bas permettant de supprimer les fréquences hautes.
- `session` : configuration de la carte d'acquisition.
- `webcam` : configuration de la webcam.
- `other_data` : ensemble des autres données nécessaires à la reprise des expériences (paramétrages de l'acquisiton)
- `color` : configuration de la couleur du marqueur de détection de position.
- `z` : hauteur du capteur.

4.2.2 Programme : présentation par fonctionnalité

Dans cette partie nous détaillerons le programme par bloc fonctionnels. En effet, de nombreuses fonctions sont imbriquées et leur explicitation serait ici longue, fastidieuse et sans intérêt. En revanche, un découpage par fonctionalité permet d'avoir une vision globale du programme et d'en comprendre les problématiques.

4.2.2.1 Initialisations

Comme indiqué dans le scénario (voir 4.1.2), l'utilisation du logiciel est précédé par l'appel des initilisations. Le programme va initialiser, dans l'ordre :

1. les variables globales présentées en 4.2.1 : des questions seront posées à l'utilisateur pour définir par exemple les dimensions de la pièce dans laquelle les mesures sont réalisées. (`init_variables.m`)
2. la carte d'acquisition : en supposant que la carte est une National Instrument, le programme va repérer les différents paramètres et se configurer pour pouvoir communiquer avec la carte. L'utilisateur n'aura plus qu'à répondre aux dernières questions permettant de choisir la fréquence et le temps d'acquisition. (`init_DAQ.m`)
3. la webcam : comme les noms donnés par Matlab aux différentes caméras diffèrent selon l'OS de l'ordinateur et la marque des webcam, l'utilisateur est un peu plus sollicité ici. Une fois un choix de caméra effectué, le programme essaiera s'il le peut de finir seul la configuration de la webcam. Sinon il sollicitera l'utilisateur. Enfin, une fenêtre dite de *preview* s'ouvre. Elle permet à l'utilisateur de positionner la webcam de sorte d'avoir un angle convenable. Ceci fait, il doit quitter la fenêtre pour que le programme passe à la dernière initialisation. (`init_webcam.m`)
4. la matrice de passage permettant de supprimer la perspective : le programme va prendre une image de la pièce, dans laquelle devra se trouver le support. L'utilisateur devra alors sélectionner les quatre coins de l'étalon le plus précisément possible. S'il est insatisfait de sa sélection, il peut recommencer, sinon il peut terminer l'initialisation. Le programme va ensuite en utilisant les mathématiques vues dans la section 4.1.1 construire la matrice de passage entre les points sélectionnés et un rectangle qu'il aura créé. L'utilisateur pourra ensuite vérifier la validité de la rectification en lançant la commande `disp_rebuilt.m`.

4.2.2.2 Acquisition et traitement des données

Acquisition Ceci est le rôle principal de cet outil. Il doit pouvoir récupérer puis traiter les données recueillies par le capteur. Le programme est conçu pour fonctionner avec une

carte d'acquisition National Instruments, mais une petite modification des initialisations permettra sans peine d'utiliser un autre modèle (si toute fois elle est reconnue par Matlab).

Traitements fréquentiels Après une acquisition d'une durée et à une fréquence d'activité définies par l'utilisateur durant l'initialisation sur les quatres capteurs, le signal enregistré est fenêtré grâce à la fonction `hanning(n)` où n est le nombre d'échantillons du signal, puis subit une transformée de Fourier discrète grâce à **fft(signal)**. Enfin, toutes les composantes du signal à fréquence supérieure à la moitié de la fréquence d'échantillonage sont mises à zéro, ce qui revient à un filtre passe-bas idéal.

Grâce aux Toolbox de Matlab, nous avons pu suivre exactement la procédure imaginée lors de la conception (voir section 4.1.1.4).

Obtention de la phase La phase devant être calculée, nos commanditaires nous ont procuré une fonction codée sur Matlab permettant, à partir de deux signaux, de trouver le déphasage. Ceci explique la nécessité du deuxième capteur positionné de manière fixe pour toute la durée de l'expérience. Cette fonction nous renvoit un délai en secondes, que nous pouvons convertir ensuite en une phase en degrés ou radians en le multipliant par la fréquence considérée.

4.2.2.3 Acquisition des coordonnées du capteur

Retrouver le capteur Une fois l'acquisition des données terminée, le programme lance la détection du capteur. Comme indiqué dans la section 4.1.1.3, il va chercher un cercle de couleur au sol, positionné sous le capteur. La webcam commence par prendre une photographie qui est présentée à l'utilisateur, grâce aux fonctions de la toolbox *Image Acquisition Toolbox*. Il clique ensuite, dans un soucis de réduction du nombre de calculs, aux alentours du cercle à détecter. Le programme recherche ensuite, par actions sur les contrastes, un cercle de la couleur souhaitée. Plus précisément, on supprime de l'image les composantes des autres couleurs puis on la convertit en nuances de gris. Le contraste est augmenté.

Finalement, il ne doit rester qu'un disque noir : le disque coloré que nous recherchions. Le centre de l'objet trouvé est montré à l'utilisateur. S'il n'y a pas d'erreur, il valide, sinon il pointe plus précisemment l'objet. Toutes ces actions sont quasi instantanées et permettent un gain de temps pour l'utilisateur qui n'a pas besoin d'être précis. On a alors les coordonnées du disque dans le référentiel de la photo.

Replacer les coordonnées dans le référentiel de la pièce [1] En utilisant les mathématiques présentées dans la section 4.1.1.3, on a créé durant les initialisation la matrice H, de passage entre les deux référentiels. Il n'y a qu'à multiplier cette matrice par les coordonnées obtenues au paragraphe précédent puis de leurs faire subir une homothetie pour obtenir la position du capteur dans la pièce, et de la concaténer avec la valeur z entrée par l'utilisateur lors de l'acquisition pour avoir la position dans les trois dimensions.

4.2.2.4 Sauvegarder les données

À la fin de l'acquisition, toutes les données importantes sont sauvegardées. Un test est réalisé afin de savoir si des données existent déjà dans le dossier, si oui, on écrit à la suite, sinon, on créé le fichier.

L'inconvénient de ceci est qu'à la fin des mesures, il faut penser à récupérer les données, sous peine d'y ajouter d'autres valeurs lors d'une prochaine expérience ; ce qui corromprait évidemment les deux études.

Les données conservées sont dans un seul fichier nommé `data.m` et contenant :

- data : une structure contenant
  - real_coordinates : une matrice contenant les cordonnées du capteur (`[x(:),y(:),z(:)]`)
  - nu_m est un vecteur linéaire contenant toutes les fréquences définies durant l'initialisation (`[`nu_1`(:)...`nu_N`(:)]`)
  - Bx_nu (respectivement By_nu, Bz_nu) : une matrice contenant pour chaque fréquence (en ligne) l'amplitude complexe du champ magnétique selon x (resptectivement y, z) (`[`Bx_{nu_1}`(:)...`Bx_{nu_N}`(:)]`)
  - delta : une matrice contenant les délais en seconde entre les signaux Bx (respectivement By, Bz) et la référence Br (`[`$delta_{B_x}$`(:),`$delta_{B_y}$`(:),`$delta_{B_z}$`(:)]`)
- other_data : une structure comprenant
  - pattern_coord_rebuilt : un vecteur contenant les coordonnées du coin supérieur gauche du gabarit dans le référentiel de la photo en pixels (`[x,y]`)
  - H : la matrice permettant de supprimer la perspective (matrice 3×3)
  - pattern_coord : un vecteur contenant les coordonnées du coin supérieur gauche du gabarit dans le référentiel de la pièce en mètres (`[x,y]`)
  - Image1 : l'image de la pièce prise lors de l'initialisation de la matrice H.
  - pattern_size : un vecteur contenant les dimensions (longueur $\times$ largeur, en mètres) du gabarit dans le référentiel de la pièce (`[L,l]`)
  - pattern_size_rebuilt : un vecteur contenant les dimensions (longueur $\times$ largeur, en pixels) du gabarit dans le référentiel de la photo (`[L,l]`)
  - dimensions : un vecteur contenant les dimensions de la pièce en mètres (`[l, L, h]`)
  - detector_raw_coord : une matrice contenant les coordonnées en pixel de disque de repérage dans le référentiel de la photo, avant tout traitement (`[x(:),y(:)]`)
  - Rate : un réel représentant la fréquence d'échantillonnage (`[`nu_e`]`)

4.2.2.5 Showme : un GUI pour afficher la cartographe 3D de la pièce

L'idée du GUI n'était pas dans le cahier des charges et est apparue au cours de l'année, lors de discussions avec les tuteurs. En effet nous manipulons des entitées numériques mais nous ne pouvons pas vérifier la cohérence physique de notre cartographie. Nous avons donc créé un GUI, c'est à dire une fenêtre graphique, qui permet à l'utilisateur de visualiser l'amplitude du champ en un point.

Comme on le voit sur la figure 4.4, l'utilisateur peut définir des plans selon x, selon y et selon z ainsi qu'une fréquence à afficher au moyen de sliders. Cela permet d'avoir une première vue des intensités de champ dans la pièce.

4.2.3 Maintenabilité et documentation

4.2.3.1 Maintenabilité

Afin de garantir une réutilisabilité de notre programme et les améliorations qu'une mise en production nécessiterait, nous avons décidé de coder l'ensemble de l'outil en anglais, et de le commenter abondamment. Notre projet s'inscrivant dans une optique de recherche,

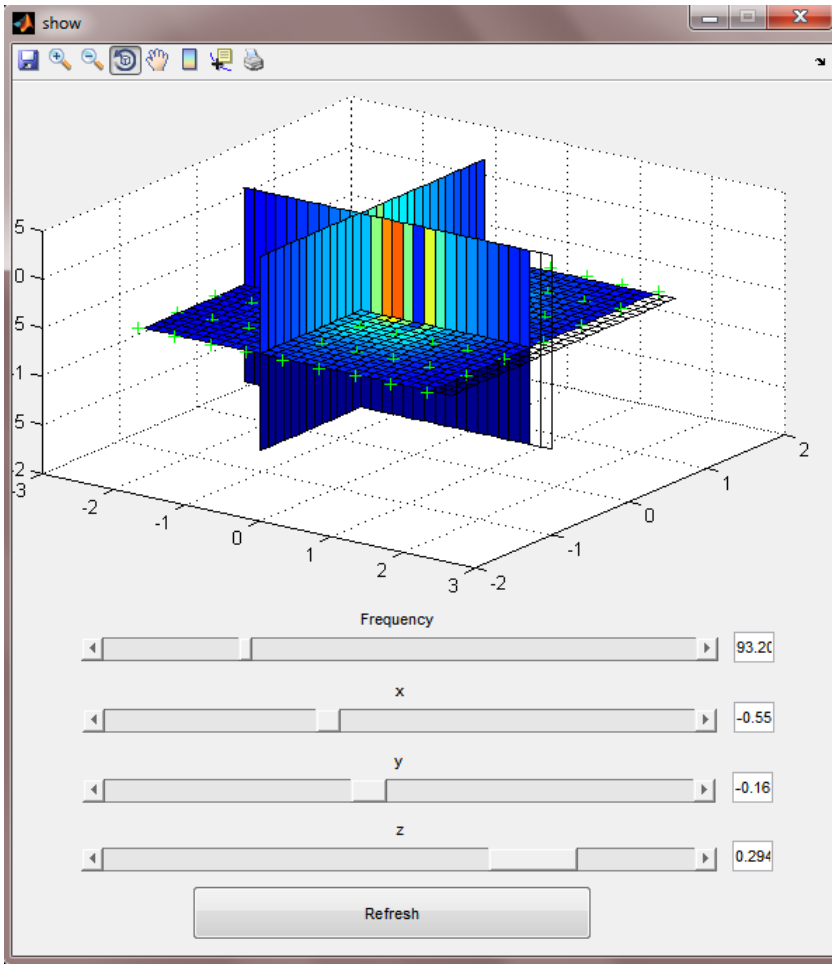

FIGURE 4.4 – GUI pour cartographier le champ d'une pièce

il nous semblait pertinent d'en respecter ces principes de base, bien que le cahier des charges ne l'ait pas spécifié.

4.2.3.2 Robustesse

Nous avons aussi implémenté des tests à chaque intervention de l'utilisateur : lorsqu'un nombre est demandé, un oui/non ou un vecteur, le programme vérifie que l'utilisateur a bien entré la chaine demandé. Dans le cas contraire, il réitère sa question. Pour simplifier les réponses que l'utilisateur doit donner, nous avons proposé des choix par défaut le plus souvent possible : si l'utilisateur ne répond rien, le choix par défaut est appliqué.

4.2.3.3 Documentation

Le programme doit pouvoir être facilement relu et compris, comme expliqué précedemment, mais aussi réutilisé ailleurs ou autrement. C'est pourquoi nous avons rédigé une documentation.

Cette documentation a deux buts :

1. une documentation utilisateur, destinée au manipulateur, afin qu'il comprenne comment se servir de l'outil et y retrouve les commandes utiles. La procédure et le protocole expérimental y seront rappelés et détaillés.

2. une documentation développeur, destinée aux développeurs voulant retravailler sur le projet et aux utilisateurs souhaitant comprendre le fonctionnement interne de notre outil. On y retrouvera le descriptif complet des fonctions commentées du programme, les entrées/sorties de chacune de ces fonctions ainsi que, à terme, des exemples d'utilisations.

La version actuelle de ce document est disponible en annexe B. Nous n'avons malheureusement pas eu le temps d'y ajouter les exemples prévus.

4.2.4 Améliorations possibles

- Factorisation des variables globales : nous aurions pu placer la variable `color` dans `other_data`. De même, la valeur `z` n'est pas lourde ; elle n'a donc pas besoin d'être passée par référence.
- Pouvoir afficher le champ selon un axe ou sa norme (au choix).
- Permettre l'ajout de points hors de l'image
- Améliorer la précision de la recherche du capteur dans l'image

4.2.5 Bilan

Nous avons décrit ici le cœur de notre outil : le programme qui acquiert les données et les traite. Grâce aux différentes *toolbox* et fonctions de MATLAB, il a été plus simple de construire les parties fonctionnelles de l'outil. Chaque fonction construite, nous avons pu relier tout cela et tester l'ensemble du programme. Si celui-ci n'est pas parfait, il reflète déjà une base solide de l'outil. C'est de plus la partie qui caractérise complètement cette aide à la mesure que nous avons créée et qui n'est pas interchangeable (puisqu'un autre capteur ou un autre support pourraient aussi permettre d'obtenir des mesures avec la même simplicité). Néanmoins, ces autres composants ont fait l'objet de réflexion lors de leurs choix et conception et seront présentés dans la suite du présent document.

4.3 Conception de la carte électronique et choix des magnétomètres

Notre outil doit pouvoir mesurer le champ magnétique basse fréquence dans une pièce, dans les trois dimensions. En pratique deux capteurs sont utilisés, l'un (le capteur unidimensionnel) est situé à un endroit fixe : il servira de point de référence pour calculer les différence de phases. Le second (tridimensionnel) est mobile afin de mesurer le champ en plusieurs points différents. Les données recueillies par les capteurs sont ensuite échantillonnées par la carte d'acquisition puis transférées vers l'ordinateur, sur lequel le programme (voir 4.2) réalisera le post-traitement.

4.3.1 Étapes suivies lors de la conception du choix du capteur et de la conception du circuit électrique

- Choix des capteurs permettant de mesurer les champs magnétiques de basse fréquence
- Achat des capteurs et composants associés
- Contrôle des composants et construction du circuit

– Vérification du circuit
– Test d'acquisition afin de valider le choix de capteur

4.3.2 Choix des capteurs

Afin de mesurer le champ en trois dimensions, nous avons d'abord pensé utiliser trois capteurs unidimensionnels sur trois axes différents pour ainsi mesurer les valeurs des composantes du champ. Cette méthode présente une difficulté pour le positionnement exact des capteurs sur les axes voulus (les axes X, Y et Z du repère orthogonal). De plus, les valeurs mesurées sur les différents axes ne sont pas exactement représentatifs de la valeur ponctuelle que nous recherchons comme le montrent les schémas4.5 et 4.6. Le déphasage risque notamment d'être faux. Nous nous sommes donc tournés vers la solution

FIGURE 4.5 – Avec trois capteurs unidimensionnels

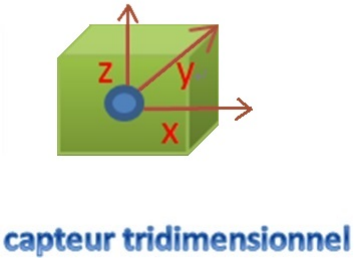

FIGURE 4.6 – Avec un unique capteur tridimensionnels

d'un unique capteur tridimensionnel, malgré son prix beaucoup plus élevé. Nous avons choisi le magnétomètre HMC2003 de HONEYWELL.

Les figures 4.7 et 4.8 permettent de comprendre comment brancher le magnétomètre.

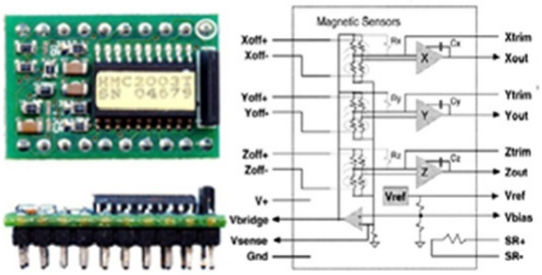

FIGURE 4.7 – Photo et concept de fonctionnement du HMC2003

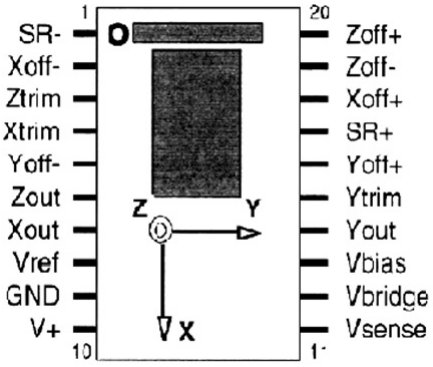

FIGURE 4.8 – Schéma de branchement du HMC2003

4.3.3 Carte de conditionnement du capteur

Le capteur tri-axe choisi délivre une tension fonction du champ magnétique dans lequel il est plongé. Pour obtenir ses tensions il est nécessaire d'alimenter le capteur et de brancher les sorties du capteur sur la carte d'acquisition.

Le capteur peut être déréglé, s'il a reçu un choc par exemple, et peut se retrouver en dehors de sa zone linéaire. Il est alors nécessaire de le réinitialiser. Le circuit qui est présenté dans la figure 4.10 permet de replacer le capteur dans sa zone linéaire [6], [7]. Il est commandé par les tensions Set et Reset (voir Figure 4.9).

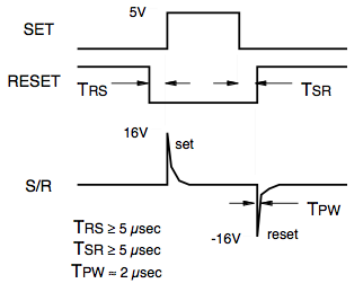

FIGURE 4.9 – Forme des signaux dans le circuit

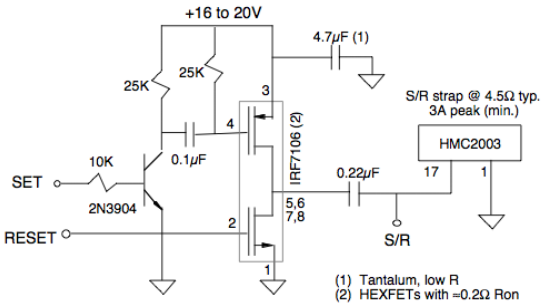

FIGURE 4.10 – Circuit d'initialisation du capteur

Quand Set et Reset sont à 0, un condensateur accumule de l'énergie qu'il va libérer sur le front montant de set, ce qui va donner un pic d'intensité d'environ 3A. Lorsque set et reset sont à 1, le condensateur va libérer son énergie sur un front descendant de reset, créant un pic d'intensité d'environ -3A (voir Figure 4.9).

Capteur de référence Ce capteur n'a pas de vocation à être précis, il est juste nécessaire pour pouvoir mettre les mesures à différents instants dans une même base de temps. Il permet donc de calculer des déphasages entre les différentes mesures prises. C'est pourquoi nous avons choisi un capteur un axe, déjà monté sur son support. Pour obtenir le champ, il suffit de l'alimenter et de brancher la carte d'acquisition sur les bornes de sortie du capteur.

4.4 Conception du support

Afin d'assurer les prises de mesures, nous avons conçu un support pour le capteur et le circuit l'accompagnant avec pour principal objectif de faciliter la manutention et le positionnement du capteur. Comme le projet est expérimental la minimisation des erreurs de positionnement sur le prototype n'a pas été prioritaire dans cette conception.

4.4.1 Cahier des charges

Les principaux critères de conception de l'ensemble du support ont donc été par ordre de priorité :

1. Simple de construction
2. Coût de construction peu élevé
3. Facile et rapide d'utilisation
4. Influence négligeable sur les mesures

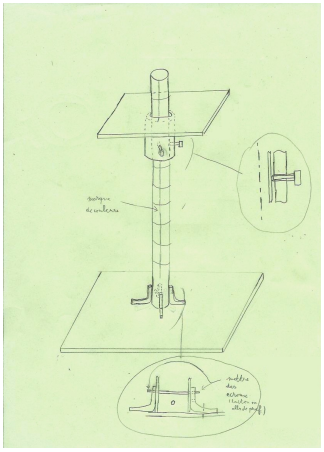

FIGURE 4.11 – Esquisse du support (réflexion)

4.4.2 Réflexion

Pour répondre aux différents critères du cahier des charges, la forme de la partie fixe du support a évolué en gardant l'idée de posséder les caractéristiques suivantes :

- Les matériaux utilisés doivent avoir peu d'influence sur les champs magnétiques (bois, PVC ou laiton).
- La base doit assurer la stabilité du support en étant suffisamment large et lourde pour éviter le basculement de celui-ci ou qu'il ne se déplace à cause de perturbations externes.
- Le poteau doit être suffisamment rigide pour maintenir le capteur toujours dans la même position verticale afin de minimiser les erreurs de positionnement du capteur.

La partie mobile doit posséder les caractéristiques suivantes :

- Permettre d'amener le capteur à différentes hauteurs, rapidement et facilement par exemple avec un système de vis de pression ou de collier.
- Minimiser les jeux entre la partie mobile et le poteau pour minimiser les erreurs de positionnement du capteur.

L'ensemble doit être le moins cher possible.

4.4.3 Construction

Pour répondre aux critères de simplicité et de coût il a été décidé de détourner des matériaux provenant du commerce.

Description du support Pour minimiser l'influence sur le champ magnétique mesuré, nous n'avons utilisé que du bois et du PVC, en dehors des équerres et de la pince (en acier) et dont on considère l'influence négligeable.

La base (Figure 4.12) est constituée d'une planche de bois ainsi qu'une bouche de vidage en PVC collés permettant d'assurer une stabilité suffisante pour maintenir le support debout et une verticalité raisonnable. Le poteau est constitué d'un tuyau de deux mètres de longs et de quarante millimètre de diamètre en PVC (utilisés en plomberie) et fixé à la base par collage. Le diamètre est suffisant pour minimiser le fléchissement du poteau sous le poids de la partie mobile et du capteur.

La mobilité du panneau mobile est assurée par un manchon en PVC. Une planche de bois servant à accueillir le capteur est maintenu en place par serrage grâce à des équerres.

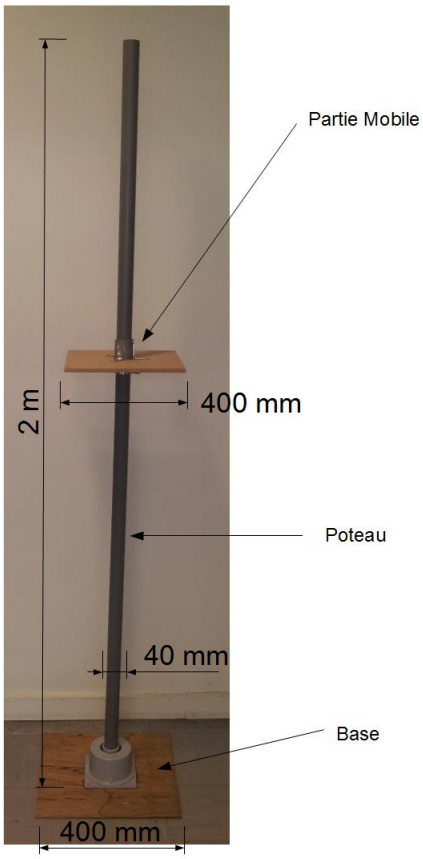

FIGURE 4.12 – Photographie du support terminé

Cette partie mobile glisse sur le poteau et est maintenue à la hauteur voulue grâce à une pince métallique permettant de repositionnement rapide et facile.

4.4.4 Améliorations

Il est possible d'améliorer le support actuel sur plusieurs points :
- Éliminer complètement l'utilisation de matériaux ferromagnétique telle que l'acier pour diminuer encore les possibles perturbations.
- Fabriquer le manchon sur mesure afin de minimiser au maximum les jeux et de pouvoir utiliser un système de maintien en position de la partie mobile plus efficace (actuellement la pince ne peut soutenir qu'un poids relativement faible).
- Utiliser un poteau un peu plus large pour réduire encore les possibilités de fléchissement (le diamètre actuel a été choisi sous la contrainte des matériaux disponible dans le commerce tel que le manchon).
- Améliorer l'isostatisme de la base en utilisant des plots sous la planche ou en utilisant une planche plus épaisse et lourde.

4.4.5 Bilan

Le détournement de matériaux venant du commerce nous a permis de permettre une construction rapide et à faible coût. Cela a en contrepartie induit des imprécisions dues aux différents jeux et à la légère flexion du poteau.

Ce problème d'imprécisions n'est pas primordial pour notre projet puisque c'est une preuve de concept, un prototype, néanmoins dans l'optique d'une expérience normée ces détails-là seront à corriger.

4.5 Validations de l'outil

Au cours de la réalisation, des tests ont été mis en place afin de corriger les erreurs le plus tôt possible et de pouvoir les repérer plus facilement. Ces tests concernent aussi bien la programmation que les tests du capteur. Au niveau de la programmation, les tests majeurs ont concerné la rectification de la perspective, l'acquisition des données de la carte, leur traitement et leur visualisation. Au niveau du capteur, les tests ont concerné l'acquisition du champ magnétique et le circuit d'initialisation du capteur.

4.5.1 Programmation

4.5.1.1 Rectification de la perspective

Matériel : le programme et une image avec un quadrillage déformé par la perspective.

Ce test s'est basé sur un code de Chaianun [1] qui permet, connaissant la matrice de passage H, de construire une image rectifiée. Si les coins de l'étalon ont été correctement désignés, alors les lignes parallèles et perpendiculaires sur le sol doivent garder cette propriété dans l'image construite. Ce test à permis la rectification de problèmes tels que l'ordre des coordonnées donnée par MATLAB sur une image ((x,y) étant inversés). Ainsi, lorsque les points sont bien choisis sur l'image de départ 4.13, les droites perpendiculaires dans la réalité le sont dans l'image reconstruite 4.14. Malgré tout, l'image se déforme sur les bords de l'image.

FIGURE 4.13 – Image acquise déformée par la perspective

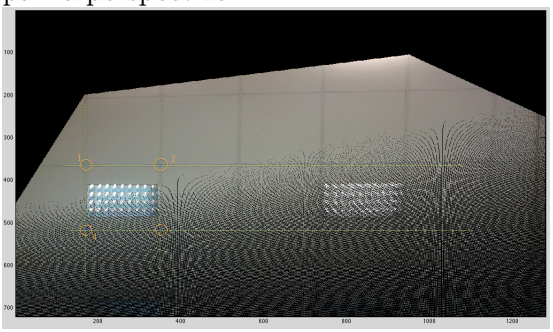

FIGURE 4.14 – Image reconstruite à partir de H

4.5.1.2 Coordonnées du capteur dans l'espace

Ce test n'a pas encore été mis en place.

Nécessite : Une pièce quasi vide et quadrillée, la webcam et le programme

Le but de ce test est d'estimer les erreurs de coordonnées lorsque le capteur est détecté par la webcam et que le programme calcule se position dans l'espace.

Des tests moins poussés ont été réalisés (avec une pièce non quadrillée) pour obtenir des coordonnées cohérentes avec le positionnement du capteur en vérifiant les erreurs de signe.

4.5.1.3 Acquisition des données

Matériel : Carte d'acquisition, programme, capteur ou alimentation stabilisée

Avec la carte d'acquisition branchée et MATLAB lancé, le but a été de vérifier que les entrées/sorties programmées étaient bien les bonnes et qu'un signal était reçu. Puis, avec l'alimentation stabilisée de branchée, d'obtenir la même tension en sortie que celle en entrée.

4.5.1.4 Traitement et visualisation des données

Un champ magnétique créé par un fil infini vertical est modélisé sur MATLAB. Il est ensuite traité par le programme. Ce test concerne la FFT, la sauvegarde et la visualisation des données. Le fil crée un champ selon l'axe $\vec{e_\theta}$ en coordonnées polaires, et sa norme ne dépend que de la distance au fil. La visualisation de ce champ est faite dans la figure 4.15.

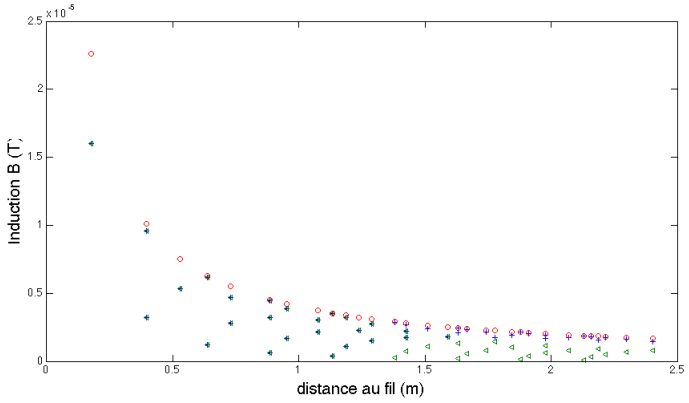

FIGURE 4.15 – Amplitude du champ créé par un fil en fonction de l'éloignement

A l'issue du traitement, il est visualisé à l'aide de l'interface graphique que nous avons programmée (voir Figure 4.16).

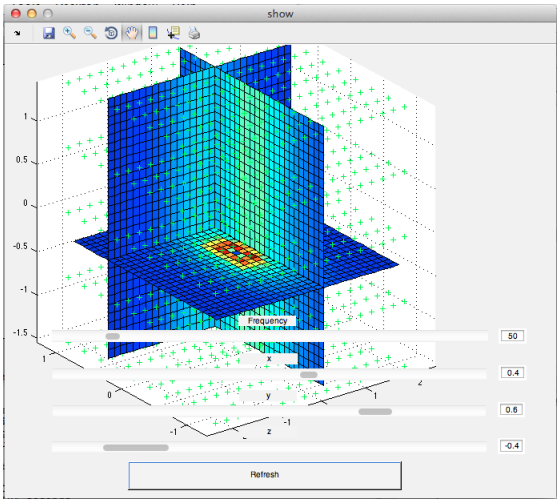

FIGURE 4.16 – Affichage du champ créé par un fil

Le nombre de points de "mesure" est important, pour prendre en compte le nombre plus important de mesure dans les zones de fort gradient. On voit que la norme du champ (couleur) forme un cercle, mais que le champ semble s'annuler au centre. Ceci vient de l'interpolation du champ qui se fait avant le passage à la valeur absolue. En effet, le champ selon x (ou y) est impair, donc l'interpolation donne le rendu de la figure 4.17 (le champ s'annule au niveau de la source). A l'issue de ce test, nous avons pu détecter entre autres : des problèmes dans notre filtrage numérique, causé par une définition de variable, une sauvegarde de données inutiles car la FFT d'un signal réel peut être entièrement reconstruite à partir de la moitié du vecteur. Ce test à permis en outre de rendre la visualisation plus efficace, rendant l'interpolation plus ciblée lors de la visualisation. En effet, l'interpolation est alors faite uniquement sur les plans affichés et plutôt que sur l'ensemble du volume.

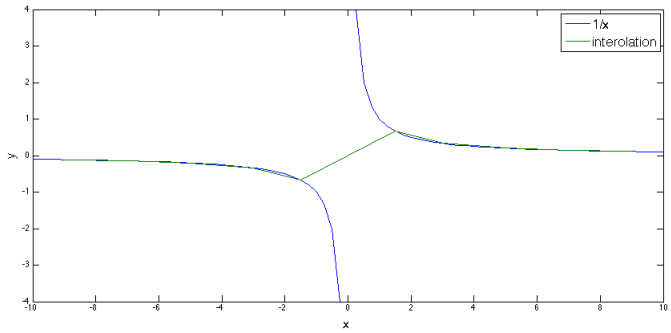

FIGURE 4.17 – résultat de l'interpolation d'une hyperbole

4.5.2 Électronique

4.5.2.1 Test du capteur

Matériel : Bobine, GBF, oscilloscope, capteur 3 axes.

Le capteur a été testé à proximité de la bobine, qui débitait un champ jusqu'à 200μT. L'oscilloscope nous a servi à visualiser le signal. Ce test à permis de vérifier que le capteur fonctionnait, et que son gain était bien de l'ordre de celui donné par le constructeur. Malheureusement, la présence d'un champ parasite important nous à empêché de le préciser.

4.5.2.2 Test du circuit d'initialisation

Matériel : Circuit, ohmmètre

L'ohmmètre à été utilisé pour vérifier que les connexions étaient bien présentes, et qu'il n'y avait pas de courts-circuits. Ce test nous à permis de détecter qu'un des composants n'avait pas le bon comportement, les branchements créaient des courts-circuits. La puce que nous avions commandée aurait dû avoir le schéma de fonctionnement visible sur la figure 4.18, mais les tests ont montré que les pattes G1, S2, D1 et D2 étaient reliées et que les pattes D1 et D1 ainsi que D2 et D2 n'étaient pas reliées. Nous avons alors vérifié sur celui de rechange, qui avait les mêmes caractéristiques. N'ayant plus de le temps d'en commander un autre et cette fonction n'étant nécessaire que pour une très grande précision, nous l'avons mise en attente.

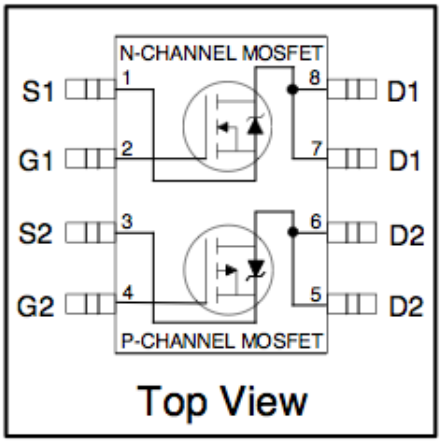

FIGURE 4.18 – schéma de principe du MOFSET

5. Perspectives d'améliorations

Au cours de la conception, nous nous avons gardé à l'esprit que notre outil était avant tout une preuve de concept. Ainsi, certains problèmes de précision n'ont pas été réglés. Nous regroupons dans cette partie les différentes améliorations possibles, déjà exprimées pour la plus part dans les parties concernées.

5.1 Programme

- Factorisation des variables globales : nous aurions pu placer la variable `color` dans `other_data`. De même, la valeur `z` n'est pas lourde ; elle n'a donc pas besoin d'être passée par référence.
- Pouvoir afficher le champ selon un axe ou sa norme (au choix).
- Permettre l'ajout de points hors de l'image
- Améliorer la précision la recherche du capteur dans l'image
- Ajouter des exemples d'utilisation pour clarifier la documentation.

5.2 Support

- Éliminer complètement l'utilisation de matériaux ferromagnétique telle que l'acier pour diminuer encore les possibles perturbations.
- Fabriquer le manchon sur mesure afin de minimiser au maximum les jeux et de pouvoir utiliser un système de maintien en position de la partie mobile plus efficace (actuellement la pince ne peut soutenir qu'un poids relativement faible).
- Utiliser un poteau un peu plus large pour réduire encore les possibilités de fléchissement (le diamètre actuel a été choisi sous la contrainte des matériaux disponible dans le commerce tel que le manchon).
- Améliorer l'isostatisme de la base en utilisant des plots sous la planche ou en utilisant une planche plus épaisse et lourde.

5.3 Outil général

- Évaluer l'erreur de mesure de position
- Évaluer l'erreur de mesure de champ

6. Conclusion

Arrivés à la fin de cette année et de ce projet, nous avons rempli la plus part des objectifs que nous nous étions fixés. Nous avons conçu un programme, pris en main un magnétomètre triaxes et construit un support mobile répondant à un cahier des charges relativement précis.

Malheureusement, suite à un problème de carte d'acquisition, nous n'avons pas pu réaliser de mesure finale, c'est à dire une cartographie complète d'une pièce contenant une source de rayonnements magnétiques.

En revanche, grâce aux différents tests réalisés, nous savons que chaque partie du projet, prise séparément, fonctionne. De plus, grâce à la documentation que nous avons rédigé, le projet pourra être repris et amélioré sur les points non traités, à savoir la précision des mesures.

Nous avons pu durant cette année expérimenter les aspects réjouissants (manipulations, créativité, autonomie, prises de responsabilité...) et moins réjouissante (gestion du temps, retards, états de l'art...) de la gestion de projet.

Nous espérons que l'outil aura une utilité et que nous aurons ainsi contribué, de près ou de loin, à l'avancée du projet de recherche de MM. Morel et Scorretti.

A. Gestion de projet

Dans cette partie vous trouverez les documents de gestion de projet, réalisés au début de l'année et présentés aux différents rendez-vous de pilotage.

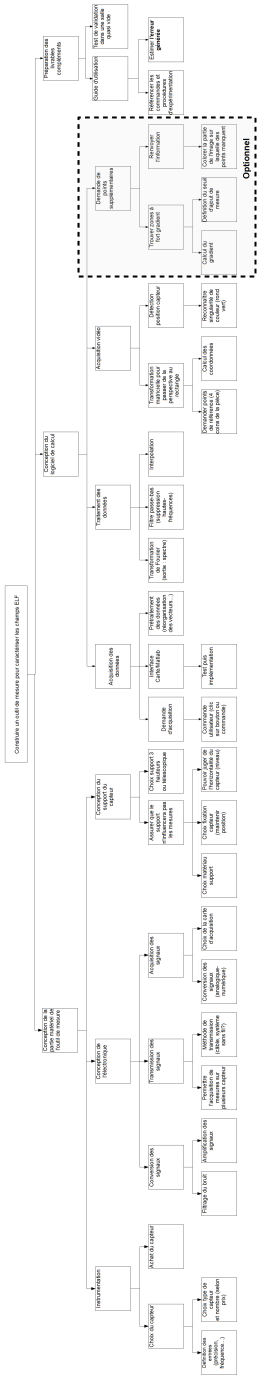

FIGURE A.1 – Diagramme de décomposition des tâches

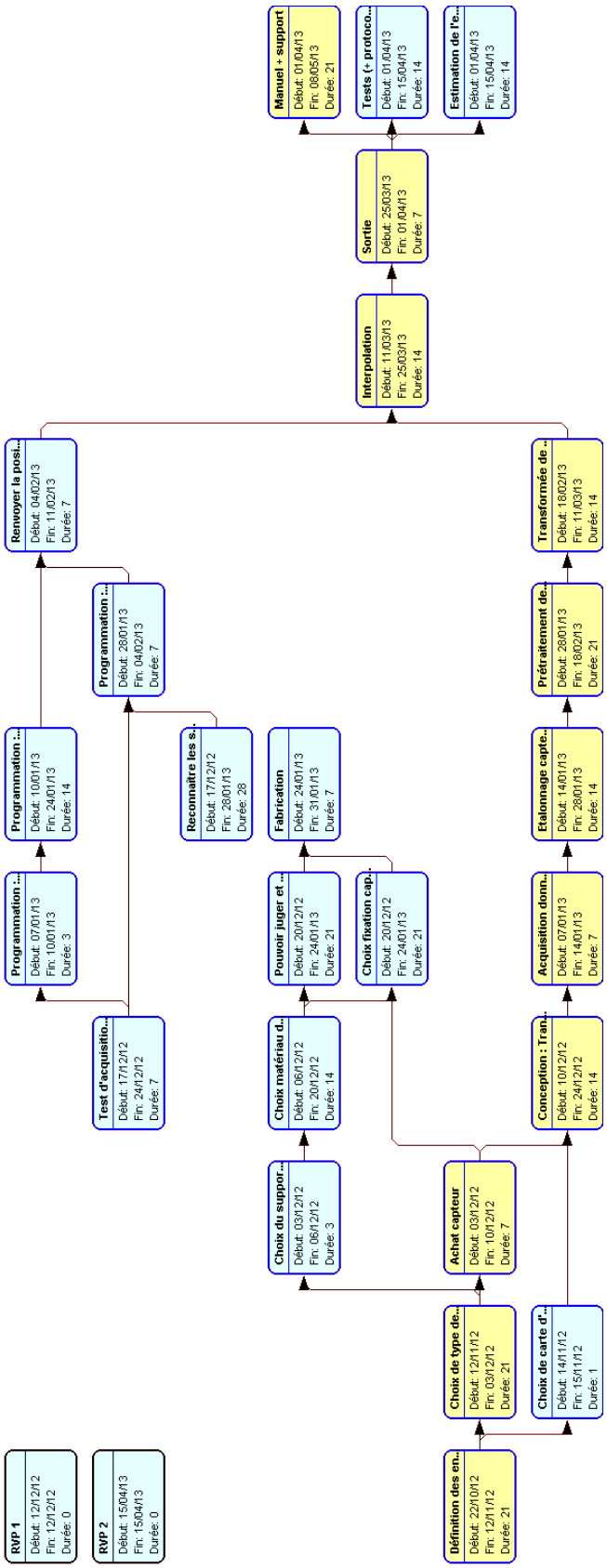

FIGURE A.2 – Diagramme Pert

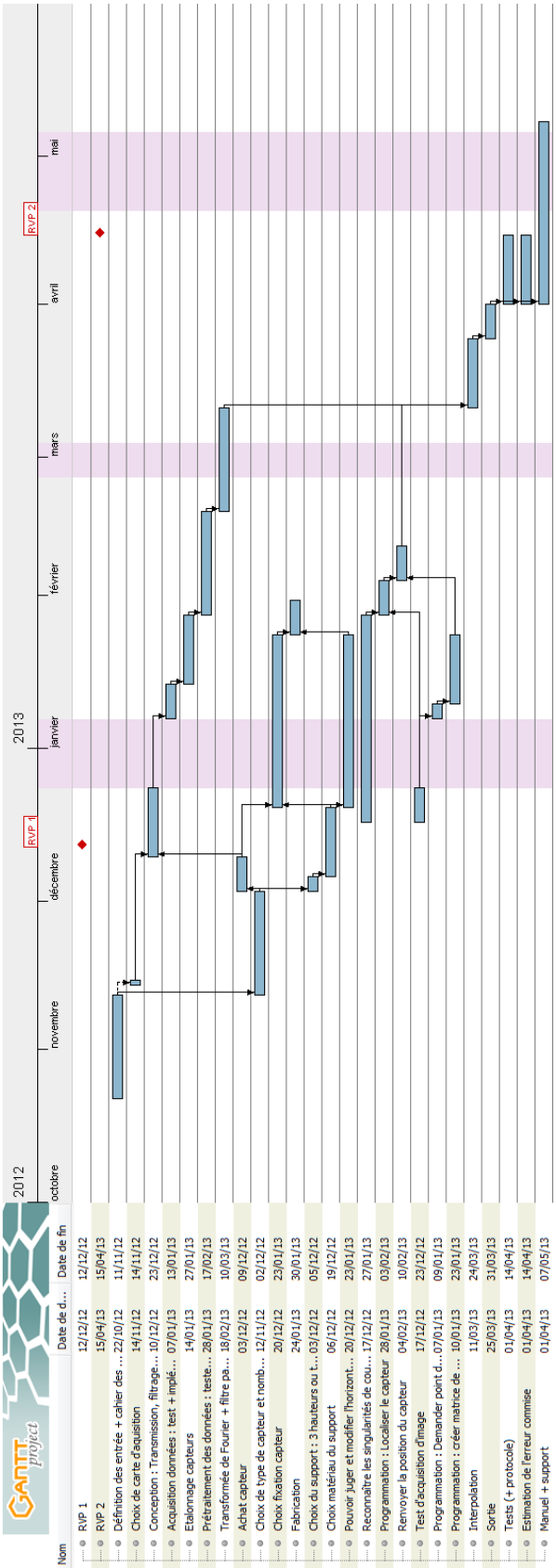

FIGURE A.3 – Diagramme Gantt

B. Documentation

Dans cette partie vous trouverez la documentation complète de notre outil. Celle-ci est évidemment disponible hors du présent rapport.

École Centrale de Lyon

Projet d'Études

26 juin 2013

PE 103 : Documentation de notre outil de mesure de champs magnétiques de basses fréquences

---

Olivier CHURLAUD, Sylvain HEMETTE, Yuechen LIU, Cédric OGER, Xiaoyi YANG

Table des matières

Introduction 1

I Documentation utilisateur 3

1 Matériel nécessaire 3

2 Utilisation 3

3 Contenu du fichier enregistré 4

II Documentation développeur 5

4 Fonctions accessibles en commande par l'utilisateur 5
4.1 Aide . . . 5
4.1.1 helpme.m . . . 5
4.2 Initialisations . . . 6
4.2.1 init.m . . . 6
4.2.2 init_variables.m . . . 6
4.2.3 init_daq.m . . . 6
4.2.4 init_webcam.m . . . 8
4.2.5 init_matrix_H.m . . . 8
4.2.6 disp_rebuilt . . . 9
4.3 Mesures et aperçus . . . 10
4.3.1 acquisition.m . . . 10
4.3.2 show.m . . . 10

5 Fonctions secondaires 12
5.1 Initialisation . . . 12
5.1.1 video_acquisition.m . . . 12
5.1.2 get_points.m . . . 13
5.1.3 create_rectangle.m . . . 13
5.1.4 get_transition_matrix.m . . . 14
5.1.5 ask4points.m . . . 14
5.2 Acquisition . . . 15
5.2.1 daq_acq.m . . . 15
5.2.2 fftdata.m . . . 15
5.2.3 get_delay.m . . . 15
5.2.4 XYcoords.m . . . 16
5.2.5 get_marker_position.m . . . 16
5.2.6 unpersp.m . . . 17
5.2.7 homothetie . . . 17
5.3 Affichage . . . 18
5.3.1 slice_me.m . . . 18
5.3.2 interpolation.m . . . 19
5.3.3 isconst.m . . . 20
5.3.4 reshape_for_slice.m . . . 20

6 Test sur du programme sur le champ théorique d'un fil infini 21
6.0.5 init_test.m . . . 21
6.0.6 acquisition_test.m . . . 21
6.0.7 DonneesFil.m . . . 22

Introduction

Dans le cadre d'un projet d'étude à l'École Central de Lyon, nous avons réalisé un outil permettant de cartographier le champ magnétique d'un espace, afin de permettre à nos commanditaires, Laurent Morel et Riccardo Scorretti, de pouvoir travailler sur son interaction avec l'organisme. Le but de cet outil est de simplifier et d'accélérer l'acquisition de données. En effet, pour avoir une cartographie précise et complète, le champ doit être mesuré en de nombreux points.

Nous avons donc implémenté une détection du capteur, permettant à l'expérimentateur de ne pas avoir à mesurer sa position dans l'espace. Une webcam prend une photographie de la pièce et le programme, préalablement calibré durant les initialisations, en déduira la position au sol du capteur. Ainsi seule la hauteur du capteur doit être donnée, ce qui est relativement simple si le support mobile (sur lequel est posé le capteur) est gradué.

Le programme réalise aussi les traitements attendus par nos commanditaires : après avoir fait subir une transformée de Fourier au signal et supprimé les fréquences hautes, il récupère les informations de phase et enregistre toutes ces données dans un fichier.

Cette documentation a pour but d'expliquer comment utiliser l'outil et le programme. Il a pour vocation d'être simple et succinct. Évidemment, une compréhension plus fine de notre travail permettrait d'utiliser l'outil dans des configurations différentes de ce que nous avons imaginé. Pour que cela soit possible et que notre travail soit réutilisable, nous avons rédigé une deuxième partie à destination d'un développeur ou d'un utilisateur plus curieux des rouages du programme.

Nous considèrons ici que l'utilisateur utilise des capteurs fonctionnels, qu'il connait et maîtrise. Ainsi, notre choix dans ce domaine n'a pas d'influence sur l'utilisation de l'outil.

Nous vous souhaitons une bonne lecture, tout en espérant que vos yeux sont affutés, et la boîte d'aspirines à proximité, car malgré la coloration syntaxique, il y a une probabilité non nulle que toutes ces lignes de code finissent par avoir raison de vous.

Le groupe de PE 103 :
Olivier CHURLAUD
Sylvain HEMETTE
Yuechen LIU
Cédric OGER
Xiaoyi YANG

4cm

Première partie

Documentation utilisateur

1 Matériel nécessaire

- Un ordinateur possédant MATLAB dans une version suffisamment récente,
- Une webcam avec une résolution raisonnable (fournie pour la preuve de concept),
- Une carte d'acquisition DAQ,
- Un magnétomètre 3 dimensions ou 3 capteurs placé en trièdre (capteur général, mobile),
- Un magnétomètre de référence (une dimension, fixe),
- Un support pour les magnétomètres, pour la webcam, dont la hauteur est connue,
- Le gabarit fourni permettant d'étalonner le programme pour déterminer les coordonnées du capteur dans la pièce (feuille A3),
- Le marqueur ou disque de couleur pour repérer le magnétomètre mobile.

2 Utilisation

1. Installer les drivers de la carte d'acquisition et de la webcam.
2. Brancher la webcam et la carte d'acquisition AVANT de lancer MATLAB.
3. Lancer MATLAB.
4. Se placer dans le dossier MagneticCartog.
5. Placer le capteur de référence dans le point que vous considérez comme le (0,0) de votre pièce.
6. Initialiser le matériel.

   ```
   >> init
   ```

   Suivez le programme en répondant aux diverses questions.
   Remarque : Si vous vous trompez dans une réponse ou dans un paramètre, finissez l'initialisation malgré tout puis réinitialisez le bloc erroné :

   ```
   >> init_variables
   ```

   pour les variables.

   ```
   >> init_DAQ
   ```

   pour la carte d'acquisition.

   ```
   >> init_webcam
   ```

   pour la webcam.

   ```
   >> init_matrix_H
   ```

   pour la création de la fonction qui supprime la perspective.
   Attention : Si vous lancez init_ webcam, il est impératif de relancer init_ matrix_ H !
7. Placez le magnétomètre de mesure à une hauteur connue z.
8. Lancez la mesure :

   ```
   >> acquisition(z)
   ```

9. Lorsque vous avez réalisé quelques mesures, vous pouvez vérifier à quoi ressemble le champ de votre pièce par :

   ```
   >> show
   ```

10. Une fois vos mesures terminées, fermez MATLAB, vos données sont déjà enregistrées dans le dossier local dans le fichier `data.m`. Pensez à le renommer !

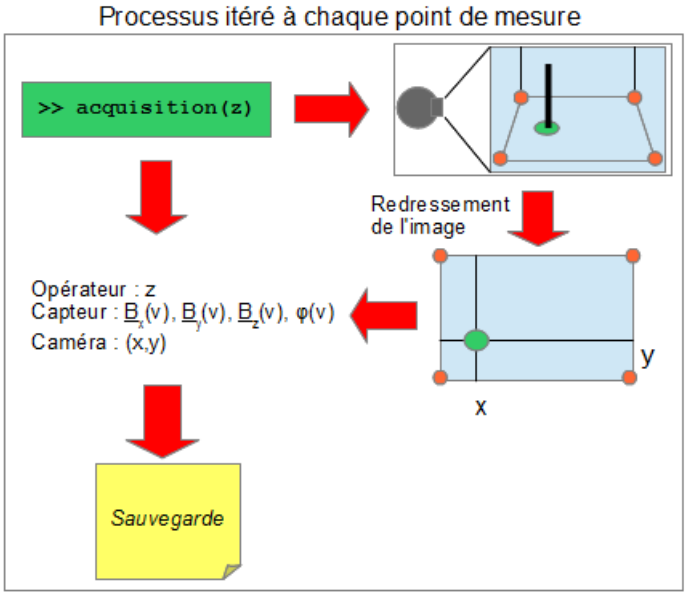

FIGURE 1 – Scénario d'utilisation du programme

3 Contenu du fichier enregistré

- data : une structure contenant
  - real_coordinates : une matrice contenant les cordonnées du capteur (`[x(:),y(:),z(:)]`)
  - nu_m est un vecteur linéaire contenant toutes les fréquences définies durant l'initialisation (`[`nu_1`(:)...`nu_N`(:)]`)
  - Bx_nu (respectivement By_nu, Bz_nu) : une matrice contenant pour chaque fréquence (en ligne) l'amplitude complexe du champ magnétique selon x (resptectivement y, z) (`[`Bx_{nu_1}`(:)...`Bx_{nu_N}`(:)]`)
  - delta : une matrice contenant les délais en seconde entre les signaux Bx (respectivement By, Bz) et la référence Br (`[`$delta_{B_x}$`(:),`$delta_{B_y}$`(:),`$delta_{B_z}$`(:)]`)
- other_data : une structure comprenant
  - pattern_coord_rebuilt : un vecteur contenant les coordonnées du coin supérieur gauche du gabarit dans le référentiel de la photo en pixels (`[x,y]`)
  - H : la matrice permettant de supprimer la perspective (matrice 3×3)
  - pattern_coord : un vecteur contenant les coordonnées du coin supérieur gauche du gabarit dans le référentiel de la pièce en mètres (`[x,y]`)
  - Image1 : l'image de la pièce prise lors de l'initialisation de la matrice H.
  - pattern_size : un vecteur contenant les dimensions (longueur $\times$ largeur, en mètres) du gabarit dans le référentiel de la pièce (`[L,l]`)
  - pattern_size_rebuilt : un vecteur contenant les dimensions (longueur $\times$ largeur, en pixels) du gabarit dans le référentiel de la photo (`[L,l]`)
  - dimensions : un vecteur contenant les dimensions de la pièce en mètres (`[l, L, h]`)
  - detector_raw_coord : une matrice contenant les coordonnées en pixel de disque de repérage dans le référentiel de la photo, avant tout traitement (`[x(:),y(:)]`)
  - Rate : un réel représentant la fréquence d'échantillonnage (`[`nu_e`]`)

Deuxième partie
Documentation développeur

4 Fonctions accessibles en commande par l'utilisateur

4.1 Aide

4.1.1 helpme.m

Récapitule l'ensemble des commandes disponibles.

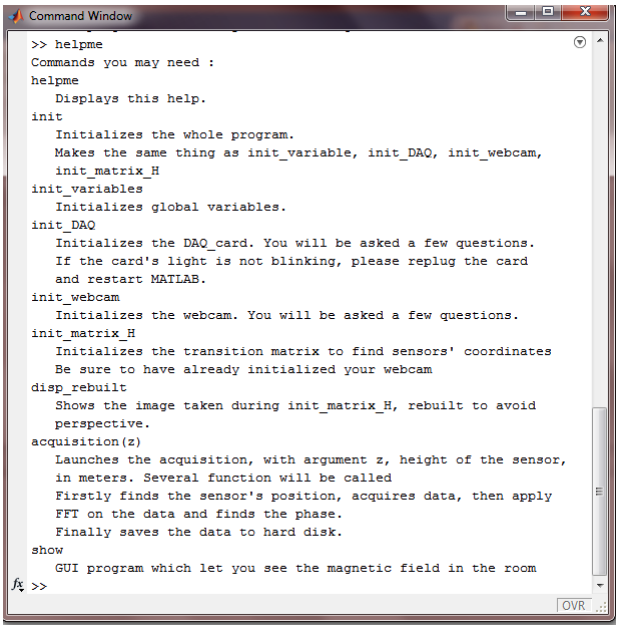

FIGURE 2 – Résultat de la commande `helpme`

```
1  disp('Commands you may need : ');
2  disp('helpme');
3  disp('   Displays this help.');
4  disp('init');
5  disp('   Initializes the whole program.');
6  disp('   Makes the same thing as init_variable, init_DAQ, init_webcam,');
7  disp('   init_matrix_H');
8  disp('init_variables');
9  disp('   Initializes global variables.');
10 disp('init_DAQ');
11 disp('   Initializes the DAQ_card. You will be asked a few questions.');
12 disp('   If the card''s light is not blinking, please replug the card');
13 disp('   and restart MATLAB.');
14 disp('init_webcam');
15 disp('   Initializes the webcam. You will be asked a few questions.');
16 disp('init_matrix_H');
17 disp('   Initializes the transition matrix to find sensors'' coordinates');
18 disp('   Be sure to have already initialized your webcam');
19 disp('disp_rebuilt');
20 disp('   Shows the image taken during init_matrix_H, rebuilt to avoid');
21 disp('   perspective.');
22 disp('acquisition(z)');
23 disp('   Launches the acquisition, with argument z, height of the sensor,');
24 disp('   in meters. Several function will be called');
25 disp('   Firstly finds the sensor''s position, acquires data, then apply');
26 disp('   FFT on the data and finds the phase.');
27 disp('   Finally saves the data to hard disk.');
28 disp('show');
29 disp('   GUI program which let you see the magnetic field in the room');
```

4.2 Initialisations

4.2.1 init.m

Un premier fichier (init.m) permet de tout initialiser. Si l'utilisateur en a besoin, il peut initialiser chaque bloc (DAQ, webcam, variables ou matrix_H) indépendamment.

```
disp('Welcome to your tool''s initialization. You will be asked a few questions...');

% Defines the path
projectpath = genpath(pwd);
addpath(projectpath);

% Defines the global variables
global other_data;
global data;
global filter_parameters;
global z;
global color;
global session;
global webcam;

% Initialization of the struct. and global variables
init_variables;

% Initialization of the acquisition card
init_DAQ;

%  Initialization of the webcam
init_webcam;

% Initialization of the transition matrix H
init_matrix_H;

disp('Initialization completed !');
```

4.2.2 init_variables.m

Permet d'intialiser les structures.

```
% Initialization of the structs and the color
global color data other_data;

% Itialization of the struct of field data
data = struct('real_coordinates',[],'nu_m',[],'Bx_nu',[],'By_nu',[],'Bz_nu',[],'delta',[]);

% Initialization of the struct of other data
other_data = struct('pattern_coord_rebuilt',[],'H',[],'pattern_coord',[],'Image1',[], '↩
    pattern_size', [.21,-.297], 'pattern_size_rebuilt', [],...
    'dimensions', [],'detector_raw_coord',[], 'Rate',[]);

% Initialization of the room dimensions
dimensions='';
while(isempty(dimensions) || ~strcmp('1  3',num2str(size(dimensions))))
    display('Enter the room dimensions [l L h]');
    dimensions =  str2num(strrep(input('where l the front length, L is the left length and h ↩
        the height of the room : ','s'),',','.'));
    if (isempty(dimensions))
        display('Room dimensions must be 3 numbers');
    end
end
other_data.dimensions = [zeros(1,length(dimensions));dimensions]';

% Initialization of the color of the marker
color = '';
while (~strcmp('red', color) && ~strcmp(color, 'green') && ~strcmp(color, 'blue'))
    color = input('Enter the color of your marker (blue/green/red) : ','s');
end

clear dimensions;

disp('Variables initialized !');
```

4.2.3 init_daq.m

Permet d'initialiser la DAQcard. Il faut bien veiller à ce que celle-ci soit déjà branchée avant l'ouverture de Matlab. Pour plus de simplicité, nous avons automatisé la plus part des réglages, hors le temps d'acquisition et la fréquence d'échantillonage.

```
% Initialization of the DAQ card
answer = '';
warning off;

global session data filter_parameters other_data;

% Checks whether the DAQ is plugged or not. If not, Matlab has to be restarted.
while (~strcmp(answer, 'yes') && ~strcmp(answer, 'Yes'))
    answer = input('Did you plug the DAQ card before opening Matlab and plug the 4 channels ? yes/[no] : ', 's');
    if isempty(answer)
        answer = 'no';
    end
end

if (strcmp(answer, 'Yes') || strcmp(answer, 'yes'))
% We know the name of the card that must be used : 'ni'
    session = daq.createSession('ni');
    d = daq.getDevices;
% If the card is plugged, the size is not 0 : we can take the id
    if (size(d) ~= 0)
        DAQ_device = d.ID;

        % The init_card should be placed here
%       session.addDigitalChannel(DAQ_device,'Port1/Line0:1','OutputOnly')
%
%       % creates the digital output object
%       session.outputSingleScan ([0,1]);
%       % resets the circuit
%       tic;
%       Timer=timer('Period',.01,'TimerFcn',@pulseInit,'ExecutionMode','fixedDelay',...
%           'TasksToExecute',50)
%       %Timer=timer('Period',1,'TimerFcn',@pulseInit,'ExecutionMode','fixedDelay',...
%       %   'TasksToExecute',50);
%       % links the lines to the pulse-making function
%
%       start(Timer)
%       % launches the clock
%       wait(Timer);
%       % waits for the dio to finish.
%       delete(Timer)
%       clear Timer
%         session.removeChannel(1:2);
%       toc;
        % deletes the Timer

% Here we define the 4 first channels
        for i=1:4
            channel = ['ai' num2str(i)];
            session.addAnalogInputChannel(DAQ_device, channel, 'Voltage');
        end
% Here we choose the rate of acquisition (by default 1000)
        rate = '';
        while (isempty(rate))
            rate = input('Choose the rate of operation in scans per second [1000] : ', 's');
            if isempty(rate)
                rate = '1000';
            end
            rate = str2num(rate);
        end
        session.Rate = rate;
% Here we choose the duration (by default 1 second)
        duration = '';
        while (isempty(duration))
            duration = input('Choose the duration in second [1] : ','s');
            if isempty(duration)
                duration = '1';
            end
            duration = str2num(duration);
        end

        session.DurationInSeconds = duration;

% Here we define the discrete frequency domain
        data.nu_m =0:1/session.DurationInSeconds:session.Rate/2-1/session.DurationInSeconds;
% And here we define the filter parameters struct
        filter_parameters = struct('Fs',session.Rate,'Rp',0.5,'Rs',20,'nu_pass',4/10*session.Rate,'nu_stop',1/2*session.Rate);
% 4/10 and 1/2 are here to verify the theorem of Shannon.
% We copy the rate in other_data (for load purposes)
        other_data.Rate = rate;
        disp(sprintf('Card initialized !'));

    else disp(sprintf('You didn''t plug the device. You may have to restart Matlab after plugging'));
    end
```

```
84 else disp(sprintf('You didn''t plug the device. You may have to restart Matlab after plugging↩
       ')); % Should be useless but...
85 end
86 % clears the unused variables
87 clear answer i d DAQ_device channel rate duration;
```

4.2.4 init_webcam.m

Permet d'initialiser la webcam. Veillez à ce que celle-ci soit connectée. Nous avons ici encore automatisé la plus part des réglages.

```
1  % Itinitalization of the webcam
2  liste_webcams = imaqhwinfo;
3  display(liste_webcams);
4
5  global webcam;
6
7  webcam_device = '0';
8
9  % Displays the available webcam and check if exists
10 while (webcam_device=='0')
11     while (~any(strcmp(liste_webcams.InstalledAdaptors, webcam_device)))
12         webcam_device = input('Enter name of the device to use (see above) without ''' : ', ↩
               's');
13         if(~any(strcmp(liste_webcams.InstalledAdaptors, webcam_device)))
14             display(liste_webcams);
15             display('');
16             display('The device you entered doesn''t exist.');
17             disp('If your webcam wasn''t detected, please relaunch MATLAB');
18         end
19     end
20
21     % Checks if available webcam can be used, or let the user choose another webcam
22     webcam_info = imaqhwinfo(webcam_device);
23     if (size(webcam_info.DeviceIDs)==[1 0])
24         disp('This webcam is not available, please choose another one.');
25
26         webcam_device='0';
27     end
28 end
29
30 % Selects the right device..
31 if (size(webcam_info.DeviceIDs) == 1) %...automatically if there is only one...
32     webcam_id = webcam_info.DeviceIDs{1,1};
33     temp=1;
34 else % ... or with the user's help if there are several
35     webcam_id = [];
36     display(webcam_info);
37     while isempty(webcam_id)
38         temp = input('Enter name of ''DeviceIDs'' : ');
39         temp = [webcam_info.DeviceIDs{:,:}] == temp ; % makes a binary vector out
40         %   of the commited search in hope of having less bugs.
41         %
42         webcam_id = webcam_info.DeviceIDs(temp); % checks if the ID is correct
43     end
44     webcam_id = webcam_id{1};
45 end
46
47 % Selects automatically the picture format
48 webcam_format = webcam_info.DeviceInfo(temp).DefaultFormat;
49
50 % Saves the webcam parameters in the struct
51 webcam = struct('device', webcam_device, 'id', webcam_id, 'format', webcam_format);
52
53 disp(sprintf('Webcam initialized !'));
54 disp(sprintf('Please put your webcam properly... Then you can close the window'));
55 vid = videoinput(webcam_device, webcam_id, webcam_format);
56 src = getselectedsource(vid);
57 preview(vid);
58 % Clears unused variables
59 clear liste_webcams webcam_info webcam_device webcam_id webcam_format temp vid;
```

4.2.5 init_matrix_H.m

Permet d'initialiser la suppression de la perspective.

```
1 % Initialization of the H matrix
2 global webcam other_data;
3 input('Please set the A3 paper pattern in portrait orientation','s')
```

```
% Saves the picture of the pattern
other_data.Image1 = video_acquisition(webcam);
% Asks the user the corners of the pattern
xy_pattern = get_points(other_data.Image1,4);

% Prepare the transition matrix H creation and the homotheties lengths
[xy_pattern_rebuilt,other_data.pattern_size_rebuilt,other_data.pattern_coord_rebuilt] = ↩
    create_rectangle(xy_pattern);
% Calculate H
other_data.H = get_transition_matrix(xy_pattern,xy_pattern_rebuilt);

% Asks the user the pattern upper left point (useful for the homothetie)
other_data.pattern_coord = ask4Point('pattern');

% Firstly check if everything else works well
% Let the user check if the calibration is well done
% disp_rebuilt(other_data.H,other_data.Image1);

% clears the unused variables
clear xy_pattern xy_pattern_rebuilt;

disp('init_matrix_H completed !');
```

4.2.6 disp_rebuilt

Permet de vérifier si la suppression de la perspective fonctionne bien.

```
function [ intArray ] = disp_rebuilt
% Displays the rebuilt image
% IN (global) :
%       none
% OUT :
%       intArray : image (RGB, m*n*3 matrix), redressed image (with a black
%       background.

global other_data
H = other_data.H;
Image = other_data.Image1;

[M,N,~] = size(Image);

tmp = zeros(2,M*N);
for i = 1:M
    for j = 1:N
        tmp(1,(N*(i-1))+j) = j; % saves the column
        tmp(2,(N*(i-1))+j) = i; % saves the line
    end
end
 %%
% converts [tmp] size 2 x M*N  to be [tmp]  size 3 x M*N by pad [1] size 1xM*N at the 3rd row
% applies X2 = H.X1
mapped = H * [ tmp ; ones(1,M*N) ]; % applies H to each pixel

% normalize [X2]
X2 = mapped(1:2,:) ./ repmat(mapped(3,:),2,1); % normalizes by the homogeneous coordinates


% map [X2] size 3xM*N back to [output] size MxNx3
output = zeros(M,N,3);
 %%
for i = 1:M
    for j = 1:N

        %%
        col = round(X2(1,(N*(i-1))+j)); % collects the abscissa of each rebuilt pixel
        row = round(X2(2,(N*(i-1))+j)); % collects the ordinate of each rebuilt pixel

        if 0 < row && row < M && 0 < col && col < N
%             disp( 'in range' )
            output(row,col,1) = Image(i,j,1); % gives back each pixel its color
            output(row,col,2) = Image(i,j,2);
            output(row,col,3) = Image(i,j,3);
        else
%             disp('out range')
        end
        %%
    end
end

intArray = uint8(output); % displays the picture in RGB
figure,image(intArray);
end
```

4.3 Mesures et aperçus

4.3.1 acquisition.m

Réalise l'acquisition, le traitement (fft, filtrage, calcul des champs), et la sauvegarde des donnéees

```
function acquisition(z0)
% Void function (no out parameters) which makes the acquisitions of the positions of the detector
% and the values magnetic field, applies fft, finds the phase and saves the datas
% IN :
%       z0 : double, height of the detector
% OUT :
%       none

global data other_data  webcam session z;
warning off
z = z0;

% Finds the x,y position (in pixels) of the detector, thanks to the webcam
[x,y] = XYcoordinates(webcam);
[n,~] = size(other_data.detector_raw_coord);
n=n+1;

% Applies the transition matrix (H) to the coordinates of the selected point
temp_rebuilt_coords=unpersp(other_data.H,[x;y]);
% Converts the coordinates in pixels to meters
xy=homothetie( temp_rebuilt_coords, other_data.pattern_size_rebuilt, other_data.pattern_size, other_data.pattern_coord, other_data.pattern_coord_rebuilt)';

% Launches acquisition of the magnetic field values
[Bx,By,Bz,Br] = daq_acq(session);


% Applies fft to the acquiered values

% Get the delay (FORMULE QUI LIE LE DELAI A LA PHASE ICI ) + CORRECTION DE LA SUITE
[delta_x,delta_y,delta_z] = get_delay(Bx,By,Bz,Br,session.Rate);
[Bx,By,Bz] = fftdata([Bx,By,Bz]);
[n,~] = size(data.Bx_nu);
n=n+1;

% Saves the  coordinates in the struc datas
other_data.detector_raw_coord(n,:) = [x,y];
data.real_coordinates(n,:)=[xy,z];
data.delta(n,:) = [delta_x,delta_y,delta_z];
data.Bx_nu(n,:) = Bx;
data.By_nu(n,:) = By;
data.Bz_nu(n,:) = Bz;

% saves de data
if(exist('data','file')) save('data','data','other_data', '-append');
else save('data','data','other_data');


% Keyword to add data to an existing file. For MAT-files, -append adds new
% variables to the file or replaces the saved values of existing variables
% with values in the workspace. For ASCII files, -append adds data to the
% end of the file.
end
```

4.3.2 show.m

Il s'agit d'un .fig créé grâce au GUIde de MATLAB et du fichier .m complété par nos soins.

```
function varargout = show(varargin)
% SHOW MATLAB code for show.fig
%      SHOW, by itself, creates a new SHOW or raises the existing
%      singleton*.
%
%      H = SHOW returns the handle to a new SHOW or the handle to
%      the existing singleton*.
%
%      SHOW('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SHOW.M with the given input arguments.
%
%      SHOW('Property','Value',...) creates a new SHOW or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before show_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to show_OpeningFcn via varargin.
%
```

```
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help show

% Last Modified by GUIDE v2.5 20-Mar-2013 16:13:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @show_OpeningFcn, ...
                   'gui_OutputFcn',  @show_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
end

% --- Executes just before show is made visible.
function show_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to show (see VARARGIN)

% Choose default command line output for show
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes show wait for user response (see UIRESUME)
% uiwait(handles.figure1);
end

% --- Outputs from this function are returned to the command line.
function varargout = show_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
end
```

La partie qui suit est identique pour les sliders selon x, y, z ou fréquences. Les répétitions ont été retirées. Si vous voulez avoir le code complet, il est dans les annexes.

```
% --- Executes on slider movement.
function slider_x_Callback(hObject, eventdata, handles)
% hObject    handle to slider_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% sets the text area to the slider value
x=num2str(get(hObject,'Value'));
set(handles.edit_x,'String',x);
end

% --- Executes during object creation, after setting all properties.
function slider_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

global other_data;
% sets the max and min values
set(hObject,'Min',other_data.dimensions(1,1));
set(hObject,'Max',other_data.dimensions(1,2));
% sets the slider step
set(hObject,'SliderStep', [0.1/other_data.dimensions(1,2),0.1/other_data.dimensions(1,2)])
```

```
26
27 if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
28     set(hObject,'BackgroundColor',[.9 .9 .9]);
29 end
30 guidata(hObject, handles);
31 end
32
33 function edit_x_Callback(hObject, eventdata, handles)
34 % hObject    handle to edit_x (see GCBO)
35 % eventdata  reserved - to be defined in a future version of MATLAB
36 % handles    structure with handles and user data (see GUIDATA)
37
38 % Hints: get(hObject,'String') returns contents of edit_x as text
39 x=str2double(get(hObject,'String'));
40 % checks if text value is a number
41 if isnan(x)
42     set(hObject, 'String', 0);
43     errordlg('Input must be a number','Error');
44 end
45
46 set(handles.slider_x,'Value',x);
47
48 handles.x = x;
49 end
50
51 % --- Executes during object creation, after setting all properties.
52 function edit_x_CreateFcn(hObject, eventdata, handles)
53 % hObject    handle to edit_x (see GCBO)
54 % eventdata  reserved - to be defined in a future version of MATLAB
55 % handles    empty - handles not created until after all CreateFcns called
56
57 % Hint: edit controls usually have a white background on Windows.
58 %       See ISPC and COMPUTER.
59 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
60     set(hObject,'BackgroundColor','white');
61 end
62 end
```

Ce qui suit n'est pas répété.

```
1  % --- Executes on button press in refresh.
2  function refresh_Callback(hObject, eventdata, handles)
3  % hObject    handle to refresh (see GCBO)
4  % eventdata  reserved - to be defined in a future version of MATLAB
5  % handles    structure with handles and user data (see GUIDATA)
6  global data;
7
8  axes(handles.axes1);
9  cla;
10
11 frequency = get(handles.slider_frequency,'Value');
12 % Gets the value of frequency chosen
13 xslice = get(handles.slider_x,'Value'); yslice = get(handles.slider_y,'Value');
14 zslice = get(handles.slider_z,'Value');
15
16 % Gets the coordinates of the slices to make.
17
18 slice_me(xslice,yslice,zslice,frequency)
19 % interpolates the data in a grid, and plots it as a slice plot. Shows the
20 % locations of the acquired points.
21 end
```

5 Fonctions secondaires

5.1 Initialisation

5.1.1 video_acquisition.m

Prend un instantanné grâce à la webcam.

```
1  function [ Image ] = video_acquisition(webcam)
2  % IN :
3  %       webcam : struct, contains webcam parameters
4  % OUT :
5  %       Image : image taken.
6
7  % configurates the webcam
8  vid = videoinput(webcam.device, webcam.id, webcam.format);
9  set(vid,'ReturnedColorSpace','rgb')
10 % number of frames = 1 (picture)
```

```
vid.FramesPerTrigger = 1;
start(vid);

% extracts image
Image = getdata(vid);

end
```

5.1.2 get_points.m

Demande à l'utilisateur de sélectionner des points sur une image.

```
function [ XYimage ] = get_points( Image,n )
% Asks to select the pattern's corners in the picture
% IN :
%       Image : image(RGB, n*m*3 matrix), image on which the positions of
%       the detector or the corners of the pattern will be selected.
%       n : integer, number of points to select on the figure.
%           During the init, n=4 is necessary to create the transition
%           matrix
%           Afterwards, n=1 can be used to have a full screen figure on
%           which the detector can be pointed out.
% OUT :
%       XYimage : 2 by n integer matrix, positions of the selected points.

scrsz = get(0,'ScreenSize');
difforme=figure('Position',[10 -10 scrsz(3) scrsz(4)],'DockControls','on','menubar','none');
image(Image);
% plots the Image in full screen.

hold on
XYimage=zeros(2,n);

for i = 1:n
    XYimage(:,i)=ginput(1)'; % asks the user the i-th corner
    plot(XYimage(1,i),XYimage(2,i),'+'); % determines the selected corner.
end
commandwindow; % comes back to workspace
input('If one of your point seems misplaced, please run again ''init_matrix_H'' after the end of the script.');
close(difforme);
end
```

5.1.3 create_rectangle.m

Ordonne les points entrés pour en faire un rectangle dont on donnera les coordonnées des sommets et les longueurs.

```
function [ XYr,DistXY, plate_coord_rebuilt ] = create_rectangle( XY )
% Automatically creates a rectangle with the entered coordinates
% IN :
%       XY : 2 by 4 integer matrix, corresponds to the positions of the
%       pattern's corner
% OUT :
%       XYr : 2 by 4 real matrix, rebuilt coordinates of the input points
%       DistXY : 2-real-vector, size of the rectangle created
%       plate_coord_rebuilt : 2-real-vector, coordinates of the upper-left
%       corner of the pattern.

if size(XY) == [4,2]
    XY=XY';
end

[unused,inds]=sort(XY,2);
x1=(unused(1,1)+unused(1,2))/2; % create the small and high values
x2=(unused(1,3)+unused(1,4))/2;
y1=(unused(2,1)+unused(2,2))/2;
y2=(unused(2,3)+unused(2,4))/2;

XYr(1,inds(1,:))=[x1,x1,x2,x2]; % sort the values in a table in fonction of
XYr(2,inds(2,:))=[y1,y1,y2,y2]; % the original points


DistXY=[x2-x1;y2-y1];
plate_coord_rebuilt=[x1,y1];

end
```

5.1.4 get_transition_matrix.m

Construit la matrice H de passage permettant de supprimer la perspective.

```
function [ H ] = get_transition_matrix( posX,posRX )
% IN :
%       posX : 2 by 4 integer matrix, positions in pixels of the corners of
%              the pattern in the altered image [x1:x4]
%                                               [y1:y4]
%       posRX : 2 by 4 integer matrix, positions of the corners of
%              the pattern in pixels in the straighten up image [x1:x4]
%                                                               [y1:y4]
% OUT :
%       H : 3 by 3 real matrix, transition matrix for the homogeneous
%       coordinates

A = [];
for i = 1:4

  x1 = [posX(:,i);1];      % Get ith point in original image in homogeneous coords
  x2x = posRX(1,i);        % Get x coordinate of destination point
  x2y = posRX(2,i);        % Get y coordinate of destination point
  ThisA = [ -x1', 0, 0, 0, x2x * x1' ;
            0, 0, 0, -x1', x2y * x1' ];
  A = [ A ; ThisA ];

end
% This loop is a resolution of a linear system. It is then reshaped into a
% 3 by 3 matrix.


H = reshape(null(A),3,3)';

end
```

5.1.5 ask4points.m

Demande à l'utilisateur de sélectionner un certain nombre de points (1 si l'argument est détector, 4, si c'est le gabarit)

```
function [ XY_real ] = ask4Point(detector_or_pattern)
% IN :
%       detector_or_pattern : string, 'pattern' or 'detector'
% OUT :
%       XY_real : vector [x,y], coordinates of the selected point.
%   ask for the coordinates of the detector the pattern

% define X and Y
X = '';
Y = '';

% Switch : whether the text displayed is about the detector or the benchmark
switch detector_or_pattern
    case 'detector'
        while (isempty(X) || isempty(Y)) % check if it's empty or NaN
            X = str2num(strrep(input('Abscissa of the detector\nx = ','s'),',','.'));
            Y = str2num(strrep(input('Ordinate of the detector\ny = ','s'),',','.'));
            if (isempty(X) || isempty(Y)) % if empty or NaN, displays :
                disp('Numeric values expected...');
            end
        end

    case 'pattern'
        while (isempty(X) || isempty(Y)) % check if it's empty or NaN
            X = str2num(strrep(input('Abscissa of the upper left corner of the benchmark\nx =↩
                ','s'),',','.'));
            Y = str2num(strrep(input('Ordinate of the upper left corner of the benchmark\ny =↩
                ','s'),',','.'));
            if (isempty(X) || isempty(Y)) % if empty or NaN, displays :
                disp('Numeric values expected...');
            end
        end
end

XY_real = [X,Y]
end
```

5.2 Acquisition

5.2.1 daq_acq.m

Lance l'acquisition sur la DAQ.

```
function [Bx,By,Bz,Br] = daq_acq(session)
% IN :
%       session : struct, contains DAQ session
% OUT :
%       Bx,By,Bz,Br : n-double vectors, acquired field

% Starts acquisition (with options declared in init_DAQ)
[B,~] = session.startForeground;

% Separates the 3 components and the reference
Bx = B(:,1);
By = B(:,2);
Bz = B(:,3);
Br = B(:,4);

end
```

5.2.2 fftdata.m

Réalise les traitements sur les données obtenues : fenêtrage, transformée de Fourier discrète, filtrage.

```
function  [Bx,By,Bz] = fftdata_test( data_to_compute )
% Returns the fast Fourier transform of the data given.
% IN :
%       data_to_compute : n*3 real matrix, temporal values of the measured
%       field on the three axis (x,y,z)
% OUT :
%       Bx,By,Bz : n/2 real vectors, spectrum of the acquired signals.

global filter_parameters;

Fs = filter_parameters.Fs;
nu_stop = filter_parameters.nu_stop;  % After this frequence, the signal is
% negated.

n=length(data_to_compute);

%% Frequency analysis

coef_ham=hanning(n)/Fs; % The division by Fs is because we digitaly analyse
% an analog signal
Bx_ham=coef_ham(:).*data_to_compute(:,1);            % x-component
By_ham=coef_ham(:).*data_to_compute(:,2);            % y-component
Bz_ham=coef_ham(:).*data_to_compute(:,3);            % z-component
% Application of an Hanning window to provide a better fourier transform

Bx_freq_unshift=fft(Bx_ham);              % x-component
By_freq_unshift=fft(By_ham);              % y-component
Bz_freq_unshift=fft(Bz_ham);              % z-component
% Generates the fast fourier transform of the acquired signals

filter=filter_parameters.nu_m<=nu_stop;
Bx = Bx_freq_unshift(1:n/2).*filter(:);
By = By_freq_unshift(1:n/2).*filter(:);
Bz = Bz_freq_unshift(1:n/2).*filter(:);
% We apply an ideal filter, nu_m's length must be half of the acquired
% signal's length. This way we save only half the data.

end
```

5.2.3 get_delay.m

Permet d'obtenir le délai (en seconde) des signaux, grâce à la fonction AlignRefSignals que M. Scorretti nous a procuré. En multipliant par la fréquence considérée, on optient la phase du signal.

```
function [delta_x,delta_y,delta_z] = get_delay(Bx,By,Bz,Br,rate)
% Gives the delay of the 3-axis field relative to the reference.
% IN :
%       Bx, By, Bz, Br : n-vectors, acquired magnetic fields
%       rate : integer, rate of acquisition
% OUT :
%       delta_x, delta_y, delta_z : real, delay of Bx, By, Bz relative to
```

```
%       Br

% calculates the delay for each signal

[delta_x, ~, ~, ~] = AlignRefSignals (Bx(:), Br(:), rate);
[delta_y, ~, ~, ~] = AlignRefSignals (By(:), Br(:), rate);
[delta_z, ~, ~, ~] = AlignRefSignals (Bz(:), Br(:), rate);
end
```

5.2.4 XYcoords.m

Prend un instantanné grâce à la webcam et appelle la fonction `get_marker_position`

```
function [x,y] = XYcoordinates(webcam)
% IN :
%       webcam : struct, contains webcam parameters
% OUT :
%       x,y : double, marker coordinates in pixels in the picture

% takes a picture
image_marker = video_acquisition(webcam);

% calls the function which find the marker in the picture
[x,y] = get_marker_position(image_marker);
end
```

5.2.5 get_marker_position.m

Recherche un disque de la couleur définie dans les initialisations. Afin de réduire les calculs, l'utilisateur doit cliquer aux alentours du disque. Ainsi on ne cherche le disque que dans une fenêtre réduite autour du point désigné par l'utilisateur.

```
function [x,y] = get_marker_position(image_obj)
% Returns x,y components of the detector position
% IN :
%       image_obj : picture to be studied
% OUT :
%       x,y : 2-real vectors,  x,y components of the detector position

% defines the colour chosen during initialisations
global color;

% definition of the half-width of the window around the point clicked below
width = 30;

answer = 'No';

while (strcmp('No' ,answer))

    % waits the user's click on the picture
    scrsz = get(0,'ScreenSize');
    h=figure('Position',[10 -10 scrsz(3) scrsz(4)],'DockControls','on','menubar','none');
    image(image_obj);
    [x,y] = ginput(1);

    % creates a window around the click to reduce computations
        % precautions for the case where the click is near the boarder of the picture
        [max_y,max_x, ~] = size(image_obj);
        x0 = max(1,x-width);
        x1 = min(x+width,max_x - 1);
        y0 = max(1,y-width);
        y1 = min(y+width,max_y - 1);

        % creates a the smaller image (in window designed above)
        smaller_image = image_obj(y0:y1,x0:x1,:);

    % suppresses other colours than the selected one
        % changes the colour chosen into the name of the raw to keep
        switch color
            case 'red'
                valeur_couleur = 2;
            case 'blue'
                valeur_couleur = 2;
            case 'green'
                valeur_couleur = 1;
        end

        % changes the coloured picture  in a B&W picture where black is the colour of the raw↩
             we kept, white its opposite
```

```
47          computed_object = smaller_image(:,:,valeur_couleur);
48      % contrast enhancement
49      image_objet=imadjust(computed_object);
50      % suppresses the parts of the image where the colour value is below 10
51      binary_image = image_objet < 10;
52
53      % looks for the center of circular objects
54      s = regionprops(binary_image, 'centroid');
55      object_center = cat(1, s.Centroid);
56      % finds the center coordinates in the large original picture
57      x = x0+object_center(1,1);
58      y = y0+object_center(1,2);
59
60      hold on; plot(x,y,'c*');
61      answer = questdlg('Is it good enough ?', 'Question', 'Yes', 'No', 'Yes');
62
63      if(ishandle(h))
64          close(h);
65      end
66  end
```

5.2.6 unpersp.m

Applique la matrice H, calculée dans `init_matrix_H`, à un couple de coordonnées.

```
1   function [ pos ] = unpersp( H,xy )
2   % Gives the rebuilt coordinates of a point in the acquired image
3   % IN :
4   %       H : 3 by 3 real matrix, transition matrix for the homogeneous
5   %       coordinates
6   %       xy : integer vector or 2 by n integer matrix, positions of the
7   %       points in the acquired image. [x1:xn]
8   %                                     [y1:yn]
9   % OUT :
10  %       pos : real vector or 2 by n real matrix, positions of the points in
11  %       the rebuilt image.
12
13
14  % p = number of points to rebuilt
15  [~,p]=size(xy);
16
17  % applies H to homogenous coordinates of acquired points
18  Hpos=H*[xy;ones(1,p)];
19
20  % normalizes the coordinates to have a 2 by n matrix of n rebuilt points
21  pos=Hpos(1:2,:) ./ repmat(Hpos(3,:),2,1);
22
23  end
```

5.2.7 homothetie

Applique l'homothétie définie dans `init_matrix_H`, à un couple de coordonnées afin de les transformer en coordonnées métriques, dans le bon référentiel.

```
1   function [ coord_in_meters ] = homothetie( XY_rebuilt, pattern_size_rebuilt, pattern_size, ↩
        pattern_coord, pattern_coord_rebuilt)
2   % IN:
3   %       XY_rebuilt : vector or matrix (2,n), coordinates of a bench of points in the rebuilt ↩
        picture
4   %       schape [x1,x2,x3;
5   %               y1,y2,y3];
6   %       pattern_coord_rebuilt : vector, coordinates of the points of the pattern (already ↩
        refered) in the rebuilt picture
7   %       pattern_size_rebuilt : vector, size of the pattern in pixels in the rebuilt picture
8   %       pattern_coord : vector, coordonates of the same point in the room reference
9   %       pattern_size : vector, size of the pattern in the room reference
10  % OUT:
11  %       coord_in_meters = coordonnates of the asked point in the room reference
12
13  [~,p] = size(XY_rebuilt);
14
15  A = repmat([pattern_size(1)./pattern_size_rebuilt(1);pattern_size(2)./pattern_size_rebuilt(2)↩
        ],1,p);
16
17  % to be able to apply the homothetie to each vector
18  B = repmat([pattern_coord_rebuilt(1) ; pattern_coord_rebuilt(2)] , 1 , p); % affine component↩
        to be removed (in the picture)
19  C = repmat([pattern_coord(1) ; pattern_coord(2)],1,p); % affine component to be removed (in ↩
        the picture) (in the room)
```

```
20
21  coord_in_meters = (XY_rebuilt - B).*A + C;
22  end
```

5.3 Affichage

5.3.1 slice_me.m

```
1   function slice_me( xs,ys,zs,freq )
2   % Creates the slice-plot of the data for the frequency freq on the xs, ys,
3   % zs planes.
4
5
6   global dimensions data;
7
8   nb_points = 30; % number of points in each direction
9   xi = xs-(dimensions(1,2)-dimensions(1,1)):(dimensions(1,2)-dimensions(1,1))/nb_points:↩
        dimensions(1,2);
10  xi = xi(xi>dimensions(1,1));
11  % creation of the 30-point vector on the x-axis, where xs appears
12
13  yi = ys+dimensions(2,1)-dimensions(2,2):(dimensions(2,2)-dimensions(2,1))/nb_points:↩
        dimensions(2,2);
14  yi = yi(yi>dimensions(2,1));
15  % creation of the 30-point vector on the y-axis, where ys appears
16
17  constant=isconstant(data.real_coordinates(:,3));
18  % indicates if the data is a plane or a volume. This has an importance when
19  % the data is interpolated.
20  if ~isnan(constant)
21      zs = constant;
22      zi = zs-1:zs+1;
23      % if the data is on a plane, the number of points is
24      % reduced to  minimum on the z-axis, to enable the slice function.
25  else
26      zi = zs+dimensions(3,1)-dimensions(3,2):(dimensions(3,2)-dimensions(3,1))/nb_points:↩
            dimensions(3,2);
27      zi = zi(zi>dimensions(3,1));
28      % if the data is acquired in a volume, a '3d' interpolation will take
29      % place, with a 30-point vector on the z-axis.
30  end
31
32  if isnan(constant)
33      % if the data is acquired in a volume :
34      [Bx1,By1,Bz1,gridx]=interpolation(xs,yi,zi,freq);
35      [Bx2,By2,Bz2,gridy]=interpolation(xi,ys,zi,freq);
36      % the data is interpolated on the x=xs plane and the y=ys plane to keep
37      % calculations to a minimum.
38
39      B1=sqrt( abs(Bx1).^2 + abs(By1).^2 + abs(Bz1).^2 );
40      B2=sqrt( abs(Bx2).^2 + abs(By2).^2 + abs(Bz2).^2 );
41      % This seems to be bullshit.
42      % A euclidian norm is used to calculate the amplitude of the signal
43      % here the data is a vector
44
45      [Xx,~,~,Vx]=reshape_for_slice(gridx,B1 ,xs,yi,zi, freq);
46      [~,Yy,~,Vy]=reshape_for_slice(gridy,B2 ,xi,ys,zi,freq);
47      % The data is transformed from vectors to 2d grids
48
49  end
50
51
52  [Bx3,By3,Bz3,gridz] = interpolation(xi,yi,zs,freq);
53  B3 = sqrt( abs(Bx3).^2 + abs(By3).^2 + abs(Bz3).^2 );
54  [~,~,Zz,Vz] = reshape_for_slice(gridz,B3 ,xi,yi,zs,freq);
55  % The same is done on the z altitude.
56
57
58  % creation of 3d grids usable in slice
59  [Xf,Yf,Zf]=meshgrid(xi,yi,zi);
60  % the coordinates grid are created here
61
62  V=zeros(length(yi),length(xi),length(zi));
63  % the creation of the amplitude grid is created here, and fulled later.
64
65  if isnan(constant)
66      % if the data is acquired in a volume :
67      XX=repmat(Xx,[1,length(xi),1]);
68      YY=repmat(Yy,[length(yi),1,1]);
69      % the 2d matrix of data created before are replicated to create 3d
70      % matrixes
71
72      V(abs(XX-Xf)<10*eps)=Vx(:,:);
```

```
    V(abs(YY-Yf)<10*eps)=Vy(:,:);
    % The data is copied where the coordinates of the interpolated points
    % correspond to the meshgrid. (2 planes of non-zeros will appear in the
    % amplitude matrix, where x=xs and y=ys)
end

ZZ=repmat(Zz,[1,1,length(zi)]);
V(abs(ZZ-Zf)<10*eps)=Vz(:,:);
% the same is done with the z=zs plane.


slice(Xf,Yf,Zf,V,xs,ys,zs);
% this new data is then ploted. Because xs,ys,zs exist in the Xf,Yf,Zf
% matrixes, the data will not be interpolated. The value on the planes are
% given by V.
hold on;
plot3(data.real_coordinates(:,1),data.real_coordinates(:,2),data.real_coordinates(:,3),'g+');
% The acquisition points are shown on the 3d plot.
hold off

end
```

5.3.2 interpolation.m

```
function [interpolated_field_x,interpolated_field_y,interpolated_field_z, interpolation_grid,↩
    are_NaNs ] = interpolation( X,Y,Z,frequencies )
% Gives the interpolation of Bx By Bz on a grid formed by the
% X,Y,Z,frequency vectors
%
% Important : the data must create a real volume or create a real polygon
% on a z=cte plane for this function to work. Else you will have an error
% saying 'The data is degenerate in at least one dimension - ND set of
% points lying in (N+1)D space.'
%
% IN :
%       X, Y, Z, frequency : n-,m-,l-,k-sized vectors of doubles, planes where the field
%        should be interpolated
% OUT :
%       interpolated_field_x (,_y, _z) : p-sized vectors of doubles (p ? n*m*l*k),
%        value of the field at each point of the grid.
%       interpolation_grid : p by 4 matrix of doubles, each line
%        of coordinates cooresponds to the same line in the
%        interpolated_field.
%       NaNs_locations : (n*m*l*k-p) by 4 matrix of doubles, coordinates
%        outside of the convex hull where interpolation is possible.

global data;
constant=isconstant(data.real_coordinates(:,3));


% looking for the indices of the boundarie values of frequencies in nu_m
mini=find(data.nu_m>=min(frequencies),1,'first');
maxi=find(data.nu_m>=max(frequencies),1,'first');

% selecting the meaningful data to accelerate the interpolation
% 2 seems to be the fastest choice for a linear interpolation.
mini=max(1,mini-2);
maxi=min(maxi+2,length(data.nu_m));

n=length(data.nu_m(mini:maxi));
[m,~]=size(data.real_coordinates);

% preparation of the interpolation
if isnan(constant)
    % if the data's coordinates form a real volume, a 4-dimensional matrix
    % is made
    [X,Y,Z,nu]=ndgrid(X,Y,Z,frequencies);
    [i,j,k,l]=size(X);
    N=i*j*k*l;
    interpolation_grid=[reshape(X,N,1),reshape(Y,N,1),reshape(Z,N,1),reshape(nu,N,1)];
 % Creation of a 2d matrix containing all interpolation coordinates

 Volume=[repmat(data.real_coordinates,[n,1]), sort(repmat(data.nu_m(mini:maxi)',[m,1]))];
% Creation of a 2d matrix containing all acquisition coordinates for each
% frequency near the chosen ones
else
    % if the data forms a plane, (same z altitude) a 3d matrix is made.
    [X,Y,nu]=ndgrid(X,Y,frequencies);
    [i,j,l]=size(X);
    N=i*j*l;
    interpolation_grid=[reshape(X,N,1),reshape(Y,N,1),reshape(nu,N,1)];
    % Creation of a 2d matrix containing all interpolation coordinates

```

```
59      Volume=[repmat(data.real_coordinates(:,1:2),[n,1]), sort(repmat(data.nu_m(mini:maxi)',[m↩
            ,1]))];
60  % Creation of a 2d matrix containing all acquisition x and y coordinates for each
61  % frequency near the chosen ones
62  end
63
64  Bx=reshape(data.Bx_nu(:,mini:maxi),m*n,1);
65  By=reshape(data.By_nu(:,mini:maxi),m*n,1);
66  Bz=reshape(data.Bz_nu(:,mini:maxi),m*n,1);
67  % Creation of a vector where each value matches its coordinates in the 2d
68  % matrix
69
70
71  tic;
72  interpolated_field_x=griddatan( Volume , Bx , interpolation_grid );
73  interpolated_field_y=griddatan( Volume , By , interpolation_grid );
74  interpolated_field_z=griddatan( Volume , Bz , interpolation_grid );
75  toc;
76  % interpolation of each field in the coordinates given by the
77  % interpolation_grid.
78
79
80
81  if ~isnan(constant)
82      interpolation_grid(:,3:4)=[repmat(constant,[N,1]),interpolation_grid(:,3)];
83      % if the acquired data is on a plane, we remaje a n-by-4 matrix, by
84      % adding the correct altitude.
85  end
86
87  are_NaNs=(isnan(interpolated_field_x));
88  % locations of NaNs in the data
89  % NaNs used to be removed, but it is necessary to have the full matrix of
90  % values to display the data into slice. This is why the binary vector
91  % are_NaNs is given if the user wants to remove them.
92
93  end
```

5.3.3 isconst.m

```
1   function [ constant ] = isconstant( vector )
2   % If all elements in vector are identical, it returns the value.
3   % If they are not, it returns nan
4   % IN :
5   %       vector : vector, in the primary use it is the z-coordinates of the
6   %       acquired points
7   % OUT :
8   %       constant : NaN or double.
9   constant=vector(1);
10      for j = 1:length(vector)
11          if (vector(j)-constant)
12              constant = NaN;
13              break
14          end
15      end
16
17  end
```

5.3.4 reshape_for_slice.m

```
1   function [ X,Y,Z,V ] = reshape_for_slice( grid , B , x,y,z , freq)
2   % Reshapes the data to import it into slice
3   % The data must come from the interpolation function.
4   % This function rearranges the data into a ndgrid format for slice to work
5   % (even if slice demands a meshgrid format, it seems to work with the ndgrid
6   % instead.
7   %
8   % IN :
9   %       grid : interpolated_grid from interpolation
10  %       B : interpolated_field_ from interpolation
11  %       x,y,z : vectors used to interpolate
12  %       freq : chosen frequency for slice, must exist in the frequency
13  %       vector used to interpolate
14  % OUT :
15  %       X,Y,Z : 3-d grids of coordinates
16  %       V : 3-d grid of rearranged values of B
17
18
19  m=length(x);
20  n=length(y);
```

```
p=length(z);
% size of the 3-d dimensional matrix whichsrved as a grid.

    ok_freq=freq==grid(:,4);
% binary vector which indicates the lines relative to the chosen frequency

X = reshape(grid(ok_freq,1),m,n,p);
Y = reshape(grid(ok_freq,2),m,n,p);
Z = reshape(grid(ok_freq,3),m,n,p);

V = reshape(B(ok_freq),m,n,p);

% data is in meshgrid format, it is needed in ndgrid format
P = [2 1 3];
   X = permute(X, P);
   Y = permute(Y, P);
   Z = permute(Z, P);
   V = permute(V, P);

end
```

6 Test sur du programme sur le champ théorique d'un fil infini

Afin de valider notre programme, nous avons modifié certaines fonctions pour les faire accepter des données non acquises. Nous avons modélisé le champ induit par un fil de courant, et l'avons mis en entré du programme pour vérifier que ce que l'on obtient en sortie est correct.

6.0.5 init_test.m

```
%clear all
projectpath = genpath(pwd);
addpath(projectpath);

disp('Welcome to your tool''s test initialization. You will be asked a few questions...');

% Define the global variables

global data Rate;
data = struct('real_coordinates',[],'nu_m',[],'Bx_nu',[],'By_nu',[],'Bz_nu',[],'delta',[]);



% Initialization of the struct. and global variables
init_variables;
Rate = 1000;
DurationInSeconds = 1;

global filter_parameters;
filter_parameters = struct('Fs',1/Rate,'Rp',0.5,'Rs',20,'nu_pass',4/10*Rate,'nu_stop',1/2*↩
    Rate,...
                           'nu_m', 0:1/DurationInSeconds:Rate/2-1/DurationInSeconds);
data.nu_m = 0:1/DurationInSeconds:Rate/2-1/DurationInSeconds;

donneesFil; % Calculates the theoric magnetic field in specified points
data.real_coordinates(:,4)=[];
% removes the 4th column of the coordinates which are used to display the magnetic field on ↩
    the first graph.

disp('Initialization completed !');

acquisition_test;
% applies the fft on the signals calculated

global other_data
other_data =struct('Rate',Rate,'dimensions',dimensions);
% saves the dimension and rate data in a struct 'show' uses
show;
% gives a preview of the data.
```

6.0.6 acquisition_test.m

```
function acquisition_test()
% Void function (no out parameters) which applies fft to the test points,
% finds the phase and saves the data
% IN :
```

```
%       none
% OUT :
%       none

global data Br Rate;
warning off
Bx = data.Bx_nu';
By = data.By_nu';
Bz = data.Bz_nu';
data.Bx_nu =[];
data.By_nu =[];
data.Bz_nu =[];


for i=1:length(data.real_coordinates)

% Applies fft to the acquired values
[delta_x,delta_y,delta_z] = get_delay(Bx(:,i),By(:,i),Bz(:,i),Br',Rate);
[Bx_nu,By_nu,Bz_nu] = fftdata_test([Bx(:,i),By(:,i),Bz(:,i)]);
% Get the delay ( FORMULE QUI LIE LE DELAI A LA PHASE ICI ) + CORRECTION DE LA SUITE

% Saves the generated data in one go preventing problems with unfinished
% acquisitions.
data.delta(i,:) = [delta_x,delta_y,delta_z];
data.Bx_nu(i,:) = Bx_nu;
data.By_nu(i,:) = By_nu;
data.Bz_nu(i,:) = Bz_nu;

end
disp('Warning may occure if a signal is null, for exemple the By field for y=0');

% saves de data
if(exist('data_test','file')) save('data_test','data', '-append');
else save('data_test','data');
end


% Keyword to add data to an existing file. For MAT-files, -append adds new
% variables to the file or replaces the saved values of existing variables
% with values in the workspace. For ASCII files, -append adds data to the
% end of the file.
end
```

6.0.7 DonneesFil.m

```
close all
global data Br

% Caracteristique de l'objet d'etude

u0=5;       % permeabilite magnetique
I0=20;      % amplitude du courant en A
f=50;       % frequence du courant en Hz
C=50;       % Amplitude des parasites

% Taille du domaine spatial etudie (ici un pave)


a=2.2;        % longueur sur x
b=1.5;        % longueur sur y
c=0.55;        % longueur sur z
global dimensions
dimensions = [-a,a;-b,b;-c,c];

% Temps de mesure

Tf = DurationInSeconds;    % en s

% Echantillonage

px = .5;      % pas spatial d'echantillonnage
py = .5;      % pas spatial d'echantillonnage
pz = 1.75;      % pas spatial d'echantillonnage

Fs=Rate;
Ts = 1/Fs; % temps d'echantillonnage en s

%% Le fil etudie passe par l'origine et est colineaire au vecteur ez

for x=-a:px:a
    for y=-b:py:b
        for z=-c:pz:c

```

```
            %% Passage en coordonnees cylindrique

            r=(x^2+y^2)^(1/2);


                        %% Echantillonnage temporel

            t=0:Ts:Tf-Ts;                                   % Abscisse temporelle
            n=length(t);                                  % Nb d'echantillons temporelle


            I=I0*sin(2*pi*f*t);                           % Calcul du courant au moment d'↩
                 echantillonnage

            B_parasite = 0;%2*C*(rand(1,n)-1/2);     % Creation de parasite magnetique

            % Mesure du champ (parasites compris)

            Bx=(u0*I/(2*pi*r)+B_parasite)*x/r;            % Composante sur x
            By=(u0*I/(2*pi*r)+B_parasite)*y/r;            % Composante sur y
            Bz=(u0*I/(2*pi*r)+B_parasite)*0/r;            % Composante sur z
             % Champs de reference

%% remplissage structure
[n,~]=size(data.real_coordinates);
    data.real_coordinates(n+1,:) = [x,y,z,r];
    data.Bx_nu(n+1,:) = Bx;
    data.By_nu(n+1,:) = By;
    data.Bz_nu(n+1,:) = Bz;

    maxiX(n+1) = u0*I0/(2*pi*r)*abs(x)/r;
    maxiY(n+1) = u0*I0/(2*pi*r)*abs(y)/r;

        end
    end
end
Br=(u0*I/(2*pi*(3^2+3^2)^(1/2))+B_parasite)*3/((3^2+3^2)^(1/2));
plot(data.real_coordinates(:,4), maxiX(:),'+', data.real_coordinates(:,4), maxiY(:),'<',data.↩
    real_coordinates(:,4),sqrt(maxiX.^2+maxiY.^2),'o');

%%
clear r Bx By Bz B_parasite Numero_de_positionnement x y z p a b c u0 I0 f C n Tf Ts Fs I ↩
    Nbline
```

Bibliographie

[1] chaianun. matlab :tutorial :perspective [AIT computer vision wiki]. http://webeng.cs.ait.ac.th/cvwiki/matlab:tutorial:perspective.

[2] Andreas Christ, René Guldimann, Barbara Bühlmann, Marcel Zefferer, Jurriaan F Bakker, Gerard C van Rhoon, and Niels Kuster. Exposure of the human body to professional and domestic induction cooktops compared to the basic restrictions. *Bioelectromagnetics*, (May) :1–11, June 2012.

[3] Matthew Dailey. Course lecture notes. http://cis.ait.asia/course_offerings/48/lecture_notes/154.

[4] Birgitta Floderus, Carin Stenlund, and Frank Carlgren. Occupational exposures to high frequency electromagnetic fields in the intermediate range (>300 hz-10 MHz). *Bioelectromagnetics*, 23(8) :568–77, December 2002.

[5] C.D. Halevidis, J.D. Koustellis, a D. Polykrati, and P.D. Bourkas. Exposure of workers to extremely low frequency magnetic fields during the temperature-rise test of electrotechnical equipment. *Measurement*, 45(8) :1960–1965, October 2012.

[6] HONEYWELL. An-201 : Set/reset pulse circuits for magnetic sensors. http://physics.ucsd.edu/neurophysics/Manuals/Honeywell/AN-201.pdf.

[7] HONEYWELL. An-202 : Magnetic sensor hybrid application circuit. http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/Magnetic__Literature_Application_notes-documents/AN202_Magnetic_Sensor_Hybrid_Application_Circuit.pdf.

[8] Aida Muharemovic, Hidajet Salkic, and Mario Klaric. The calculation of electromagnetic fields (EMF) in substations of shopping centers. page 1081–1088, 2012.

[9] Mohammad Nadeem, Yngve Hamnerius, Kjell Hansson Mild, and Mikael Persson. Magnetic field from spot welding equipment–is the basic restriction exceeded ? *Bioelectromagnetics*, 25(4) :278–84, May 2004.