Hes·so VALAIS WALLIS
School of Engineering

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Filière systèmes industriels

Orientation infotronics

Travail de bachelor Diplôme 2020

Borgeat Rémy

PlongDir

Professeur
Andersson Alexandra

Expert
Gianni Di Marco

Date de la remise du rapport
14.8.2020

Données du travail de diplôme
Daten der Diplomarbeit

Filière / Studiengang	Année académique / *Studienjahr*	No TD / *Nr. DA*
SYND	**2019/20**	**it/2020/46**

Mandant / *Auftraggeber*
- ☐ HES—SO Valais
- ☒ Particulier
- ☐ Etablissement partenaire *Partnerinstitution*

Etudiant / *Student*
Rémy Borgeat

Professeur / *Dozent*
Alexandra Andersson

Lieu d'exécution / *Ausführungsort*
- ☒ HES—SO Valais
- ☐ Industrie
- ☐ Etablissement partenaire *Partnerinstitution*

Travail confidentiel / *vertrauliche Arbeit*
☐ oui / ja [1] ☒ non / nein

Expert / *Experte (données complètes)*
Gianni Di Marco
Route des Véroz 12, 1872 Troistorrents
gianni.dimarco@bluewin.ch

Titre / *Titel*

PlongDir

Description / *Beschreibung*

Les plongeurs travaillent souvent à deux sous l'eau. Parfois, ils perdent le contact visuel les uns avec les autres. Il peut alors être assez difficile de localiser le partenaire en raison d'une visibilité limitée sous l'eau ou d'une ligne de vue interrompue. Les systèmes qui existent pour résoudre ce problème sont encombrants et impliquent l'utilisation d'un relais sur le bateau. Le but de ce projet est de développer un petit système pour la localisation acoustique sous l'eau, où chaque plongeur a un tel appareil, et chaque appareil indiquera dans quelle direction se trouve l'autre.

Objectifs / *Ziele*

- Design a small battery powered PCB with a micro-controller, three microphones, and appropriate data storage and data transfer. Design a drive circuit and PCB for the transducer. Understand how the microphones work and design the appropriate digital signal processing algorithm to extract signal.
- In air: Use the PCBs to collect data for various angles for sound between the two devices without and with enclosure. Construct an algorithm to compute sound arrival angle from data collected. Repeat experiment in water.
- If time allows: Demonstrate sound arrival detection in water and air with calculation done on embedded system with display of direction on the device.

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation / filière
Leiter der Vertiefungsrichtung / Studiengang:

.....

[1] Etudiant / *Student* :

.....

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
25.05.2020

Présentation intermédiaire / *Zwischenpräsentation*
Semaine / Woche 26 (22.06 – 26.06.2020)

Remise du rapport / *Abgabe des Schlussberichts:*
14.08.2020, 12:00

Exposition / *Ausstellung der Diplomarbeiten:*
28.08.2020 (si autorisé / *falls genehmigt*)

Défense orale / *Mündliche Verfechtung:*
Semaine / Woche 36 (31.08 – 04.09.2020)

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

PlongDir

Diplômant/e Rémy Borgeat

Objectif du projet

Lors d'une plongée, il faut garder le contact visuel avec son binôme de plongée. Cependant, il peut arriver que le contact visuel soit perdu. L'objectif de ce travail est de permettre aux plongeurs de connaître la direction de son binôme.

Méthodes | Expériences | Résultats

Pour déterminer la direction de son partenaire de plongée, des microphones et des émetteurs ultrasoniques sont utilisés. Chaque plongeur comporte des émetteurs ultrasoniques et 3 microphones qui sont disposés en triangle.

Dans un premier temps, un PCB a été développé. Ce PCB s'interface avec la carte STM32F746G-Discovery. Les données des microphones sont récupérées et enregistrées sur une carte SD. Ces données sont ensuite utilisées dans des scripts Matlab pour déterminer la direction du son partenaire de plongée. Dans un deuxième temps, les calculs sont effectués sur le uContrôleur.

Plusieurs tests ont été effectués. Le premier consiste à tester le projet dans l'air pour savoir s'il est possible de récupérer la direction du partenaire de plongée. Les tests ont montré qu'il est possible de récupérer la direction de son partenaire de plongée jusqu'à une certaine distance.

Le projet devant aller dans l'eau, il est nécessaire de mettre les microphones et l'émetteur ultrasonique dans des boîtiers. Des tests ont été effectués avec différents matériaux comme boîtier. Des difficultés ont été rencontrées avec les matériaux testés et aucun des matériaux testés a eu des résultats concluants.

Travail de diplôme
| édition 2020 |

Filière
Système industiel

Domaine d'application
Infotronics

Professeur responsable
Andersson Alexandra
alexandra.andersson@hevs.ch

Partenaire
Gianni Di Marco

Table des matières

Table des figures

Table des équations

Table des tableaux

Liste des abréviations

- **DMA** : Direct Memory Access : Accès direct à la mémoire
- **SPI** : Serial Peripheral Interface : Interface périphérique série
- **MOSI** : Master output, Slave Input : Sortie maître, Entrée esclave
- **MISO** : Master input, Slave output : Entrée maître, Sortie esclave
- **CLK** : Clock : Signal d'horloge
- **HAL** : Hardware Abstraction Layer : Couche d'abstraction matérielle
- **PDM** : Pulse Density Modulation : Modulation a densité d'impulsions
- **PWM** : Pulse Width Modulation : Modulation à largeur d'impulsions
- **LED** : light-emitting diode : Diode électroluminescente
- **PCB** : Printed Circuit Board : Circuit imprimé
- **LCD** : Liquid Crystal Display : Affichage à cristaux liquides
- **SD** : Secure Digital : Numérique sécurisée
- **UML** : Unified Modeling Language : Langage de modélisation unifié

I. INTRODUCTION

1 PRÉAMBULE

Lors d'une plongée, les plongeurs sont par binôme. Pour des raisons de sécurité, il est essentiel que chaque plongeur garde le contact visuel avec son partenaire. Il est, néanmoins, possible que le contact visuel entre les plongeurs soient perdus (eau trouble, algues, nuit, etc…) voir Figure 1. C'est dans cet optique que ce projet a été développé.

Figure 1 : Illustration de l'obstruction de la vue entre les 2 plongeurs

Ce projet a pour but de connaître la direction de son partenaire de plongée, même si le contact visuel est perdu.

Il existe des appareils permettant de localiser son partenaire de plongée, mais ces derniers sont encombrants et nécessite un relais sur le bateau.

2 CONCEPT

Le concept de ce travail consiste à utiliser des microphones et des émetteurs ultrasoniques pour déterminer la direction de son partenaire de plongée.

Chaque plongeur est équipé de microphones et d'émetteurs ultrasoniques. Les microphones présents sur chaque plongeur permettent de déterminer la direction de son partenaire de plongée.

Un affichage composé de 8 LEDs indiquera la direction de son partenaire de plongée. Cet affichage permet de montrer la direction de son partenaire de plongée à 360° autour du lui.

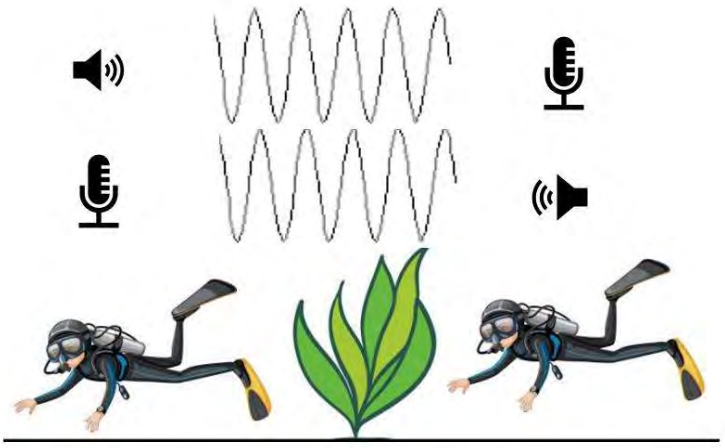


Figure 2 : Illustration du concept de ce travail de bachelor

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

3 PLANNIFICATION

Une planification a été faites et est présentée en Figure 3. La planification est divisée en bloc d'une semaine et présente le déroulement général du projet.

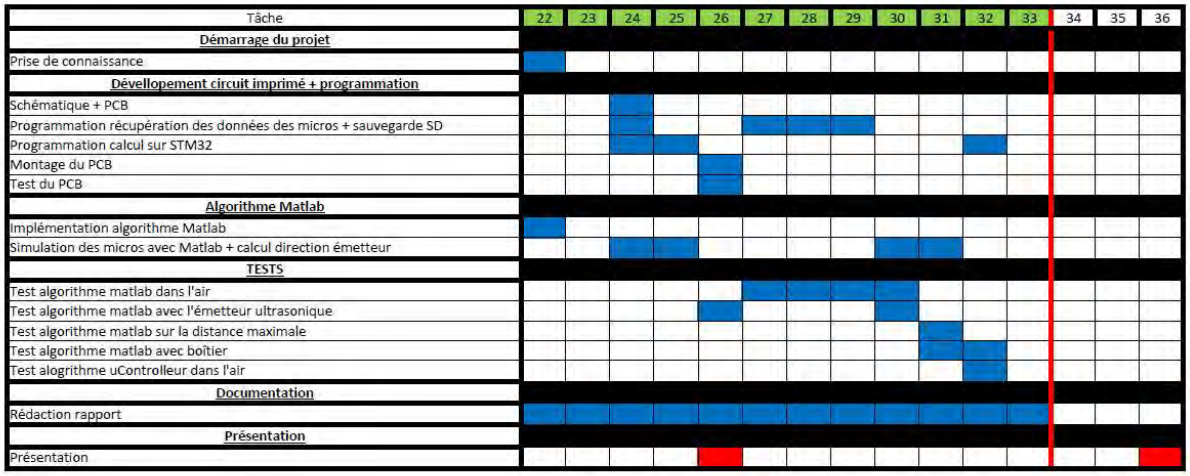


Figure 3 : Planification du travail de bachelor

Il y a 3 blocs principales différents dans ce planning :

- Développement circuit imprimé + programmation en C/C++
- Développement algorithme Matlab
- Tests

Chaque bloc comporte plusieurs tâches qui sont décrites dans la planification.

4 STRUCTURE DU RAPPORT

Ce rapport présente le développement et les résultats obtenus de ce travail de bachelor. On y trouve :

- Un cahier des charges qui présente les objectifs du projet et les contraintes imposées.
- Une présentation des logiciels utilisés et du matériel utilisé.
- Une partie théorique qui présente les différents algorithmes utilisés dans ce projet.
- Le développement réalisé pour répondre à la problématique du cahier des charges.
- Une présentation des tests effectués et des résultats obtenus.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

II. ANALYSES

5 CAHIER DES CHARGES

Ce chapitre présente le cahier des charges de ce travail de bachelor.

Le but de ce travail est de déterminer s'il est possible de déterminer la direction de son partenaire de plongée en utilisant des microphones. Utiliser des microphones permet de réaliser un produit qui n'est pas encombrant. De plus, cela permet de réaliser un produit qui ne coûte pas cher.

La direction du partenaire de plongée doit être déterminer en 2 dimensions seulement. La différence de profondeur entre les plongeurs n'est pas prise en compte. Les microphones ont été imposés pour ce travail. Ces derniers sont bon marché. La direction de son partenaire de plongée doit être affichée avec 8 LEDs.

5.1 Cahier des charges initial

Dans un premier temps, un PCB doit être développé. Ce PCB comporte les microphones et est alimenté avec des batteries. Les données des microphones doivent être sauvegardées sur une carte SD et les calculs sont faits sur le logiciel Matlab. Dans un premier temps, des tests doivent être effectués avec les microphones dans l'air et dans un boîtier. Ensuite, il faudra refaire l'expérience dans l'eau.

Si le temps le permet, les calculs doivent être implémentés sur un uContrôleur. Ce uContrôler doit aussi gérer l'affichage de la direction de son partenaire de plongée.

5.2 Cahier des charges final

Des modifications sur le cahier des charges ont été apportées. Les modifications sont les suivantes :

- Développer un PCB avec uniquement les microphones qui s'interface avec la carte de développement STM32F746G-Discovery.
- Faire les calculs sur uContrôleur.
- Afficher la direction de son partenaire de plongée sur un écran LCD.

6 LOGICIEL UTILISÉ

Plusieurs logiciels ont été utilisés pour la réalisation de ce projet. Le Tableau 1 présente les logiciels utilisés dans ce travail de bachelor.

LOGICIEL	FONCTION
MATLAB R2019B	Utilisé pour les simulations et divers calculs.
SYSTEM WORKBENCH FOR STM32 BASED ON ECLIPSE	Programmation du uContrôleur.
STM32CUBEMX 5.2.1	Configuration des registres et des périphériques du uContrôleur.
GITHUB	Synchronisation du projet entre plusieurs ordinateurs.
UMLET 14.3.0	Création des diagrammes UML.
DOXYGEN 1.8.16	Création de la documention logicielle.

Tableau 1: Logiciel utilisé dans ce travail de bachelor.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

7 MATÉRIEL UTILISÉ

Ce chapitre traite des composants imposés et utilisés dans ce projet.

7.1 Émetteur ultrasonique

Un émetteur ultrasonique est utilisé pour émettre un signal ultrasonique à 40[kHz].

2 émetteurs ultrasoniques peuvent être utilisés dans ce projet : UT-1240K-TT-R[1] et UT-1640K-TT-2-R[2].

Des tests sur la consommation de ces composants doivent être effectués, car les fiches techniques n'indiquent pas la consommation de ces derniers. Il est nécessaire de connaître la consommation des émetteurs ultrasoniques, afin de connaître le circuit électronique nécessaire pour piloter le composant.

Les tests sur la consommation des 2 émetteurs ultrasoniques sont présentés en chapitre 12.

Figure 4 : Émetteur ultrasonique (*source de l'image, voir note de fin 1*)

7.2 Microphones SPH0641LU4H-1

Le microphone SPH0641LU4H-1[3] est utilisé dans ce projet. Ce microphone possède une sortie digitale. Cette sortie est une modulation à densité d'impulsions (PDM) qui est illustrée en Figure 5. Il y a un convertisseur sigma-delta 1 bit à l'intérieur du microphone.

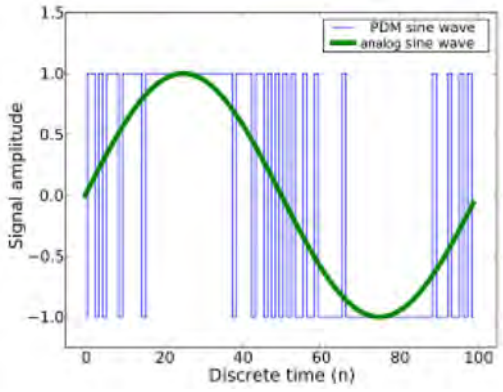


Figure 5 : Modulation à densité d'impulsions (*source de l'image[4]*)

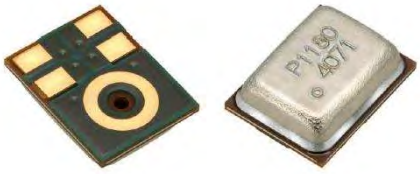


Figure 6 : Microphones (*source de l'image, voir note de fin 3*)

Le microphone a besoin d'un signal d'horloge pour fonctionner. Dans ce travail, la fréquence du signal d'horloge est de 4[MHz].

La résolution de la sortie pour un signal ultrasonique d'une fréquence à 40[kHz] et une fréquence d'horloge à 4[MHz] est calculée dans l'Équation 1.

$$résolution = \frac{fHorloge}{fSon} = \frac{4\ [Mhz]}{40\ [kHz]} = 100\ [\frac{bits}{période}]$$

Équation 1 : Résolution sortie des microphones

Selon l'Équation 1, 100 bits constituent une période du signal capté par les microphones.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

7.3 Carte de développement STM32F746G-Disco

La carte de développement STM32F746G-Disco[5] a été utilisée dans ce projet. Cette carte de développement possède un uContrôleur STM32F746NGH. Ce uContrôleur est basé sur une cœur Arm Cortex-M7.

Cette carte a été utilisée, car elle comporte tous les périphériques voulus dans ce projet (SPI, carte SD, écran LCD). De plus, ce projet étant un prototype, c'est un gain de temps d'utiliser une carte de développement que de développer une carte à uContrôleur.

Figure 7 : Carte de développement STM32F736G-Disco (*source de l'image*[6])

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

III. DÉVELOPPEMENT RÉALISÉ

8 CALCUL DIRECTION DU SON

Ce chapitre traite des calculs effectués pour déterminer la direction du son.

Dans ce rapport, plusieurs angles différents sont employés. Le Tableau 2 présente la nomenclature utilisée pour ces angles.

NOM DE L'ANGLE	SYMBOLE
ANGLE D'ARRIVÉE DU SON	θ
DÉPHASAGE ENTRE LE MICROPHONE NUMÉRO 2 ET LE MICROPHONE NUMÉRO 1	$\varphi_2 - \varphi_1$
DÉPHASAGE ENTRE LE MICROPHONE NUMÉRO 3 ET LE MICROPHONE NUMÉRO 1	$\varphi_3 - \varphi_1$
ANGLE ENTRE LES 2 DÉPHASAGES	α

Tableau 2 : Nomenclature des angles

8.1 Principe

Pour déterminer la direction de son binôme de plongée, un plongeur est équipé de 3 microphones disposés sur la périphérie d'un cercle et l'autre plongeur est équipé d'un émetteur ultrasonique. M1, M2 et M3 représentent les microphones, T représente l'émetteur ultrasonique et θ est l'angle recherché.

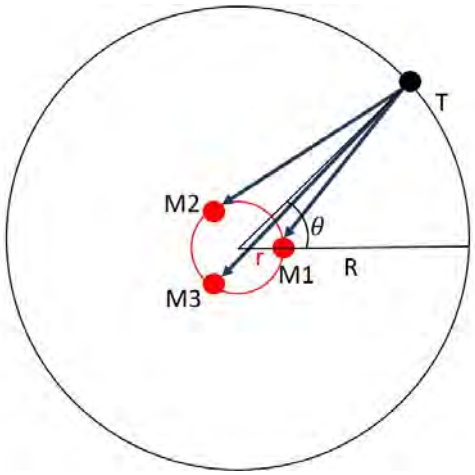


Figure 8 : Schéma triangulation

Chaque microphone va détecter le son provenant de l'émetteur ultrasonique. Le son capté par chacun des microphones sera déphasé dans le temps, comme l'illustre la Figure 9. Les déphasages entre les signaux captés par les microphones varient en fonction de l'angle θ.

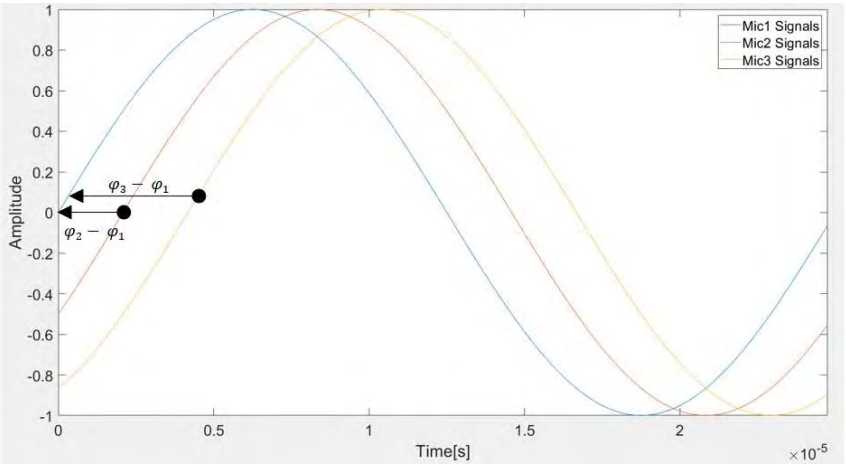


Figure 9 : Signaux capté par les microphones

Sur la Figure 9, il y a 3 signaux, ces signaux sont documentés ci-dessous.

- Le signal bleu représente le signal capté par le microphone numéro 1
- Le signal orange représente le signal capté pour le micro 2
- Le signal jaune représente le signal capté par le micro 3.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

8.2 Théorie

Ce sous-chapitre présente l'aspect théorique du calcul de l'angle d'arrivée du son.

Soit M1, M2 et M3 sont les positions des microphones et r le rayon du cercle sur lequel sont disposés les microphones. Les cordonnées cartésiennes des 3 microphones sont les suivantes :

- M1 = (r, 0)
- M2 = $(-\frac{1}{2}r, \frac{\sqrt{3}}{2}r)$
- M3 = $(-\frac{1}{2}r, -\frac{\sqrt{3}}{2}r)$

Soit T la position de l'émetteur ultrasonique, R est le rayon du cercle sur lequel se trouve l'émetteur ultrasonique et θ est l'angle d'arrivée du son. La position de l'émetteur ultrasonique est la suivante :

- T = $(R * \cos\theta\,, R * \sin\theta)$

3 distances peuvent être calculées entre l'émetteur ultrasonique et chacun des microphones :

- d1 : T → M1 = $\sqrt{(R * \cos(l) - r)^2 + (\mathrm{R} * \sin(\theta))^2}$
- d2 : T → M2 = $\sqrt{(R * \cos(\theta) + \frac{r}{2})^2 + (\mathrm{R} * \sin(\theta) - \frac{r\sqrt{3}}{2})^2}$
- d3 : T → M3 = $\sqrt{(R * \cos(\theta) + \frac{r}{2})^2 + (\mathrm{R} * \sin(\theta) + \frac{r\sqrt{3}}{2})^2}$

On peut noter que les distances entre les microphones et l'émetteur ultrasonique varient en fonction de l'angle θ.

La longueur d'onde d'un signal sonore dans l'air est définie dans l'équation suivante :

$$\lambda = \frac{c_{air}}{f}$$

Pour calculer la phase du signal sonore, l'équation suivante est utilisée :

$$\varphi = \frac{2\pi}{\lambda} * d = \frac{2 * \pi * f}{c_{air}} * d$$

Grâce à l'équation précédente, la phase de chaque microphone peut être calculée. Cependant, il n'existe pas de phase absolue, c'est pourquoi ce sont les différences de phases entre les microphones qui sont considérées dans les calculs.

$$\varphi_2 - \varphi_1 = \frac{2 * \pi * f}{c_{air}} * (d2 - d1)$$

Équation 2 : Déphasage entre les microphones 2 et 1

$$\varphi_3 - \varphi_1 = \frac{2 * \pi * f}{c_{air}} * (d3 - d1)$$

Équation 3 : Déphasage entre les microphones 3 et 1

Sur la Figure 9, les déphasages $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$ sont illustrés.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

9 RÉCUPÉRER L'ANGLE DEPUIS LES MESURES

Ce chapitre traite des algorithmes utilisé pour récupérer l'angle θ (angle d'arrivée du son) en utilisant les microphones. La sortie des microphones est une modulation à densité d'impulsions.

9.1 Filtrage de la sortie des micros

La première étape consiste à utiliser un filtre passe-bas numérique pour filtrer la modulation à densité d'impulsions (sortie des microphones). Ce filtre permet de récupérer un sinus qui est le signal analogique capté par le microphone. La Figure 10 montre le résultat du filtrage.

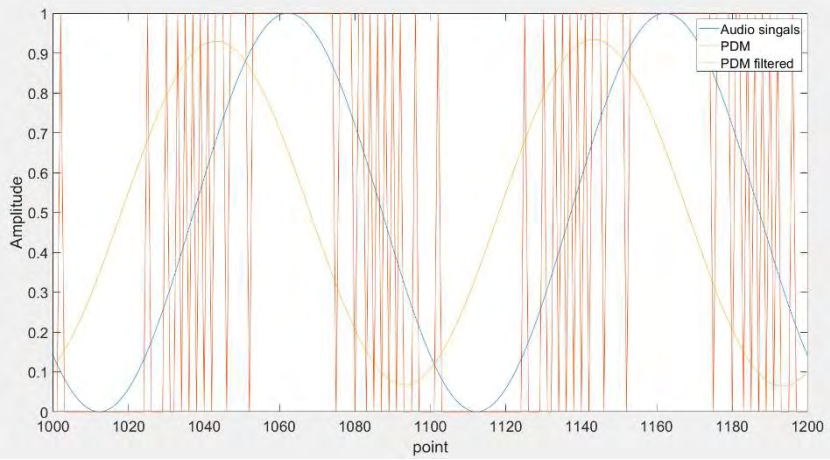


Figure 10 : Filtrage de la modulation à densité d'impulsions.

Sur la Figure 10, il y a plusieurs signaux. Ces signaux sont les suivants :

- Le signal orange représente la sortie du microphone
- Le signal jaune représente le signal filtré
- Le signal bleu représente le signal analogique captés par le microphone.

Le signal filtré est déphasé par rapport au signal analogique capté par le microphone. Cela est dû au filtre passe-bas numérique utilisé. Le diagramme de Bode de ce filtre est illustré en Figure 11. À la fréquence de coupure(40[kHz]), le déphasage est de 68°. Ce déphasage n'impacte pas les résultats obtenus, car il est le même pour les 3 modulations à densité d'impulsions filtrées. De plus, le signal filtré à une amplitude plus faible que le signal analogique capté par les microphones. Cela est aussi dû au filtre passe-bas. À la fréquence de coupure(40[kHz]), le gain est de -1[dB]. Ce gain fait que l'amplitude du signal filtré aura une atténuation de 10.8[%] par rapport au signal analogique capté par le microphone. Cette valeur est obtenue dans l'équation suivante :

$$atténuation = 1 - 10^{\frac{Gain}{20}} = 1 - 10^{\frac{-1}{20}} = 0.108$$

Cette atténuation n'impacte pas les résultats obtenus.

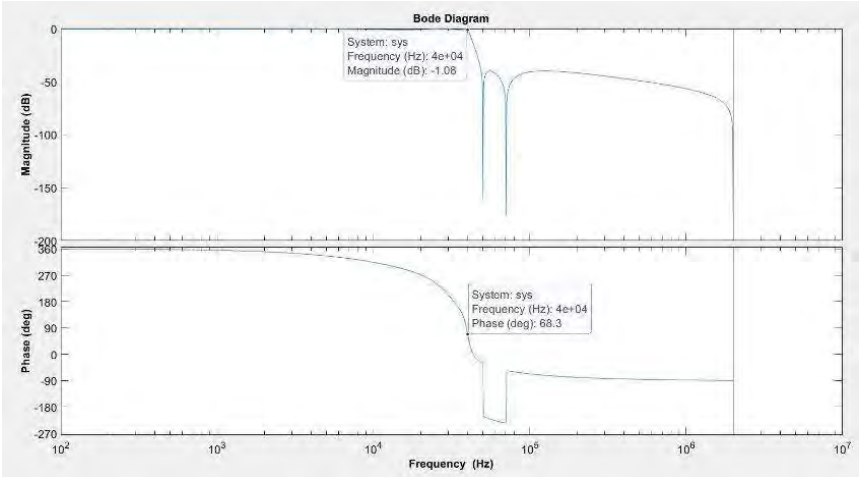


Figure 11 : Filtre passe-bas : Diagramme de Bode

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

9.2 Filtrage sur uContrôleur

Le filtre passe-bas numérique présenté dans le chapitre 9.1 devra être implémenté sur uContrôleur lorsque les calculs seront faits sur ce dernier.

Le filtre passe-bas a été dimensionné sur Matlab. Le filtre dimensionné sur Matlab a ensuite été divisé en section du 2ème ordre. L'Équation 4 présente la fonction de transfert en Z d'un filtre passe-bas numérique du 2ème ordre.

$$H(z) = \frac{b0 + b1 * z^{-1} + b2 * z^{-2}}{a0 + a1 * z^{-1} + a2 * z^{-2}}$$

Équation 4 : Fonction de transfert en Z

Avec la fonction de transfert présentée en Équation 4, on peut trouver l'équations aux différences. L'équations aux différences obtenues est présentées en Équation 5 (a0 = 1).

$$y(n) = x(n) * b0 + x(n-1) * b1 - y(n-1) * a1 + x(n-2) * b2 - y(n-2) * a2$$

Équation 5 : Equations aux différences

La Figure 12 représente le schéma bloc de cette réalisation pour le cas où a0 = 1. Cette structure est appelée Forme directe II transposée.

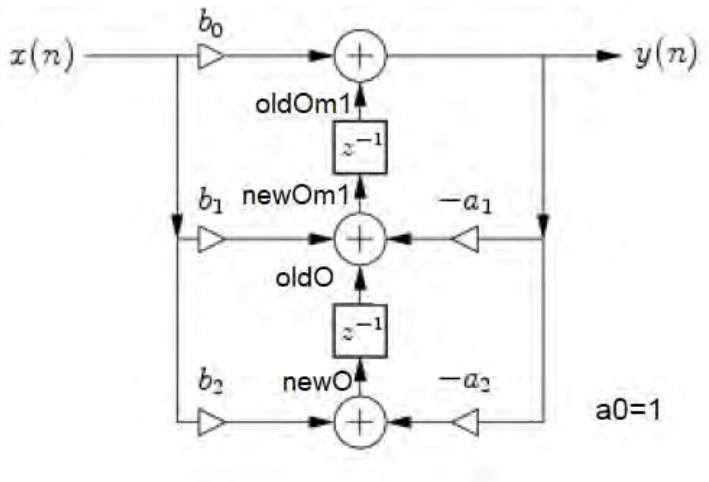


Figure 12 : Forme directe II transposée

Grâce à la Figure 12, on peut déterminer les équations partielles. Ces équations partielles sont les suivantes :

$$newO = x(n) * b2 - y(n) * a2$$

Équation 6 : Équations partielles 1/3

$$newOm1 = x(n) * b1 - y(n) * a1 + oldO$$

Équation 7 : Équations partielles 2/3

$$y(n) = x(n) * bo + oldOm1$$

Équation 8 : Équations partielles 3/3

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

En combinant les 3 équations précédentes, on trouve :

$$y(n) = x(n) * b0 \ + x(n-1) * b1 - y(n-1) * a1 + x(n-2) * b2 - y(n-2) * a2$$

Équation 9 : Équations aux différences

L'Équation 9 est équivalente à l'Équation 5, cela signifie que les 3 équations partielles tirées de la Figure 12 sont correctes.

Ce sont les 3 équations partielles qui seront implémentées dans le uContrôleur pour la réalisation du filtre passe-bas.

9.3 Calcul du déphasage entre les 3 sinus

Une détection de phase est faite, afin de récupérer les déphasage $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$. Cette détection de phase est démodulation I-Q. Un des avantages d'employer cette méthode, c'est que tous les points du signal d'entrée sont employés pour déterminer la phase.

Le signal filtré obtenu au chapitre 9.1 à la forme suivante :

$$A * \sin(\omega t \ + \ \rho\ )$$

La première étape consiste à multiplier une fois ce signal avec un $\sin(\omega t)$ (composante I) et une fois avec un $\cos(\omega t)$ (composante Q).

$$I = \sin(\omega t) * A * \sin(\omega t \ + \ \rho\ )$$

$$Q = \cos(\omega t) * A * \sin(\omega t \ + \ \rho\ )$$

Ces 2 équations peuvent être développées en employant les propriétés des sinus et cosinus.

La première propriété est la suivante :

$$A * \sin(\omega t \ + \ \rho\ ) = A * (\sin(\rho) * \cos(\omega t) + \ \cos\ (\rho) * \sin(\omega t))^{7}$$

On obtient les équations suivantes :

$$I = \mathrm{A} * (\sin(\rho) * \ \sin(\omega t) * \cos\ (\omega t) + \cos(\varphi) * \ \sin^2(\omega t))$$

$$Q = \mathrm{A} * (\sin(\rho) * \cos^2(\omega t) + \cos(\varphi) * \ \sin(\omega t\ ) * \cos\ (\omega t))$$

2 autres propriétés sont utilisées :

$$\sin^2(\omega t) = \frac{1}{2} * (1 - cos(2\omega t))^{8}$$

$$\cos^2(\omega t) = \frac{1}{2} * (1 + cos(2\omega t))^{9}$$

Les 2 équations suivantes sont obtenues :

$$I = \mathrm{A} * (\sin(\rho) * \frac{1}{2}\sin(2\omega t) + \cos(\varphi) * \frac{1}{2} * (1 - cos(2\omega t))$$

$$Q = \mathrm{A} * (\sin(\rho) * \frac{1}{2} * (1 + \cos\ (2\omega t)) + \cos(\varphi) * \ \frac{1}{2}\sin(2\omega t\ ))$$

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

En moyennant les équations sur 2ω, les sinus(2wt) et cos(2wt) valent 0. L'algorithme utilisé est présenté dans le chapitre 9.5. Les 2 équations suivantes sont donc obtenues :

$$I_{mean} = A * \frac{1}{2}\cos(\varphi))$$

$$Q_{mean} = A * \frac{1}{2}\sin(\varphi))$$

Finalement pour obtenir la phase, la fonction atan2 est utilisée.

$$\rho = atan2[1](Q_{mean}, I_{mean}))$$

Équation 10 : phase ρ du sinus

La phase obtenue est un tableau représentant la phase du signal d'entrée sur toute la plage du signal d'entrée. Pour les calculs, il est nécessaire d'avoir un seul nombre et pas un tableau. Il faut donc moyenner la phase obtenue. Cependant, il faut faire attention en moyennant la phase obtenue. En effet, en moyennant la phase obtenue avec une méthode traditionnel, cela peut entraîner des erreurs. Par exemple, si la phase obtenue varie entre la valeur π et la valeur -π, la moyenne donne 0. Cela est une erreur, car une phase de π est équivalent à une phase de -π.

Alors pour moyenner le signal, la méthode suivante a été utilisée. L'angle ρ est représenté comme un angle complexe, comme illustré dans la Figure 13.

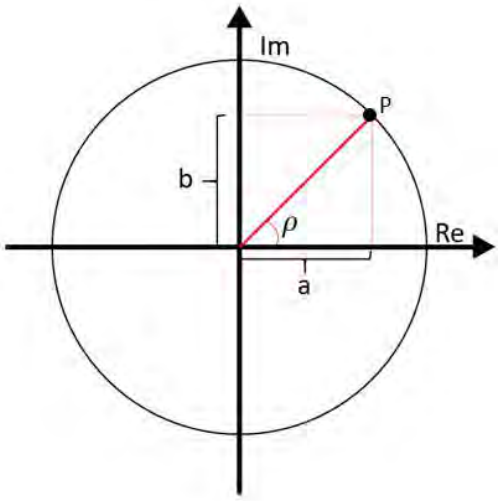


Figure 13 : Nombre complexe

Ainsi, le nombre complexe en coordonnée polaire est le suivant :

$$P = e^{i * \rho}$$

Le nombre complexe obtenu en coordonnée cartésienne est le suivant :

$$P = a + b * i \; = \; \cos(\rho) + i * \sin(\rho)$$

Les points P sont calculés pour toutes les phases obtenues par l'Équation 10.

On peut alors moyenner la partie réel("a") de tous les nombres complexes "P" calculés ensemble et la partie imaginaire("b") de tous les nombres complexes "P" calculés ensemble.

[1] La fonction trigonométrique est expliquée en chapitre 9.4

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Un seul et unique point "P" est alors obtenu. La phase ρ peut alors être récupérée avec l'équation suivante :

$$\rho = \text{atan2}\ (b, a)$$

Cette manière de calculer la moyenne résout le problème rencontré. Si l'on reprend l'exemple d'une phase qui varie entre les valeurs π et -π, les nombres complexes obtenus sont les suivants :

$$P_1 = \cos(\pi) + i * \sin\ (\pi) = -1$$

$$P_2 = \cos(-\pi) + i * \sin\ (-\pi) = -1$$

En moyennant ces nombres complexes, on obtient le résultat suivant :

$$P = a + b * i = -1 + 0 * i = -1$$

Le déphasage ρ alors obtenu est le suivant :

$$\rho = \text{atan}\,2(b, a) = \text{atan}\,2(0, -1) = \pi$$

Avec cette méthode on trouve un déphasage de π qui est la valeur attendue et pas de 0.

Pour obtenir les déphasages $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$, il est nécessaire de réaliser cet algorithme avec les 3 sorties des microphones qui ont été filtrées. On obtient alors les 3 phases $\varphi_1, \varphi_2\ et\ \varphi_3$. Il ne reste plus qu'à les soustraire entre elles.

9.4 Fonction trigonométrique atan2(y,x)

La fonction trigonométrique atan2(y,x) est la même fonction trigonométrique que atan $(\frac{y}{x})$ lorsque x est plus grand que 0. Cependant lorsque x est plus petit que 0, la fonction atan $(\frac{y}{x})$ retourne un angle incorrect. Π radians doit être ajouté à l'angle retourné par la fonction atan $\left(\frac{y}{x}\right)$ pour obtenir l'angle correct. Ce n'est pas le cas avec la fonction atan2(y,x). Avec la fonction atan2(y,x), π radian est ajouté lorsque c'est nécessaire. Ainsi sur la Figure 14 on peut voir que la méthode atan2(x,y) à un plus grand domaine d'arrivée que la méthode atan $(\frac{y}{x})$ illustré sur la Figure 15.

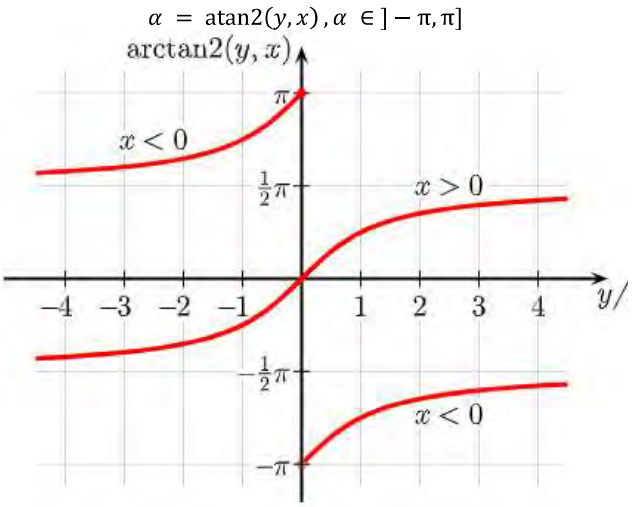


Figure 14 : Fonction trigonométrique atan2(y,x)[10]

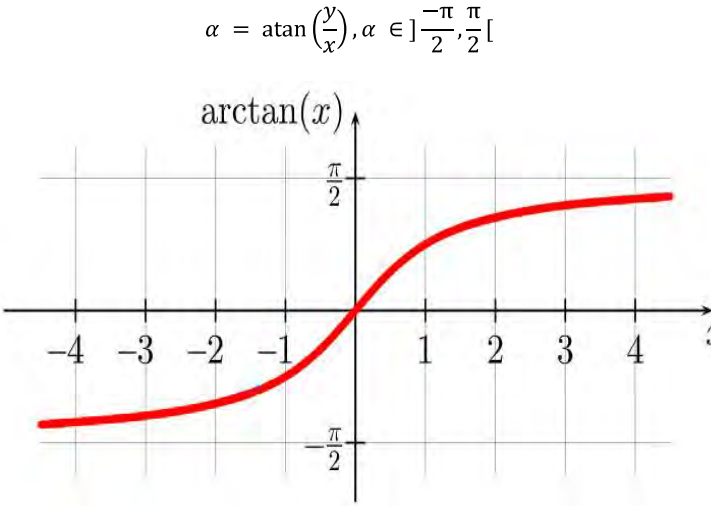


Figure 15 : Fonction trigonométrique atan $\left(\frac{y}{x}\right)$[11]

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

9.5 Algorithme "movmean"

La fonction movmean[12] (A,k) est une méthode fournie par Matlab. Cette fonction est utilisée dans le chapitre 9.3 lorsqu'il est faut moyenner le signal à 80[kHz].

La fonction movmean(A,k) retourne un tableau d'une moyenne local de k points. Lorsque k est impaire, la fenêtre est centrée sur l'élément à la position actuel. Lorsque k est paire, la fenêtre est centrée sur l'élément à la position actuel et à l'élément précédent. La taille de la fenêtre est tronquée lorsqu'il n'y a pas assez d'éléments pour remplir la fenêtre et la moyenne est prise uniquement sur les éléments ayant rempli la fenêtre.

Une période du signal de 40[kHz] est représentée sur 100 bits (voir Équation 1). Par conséquent, la période d'un signal de 80[kHz] est représentée sur 50 bits. C'est pourquoi, les méthodes suivantes sont utilisées :

$$I_{mean} = movmean(I, 50) \; et \; Q_{mean} = movmean(Q, 50)$$

Cette méthode n'existant pas en C++, il est nécessaire de l'implémenter. La méthode implémentée a été faite de la même façon que la méthode sur Matlab.

9.6 Calcul de l'angle d'arrivée du son

Plusieurs méthodes de calcul pour récupérer l'angle d'arrivée du son ont été testées. Ce rapport ne présente qu'une seule méthode pour récupérer l'angle d'arrivée du son. Cette méthode donne les meilleurs résultats par rapport aux autres méthodes testées.

La Figure 16 présente les résultats théoriques possible. Ce graphique est obtenu avec la méthode Matlab suivante :

$$scatter[13](\varphi_3 - \varphi_1, \varphi_2 - \varphi_1, 10, theta)$$

- $\varphi_2 - \varphi_1$: est un vecteur qui représente toutes les différences de phases entre le microphone numéro 2 et le microphone numéro 1 pour un angle θ variant entre 0° et 360°.
- $\varphi_3 - \varphi_1$: est un vecteur qui représente toutes les différences de phases entre le microphone numéro 3 et le microphone numéro 1 pour un angle θ variant entre 0° et 360°.
- 10 : Taille des points affichés sur le graphique
- $theta$: Vecteur de l'angle θ variant entre 0° et 360°, qui a servi à calculer les différences de phases $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$. Ces différences de phases ont été calculées comme expliquer en chapitre 8.2.

Cette commande crée un graphe et dessine des cercles aux locations spécifiées par les vecteurs $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$. L'abscisse de ce graphe est la différence de phase $\varphi_2 - \varphi_1$ et l'ordonnée de ce graphe est la différence de phase $\varphi_3 - \varphi_1$. L'ellipse représente tous les résultats possibles. De plus cette ellipse comporte une couleur qui représente l'angle θ. Sur la droite du graphique, une échelle fait correspondre une couleur à la valeur de l'angle θ.

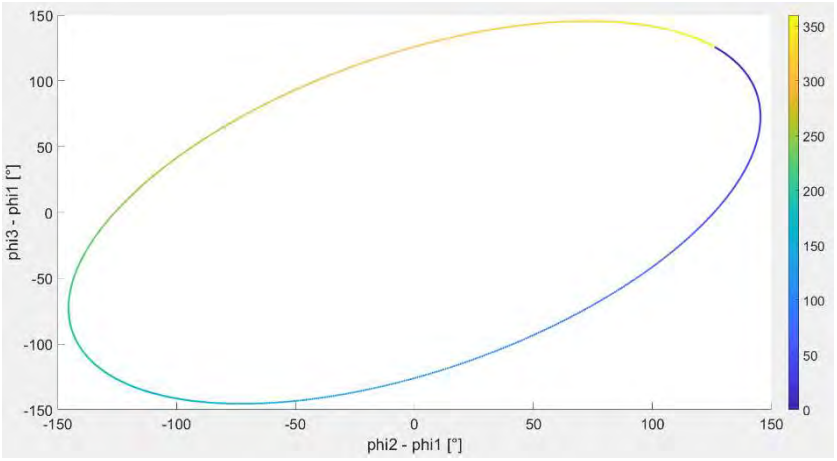


Figure 16 : Calcul de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

La Figure 17 représente le même graphe que le la Figure 16. La différence c'est que ce graphe illustre plusieurs distances différentes entre les micros. Plus la distance entre les microphones est petite, plus l'ellipse est petite et inversement. A noter que c'est le cas tant que la distance entre les microphones est plus petite que $\frac{\lambda}{2}$.

On peut remarquer sur ce graphe, que peu importe la distance entre les microphones, l'angle α pour un même angle θ reste le même.

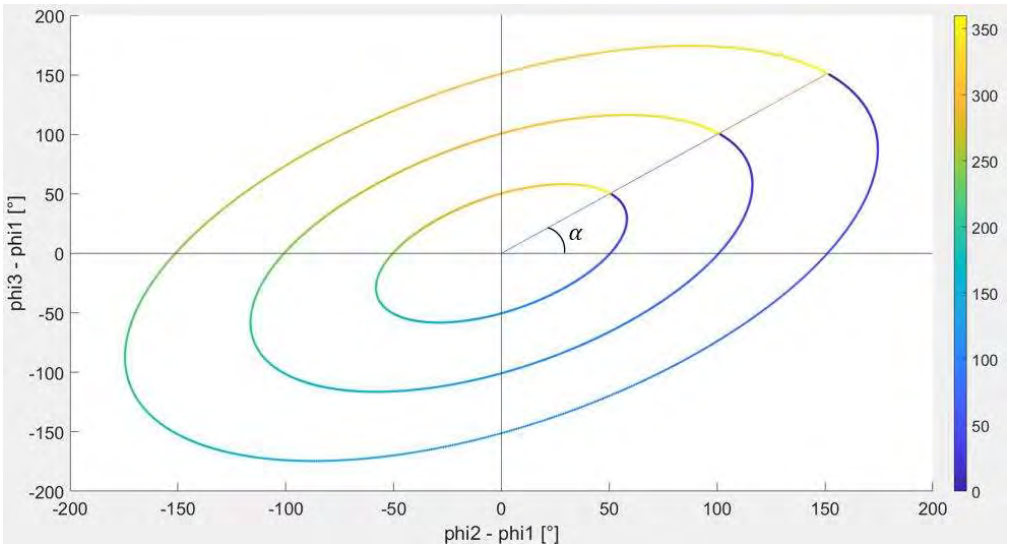


Figure 17 : Calcul de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ avec plusieurs distances entre les microphones

Il se peut que les 2 plongeurs ne soient pas à la même profondeur. Lorsque les 2 plongeurs ne sont pas à la même profondeur, les déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ seront impactés. Ces déphasages seront impactés de la même manière que lorsque la distance entre les microphones varie. Plus l'angle d'incidence du son est élevé, plus l'ellipse sera petite et inversement.

L'équation suivante permet de trouver l'angle α :

$$\alpha = atan2^2(\varphi_3 - \varphi_1, \varphi_2 - \varphi_1)$$

La Figure 18 présente les calculs dans le cas où la distance entre les microphones est plus grande que $\frac{\lambda}{2}$.

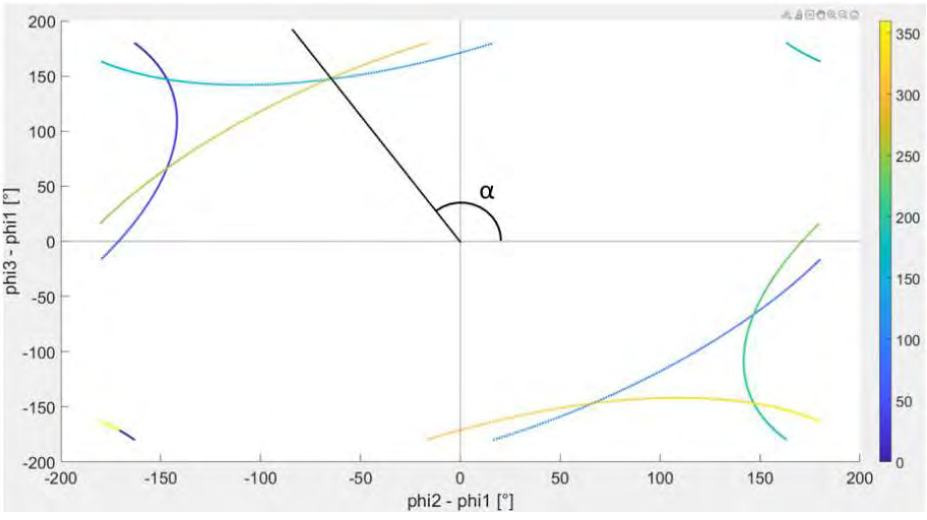


Figure 18 : Calcul de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ avec une distance entre les microphones plus grande que $\frac{\lambda}{2}$.

[2] La fonction trigonométrique atan2() expliquée au chapitre 9.4.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Avec une distance entre les microphones plus grande que $\frac{\lambda}{2}$, il n'est plus possible d'utiliser cette méthode pour récupérer l'angle. En effet, sur la Figure 18, un seul angle α indique plusieurs angles θ différents. C'est pourquoi, la distance entre les microphones dans ce projet est plus petite que $\frac{\lambda}{2}$.

En se basant sur l'observation faite à la Figure 17, un graphe a été effectué entre l'angle α et l'angle θ. Ce graphe est illustré en Figure 19.

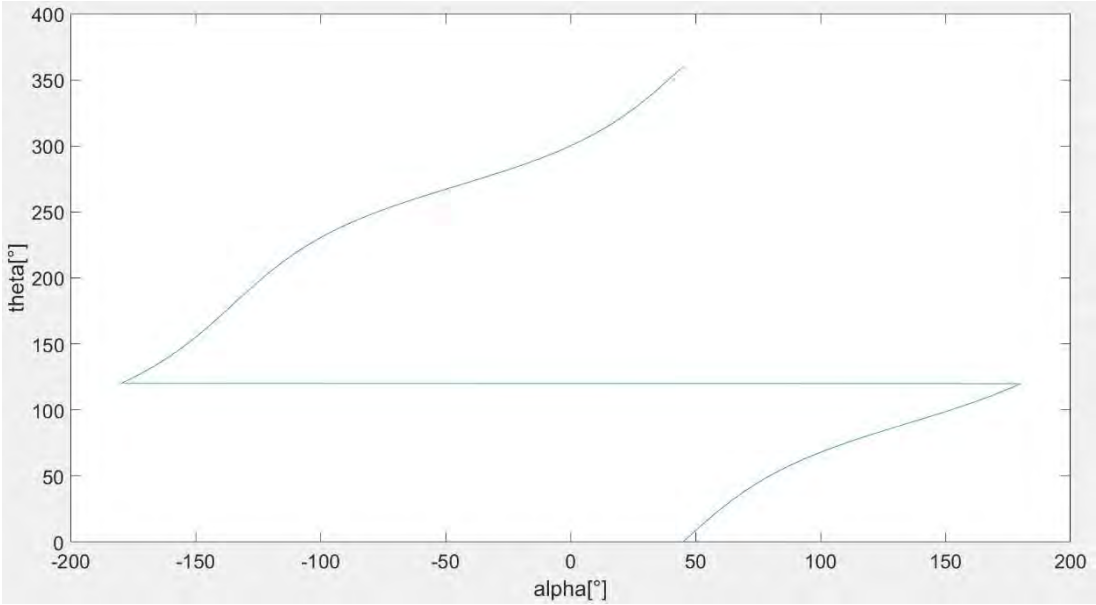


Figure 19 : Fonction $\theta = f(\alpha)$ avec α pas trié.

Si l'on trie le l'angle α par ordre croissant on trouve le graphe en Figure 20. L'algorithme de tri à bulles est utilisé.

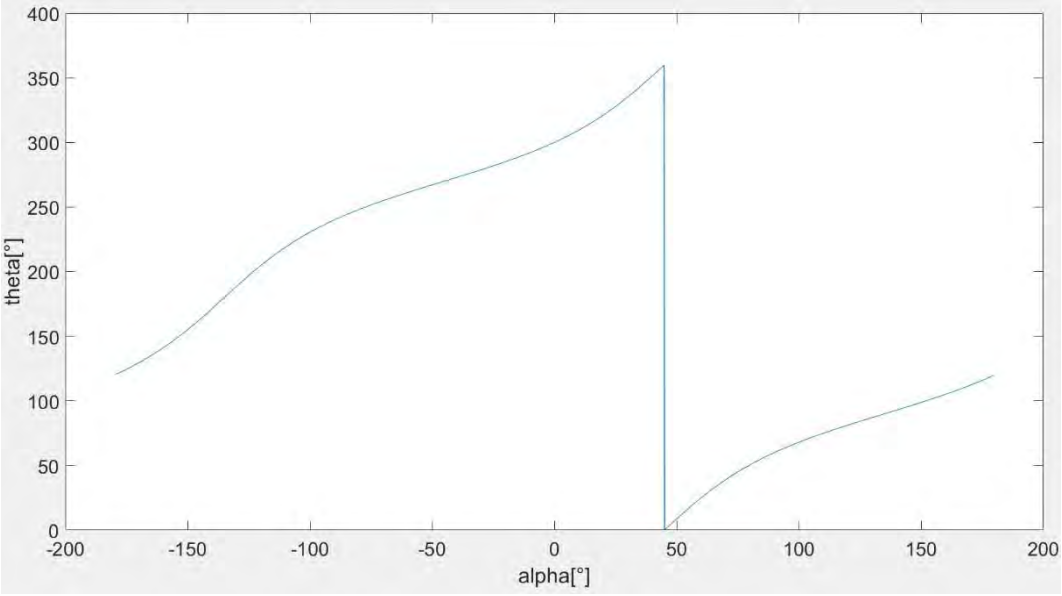


Figure 20 : Fonction $\theta = f(\alpha)$ avec α trié

Le graphe en Figure 20 est utilisé pour récupérer l'angle θ, lors d'une mesure. Lorsqu'une mesure est prise, les déphasages $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$ sont calculés. Avec ces déphasages, l'angle α correspondant est calculé. Cet angle est comparé à la théorie pour trouver l'angle θ correspondant.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

10 SIMULATION MICROPHONES SUR MATLAB

Ce chapitre traite des calculs fait sur Matlab pour simuler les microphones et trouver l'angle d'arrivée du son. La Figure 21 présente la séquence de calculs faites sur un script Matlab.

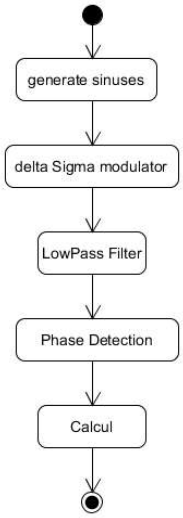


Figure 21 : Matlab séquence calculs

La première étape consiste à générer 3 sinus. Chaque sinus à la même période, mais une phase différente. La phase est calculée de la même façon qu'au chapitre 8.2. Une erreur aléatoire de 0 à 25° a été ajoutée sur les 3 phases pour simuler des erreurs de mesures.

Ensuite, chaque sinus passe dans un convertisseur sigma-delta d'ordre 1 avec 1 bit en sortie. Le signal de sortie est alors une modulation à densité d'impulsions. Il y a 100 échantillons par sinus, comme c'est le cas avec les microphones utilisés dans ce travail.

Finalement, les algorithmes utilisés en chapitre 9 sont utilisés pour déterminer l'angle θ.

Ces étapes ont été répétées pour des angles θ variant entre 0 et 360°. La Figure 22 présente les différences de phases $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$ obtenues et les différences de phases théoriques.

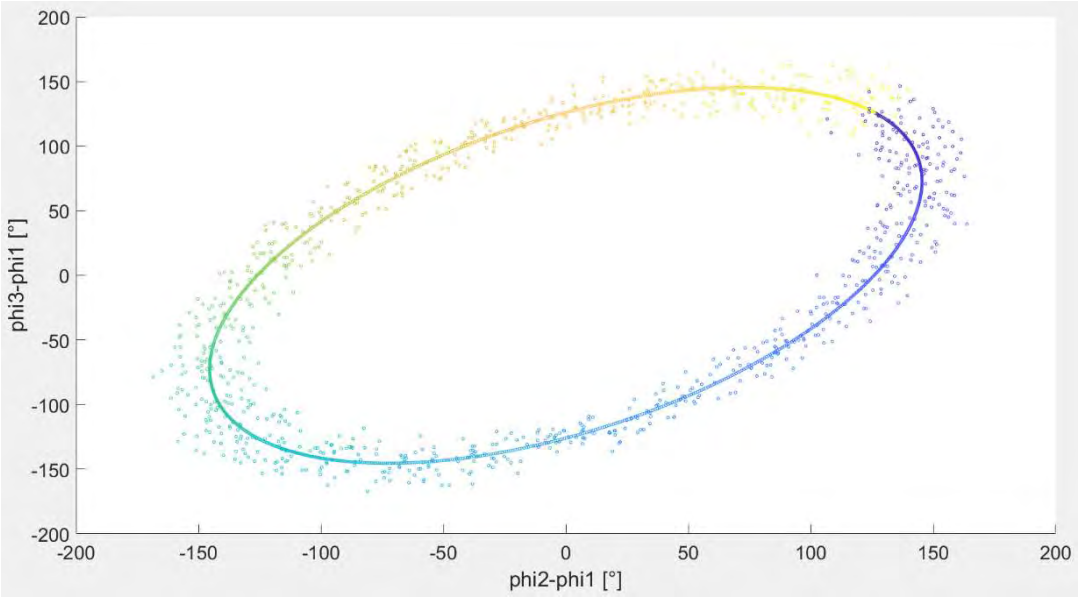


Figure 22 : Simulation de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

La Figure 23 présente l'angle θ obtenus comparé à l'angle θ théorique.

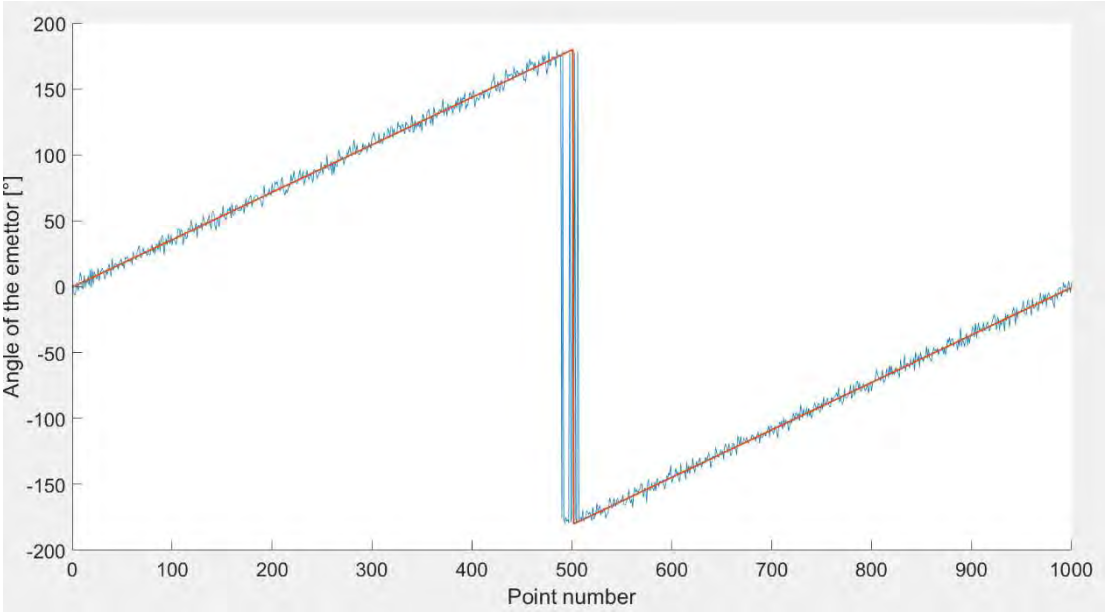


Figure 23 : Simulation : Comparaison angles θ obtenus Vs angles θ attendus

La courbe bleue représente les résultats obtenus et la courbe orange présente les résultats théoriques. Il y a très peu d'erreurs entre les résultats obtenus et les résultats théorique. Lorsque l'angle θ obtenus approche de 180°, il y a des sauts entre -180° et 180°. Cependant ce n'est pas un problème, car un angle de 180° et le même qu'un angle de -180°.

La Figure 24 présente l'erreur de calcul effectués.

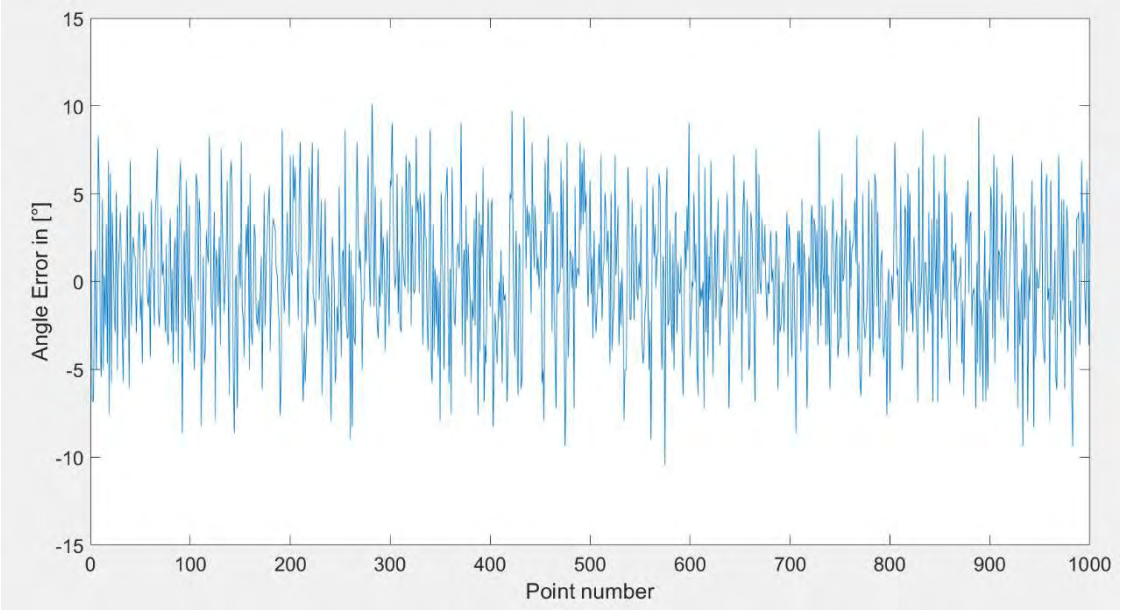


Figure 24 : Simulation : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 24, l'erreur obtenu varie entre -10° et 10° au maximum.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans le cadre de ce projet. L'erreur acceptable est de ±22.5°. C'est pourquoi, on peut conclure que les simulations sont fonctionnelles. De plus ces résultats ont été obtenus avec des erreurs de mesures allant jusqu'à 25°.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

10.1 Erreur de mesures

Le cahier des charges initiale, spécifie que 8 LEDs doivent être utilisées pour afficher la direction du son partenaire de plongée.

La Figure 25 illustre l'affichage de la direction de son partenaire de plongée.

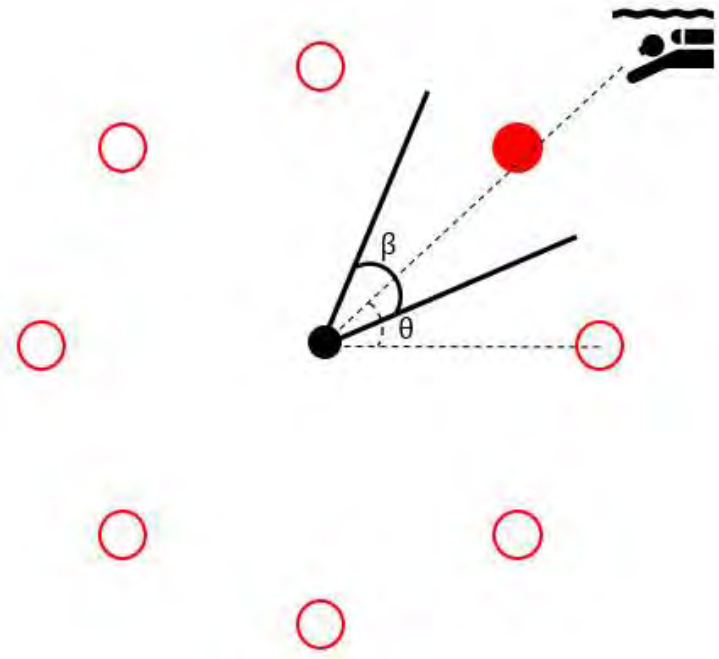


Figure 25 : Affichage de la direction du plongeur

Sur la Figure 25, il y a 8 LEDs qui sont disposées sur la périphérie d'un cercle. Chaque LED est espacée équitablement sur la périphérie du cercle. Ainsi, il y a une LED chaque 45°.

Sur la Figure 25, le partenaire de plongée se situe à un angle θ. L'angle θ à une valeur d'environ 40°. La LED étant à un angle de 45° s'allume donc pour indiquer la direction du plongeur. Lorsque cette LED est allumée, l'angle θ peut être dans l'intervalle suivant :

$$\left[45° - \frac{\beta}{2}, 45° + \frac{\beta}{2}\right], avec\ \beta = 45°$$

Ainsi, lorsqu'une LED est allumée, le plongeur peut se trouver dans un intervalle de 45°.

L'erreur de mesure acceptable est par conséquent de $\pm\frac{\beta}{2}$, soit ±22.5°.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11 DÉVELLOPEMENT PCB MICROPHONE

Cette partie traite du développement de la carte électronique contenant les microphones et qui s'interface avec la carte de développement "STM32F746G-Discovery".

11.1 Circuit imprimé V1.1

Un circuit électronique a été développé et s'interface avec la carte de développement STM32F746G-Discovery. Ce sont les connecteurs CN4 à CN7 qui servent à s'interfacer avec la carte de développement STM32F746G-Discovery. La Figure 26 représente les connecteurs présents sur cette carte.

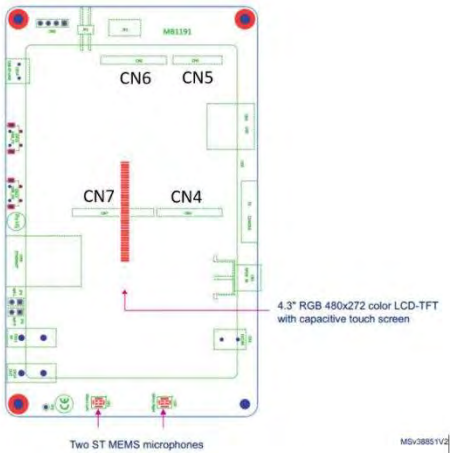


Figure 26 : Connecteurs sur la carte STM32F746G-Discovery (*source de l'image voire note de fin*[14])

Le circuit électronique développé comporte 3 microphones pour détecter l'angle d'arrivée du son à une fréquence de 40[kHz] et 3 autres microphones pour détecter l'angle d'arrivée du son à une fréquence de 1[kHz]. Les microphones ayant pour but de détecter l'angle d'arrivée d'un son à une fréquence de 40[kHz] sont disposés sur un cercle de 2[mm] de rayon. Les microphones ayant pour but de détecter l'angle d'arrivée du son à une fréquence de 1[kHz] sont disposés sur un cercle de 10[mm] de rayon. Ces derniers sont là dans le cas où la détection de son à une fréquence de 40[kHz] ne fonctionne pas.

La Figure 27 représente la face avant du PCB. Le cercle numéro 1 comporte les microphones pour la détection de l'angle d'arrivée du son à une fréquence de 1[kHz]. Le cercle numéro 2 comporte les microphones pour la détection d'un signal ultrasonique à une fréquence de 40[kHz]. La Figure 28 représente la face arrière du PCB.

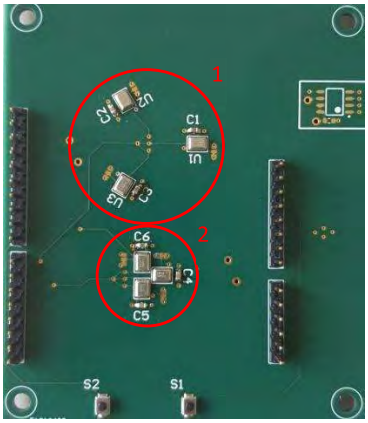


Figure 27 : PCB face avant

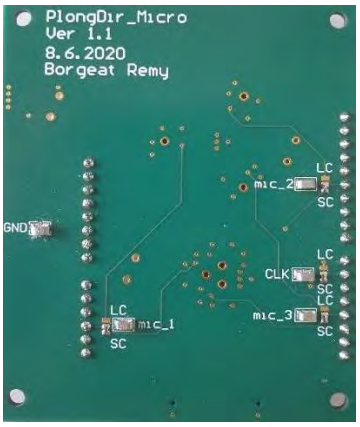


Figure 28 : PCB face arrière

Il faut noter que les rayons des cercles sur lesquels sont montés les microphones ont été choisis, afin que la distance entre les microphones est plus petite que $\frac{\lambda}{2}$. Ainsi, on évite le problème soulevé en chapitre 9.5.

Le dossier de fabrication de ce PCB se trouve en annexe.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.1.1 RÉCUPÉRATION DES DONNÉES DES MICROPHONES

Les données qui sortent des microphones doivent être récupérées et stockées en mémoire. Les données du microphone sont des bits cadencés à 4[MHz]. Pour récupérer les données, les interruptions sont inutilisables, car les données sont à trop haute fréquence.

En se référant à l'application note AN5027[15], des canaux de communications comme le SPI, I2S, SAI peuvent être utilisé pour transférer les données reçues par les microphones dans la mémoire.

Le SPI a été choisi, car sur les connecteurs de la carte STM32F746G-Disco, il y a 3 canaux de communication SPI disponible.

Les 3 transferts de données des microphones vers la mémoire doivent être faites de manière synchrone. En effet, ce sont les déphasages des signaux qui sont utiles pour connaître la direction du son. Or, si les transferts de données ne sont pas synchrones, cela va ajouter du déphasage entre les différents signaux et par conséquent fausser les résultats.

Le transfert des données depuis le SPI vers la mémoire peut se faire de 2 façons : avec des interruptions ou avec la DMA. Les interruptions ne peuvent pas être utilisées car elles seraient trop fréquentes. La taille des données du SPI peut être sélectionné entre 4 et 16 bits. Dans le cas où l'on emploie 16 bits, une interruption arriverait à une fréquence de 250[kHz], ce qui est beaucoup trop rapide pour le uContrôleur. La DMA est parfaitement adaptée pour des transferts de données à des fréquences élevées.

Dans un premier temps le SPI en mode maître a été utilisé avec des méthodes HAL fournie par ST. Lorsqu'il faut démarrer le transfert de données, chaque SPI est démarré l'un après l'autre, mais cela prend du temps et déphase donc les signaux. La Figure 29 représente le temps nécessaire à démarrer les 3 transferts de données. Une sortie est mise à l'état haut lorsque le premier transfert de données est démarré et elle est mise à l'état bas lorsque dernier transfert de données a été démarré. Activer et désactiver la sortie prend environ 400[ns]. Sur la Figure 29, le démarrage des transferts de données SPI vers la mémoire prends 1.4[us] ce qui déphase forcément le signal étant donné qu'on reçoit un bit chaque 250[ns].

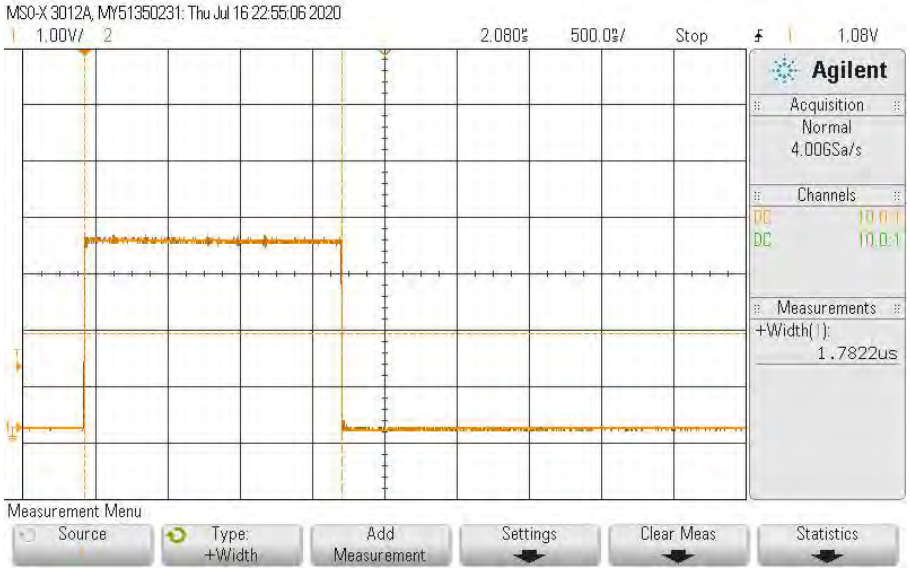


Figure 29 : Durée de démarrage des transfert DMA avec les méthodes HAL

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Pour limiter le temps de démarrage des transferts de données, les méthodes fournies HAL n'ont plus été utilisée. Les transferts de données DMA ont été directement démarrées en modifiant les bits des différents registres SPI. Sur la Figure 30, le temps de démarrage des SPI a été considérablement réduit, il dure uniquement 100[ns].

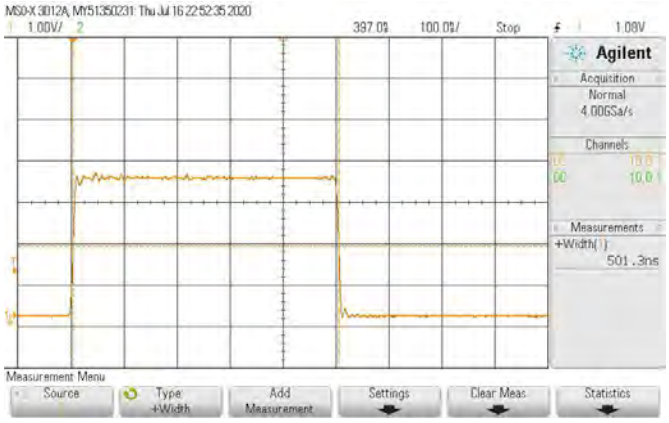


Figure 30 : Durée de démarrage des transfert DMA sans les méthodes HAL

Le temps de démarrage des transferts de données ne déphase donc pas les signaux. Cependant, un autre problème survient : il arrive que les signaux soient déphasés de 1 byte ou 2 bytes. Ce problème n'a pas été résolu, cependant une piste a été trouvée.

La Figure 31 présente le schéma bloc du périphérique SPI du uContrôleur. L'entrée MISO est utilisée pour récupérer les données des microphones.

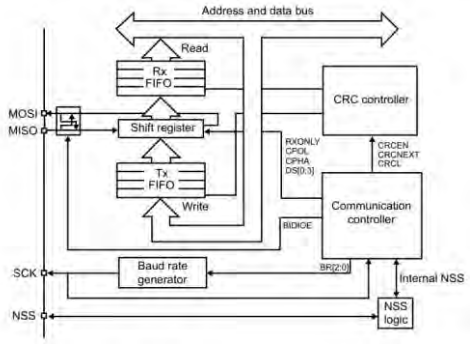


Figure 31 : Schéma bloc SPI (*source de l'image : voire de note de fin*[16])

À chaque coup du signal d'horloge, une donnée sur l'entrée MISO est lue et est déplacée dans un registre à décalage. Ce registre à décalage à une taille variant entre 4 à 16 bits dépendant de la taille des données du SPI. D'une fois que le registre à décalage est rempli, la donnée est déplacée dans une mémoire FIFO. C'est alors que la DMA vient récupérer les données et les stocker en mémoire.

Dans le programme écrit, le périphérique SPI est démarré et lorsque des mesures doivent être prise, le transfert de données DMA est démarré. Cependant, avant que le transfert de données DMA ne soit démarré, des données sont déjà lues et remplissent le registre à décalage et la mémoire FIFO. Le périphérique SPI est démarré avant la prise de mesure pour gagner du temps lors du démarrage du périphérique.

Juste avant que les transferts de données DMA soit démarré, la mémoire FIFO est vidée. Cependant, le registre à décalage n'est pas vidé et il est probable que cela entraîne des décalages entre les différents signaux. Ce problème n'a pas été réglé.

Finalement, la solution retenue est la suivante : employer le SPI en mode esclave et générer le signal d'horloge à l'aide d'une PWM. De cette façon le transfert de données commence de manière totalement synchrone.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.1.2 SCHÉMATIQUE

Les connecteurs CN4 à CN7 ont été utilisés afin de s'interfacer à la carte de développement STM32F746G-Disovery. Dans un premier temps, les problèmes rencontrés avec la communication SPI n'étaient arrivés et le PCB a été développé pour fonctionner avec le SPI en mode maître. Les sorties des microphones ont été connectées sur les SPIx_MISO du uContrôleur STM32F746NG.

Le Tableau 3 présente les connexions réalisées entre le circuit imprimé comportant les microphones et la carte de développement STM32F746G-Discovery.

SIGNAL	CONNEXION SUR LE STM32F746G-DISCOVERY	PÉRIPHÉRIQUE UCONTRÔLEUR
DONNÉES DU MICROS 1	CN5.4	SPI5_MISO
DONNÉES DU MICROS 2	CN7.5	SPI2_MISO
DONNÉES DU MICROS 3	CN4.4	SPI3_MISO
SIGNAL D'HORLOGE DES MICROPHONES	CN4.7	SPI5_CLK

Tableau 3 : Connexion à la carte STM32F746G-Discovery

La cartes des microphones est alimentées depuis la carte de développement STM32F746G-Discovery. Le circuit imprimé comportant les microphones est alimenté en 3.3[V]. Le Tableau 4 présente les connexions réalisées entre le circuit imprimé comportant les microphones et la carte de développement STM32F746G-Discovery, afin d'alimenter le circuit imprimé.

ALIMENTATION	CONNEXION SUR LE STM32F746G-DISCOVERY
3.3 [V]	CN6.4
0 [V]	CN6.6

Tableau 4 : Connexion alimentation

11.1.3 PLAN MÉCANIQUE

Il y 4 connecteurs femelles (CN4 à CN7) qui sont disponibles et utilisés pour s'interfacer avec la carte STM32F746G-Discovery. Les connecteurs mâles sur le circuit imprimé comportant les microphones doivent donc être placé mécaniquement juste, selon le plan mécanique ci-dessous (*plan mécanique : voire note de fin*[17]).

La Figure 33 représente le montage du PCB développé pour ce travail de bachelor sur la carte de développement STM32F746G-Discovery.

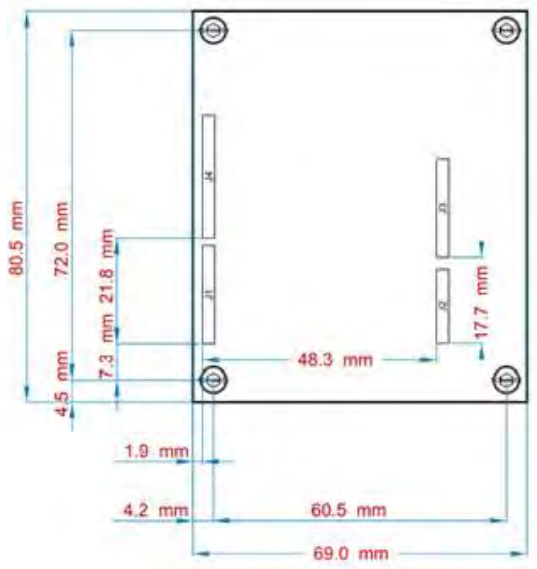


Figure 32 : Plan mécanique PCB

Figure 33 : Montage sur la carte STM32F746G-Discovery

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.1.4 Liste de matériel

La liste de matériel utilisé dans ce projet est présentée de le Tableau 5.

		MOUSER	MONNAIE
Quantité	**Référence**	**Désignation**	**Prix [CHF]**
6	721-SPH0641LU4H-1	Microphones MEMS DIG MEMs Mic, BP, Multimode,Ultrasonic	1,82
1	243-AMP2.2LN	Haut-parleurs et transducteurs Stereo amplifier 2 fixings lugs, 12VDC	35,30
1	665-AS04004MRN50WP-R	Speakers & Transducers SPEAKER 5W 4 OHM 84DB 160HZ	7,08
1	855-M20-9771046	Header pins 10 Pins, Pas 2.54mm	0,37
2	855-M20-9730846	Header pins 8 Pins, Pas 2.54mm	0,32
1	855-M20-9990645	Header pins 6 Pins, Pas 2.54mm	0,26
1	511-STM32F746G-DISCO	STM32F746G-Discovery	55,81

Tableau 5 : Liste de matériel

11.1.5 Modification PCB et carte STM32F746G-Discovery

Des modifications ont dû effectués sur la carte STM32F746G-Discovery et sur le PCB.

Le SPI était utilisé en mode maître, cependant avec les observations effectuées, le SPI doit être mis en mode esclave. Lorsque le SPI est en mode esclave, les entrées utilisées du uContrôleur sont SPIX_MOSI (les données des microphones) et SPIX_CLK (signal d'horloge du SPI). Lorsque le SPI est en mode esclave, c'est le maître qui lui fournit un signal d'horloge.

Le Tableau 6 présente les connexions réalisées entre le circuit imprimé comportant les microphones et la carte de développement STM32F746G-Discovery.

SIGNAL	CONNEXION SUR LE STM32F746G-DISCOVERY	PÉRIPHÉRIQUE UCONTRÔLEUR
Données du micro 1	CN5.3	SPI5_MOSI (PF9)
Données du micro 2	CN7.4	SPI2_MOSI (PB15)
Données du micro 3	CN4.0	SPI1_MOSI (PA7)
Signal d'horloge des microphones	CN7.2	TIM2_CH1 (PA15)
Signal d'horloge pour la communication SPI 1	CN4.7	SPI5_CLK (PH6)
Signal d'horloge pour la communication SPI 2	CN7.6	SPI2_CLK (PI1)
Signal d'horloge pour la communication SPI 3	CN4.1	SPI1_CLK (PA5)
Signal d'horloge pour les communications SPIX	CN4.3	TIM3_CH1 (PB4)

Tableau 6 : Connexion à la carte STM32F746G-Discovery

La cartes des microphones est alimentées depuis la carte de développement STM32F746G-Discovery. Le circuit imprimé comportant les microphones est alimenté en 3.3[V]. Le Tableau 7 présente les connexions réalisées entre le circuit imprimé comportant les microphones et la carte de développement STM32F746G-Discovery, afin d'alimenter le circuit imprimé.

ALIMENTATION	CONNEXION SUR LE STM32F746G-DISCOVERY
3.3 [V]	CN6.4
0 [V]	CN6.6

Tableau 7 : Connexion alimentaiton

La carte STM32F746G-Discovery a dû être modifiée, car les signaux SPI1_MOSI et SPI1_CLK ne sont pas connectés sur les connecteurs CN4 à CN7.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Sur la Figure 34, la résistance R66 qui est connectée à la pin PA7(SPI1_MOSI) a été dessoudée et la pastille de la R66 est connectée sur le connecteur CN4.0.

Sur la Figure 35, la broche CN4.1 est connectée sur un via. Ce via est connecté d'une part à la pin PA5 (SPI1_CLK) du uContrôleur et à la pin U15.1. La piste reliant le via à la pin U15.1 a été coupée.

De plus, les pistes des signaux connectés de base sur les broches CN4.0 et CN4.1 ont été coupées.

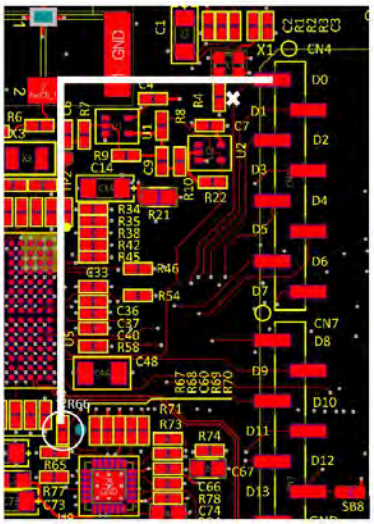


Figure 34 : Modification CN4.0 = SPI1_MOSI

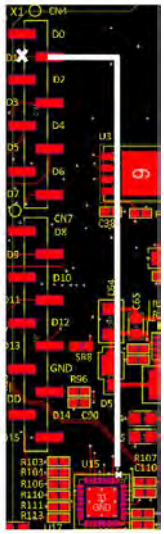


Figure 35 : Modification CN4.1 = SPI1_CLK

Les modifications faites sur le PCB comportant les microphones sont illustrées en Figure 36. Des fils ont été soudés afin d'établir les connexions adéquates.

Les modifications faites sur la carte STM32F746G-Discovery sont illustrées sur la Figure 37.

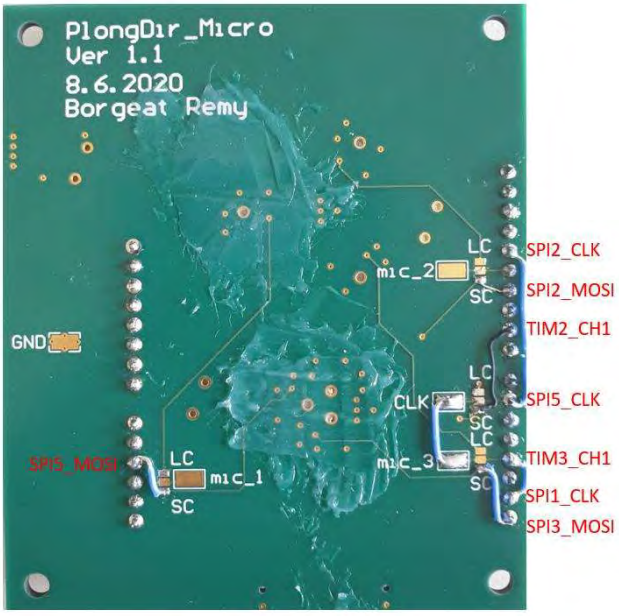


Figure 36 : Modification PCB

Figure 37 : Modification STM32F746G-Discovery

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.2 Circuit imprimé V2.0

Une version 2.0 du PCB a été développé par Mme. Andersson. Ce PCB comporte les mêmes composants et la même schématique que la version 1.1.

La Figure 38 présente la face arrière de la version 2.0 du circuit imprimé. La Figure 39 présente la face avant de la version 2.0 du circuit imprimé.

Figure 38 : PCB microphones V2.0 : Face arrière

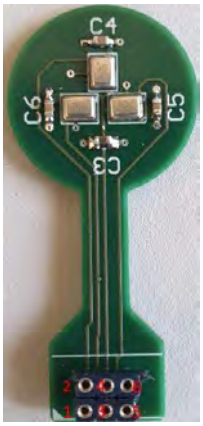


Figure 39 : PCB microphones V2.0 : Face avant

Bien que ce PCB comporte les mêmes composants que la version 1.1, il y a tout de même 3 différences entre la version 1.1 et la version 2.0.

La première différence est la taille du PCB. La version 2.0 du PCB est bien plus petite que la version 1.1 du PCB. La version 2.0 du PCB a été développé afin de pouvoir mettre le circuit imprimé dans un boîtier.

La deuxième différence est le fait que la version 2.0 du PCB ne s'interface pas directement avec la carte de développement STM32F746G-Discovery. En effet, la version 1.1 venait se fixer sur la carte de développement STM32F746G-Discovery. Ce n'est pas le cas avec la version 2.0. La carte de développement STM32F746G-Discovery est tout de même utilisée. Un câble assure la liaison entre le PCB V2.0 et la carte de développement STM32F746G-Discovery. La connectique du connecteur présent sur la version 2.0 du PCB est présentée dans le Tableau 8.

NUMÉRO DE LA PIN	FONCTION	CONNEXION SUR LE STM32F746G-DISCOVERY
1	3.3 [V]	CN6.4
2	Données micro 1	CN5.3
3	Données micro 2	CN7.4
4	Signal horloge micros	CN7.2
5	Données micros 3	CN4.0
6	Masse	CN6.6

Tableau 8 : Connectique de la version 2.0 du PCB

La dernière différence est le fait que sur la version 2.0 du PCB, il n'y a que 3 microphones qui sont monté pour fonctionner à 40[kHz].

Ce PCB sert de démonstrateur, les tests ont été effectués avec la version 1.1 du PCB.

Le dossier de fabrication de ce PCB se trouve en annexe.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.3 Ingénierie logicielle : sauvegarde des données des micros dans la carte SD

Ce chapitre traite de l'ingénierie logicielle. Chaque seconde, le programme récupère les données des microphones et les sauvegarde sur une carte SD pour ensuite faire les calculs de l'angle d'arrivée du son sur Matlab.

11.3.1 *Diagramme de séquence*

Ce chapitre présente les diagrammes de séquences utiles à la compréhension du code, d'autre diagrammes de séquence se trouvent en annexe.

La Figure 40 représente le diagramme de séquence de la classe "Controller".

Premièrement, les microphones doivent être démarrés. Pour démarrer les microphones, un signal d'horloge à 4[MHz] est injecté sur les entrées des microphones.

Les microphones comportent plusieurs modes de fonctionnement. Lorsqu'aucun signal d'horloge est appliqué sur les microphones, les microphones sont en mode repos. Dès qu'un signal d'horloge est appliqué les microphones, les microphones vont changer de mode de fonctionnement (dans notre cas, ils passent en mode ultrasonique). Cette transition d'états prend environ 10[ms] (*voir note de fin 3, chapitre 2*). C'est pour cela, que les mesures sont démarrées uniquement après "START_MIC_TIME [ms]".

Les mesures sont démarrées en injectant un signal d'horloge sur les entrées "clock" du SPI.

La DMA est utilisée pour transférer les données reçues sur le SPI dans la mémoire. D'une fois le transfert terminé (tableau pleins), une interruption est générée et la classe "Controller" va alors sauvegarder les données des micros sur la carte SD. Il y a 1 fichier texte par microphone.

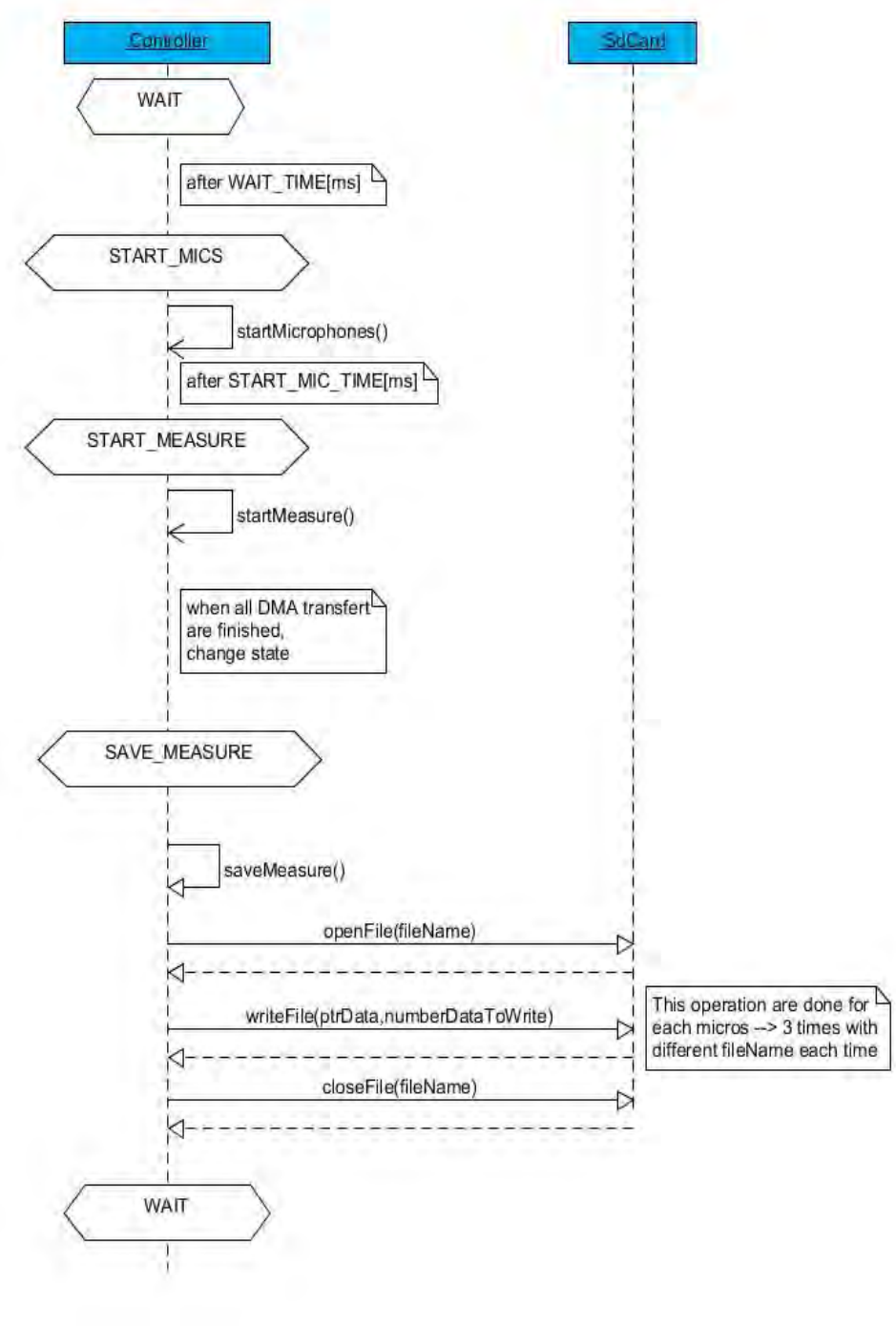


Figure 40 : Diagramme de séquence : Classe Controller

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.3.2 DIAGRAMME D'ÉTATS TRANSITIONS

La figure suivante présente la machine d'états transitions de la classe "Controller". La machine d'états démarre dans l'états "STATE_NULL", lorsqu'elle est démarrée, l'évènement "evInit" est généré et la machine d'états passe dans l'états "WAIT".

Lorsque les mesures sur tous les micros sont terminées, l'événement "evMeasureMicsFinish" est généré.

Pour plus de détails sur cette machine d'états transitions, voir chapitre 11.3.1.

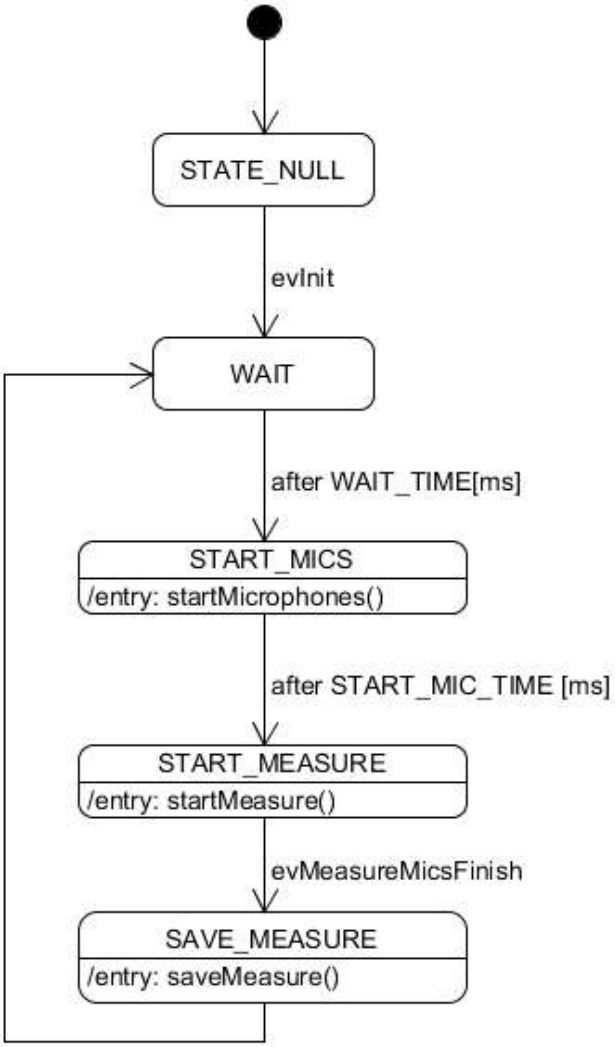


Figure 41 : Machine d'états transition : Classe "Controller"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.3.3 DIAGRAMME DE DÉPLOIEMENT

La Figure 42 présente le diagramme de déploiement de ce projet. L'objectifs de ce diagramme est de montrer les périphériques utilisés dans ce projet.

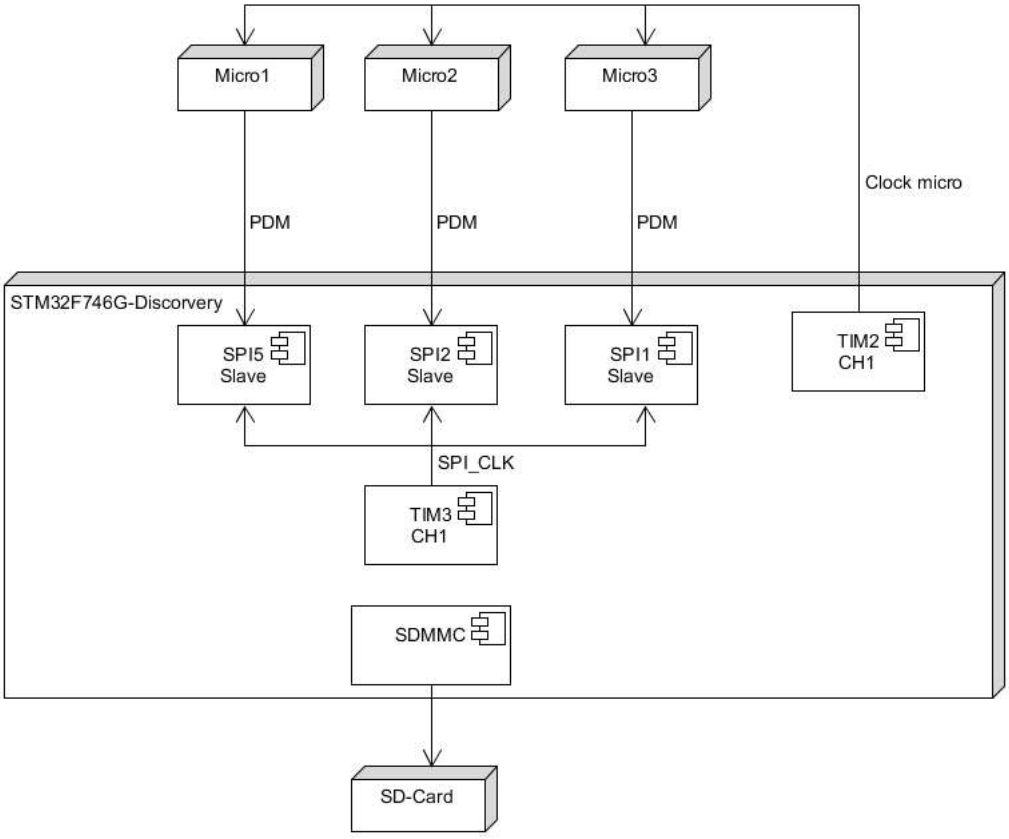


Figure 42 : Diagramme de déploiement

Sur la Figure 42, les périphériques du uContrôleur STM32F746NGH sont présentés de le Tableau 17.

PÉRIPHÉRIQUE	BUT
SPI1	Communication avec le micro 3
SPI2	Communication avec le micro 2
SPI5	Communication avec le micro 1
TIM2_CH1	Signal d'horloge des microphones
TIM3_CH1	Signal d'horloge des communication SPI
SDMMC	Sauvegarde de données sur la carte SD

Tableau 9 : Liste des périphériques de uContrôleur

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.3.4 DIAGRAMME DE CLASSE

La Figure 43 présente le diagramme de classe. Toutes les classes sont expliquées aux pages indiquées dans le Tableau 10.

CLASSE	PAGE
"CONTROLLER"	Page numéro 30
"FACTORY"	Page numéro 30
"SDCARD"	Page numéro 29
"SPICONTROLLER"	Page numéro 31

Tableau 10 : Liste des classes

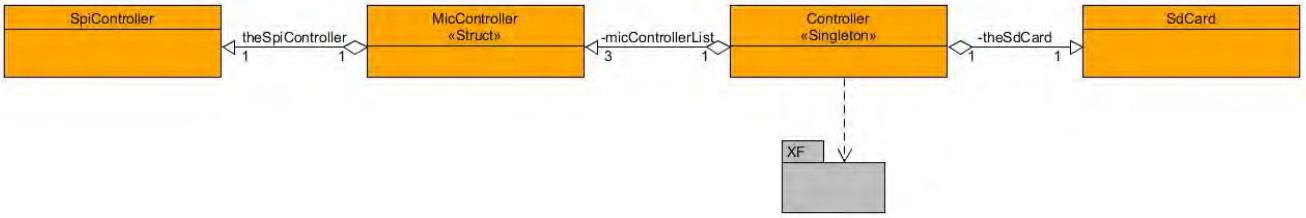


Figure 43 : Diagramme de classe

La Figure 44 représente la classe "Factory". La classe "Factory" crée les différentes instances des classes utiles dans le code. Elle permet aussi d'initialiser les différentes instances des classes et de construire les relations entre les classes.

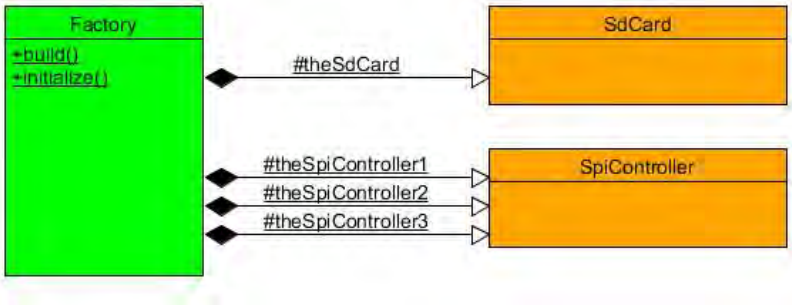


Figure 44 : Relations classe "Factory"

11.3.4.1 Classe "SdCard"

La classe "SdCard" permet de sauvegarder une chaîne de caractère dans un fichier texte sur une carte SD. Pour écrire des données sur la carte SD, il est nécessaire d'employer la méthode "openFile()" qui prend en paramètre le nom du fichier. Ensuite la méthode "writeFile()" va écrire une chaîne de caractère d'une certaine longueur sur la carte SD. Finalement, il est nécessaire de fermer le fichier en employant la méthode "closeFile()". Les attributs de cette classe, servent à gérer l'écriture sur la carte SD.

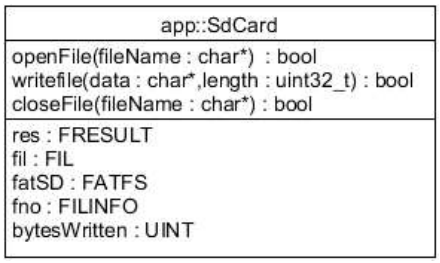


Figure 45 : Classe "SdCard"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.3.4.2 Classe "Factory"

La classe "Factory" crée les instances des différentes classes utilisées dans ce projet. De plus, elle permet d'initialiser les différents objets créés et finalement elle permet de construire les relations entre les objets.

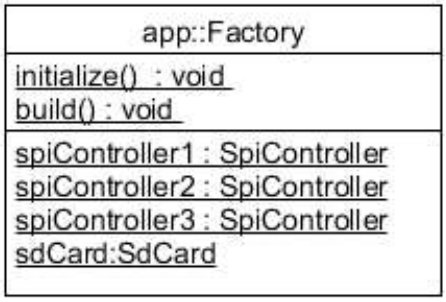


Figure 46 : Classe "Factory"

11.3.4.3 Classe "Controller"

La classe "Controller" permet de contrôler le programme. Chaque seconde, cette classe démarre la prise de mesures de microphones. D'une fois la quantité de mesures voulues prises, les mesures sont arrêtées et cette classe sauvegarde les données des mesures sur une carte SD.

Cette classe utilise un XF. Un XF est un système permettant la gestion de "timers" et d'événements. Ceci est utile lorsqu'une classe a le besoin d'implémenter une machine d'états transitions. Le XF permettra de gérer les événements et les "timers" et appellera la méthode implémentant la machine d'états transitions("processEvent()") de la classe "Controller" lorsque le "timer" a échoué ou qu'un événement est intervenu.

Cette classe comporte aussi une liste de "MicController". "MicController" est une structure stockant un pointeur vers l'instance de la classe qui gère la communication SPI et l'index du microphone sur le circuit imprimé. Dans cette liste, il y a 3 instances de la structure "MicController" pour les 3 communications SPI qui récupèrent les données des 3 microphones.

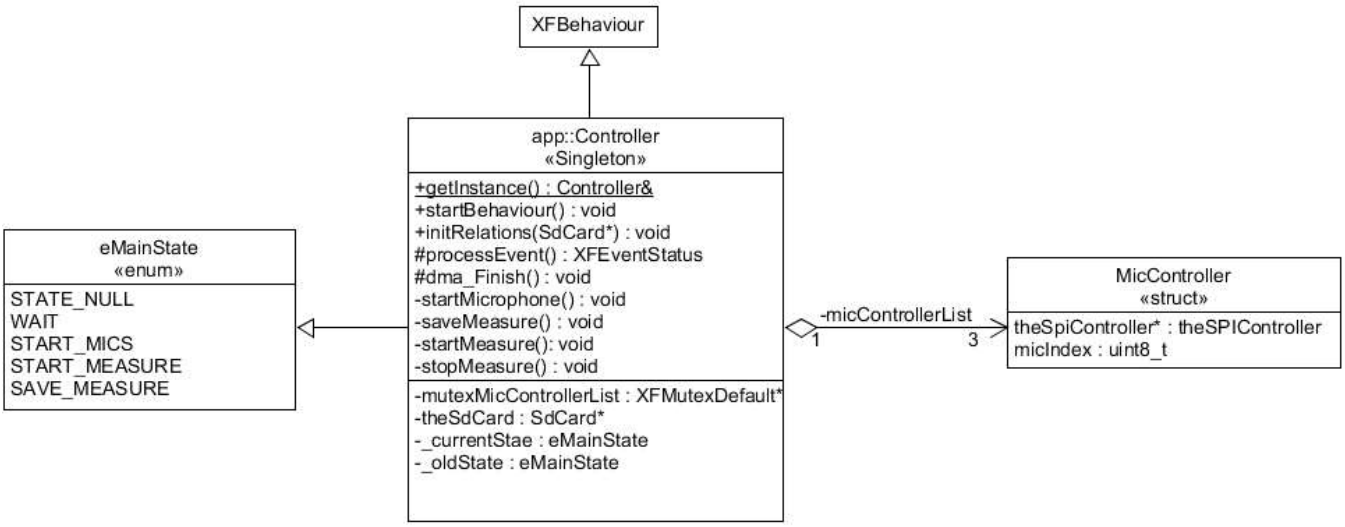


Figure 47 : Classe "Controller"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.3.4.4 Classe "SpiController"

La classe "SpiController" gère la communication SPI entre les microphones et le uContrôleur. Elle contient un pointeur vers les données stockées en mémoire ("data"). L'attribut "hspi" est un pointer vers une structure contenant les configurations de la communication SPI.

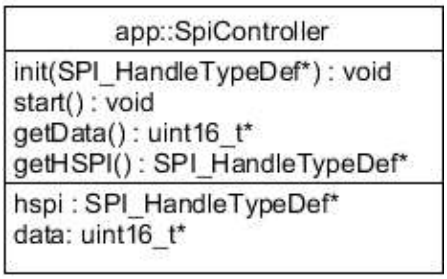


Figure 48 : Classe "SpiController"

11.4 Configuration périphérique

Ce chapitre traite de la configuration des périphériques du uContrôleur utilisés dans ce programme. Les périphériques utilisés sont présentés dans le Tableau 11.

PÉRIPHÉRIQUES	BUT
SPI	Communication avec les microphones
TIMER	Génération des PWM pour générer les signaux d'horloge du microphones
DMA	Transfert de données depuis les périphérique SPI vers la mémoire
SDMMC1	Sauvegarder les données des microphones dans la carte SD

Tableau 11 : Liste des périphériques utilisés

Les configurations des registres ont été faites avec STM32CubeMX. Les entrées/sorties des différents périphériques sont présentées dans le chapitre 11.1.5.

11.4.1 CONFIGURATION DE LA COMMUNICATION SPI

La configuration des communications SPI est présentée de le Tableau 12.

PARAMÈTRES	VALEURS
MODE	Esclave en réception uniquement
TAILLE DES DONNÉES	16 bits
PREMIER BIT	MSB
POLARITÉ DU SIGNAL D'HORLOGE (CPOL)	Bas
PHASE DU SIGNAL D'HORLOGE (CPHA)	1 flanc
CALCUL DU CRC	Désactivé

Tableau 12 : Configuration communication SPI

Le SPI travaille en mode esclave et ne fait que recevoir des données. La tailles de ces données est de 16 bits.

11.4.2 CONFIGURATION CARTE SD

Le Tableau 13 présente la configuration nécessaire pour l'écriture sur la carte SD.

PARAMÈTRES SDMMC1	VALEURS
MODE	SD 1 bits
PARAMÈTRES FATFS	**VALEURS**
MODE	SDCARD

Tableau 13 : Configuration des périphériques pour la carte SD

Le périphérique SDMMC1 est utilisé avec le mode SD 1 bit et pas avec le mode SD 4 bit. La sortie PC10 du uContrôleur est utilisée soit pour le signal d'horloge de la communication SPI 3 (SPI3_CLK) soit sur une ligne de données pour l'accès à la carte SD (SDMMC1_D2). Dans ce projet, la communication SPI 3 est utilisée pour récupérer les données des microphones. La ligne de données pour l'accès à la carte SD ne peut donc pas être utilisée. Cela n'impacte pas le fonctionnement de l'écriture sur la carte SD.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.4.3 CONFIGURATION DES PWM

Les "timers" ont pour but de générer un signal d'horloge à 4[MHz]. Ces signaux d'horloges sont utilisés pour les microphones et pour la communication SPI.

Les "timer" 2 et "timer" 3 sont sur le bus APB1. La fréquence du bus APB1 est de 96[MHz]. Il y a 2 paramètres à calculer pour obtenir un signal d'horloge à la fréquence voulue. Ces paramètres sont le prédiviseur et la période du compteur. Ces paramètres sont utilisés dans l'équation suivante pour déterminer le facteur de division.

$$facteur\ de\ divison = (prédiviseur + 1) * (période\ du\ compteur + 1) = \frac{fAPB1}{fClk} = \frac{96[MHz]}{4[MHz]} = 24$$

Le facteur de division doit être de 24. Il faut alors calculer un prédiviseur et la période du compteur de façon à obtenir un facteur de division de 24. En choisissant un prédiviseur de 2, la période du compteur est donnée d'après l'équation suivante.

$$période\ du\ compteur = \frac{fAPB1}{fClk\ *\ (prédiviseur + 1)} - 1 = \frac{96[MHz]}{4[MHz] * (2\ +\ 1)} - 1 = 7$$

Le Tableau 21 présente la configuration des périphériques des "timers".

PARAMÈTRES DU COMPTEUR	VALEURS
CANAL 1	Génération PWM CH1
PRÉDIVISEUR	2
PÉRIODE DU COMPTEUR	7
PARAMÈTRES DE LA PWM	**VALEURS**
IMPULSION	4

Tableau 14 : Configuration des "timers"

Afin de générer le signal d'horloge à 4[Mhz], les "timer" sont utilisés en mode PWM. Le paramètre impulsion sert à déterminer le rapport cycle de la PWM.

$$Impulsion = (Période\ du\ compteur + 1)$$ * rapport cyclique - 1= (7 + 1) * 0.5 = 4

Le paramètre impulsion a été configuré à une valeur de 4, ce qui donne le rapport cyclique suivant :

$$rapport\ cyclique = \frac{Impulsion}{période\ du\ compteur + 1} * 100 = \frac{4}{8} * 100 = 50\%$$

La Figure 49 présente le signal d'horloge généré par le "timer" 2.

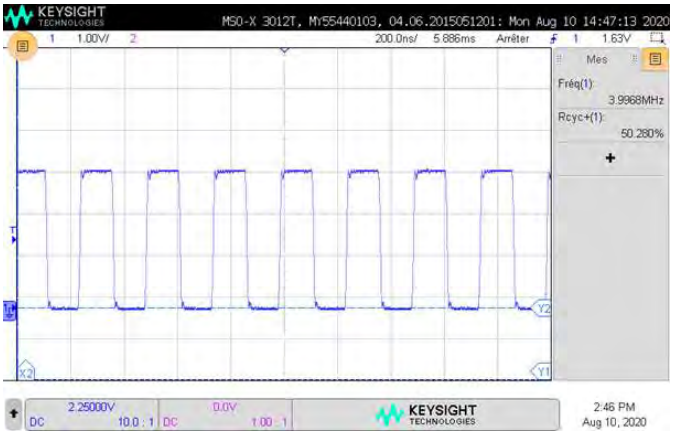


Figure 49 : Signal d'horloge généré.

La fréquence d'horloge voulue est de 4[MHz] et la fréquence d'horloge mesurée est de 3.99[MHz]. Le signal généré a donc la fréquence voulue. De plus, le rapport cyclique voulu est de 50% et le rapport cyclique mesuré est de 50.28%. Le signal généré a donc le rapport cyclique.

On peut conclure que le signal d'horloge généré par les "timers" est correct.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.4.4 CONFIGURATION DMA

La DMA est utilisée pour transférer les données des périphériques SPI directement vers la mémoire. Il y a 3 transferts DMA utilisés pour les 3 communications SPI utilisées. Le Tableau 15**Erreur ! Source du renvoi introuvable.** présente les transferts DMA utilisés.

REQUÊTE DMA	FLOT	DIRECTION
SPI1_RX	DMA2 Stream 2	Périphérique vers mémoire
SPI2_RX	DMA1 Stream 3	Périphérique vers mémoire
SPI5_RX	DMA2 Stream 3	Périphérique vers mémoire

Tableau 15 : Transferts DMA utilisés

Le Tableau 16 présente la configuration des transferts DMA.

PARAMÈTRES	VALEURS
MODE	Circulaire
INCRÉMENTATION D'ADRESSE	Mémoire
TAILLE DES DONNÉES	16 bits

Tableau 16 : Configuration communication SPI

Les transferts DMA utilisent le mode circulaire. La DMA rempli un tableau en mémoire, lorsque ce tableau est plein, en utilisant le mode circulaire, la DMA recommence à écrire des données sur le début du tableau. De plus, une interruption est générée lorsque le tableau en mémoire est plein.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.5 Ingénierie logicielle : Calcul sur STM32

Ce chapitre traite de l'ingénierie logicielle pour calculer l'angle d'arrivée du son. Ces calculs sont effectués, cette fois-ci, sur un uContrôleur.

11.5.1 Diagrammes de séquences

Ce chapitre présente les diagrammes de séquences utiles à la compréhension du code, d'autre diagrammes de séquence se trouvent en annexe.

La Figure 50 présente un diagramme de séquence qui illustre la prise de mesures et les calculs de l'angle d'arrivée du son.

Chaque "WAIT_TIME [ms]", les mesures et les calculs sont effectués.

Premièrement, les microphones doivent être démarrés. Pour démarrer les microphones, un signal d'horloge à 4[MHz] est injecté sur les entrées des microphones.

Les microphones comportent plusieurs modes de fonctionnement. Lorsqu'aucun signal d'horloge est appliqué sur les microphones, les microphones sont en mode repos. Dès qu'un signal d'horloge est appliqué sur les microphones, les microphones vont changer de mode de fonctionnement (dans notre cas, ils passent en mode ultrasonique). Cette transition d'états prend environ 10[ms] (*voir note de fin 3, chapitre 2*). C'est pour cela, que les mesures ne sont démarrées uniquement après "START_MIC_TIME [ms]".

Les mesures sont démarrées en injectant un signal d'horloge sur les entrées "clock" du SPI.

D'une fois que toutes les mesures ont été prises, la machine d'états transite dans l'états "CALCUL_DIRECTION". Cet états appel une méthode qui va effectuer les calculs nécessaires pour trouver l'angle d'arrivée du son. Cette méthode va effectuer les mêmes calculs que le script Matlab au chapitre 9.

Finalement, l'angle calculé est affiché sur l'écran LCD présents sur la carte STM32F746G-Discovery.

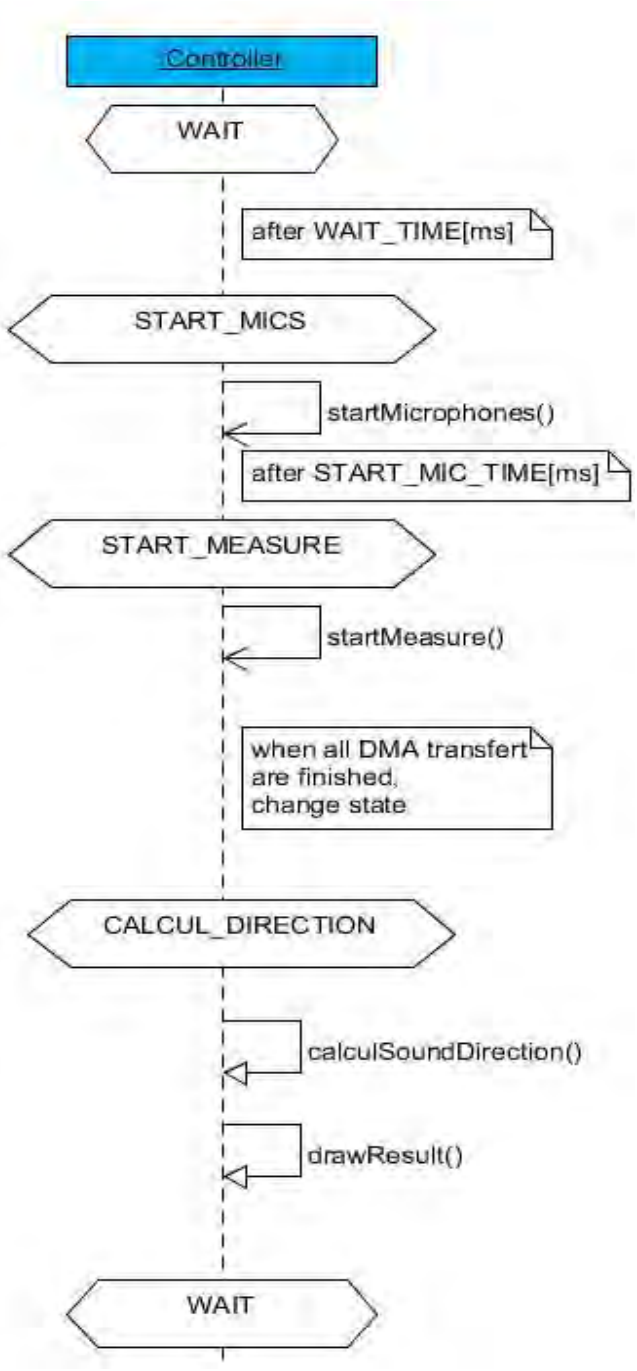


Figure 50 : Diagramme de séquence

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.5.2 DIAGRAMME D'ÉTATS TRANSITIONS

La Figure 51 présente la machine d'états transitions de la classe "Controller". Cette machine d'états démarre dans l'états "WAIT".

L'événement "evMeasureMicsFinish" est générer lorsque les mesures sur tous les micros sont terminées.

Pour plus d'explications sur cette machine d'états transitions, voir le chapitre 11.5.1.

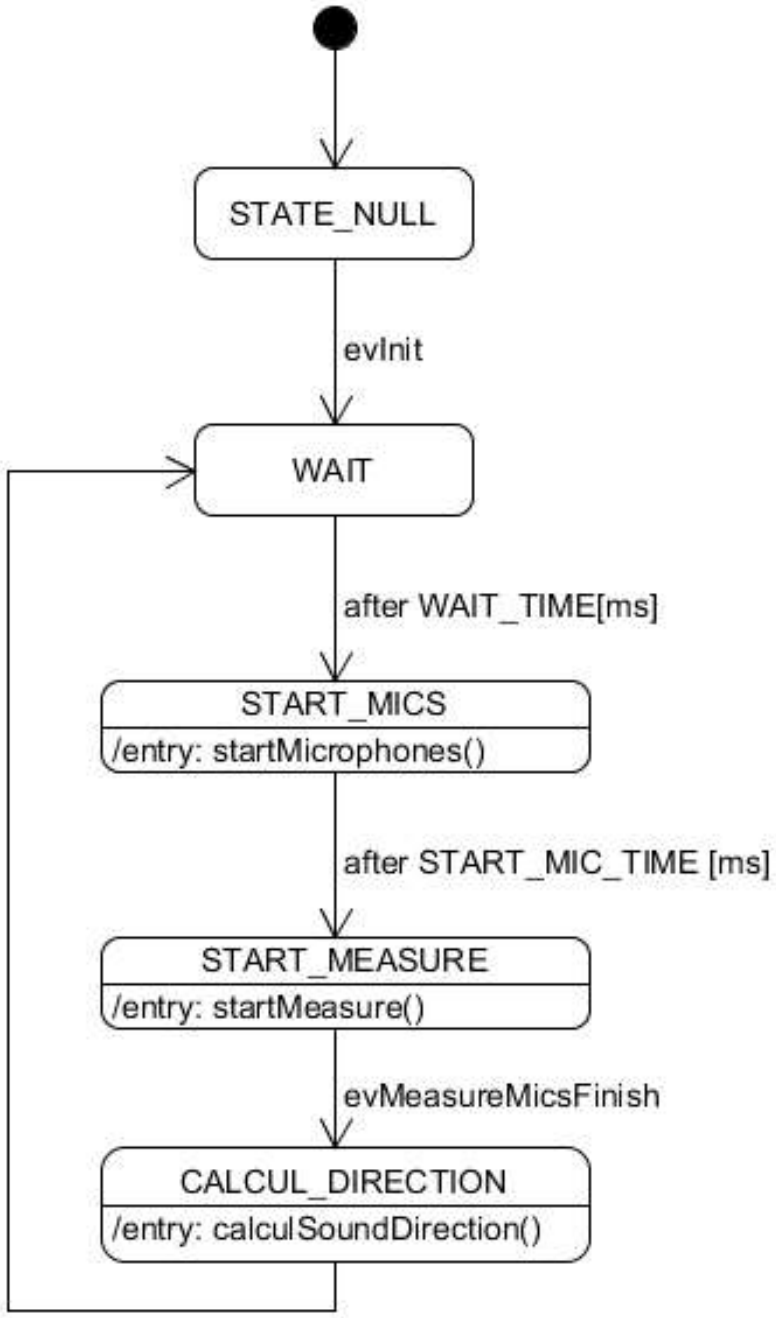


Figure 51 : Machine d'états transition : Classe "Controller"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.5.3 DIAGRAMME DE DÉPLOIEMENT

La Figure 52 présente le diagramme de déploiement de ce projet. L'objectifs de ce diagramme est de montrer les périphériques utiliser dans ce projet.

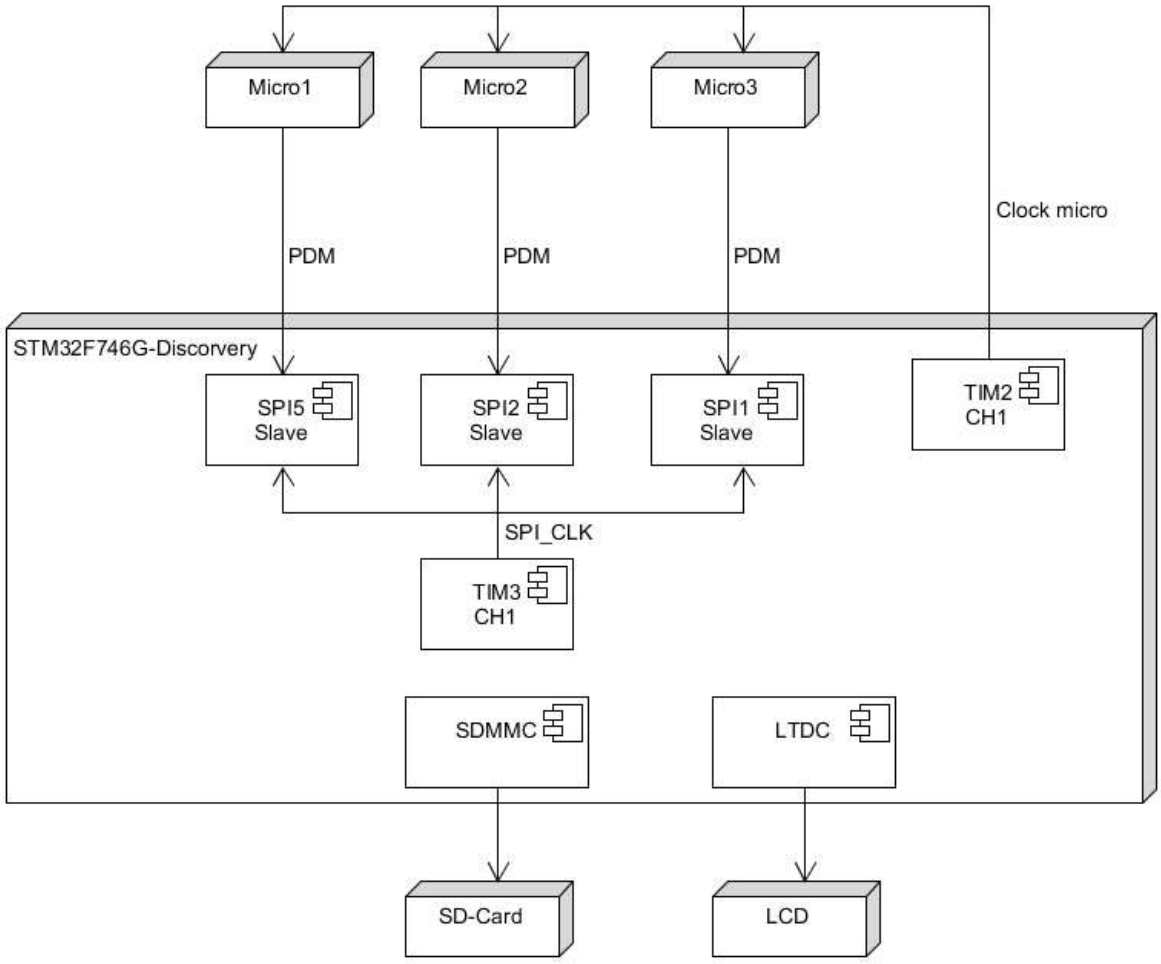


Figure 52 : Diagramme de déploiement

Sur la Figure 52, les périphériques du uContrôleur STM32F746NGH sont présenté de le Tableau 17.

PÉRIPHÉRIQUE	BUT
SPI1	Communication avec le micro 3
SPI2	Communication avec le micro 2
SPI5	Communication avec le micro 1
TIM2_CH1	Signal d'horloge des microphones
TIM3_CH1	Signal d'horloge des communication SPI
SDMMC	Sauvegarde de données sur la carte SD
LTDC	Affichage sur l'écran LCD

Tableau 17 : Liste des périphériques de uContrôleur

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.5.4 DIAGRAMME DE CLASSE

La Figure 53 présente le diagramme de classe. Toutes les classes du "package app" sont expliquées aux pages indiquées dans le Tableau 18.

CLASSE	PAGE
"CONTROLLER"	Page numéro 38
"FACTORY"	Page numéro 39
"GUI"	Page numéro 39
"SDCARD"	Page numéro 39
"POINT"	Page numéro 40
"SPICONTROLLER"	Page numéro 40
"LOWPASSFILTER"	Page numéro 40

Tableau 18 : Liste des classes

La librairie graphique ugfx (*voir note de fin*[18]) est utilisée pour l'affichage sur l'écran LCD.

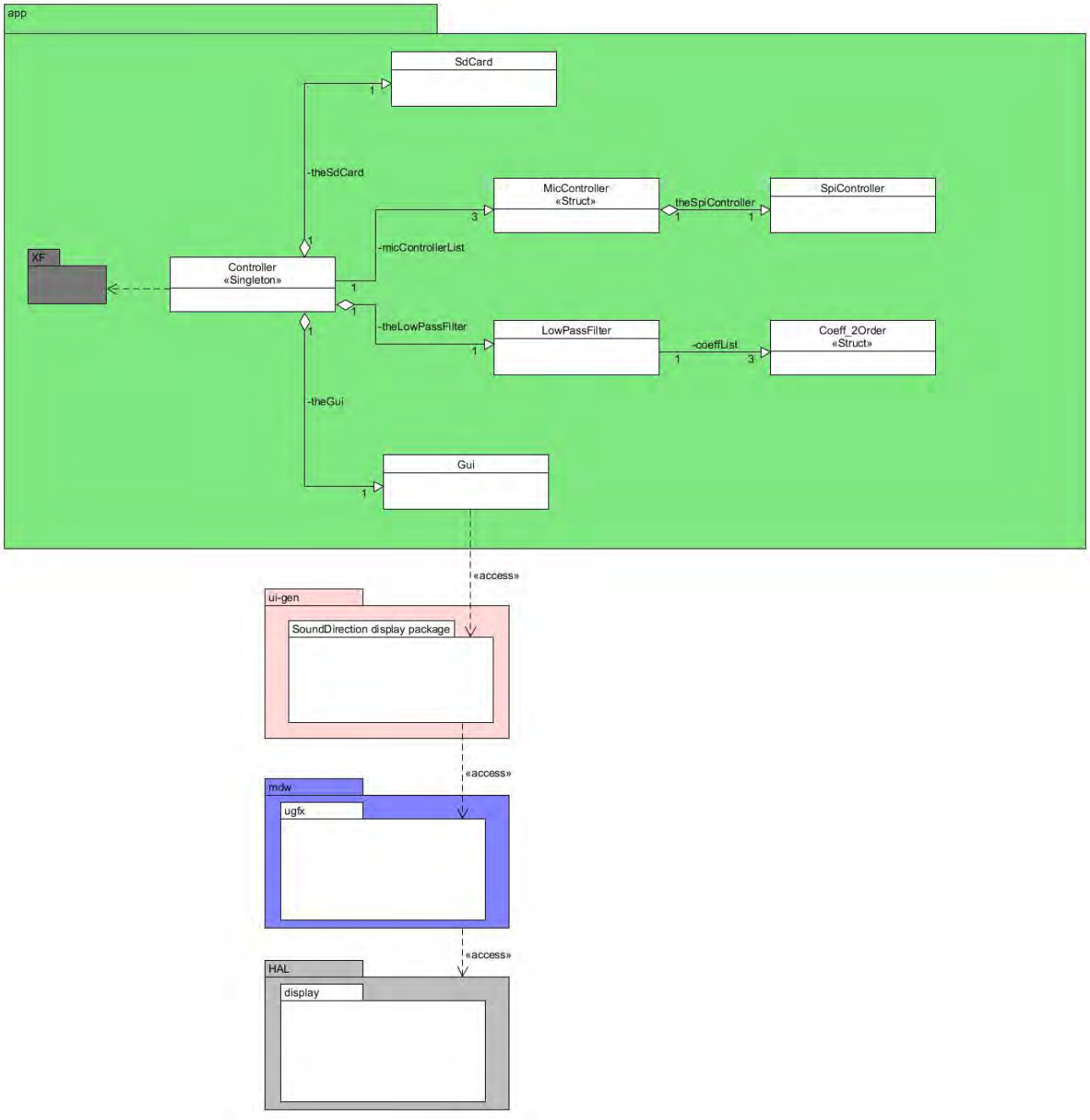


Figure 53 : Diagramme de classe

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

La Figure 54 représente la classe "Factory". La classe "Factory" crée les différentes instances des classes utiles dans le code.

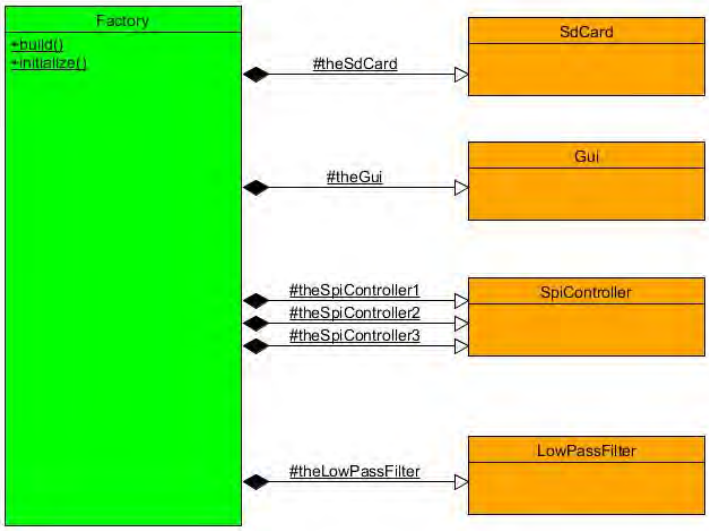


Figure 54 : Relations classe "Factory"

11.5.4.1 Classe "Controller"

La classe "Controller" est la classe qui gère le programme. Chaque seconde, cette classe démarre la prise de mesures de microphones. D'une fois la quantité de mesures voulues prises, les mesures sont arrêtées et cette classe calculs l'angle d'arrivée du son.

Cette classe utilise un XF. Un XF est un système permettant la gestion de "timers" et d'événements. Ceci est utile lorsqu'une classe a le besoin d'implémenter une machine d'états transition. Le XF permettra de gérer les événements et les "timers" et appellera la méthode implémentant la machine d'états transitions("processEvent()") de la classe "Controller" lorsque le "timer" a échoué ou qu'un événement est intervenu.

Cette classe comporte aussi une liste de "MicController". "MicController" est une structure stockant un pointeur vers l'instance de la classe qui gère la communication SPI et l'index du microphone sur le circuit imprimé. Dans cette liste, il y a 3 instances de la structure "MicController" pour les 3 communications SPI qui récupèrent les données des 3 microphones.

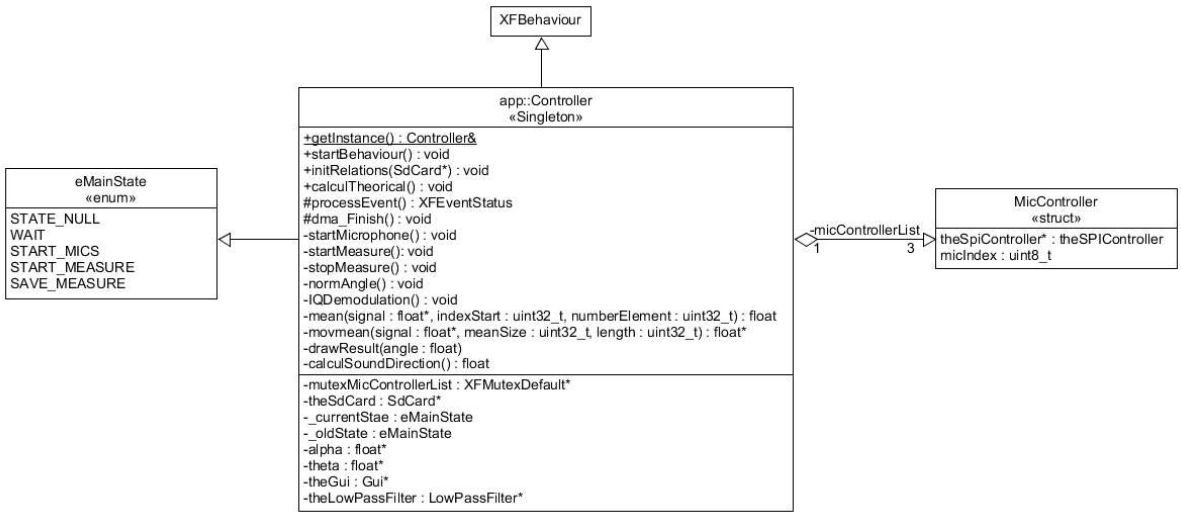


Figure 55 : Classe "Controller"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.5.4.2 Classe "Factory"

La classe "Factory" permet de créer les différentes instances des classes utilisées dans ce projets. De plus elle comporte 2 méthodes permettant d'initialiser les différents objets créer et de construire les relations entre les différents objets.

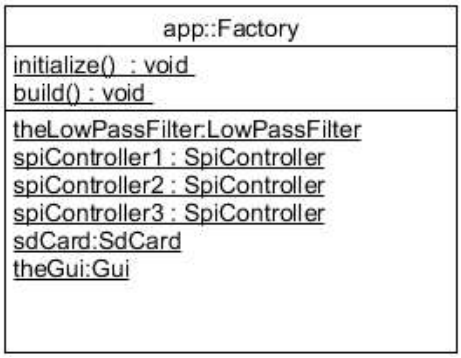


Figure 56 : Classe "Factory"

11.5.4.3 Classe Gui

La classe "Gui" permet l'affichage d'une ligne et d'un disque sur l'écran LCD. Pour afficher des éléments sur l'écran LCD, cette classe utilise la librairie graphique ugfx. Elle sert à afficher la direction de son partenaire de plongée.

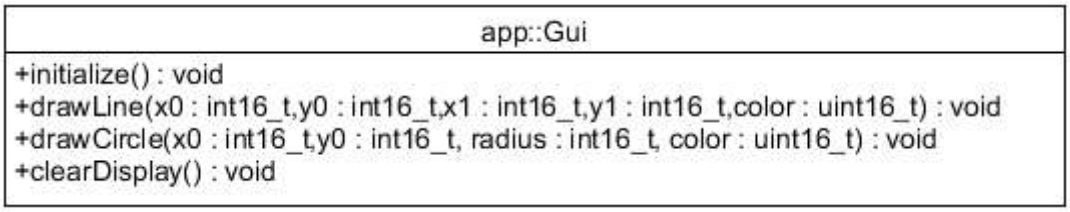


Figure 57 : Classe "Gui"

11.5.4.4 Classe SdCard

La classe "SdCard" permet de sauvegarder une chaîne de caractère dans un fichier texte sur une carte SD. Pour écrire des données sur la carte SD, il est nécessaire d'employer la méthode "openFile()" qui prend en paramètre le nom du fichier. Ensuite la méthode "writeFile()" va écrire une chaîne de caractère d'une certaine longueur sur la carte SD. Finalement, il est nécessaire de fermer le fichier en employant la méthode "closeFile()". Les attributs de cette classe, servent à gérer l'écriture sur la carte SD.

Cette classe est utilisée dans ce programme pour tester et débugger les différentes équations mathématiques.

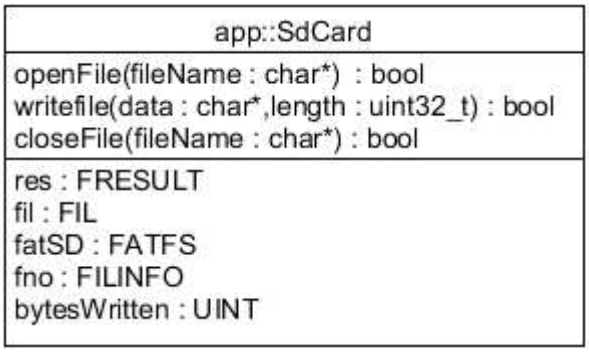


Figure 58 : Classe "SdCard"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.5.4.5 Classe Point

La classe "Point" est une classe représentant un point dans l'espace cartésien en 2 dimensions. Cette classe comporte ainsi une coordonnée x et une coordonnée y. La méthode "dist()" permet de déterminer la distance entre 2 points. Cette classe est utilisée dans les calculs théoriques pour déterminer les phases des microphones (voir chapitre 8.2)

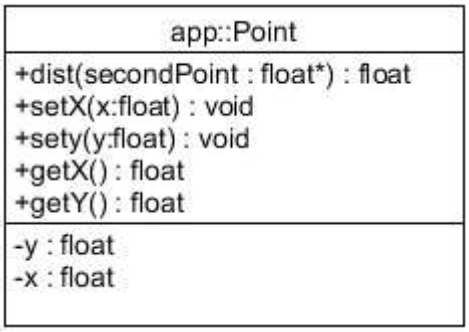


Figure 59 : Classe "Point"

11.5.4.6 Classe SpiController

La classe "SpiController" gère la communication SPI entre les microphones et le uContrôleur. Elle contient un pointeur vers les données stockées en mémoire ("data"). L'attribut "hspi" est un pointer vers une structure contenant les configurations de la communication SPI.

Figure 60 : Classe "SpiController"

11.5.4.7 Classe LowPassFilter

La classe "LowPassFilter" implémente un filtre passe-bas numérique. Ce filtre passe-bas a été dimensionné sur Matlab. Le filtre dimensionné sur Matlab a ensuite été divisé en section du 2ème ordre. Chaque section de 2ème ordre a été ajoutée dans la liste "coeffList" qui contient une toutes les sections du 2ème ordre du filtre.

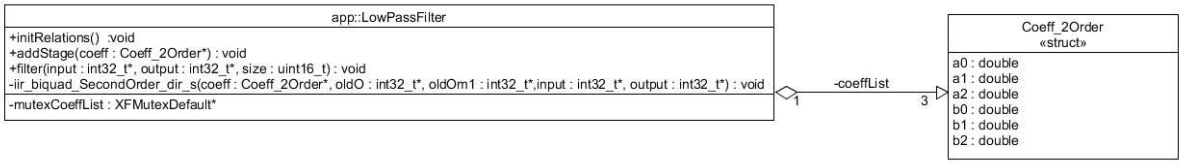


Figure 61 : Classe "LowPassFilter"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

11.6 Configuration périphérique

Ce chapitre traite de la configuration des périphériques du uContrôleur utilisés dans ce programme. Les périphériques utilisés sont présentés dans le Tableau 19.

PÉRIPHÉRIQUES	BUT
SPI	Communication avec les microphones
TIMER	Génération des PWM pour générer les signaux d'horloge du microphones
DMA	Transfert de données depuis les périphérique SPI vers la mémoire
SDMMC1	Sauvegarder les données des microphones dans la carte SD
LTDC	Affichage sur l'écran LCD

Tableau 19 : Liste des périphériques utilisés

Les configurations des registres ont été faites avec STM32CubeMX. Les entrées/sorties des différents périphériques sont présentées dans le chapitre 11.1.5.

11.6.1 CONFIGURATION DE LA COMMUNICATION SPI

La configuration des communications SPI est présentée de le Tableau 20.

PARAMÈTRES	VALEURS
MODE	Esclave en réception uniquement
TAILLE DES DONNÉES	16 bits
PREMIER BIT	MSB
POLARITÉ DU SIGNAL D'HORLOGE (CPOL)	Bas
PHASE DU SIGNAL D'HORLOGE (CPHA)	1 flanc
CALCUL DU CRC	Désactivé

Tableau 20 : Configuration communication SPI

Le SPI travaille en mode esclave et ne fait que recevoir des données. La tailles de ces données est de 16 bits.

11.6.2 CONFIGURATION DES PWM

Les "timers" ont pour but de générer un signal d'horloge à 4[MHz]. Ces signaux d'horloges sont utilisés pour les microphones et pour la communication SPI.

Les "timer" 2 et "timer" 3 sont sur le bus APB1 du uContrôleur. La fréquence de ce bus APB1 est de 96[MHz]. Il y a 2 paramètres à calculer pour obtenir un signal d'horloge à la fréquence voulue. Ces paramètres sont le prédiviseur et la période du compteur. Ces paramètres sont utilisés dans l'équation suivante pour déterminer le facteur de division.

$$facteur\ de\ divison = (prédiviseur + 1) * (période\ du\ compteur + 1) = \frac{fAPB1}{fClk} = \frac{96[MHz]}{4[MHz]} = 24$$

Le facteur de division doit être de 24. Il faut alors calculer un prédiviseur et la période du compteur de façon à obtenir un facteur de division de 24. En choisissant un prédiviseur de 2, la période du compteur est donnée d'après l'équation suivante.

$$période\ du\ compteur = \frac{fAPB1}{fClk\ *\ (prédiviseur + 1)} - 1 = \frac{96[MHz]}{4[MHz] * (2\ +\ 1)} - 1 = 7$$

Le Tableau 21 présente la configuration des périphériques des "timers".

PARAMÈTRES DU COMPTEUR	VALEURS
CANAL 1	Génération PWM CH1
PRÉDIVISEUR	2
PÉRIODE DU COMPTEUR	7
PARAMÈTRES DE LA PWM	**VALEURS**
IMPULSION	4

Tableau 21 : Configuration des "timers"

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Afin de générer le signal d'horloge à 4[Mhz], les "timer" sont utilisés en mode PWM. Le paramètre impulsion sert à déterminer le rapport cycle de la PWM.

$$Impulsion = (Période\ du\ compteur + 1) * \text{rapport cyclique} - 1 = (7 + 1) * 0.5 = 4$$

Le paramètre impulsion a été configuré à une valeur de 4, ce qui donne le rapport cyclique suivant :

$$rapport\ cyclique = \frac{Impulsion}{période\ du\ compteur + 1} * 100 = \frac{4}{8} * 100 = 50\%$$

La Figure 62 présente le signal d'horloge généré par le "timer" 2.

Figure 62 : Signal d'horloge généré

La fréquence d'horloge voulue est de 4[MHz] et la fréquence d'horloge mesurée est de 3.99[MHz]. Le signal généré a donc la fréquence voulue. De plus, le rapport cyclique voulu est de 50% et le rapport cyclique mesuré est de 50.28%. Le signal généré a donc le rapport cyclique.

On peut conclure que le signal d'horloge généré par les "timers" est correct.

11.6.3 CONFIGURATION CARTE SD

Le Tableau 22 présente la configuration nécessaire pour l'écriture sur la carte SD.

PARAMÈTRES SDMMC1	VALEURS
MODE	SD 1 bits
PARAMÈTRES FATFS	**VALEURS**
MODE	SDCARD

Tableau 22 : Configuration des périphériques pour la carte SD

Le périphérique SDMMC1 est utilisé avec le mode SD 1 bit et pas avec le mode SD 4 bit. La sortie PC10 du uContrôleur est utilisée soit pour le signal d'horloge de la communication SPI 3 (SPI3_CLK) soit sur une ligne de données pour l'accès à la carte SD (SDMMC1_D2). Dans ce projet, la communication SPI 3 est utilisée pour récupérer les données des microphones. La ligne de données pour l'accès à la carte SD ne peut donc pas être utilisée. Cela n'impacte pas le fonctionnement de l'écriture sur la carte SD.

11.6.4 CONFIGURATION DMA

La DMA est utilisée pour transférer les données des périphériques SPI directement vers la mémoire. Il y a 3 transferts DMA utilisés pour les 3 communications SPI utilisées. Le Tableau 23 illustre les transfert DMA.

REQUÊTE DMA	FLOT	DIRECTION
SPI1_RX	DMA2 Stream 2	Périphérique vers mémoire
SPI2_RX	DMA1 Stream 3	Périphérique vers mémoire
SPI5_RX	DMA2 Stream 3	Périphérique vers mémoire

Tableau 23 : Transferts DMA utilisés

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Le Tableau 24 présente la configuration des transferts DMA.

PARAMÈTRES	VALEURS
MODE	Circulaire
INCRÉMENTATION D'ADRESSE	Mémoire
TAILLE DES DONNÉES	16 bits

Tableau 24 : Configuration communication SPI

Les transferts DMA utilisent le mode circulaire. La DMA rempli un tableau en mémoire, lorsque ce tableau est plein, en utilisant le mode circulaire, la DMA recommence à écrire des données sur le début du tableau. De plus, une interruption est générée lorsque le tableau en mémoire est plein.

11.6.5 CONFIGURATION ÉCRAN LCD

L'écran LCD présent sur la carte de développement STM32F746G-Discovery est utilisé dans ce projet, afin d'afficher la direction de son partenaire de plongée. La taille des cet écran LCD est de 480 pixels par 272 pixels. Les figures suivantes présentes les configurations faites pour l'affichage sur l'écran LCD.

Figure 63 : Configuration gfx

Figure 64 : Configuration périphérique LTDC

Figure 65 : LTDC configuration des couches

Afin de pouvoir afficher des données sur l'écran LCD, la SDRAM1 est utilisée. Les Figure 66 et Figure 67 présentent les configurations de la SDRAM1.

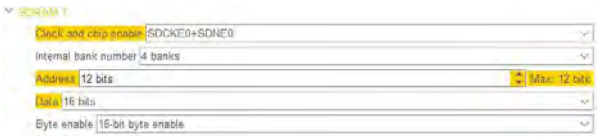


Figure 66 : Configuration périphérique SDRAM 1/2

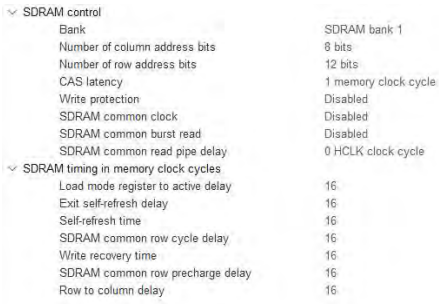


Figure 67 : Configuration périphérique SDRAM 1 2/2

La DMA2D est aussi utilisée. La Figure 68 présente la configuration de la DMA2D. La DMA2D est utilisée pour accélérer l'affichage sur l'écran LCD.

Figure 68 : Configuration périphérique DMA2D

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

IV. TESTS ET RÉSULTATS

Cette partie du rapport présente les tests et les résultats obtenus. Plusieurs tests ont été effectués et sont présentés dans cette partie du rapport. Ces tests sont présentés dans le Tableau 25.

TEST	CHAPITRE
CONSOMMATION DES ÉMETTEURS ULTRASONIQUES	Chapitre 12
RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON DANS L'AIR	Chapitre 14
ÉMETTEUR ULTRASONIQUE OBSTRUÉ AVEC DU SILICONE	Chapitre 15
DISTANCE MAXIMALE D'ÉMISSION	Chapitre 16
ÉMETTEUR ULTRASONIQUE PLUS HAUT QUE LES MICROPHONES	Chapitre 17
DIRECTIVITÉ DES ÉMETTEURS ULTRASONIQUES	Chapitre 18
TEST AVEC DES BOÎTIER	Chapitre 19
CALCUL SUR UCONTRÔLEUR	Chapitre 20

Tableau 25 : Liste des tests effectués

Une synthèse des résultats obtenus est présenté dans le chapitre 21.

12 COMSOMMATION DES ÉMETTEURS ULTRASONIQUES

2 émetteurs ultrasoniques peuvent être utilisés dans ce projet : UT-1240K-TT-R et UT-1640K-TT-2-R.

Des tests sur la consommation de ces composants doivent être effectués, car la datasheet n'indique pas la consommation de ce dernier. Il est nécessaire de connaître la consommation des émetteurs ultrasoniques afin de connaître le circuit électronique pour piloter le composant.

Le schéma de mesure est le même pour les 2 composants. Ce schéma est présenté en Figure 69.

Un générateur de fonctions est utilisé pour injecter un signal sinusoïdal sur l'émetteur ultrasonique. Le générateur de fonction possède une résistance interne $Rin = 50[\Omega]$.

Le schéma équivalent de l'émetteur ultrasonique est représenté sur la Figure 69. L'émetteur ultrasonique entre en résonnance à la fréquence de $40[kHz] \pm 1[kHz]$ voir fiches techniques $(1\ et\ 2)$. L'émetteur ultrasonique est utilisé à sa fréquence de résonnance. À la fréquence de résonnance, les composants C et L s'annule, il reste uniquement la résistance R (le condensateur Cp est ignoré). On peut donc calculer la consommation de ce composant. Pour les mesures, la tension Vsc est mesurée hors-résonance et en résonnance.

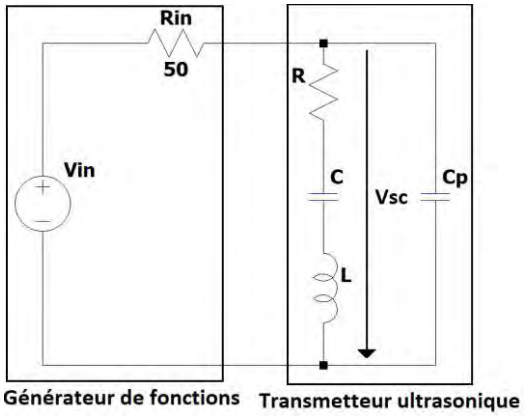


Figure 69 : schéma équivalent émetteur ultrasonique

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Le matériel utilisé pour ce test présenté dans le Tableau 26.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
OSCILLIOSCOPE	KEYSIGHT InfiniiVision MSOX3012T
ÉMETTEUR ULTRASONIQUE	UT-1240K-TT-R
ÉMETTEUR ULTRASONIQUE	UT-1640K-TT-2-R

Tableau 26 : Test de la consommation des émetteurs ultrasoniques : Matériel utilisé

Le test de consommation du composants UT-1240K-TT-R sont effectué en chapitre 12.1

Les tests de consommation du composant UT-1640K-TT-2-R sont effectué en chapitre 12.2.

Une synthèse des résultats obtenus est présenté dans le chapitre 12.3.

12.1 UT-1240K-TT-R

Les 2 figures ci-dessous, présente le résultat des mesures effectuées.

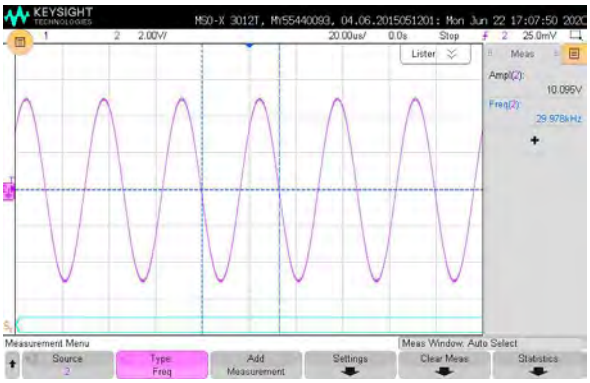


Figure 70 : Vsc hors de la fréquence de résonnance

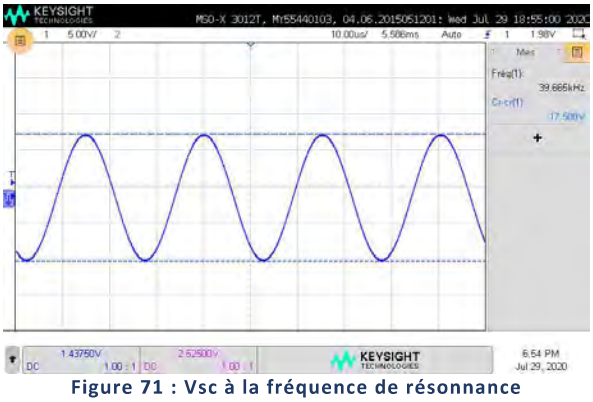


Figure 71 : Vsc à la fréquence de résonnance

La Figure 70 représente la tension Vsc lorsque l'on n'est pas en résonnance, la Figure 71 représente la tension Vsc en résonnance.

La résonnance de ce composant se situe à environ 40 [kHz]. Lors de la résonnance, l'amplitude de la tension Vsc vaut 8.8[V] soit une perte de 1.2[V] par rapport à l'amplitude de la tension hors résonnance.

La résistance R de l'émetteur ultrasonique peut donc être calculée :

$$\frac{R}{R + Rin} * Vin = Vsc$$

$$R * Vin = Vsc * (R + Rin)$$

$$R = \frac{Rin * Vsc}{Vin - Vsc} = \frac{50[\Omega] * 8.8[V]}{10[V] - 8.8[V]} = 366.6[\Omega]$$

La valeur efficace du courant consommé par le composant est calculée dans l'Équation 11.

$$I_{rms} = \frac{Vrms}{Rin + R} = \frac{\frac{Vin}{\sqrt{2}}}{Rin + R} = \frac{\frac{10[V]}{\sqrt{2}}}{50[\Omega] + 366.6[\Omega]} = 16.9\ [mA]$$

Équation 11 : Courant consommé par le composant

Le courant consommé est faible, un amplificateur à transistor peut donc être utilisé pour piloter ce composant.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

12.2 UT-1640K-TT-2-R

Les 2 figures ci-dessous, présente le résultat des mesures effectuées sur le composants UT-1640K-TT-2-R.

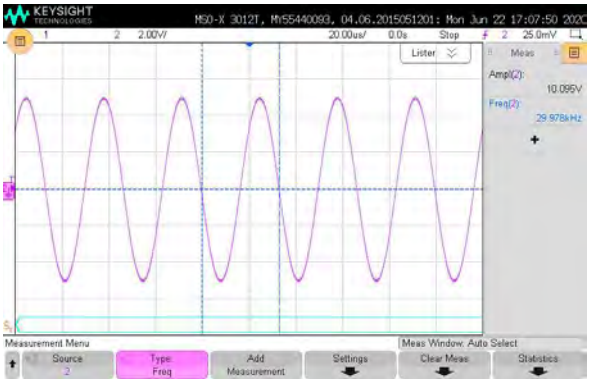


Figure 72 : Vsc hors de la fréquence de résonnance

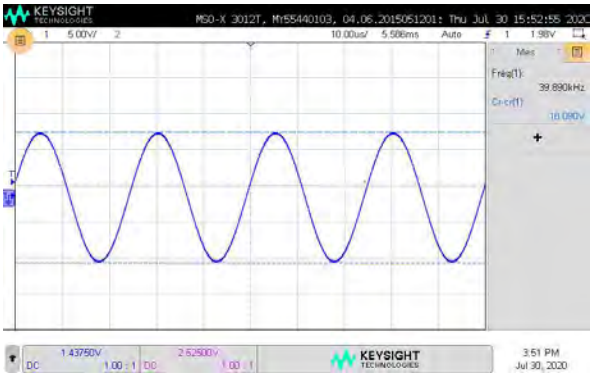


Figure 73 : Vsc à la fréquence de résonnance

La Figure 72 représente la tension Vsc lorsque l'on n'est pas en résonnance, la Figure 73 représente la tension Vsc en résonnance.

La résonnance de ce composant se situe à environ 40 [kHz]. Lors de la résonnance, l'amplitude de la tension Vsc vaut 9[V] soit une perte de 1[V] par rapport à l'amplitude de la tension hors résonnance.

La résistance R de l'émetteur ultrasonique peut donc être calculée :

$$\frac{R}{R + Rin} * Vin = Vsc$$

$$R * Vin = Vsc * (R + Rin)$$

$$R = \frac{Rin * Vsc}{Vin - Vsc} = \frac{50[\Omega] * 9[V]}{10[V] - 9[V]} = 450[\Omega]$$

La valeur efficace du courant consommé par le composant est calculée dans l'Équation 12.

$$I_{rms} = \frac{Vrms}{Rin + R} = \frac{\frac{Vin}{\sqrt{2}}}{Rin + R} = \frac{\frac{10[V]}{\sqrt{2}}}{50[\Omega] + 450[\Omega]} = 14.1\ [mA]$$

Équation 12 : Courant consommé par le composant

Le courant consommé est faible, un amplificateur à transistor peut donc être utilisé pour piloter ce composant.

12.3 Synthèse des résultats

La consommation des 2 composants est présenté dans le Tableau 27 pour une tension d'alimentation de 20[Vpp].

COMPOSANTS	COURANT CONSOMMÉ À LA FRÉQUENCE DE RÉSONNANCE
UT-1240K-TT-R	16.9 [mA]
UT-1640K-TT-2-R	14.1 [mA]

Tableau 27 : Consommation des émetteurs ultrasoniques

Les 2 composants consomment peu de courant à la fréquence de résonnance pour une tension d'alimentation de 20[Vpp]. Pour les 2 composants, un amplificateur à transistor peut être utilisés pour piloter l'émetteur ultrasonique.

Cependant, les émetteurs ultrasoniques peuvent être alimenté avec des tensions bien supérieur à 20[Vpp]. En effet, les fiches techniques des composants donnent la tension efficace maximales applicable sur les émetteurs ultrasoniques.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Le Tableau 28 présente la tension efficace d'alimentation maximales des émetteurs ultrasoniques.

ÉMETTEUR ULTRASONIQUE	TENSION D'ALIMENTATION MAXIMALE
UT-1240K-TT-R[1]	30[Vrms]
UT-1640K-TT-2-R[2]	20[Vrms]

Tableau 28 : Tension d'alimentation maximale des émetteurs ultrasoniques (voir note 1et 2)

Étant donné que la résistance de l'émetteur ultrasonique est connue, on peut calculer le courant consommé par les émetteurs ultrasonique avec une telle amplitude du signal. L'Équation 13 présente comment le courant consommé par les émetteurs ultrasonique est calculé lorsque les émetteurs ultrasoniques sont en résonnance.

$$I_{rms} = \frac{Vrms}{R}$$

Équation 13 : Calcul du courant consommé par les émetteurs ultrasoniques à la fréquence de résonnance

Le Tableau 29 présente la résistance des émetteurs ultrasoniques à leur fréquence de résonnance.

COMPOSANTS	RÉSITANCE À LA FRÉQUENCE DE RÉSONNANCE
UT-1240K-TT-R	366.6 [Ω]
UT-1640K-TT-2-R	450 [Ω]

Tableau 29 : Résistance des émetteurs ultrasonique à la fréquence de résonnance

On peut donc calculer le courant consommé par les émetteurs ultrasonique lorsqu'ils sont alimentés avec leur tension d'alimentation maximale.

COMPOSANTS	COURANT CONSOMMÉ À LA FRÉQUENCE DE RÉSONNANCE
UT-1240K-TT-R	81.8[mA]
UT-1640K-TT-2-R	44.4[mA]

Tableau 30 : Courant consommé par les émetteurs ultrasonique à la fréquence de résonnance

Le courant consommé par les émetteurs ultrasoniques est beaucoup plus important. On peut calculer la valeur pic du courant consommé par les émetteurs ultrasoniques. L'Équation 14 présente le calcul de la valeur pic du courant consommé par les émetteurs ultrasoniques à la fréquence de résonnance.

$$I_p = \frac{V_p}{R}$$

Équation 14 : Valeur pic du courant consommé par les émetteurs ultrasoniques à la fréquence de résonnance

La tension d'alimentation des émetteurs ultrasoniques est un signal sinusoïdal. On peut calculer la valeur pic de la tension avec l'Équation 15.

$$V_p = V_{rms} * \sqrt{2}$$

Équation 15 : Calcul tension pic d'un signal sinusoïdal à partir de la valeur efficace d'un signal sinusoïdal

On peut donc calculer le courant pic consommé par les émetteurs ultrasonique lorsqu'ils sont alimentés avec leur tension d'alimentation maximale.

COMPOSANTS	COURANT PIC CONSOMMÉ À LA FRÉQUENCE DE RÉSONNANCE
UT-1240K-TT-R	115.7[mA]
UT-1640K-TT-2-R	62.9[mA]

Tableau 31 : Courant pic consommé par les émetteurs ultrasonique à la fréquence de résonnance

Ces courant sont beaucoup plus importants, il faut donc prévoir une électronique adaptée pour générer les signaux d'alimentation des émetteurs ultrasoniques.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

13 MONTAGE POUR LES TESTS

La Figure 74 présente le montage effectué pour tester le projet. Les microphones sont connectés sur la carte de dévellopement STM32F746G-Discovery.

Lors d'une prise de mesures, le disque est fixe. De plus, le disque comporte des trais noir. Ces traits sont espacés de 22.5°. Il y en a par conséquents 16 sur le disque.

Un axe permet à la carte STM32F746G-Discovery de tourner autour de l'axe. L'axe à été positionner de tel manière à ce qu'il soit aligner sur le centre du cercle sur lequel les microphones sont disposés.De cet manière,en tournant la carte STM32F746G-Discovery et en ayant l'émetteur ultrasonique fixe, ce sytème peut tester des angles d'arrivée du son entre 0° et 360°.

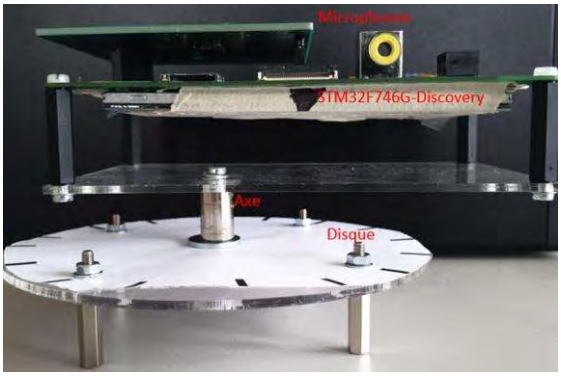


Figure 74 : Montage pour les tests 1/2

Sur la Figure 75, une marque sur le Plexiglas (dans le cercle rouge) permet de connaître l'angle actuel des microphones par rapport à l'émetteur ultrasonique. En effet, cette marque est alignée avec le centre du cercle sur lequel sont disposé les microphones.

Figure 75 : Montage pour les tests 2/2

Il y a 16 marques sur le disque, une série de mesures est prises sur chacune de ces marques (chaque 22.5°). Pour prendre une série de mesures, il est nécessaire d'aligner la marque présentée sur la Figure 75 sur un des traits sur le disque.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

14 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON DANS L'AIR

Ce chapitre explique les tests effectués et des résultats obtenus pour récupérer l'angle d'arrivée du son, lorsque les microphones et l'émetteur ultrasonique sont dans l'air.

Le matériel utilisé pour ce test est illustré dans le Tableau 32.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
ÉMETTEUR ULTRASONIQUE	UT-1240K-TT-R
CIRCUIT IMPRIMÉ CARTE MICROPHONE	V1.1

Tableau 32 : Test dans l'air : Matériel utilisé

L'émetteur ultrasonique est alimenté en 20[Vpp] dans ce test.

14.1 Protocole de tests

Ce test utilise le montage présenté dans chapitre 13.

Des mesures sont prises en positionnant la cartes des microphones à 16 angles différents. Ces 16 angles sont dispersés équitablement sur l'intervalle 0° à 360°, soit chaque 22.5°. La distance entre le circuit imprimé des microphones et l'émetteur ultrasonique est de 1 mètre.

Dans un premier temps, le test a été effectué dans un environnement avec aucun objet à moins de 2 mètres autour de l'émetteur ultrasonique et des microphones.

Dans un deuxième temps, le test a été effectué dans un environnement avec des objets à environ 1 mètre autour de l'émetteur ultrasonique et des microphones.

14.2 Résultats dans un environnement contrôlé

Ce chapitre présente le résultat des tests effectués dans un environnement contrôlé. Le but de ce test est de déterminer s'il est possible de récupérer l'angle d'arrivée du son.

La Figure 76 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 16 angles différents sont affichés sur le graphique.

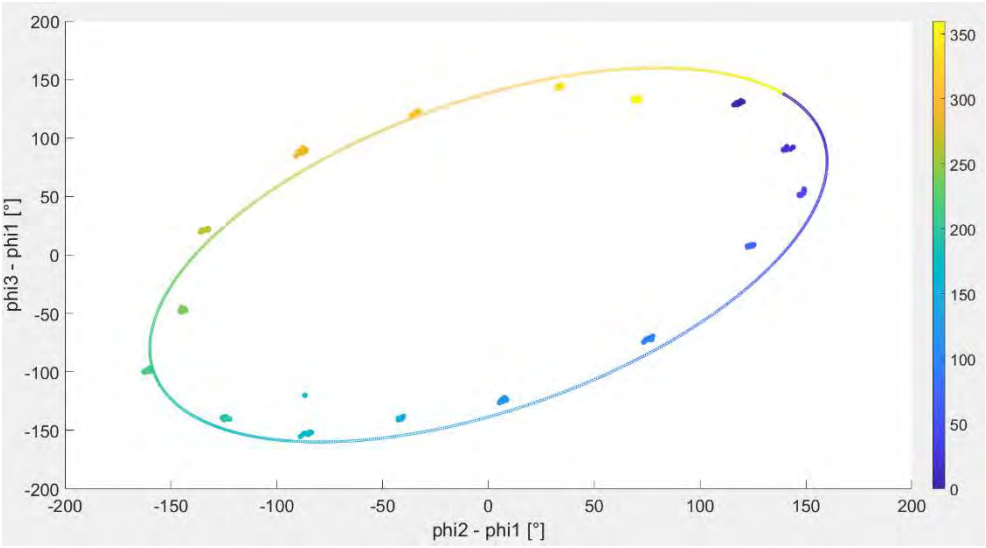


Figure 76 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 77 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

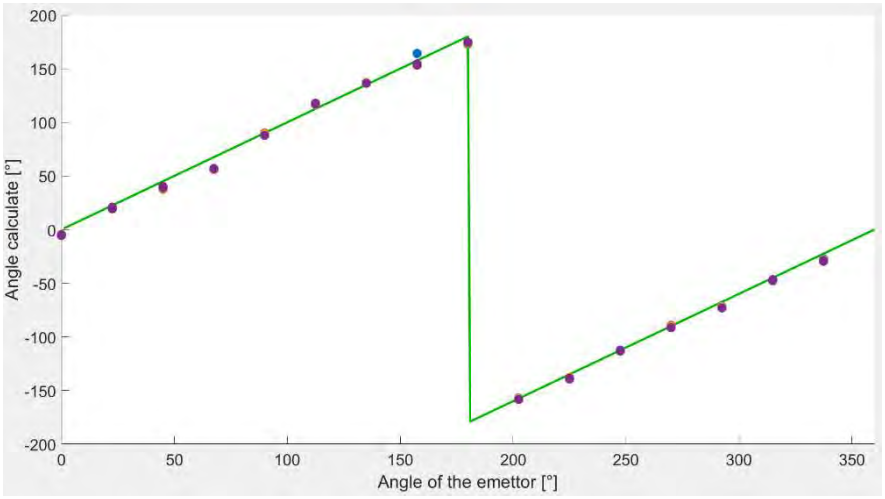


Figure 77 : Test dans l'air : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les mesures faites. La Figure 78 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

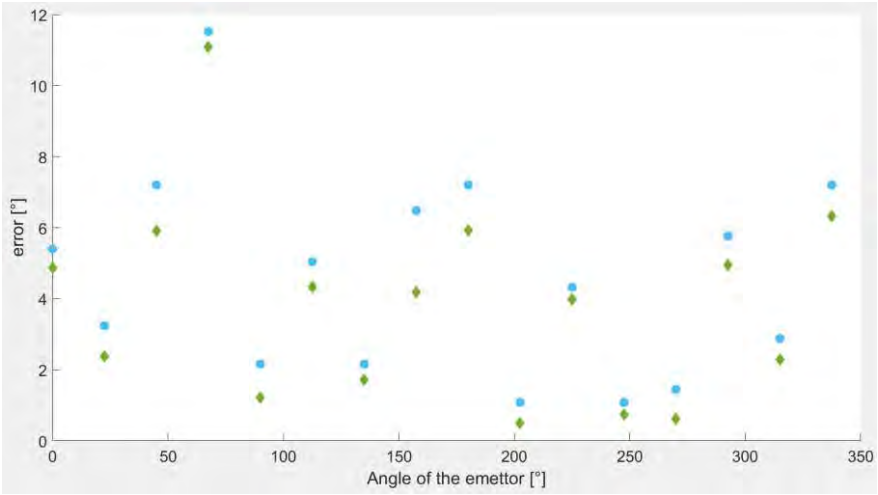


Figure 78 : Test dans l'air : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 78, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 1° et 11°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 1° et 12°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont bien inférieures à l'erreur de mesure acceptable de son projet.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

14.3 Résultats dans un environnement non-contrôlé

Ce chapitre présente le résultat des tests effectués dans un environnement non-contrôlé. Le but de ce test est de déterminer s'il est possible de récupérer l'angle d'arrivée du son et d'observer la différence avec les tests effectués dans un environnement contrôlé.

La Figure 79 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 16 angles différents sont affichés sur le graphique.

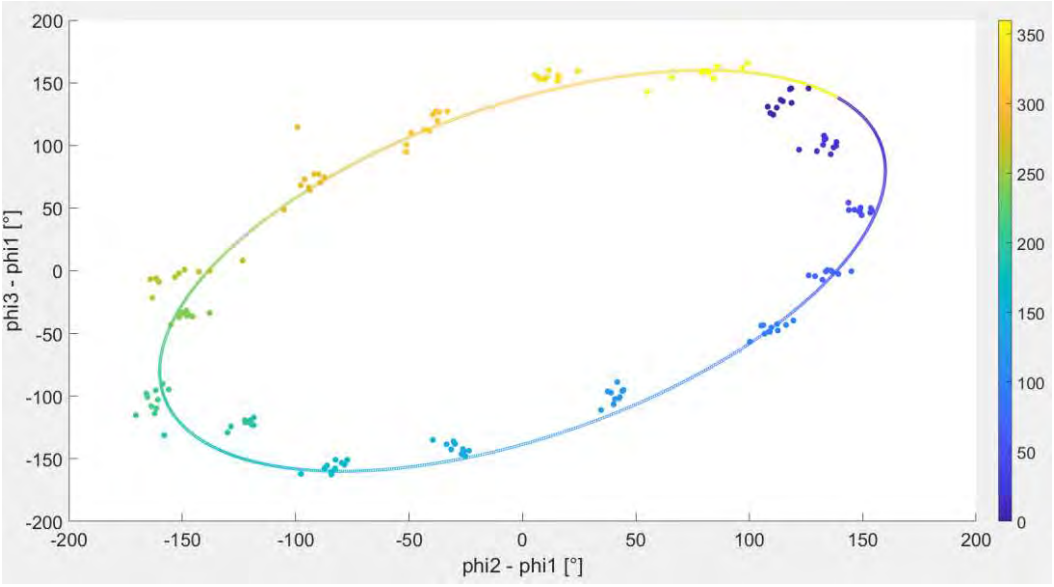


Figure 79 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont plus dispersées que lors du test dans un environnement contrôlé.

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 80 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

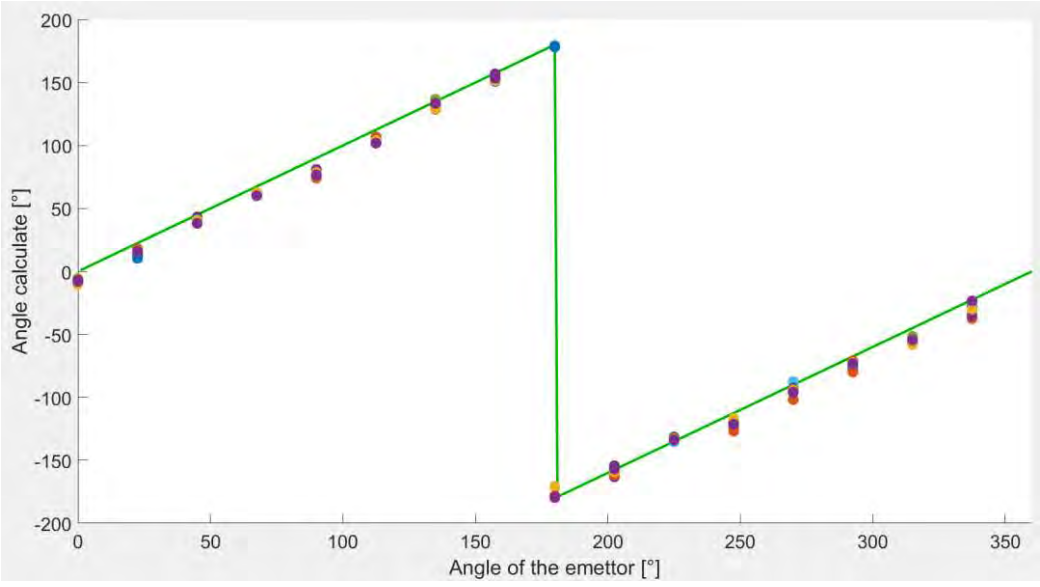


Figure 80 : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a de l'erreur sur les mesures faites. La Figure 81 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

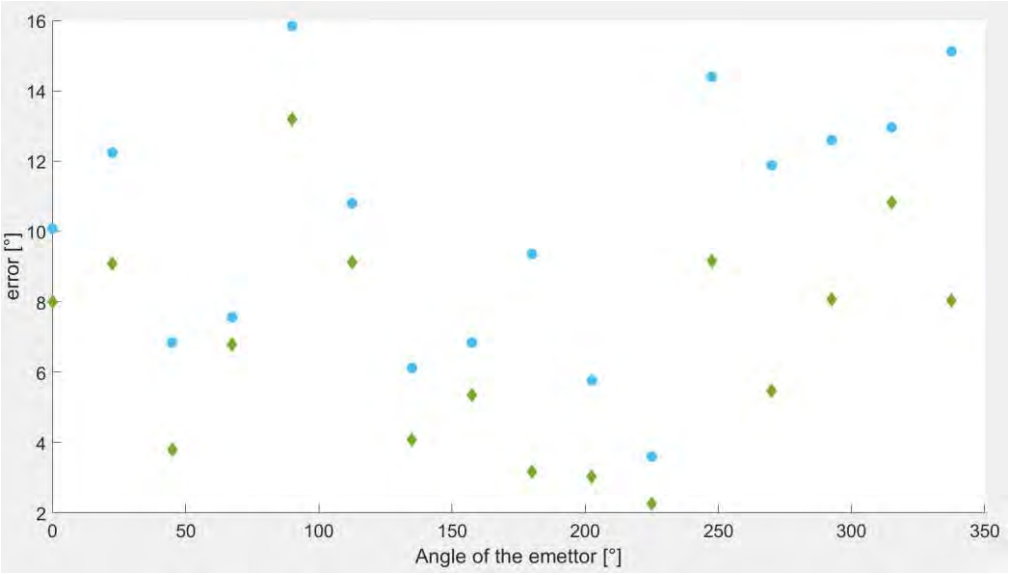


Figure 81 : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 81, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 2° et 14°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 4° et 16°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont bien inférieures à l'erreur de mesure acceptable de son projet.

14.4 Synthèse des résultats

Le test présenté en chapitre 14.2 a montré qu'il est possible de récupérer l'angle d'arrivée du son dans un environnement contrôlé, c'est-à-dire avec aucun obstacle à moins de 2 mètres des microphones et de l'émetteur ultrasonique.

Un deuxième a été effectué en chapitre 14.3 a montré qu'il est possible de récupérer l'angle d'arrivée du son dans un environnement moins contrôlé, c'est-à-dire avec des obstacles à environ 1 mètres des microphones et de l'émetteur ultrasonique.

Cependant, il y avait beaucoup plus d'erreur de mesures dans le deuxième test. De plus, les séries de mesures prises à chaque angle ont des résultats beaucoup plus variés dans le deuxième test. Toutes ces observations montrent que l'environnement à un impact sur les résultats obtenus.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

15 TEST AVEC L'ÉMETTEUR ULTRASONIQUE AVEC DU SILICONE

L'émetteur ultrasonique doit aller dans l'eau, c'est pourquoi un boîter doit être fait autour de ce dernier. Du silicone a été posé sur l'émetteur ultrasonique. Le montage effectué est présenté en Figure 82. La partie blanche représente le silicone qui a été déposé sur la sortie de l'émetteur ultrasonique.

Figure 82 : Silicone sur l'émetteur ultrasonique

Le matériel utilisé pour ce test est présenté dans le Tableau 33.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
AMPLIFICATEUR DE TENSION	TOELLNER TOE 7607
ÉMETTEUR ULTRASONIQUE	UT-1640K-TT-2-R
CIRCUIT IMPRIMÉ CARTE MICROPHONE	V1.1

Tableau 33 : Émetteur ultrasonique avec du silicone : Matériel utilisé

La tension d'alimentation de l'émetteur ultrasonique est de 50[Vpp].

15.1 Protocole de test

Ce test utilise le montage présenté dans chapitre 13.

La carte avec les microphones est placée à 25 centimètres de l'émetteur ultrasonique.

Des mesures sont prises en positionnant la cartes des microphones à 8 angles différents. Ces 8 angles sont dispersés équitablement sur l'intervalle 0° à 360°, soit chaque 45°.

15.2 Amplitude du signal

Un test a été effectué. La Figure 83 présente les résultats des signaux filtrés. Les signaux sont bruités, de plus, l'amplitude des signaux de sortie des microphones filtré est très faible. L'amplitude des sorties des microphones filtrées est d'environ 0.03. La Figure 84 présente l'amplitude des signaux reçus par les microphones lorsqu'il n'y a pas d'émetteurs ultrasoniques utilisés. L'amplitudes des signaux est de 0.008.

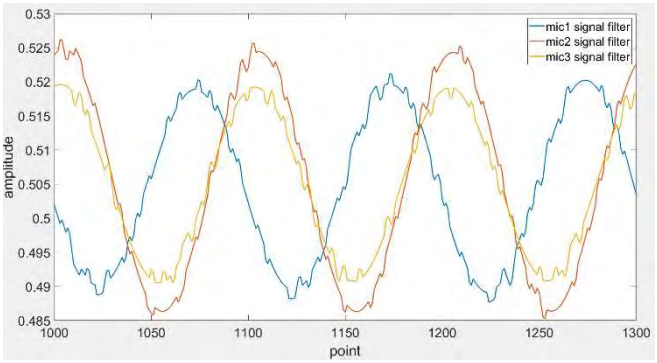


Figure 83 : Signaux microphones avec l'émetteur ultrasonique dans du silicone

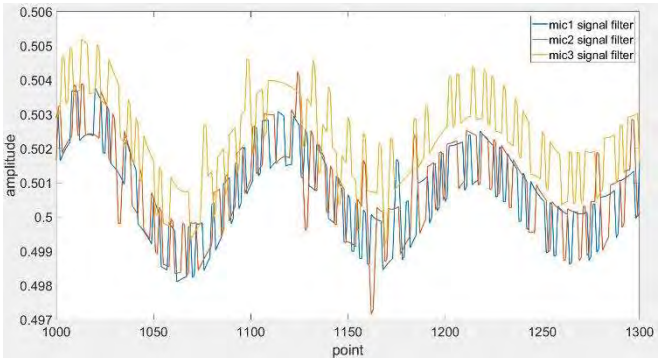


Figure 84 : Signaux des microphones sans émetteurs ultrasoniques

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Lorsque l'émetteur ultrasonique est couvert par du silicone, le signal reçu est faible, mais il y a tout de même un signal qui est reçu. Une série de mesure a été faite pour déterminer s'il est possible de récupérer l'angle d'arrivée du son. Ce test est présenté en chapitre 15.3.

15.3 Récupération angle d'arrivée du son

La Figure 85 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

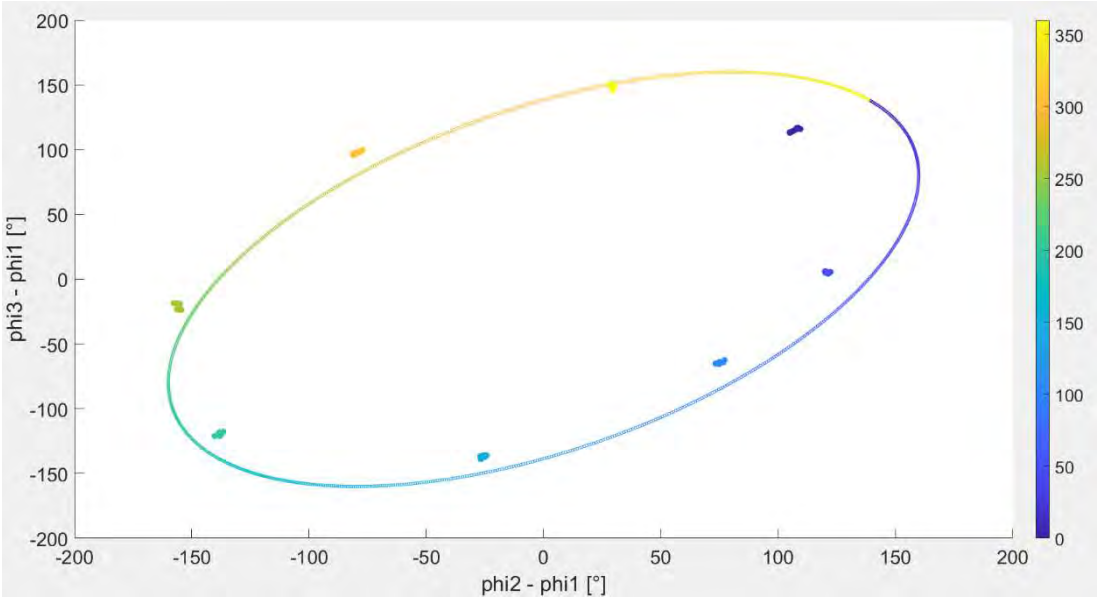


Figure 85 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées.

La Figure 86 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les disques représentent les résultats obtenus.

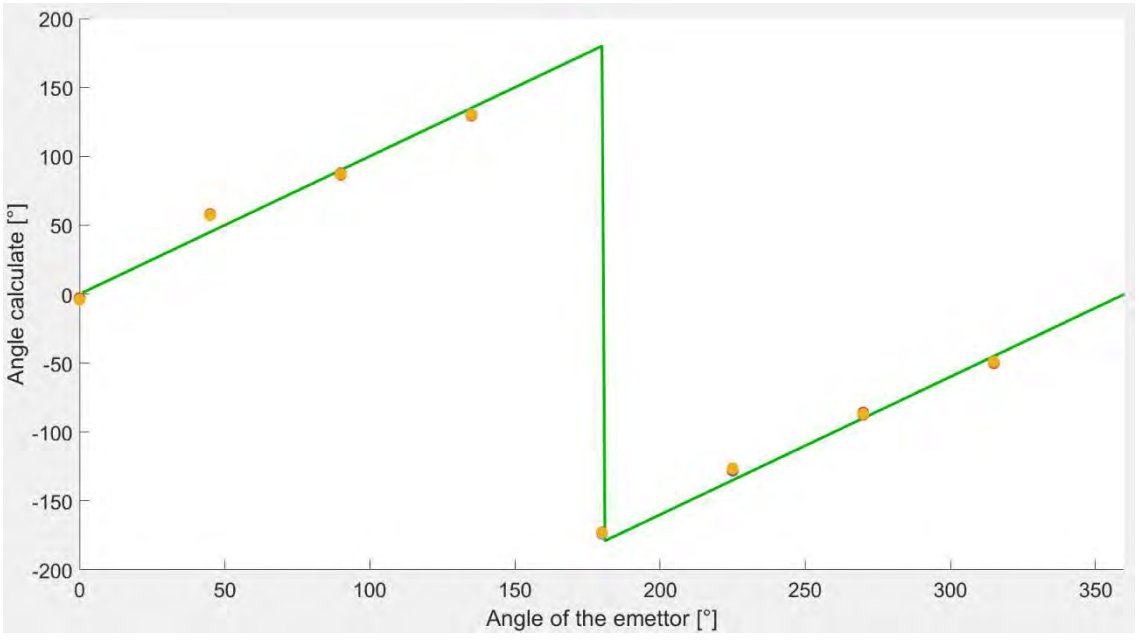


Figure 86 : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a de l'erreur sur les angles θ obtenus. La Figure 87 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

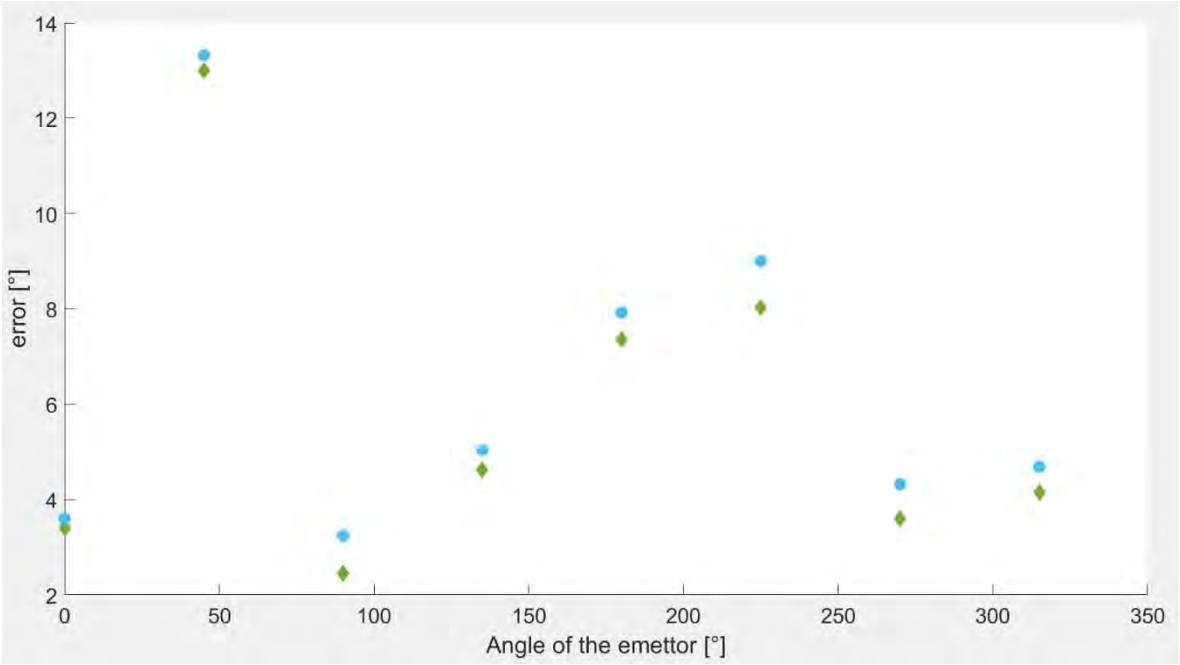


Figure 87 : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 87, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 3° et 13°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 4° et 13°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont inférieures à l'erreur de mesure acceptable de ce projet.

15.4 Synthèses des résultats

La récupération de l'angle d'arrivée du son est fonctionnelle avec les microphones dans l'air à une distance de 25 centimètres de l'émetteur ultrasonique couvert par du silicone.

Le signal reçu par les microphones est très faible, cependant il a été possible de récupérer l'angle d'arrivée du son. Lorsque l'émetteur ultrasonique est couvert par du silicone, la distance maximale d'émission est très faible. En effet, à 25 centimètres l'amplitude du signal reçus par les microphones est presque nul.

L'émetteur ultrasonique doit être placé dans un boîtier, car il doit aussi aller dans l'eau. Avec du silicone sur l'émetteur ultrasonique, la distance maximale entre les 2 plongeurs est beaucoup trop faible.

Il y a 2 options possible pour augmenter la distance maximale entre les 2 plongeurs. La première option consiste à utiliser un matériau étanche, résistant à la pression et atténuant très peu le signal sonore. La deuxième option consiste à remplacer les microphones digitaux utilisés par des microphones analogiques. Un traitement du signal analogique devra être effectué. De cette manière, il est probable d'obtenir un signal avec une meilleure amplitude.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

16 TEST SUR LA DISTANCE MAXIMALE D'ÉMISSION

Un test sur la distance maximale d'émission a été effectué.

Les fiches techniques des émetteurs ultrasonique indique la valeur efficace maximale de la tension à appliquer sur les émetteurs ultrasoniques. Ces valeurs sont les suivantes :

ÉMETTEUR ULTRASONIQUE	TENSION D'ALIMENTATION MAXIMALE
UT-1240K-TT-R	30[Vrms]
UT-1640K-TT-2-R	20[Vrms]

Tableau 34 : Tension d'alimentation des émetteurs ultrasoniques (*voir note de fin 1 et 2*)

Le matériel utilisé pour ce test est illustré dans le Tableau 32.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
AMPLIFICATEUR DE TENSION	TOELLNER TOE 7607
ÉMETTEUR ULTRASONIQUE	UT-1240K-TT-R
ÉMETTEUR ULTRASONIQUE	UT-1640K-TT-2-R
CIRCUIT IMPRIMÉ CARTE MICROPHONE	V1.1

Tableau 35 : Test sur la distance maximale : Matériel utilisé

Un amplificateur de tension a dû être utilisé, car la tension de sortie du générateur de fonctions utilisé est de 20[Vpp]. Cette tension est bien inférieure aux tensions applicables sur les émetteurs ultrasoniques présentées dans le Tableau 34. L'Équation 16 présente le calcul de la valeur efficace d'un signal sinusoïdal.

$$Vrms = \frac{Vpp}{2\sqrt{2}}$$

Équation 16 : Définition de la valeur efficace

Pour alimenter les émetteurs ultrasoniques avec la tension efficace maximale, il faut donc alimenter les émetteurs ultrasoniques avec les tensions suivantes :

ÉMETTEUR ULTRASONIQUE	TENSION D'ALIMENTATION MAXIMALE
UT-1240K-TT-R[1]	84.8[Vpp]
UT-1640K-TT-2-R[2]	56.5[Vpp]

Tableau 36 : Tension d'alimentation des émetteurs ultrasoniques

L'amplificateur de tension utilisé à une tension sortie maximale de 50[Vpp]. Les émetteurs ultrasoniques ont donc été alimenté avec du 50[Vpp].

La fiche technique des composants n'indique pas de relations entre la pression acoustique émise par l'émetteur ultrasonique et la tension appliquée sur l'émetteur ultrasonique. Il n'est donc pas possible de connaître l'impact sur la pression acoustique du signal émis. La pression acoustique du signal émis sera certainement plus faible que la pression acoustique maximale du signal émis que l'émetteur ultrasonique peut produire. Ces tests donnent par conséquent une idée de la distance maximale où l'on peut détecter un signal provenant de l'émetteur ultrasonique.

16.1 Protocole de tests

Ce test utilise le montage présenté dans chapitre 13.

Les 2 émetteurs ultrasoniques seront testés. La tension d'alimentation des 2 émetteurs ultrasoniques est de 50[Vpp].

La carte est placée aux distances suivantes :

- 3 mètres
- 5 mètres
- 7.5 mètres
- 10 mètres
- 12 mètres

Pour chaque distance, des mesures sont prises en positionnant la cartes des microphones à 8 angles différents. Ces 8 angles sont dispersés équitablement sur l'intervalle 0° à 360°, soit chaque 45°.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

16.2 Émetteur ultrasonique UT-1240K-TT-R

Ce chapitre traites des tests effectué avec l'émetteur ultrasonique UT-1240K-TT-R. Ces tests servent à connaitre la distance maximale d'émission. Les microphones ont été placés à différentes distances de l'émetteur ultrasonique. L'amplitude des signaux est calculée et les résultats sont présentés dans le Tableau 37.

FIGURE	DISTANCE ENTRE LES MICROPHONES ET L'ÉMTTEUR ULTRASONIQUE	AMPLITUDES DES SIGNAUX
FIGURE 88	3 mètres	0.3
FIGURE 89	5 mètres	0.06
FIGURE 90	7.5 mètres	0.03
FIGURE 91	10 mètres	0.01

Tableau 37 : Amplitude des signaux des microphones pour des distances différentes

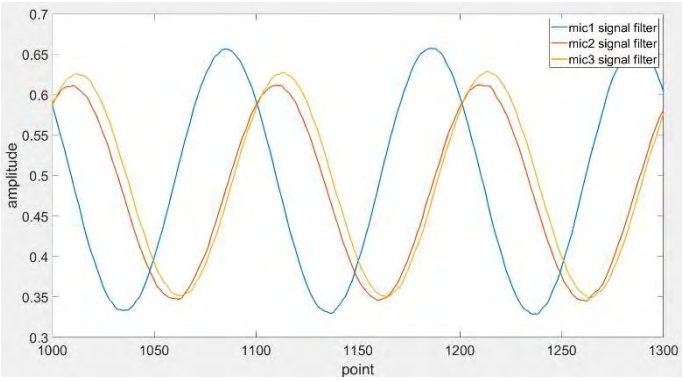


Figure 88 : Signaux microphones à une distance de 3 mètres

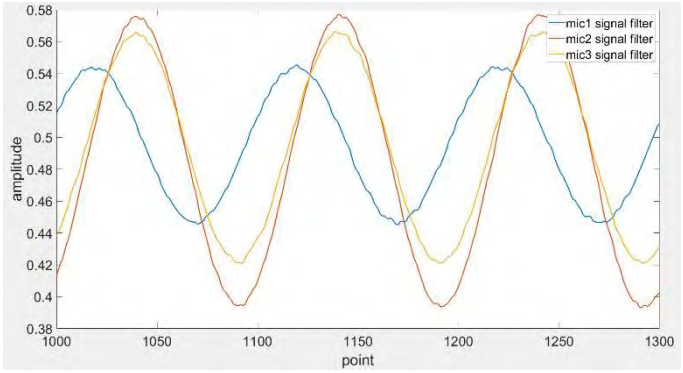


Figure 89 : Signaux microphones à une distance de 5 mètres

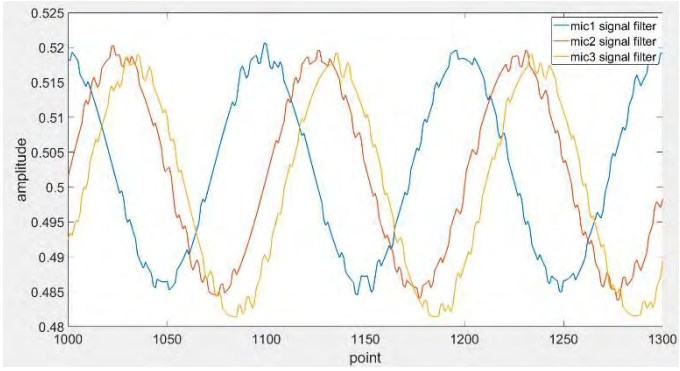


Figure 90 : Signaux microphones à une distance de 7.5 mètres

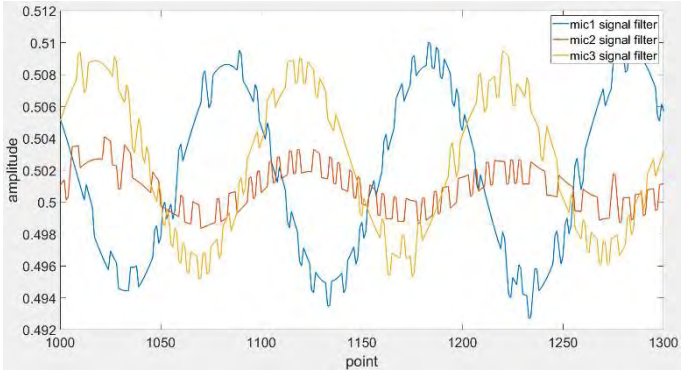


Figure 91 : Signaux microphones à une distance de 10 mètres

Ce qu'on aperçoit c'est que plus la distance entre l'émetteur ultrasonique et les microphones est grande, plus l'amplitude du son capté par le microphones diminue.

Lorsque la distance entre l'émetteur ultrasonique et les microphones est plus petite ou égale à 7.5 mètres, il devrait être possible de récupérer l'angle d'arrivée du son. En effet, le test pésenté en chapitre 15 avait l'amplitude des signaux des microphones de 0.03 et ce test a montré qu'il était possible de récupérer l'angle d'arrivée du son avec de tels amplitudes des signaux des microphones.

Lorsque la distance entre l'émetteur ultrasonique et les microphones est plus grande ou égale à 10 mètres, l'amplitude des signaux captés est quasiment nulle.

Il faut noter que la distance maximale entre l'émetteur ultrasonique et les microphones peut être impactée par la tension d'alimentation de l'émetteur ultrasonique. En effet, la tension d'alimention de cet émetteur ultrasonique est de 30[Vrms], mais dans ce test il a été alimenté avec une tension d'alimentation de 17.7[Vrms]. Il est donc probable que la distance maximale entre les microphones et l'émetteur ultrasonique puisse être plus grande si l'émetteur ultrasonique est alimenté en 30[Vrms].

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

16.2.1 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON À UNE DISTANCE DE 3 MÈTRES

La distance entre les microphones et l'émetteur ultrasonique est de 3 mètres.

La Figure 92 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

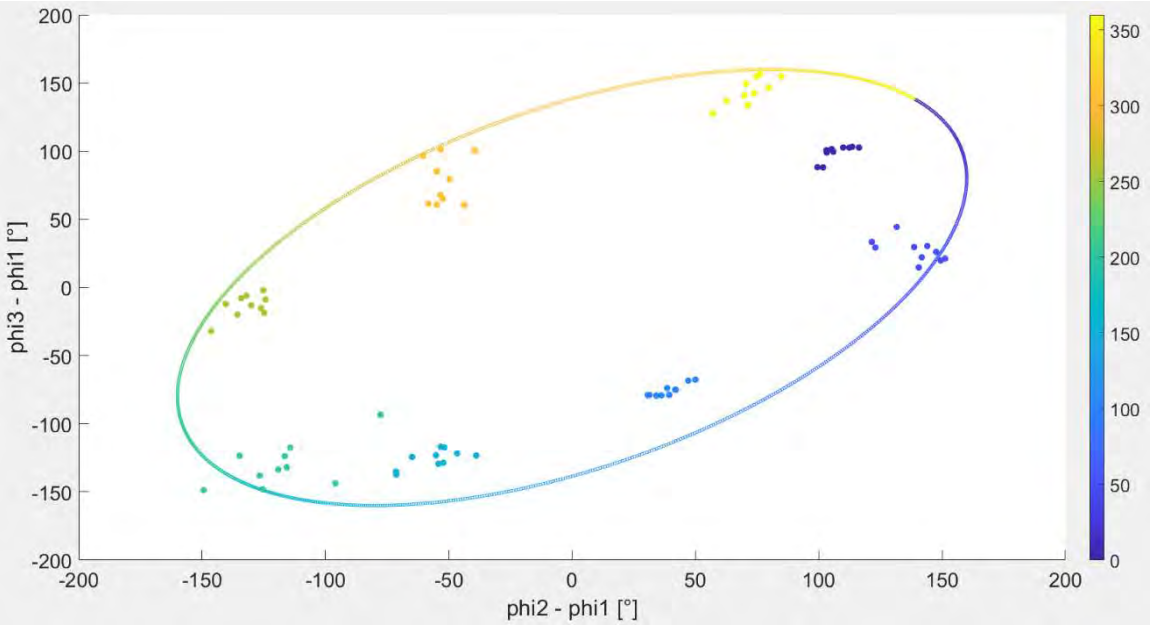


Figure 92 : Distance 3 mètres : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont un peu dispersées.

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 93 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

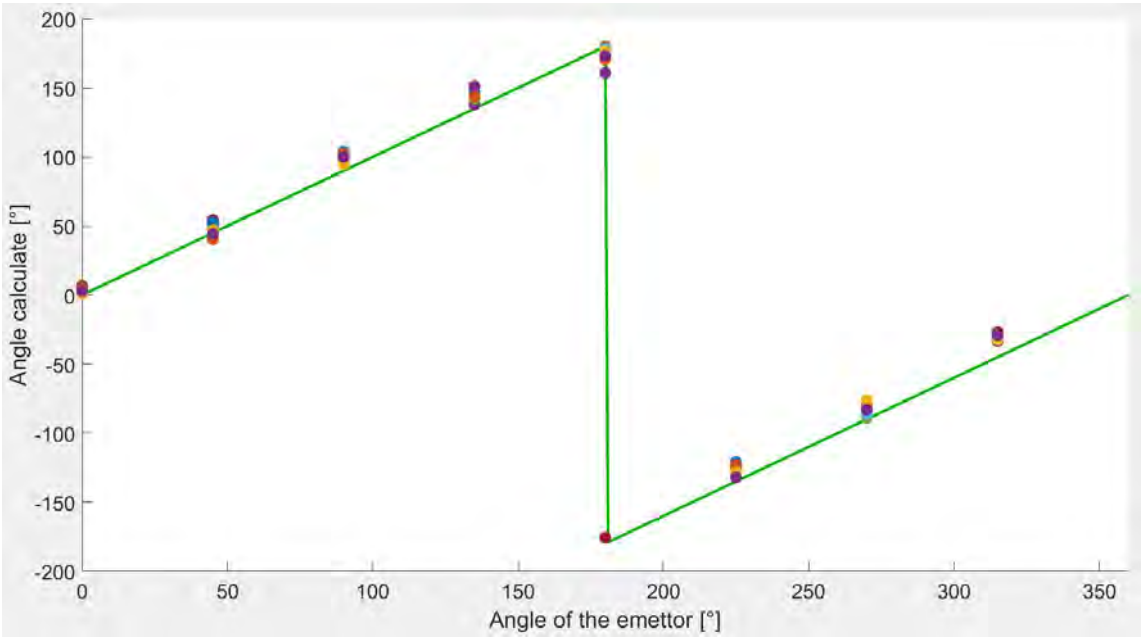


Figure 93 : Distance 3 mètres : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a de l'erreur sur les mesures faites. La Figure 94 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

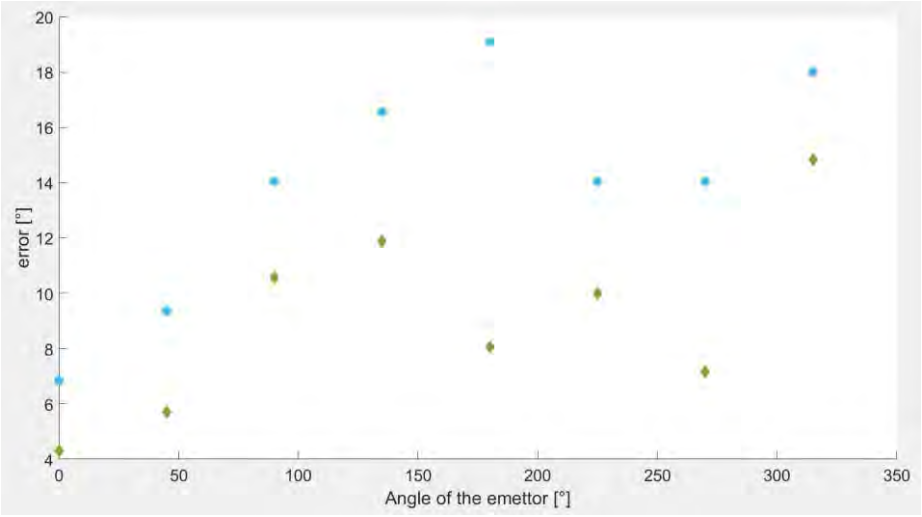


Figure 94 : Distance 3 mètres : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 94, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 4° et 15°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 7° et 19°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont inférieures à l'erreur de mesure acceptable de son projet.

16.2.2 *Récupération de l'angle d'arrivée du son à une distance de 5 mètres*

La distance entre les microphones et l'émetteur ultrasonique est de 5 mètres.

La Figure 95 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

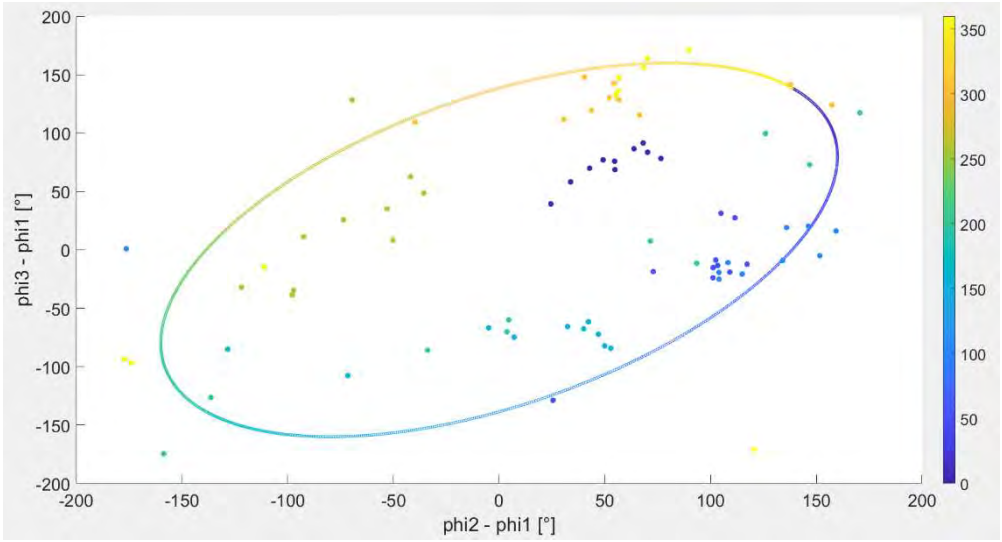


Figure 95 : Distance 5 mètres : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont énormément dispersées. Les résultats obtenus sont beaucoup plus dispersés qu'à une distance de 3 mètres.

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 96 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

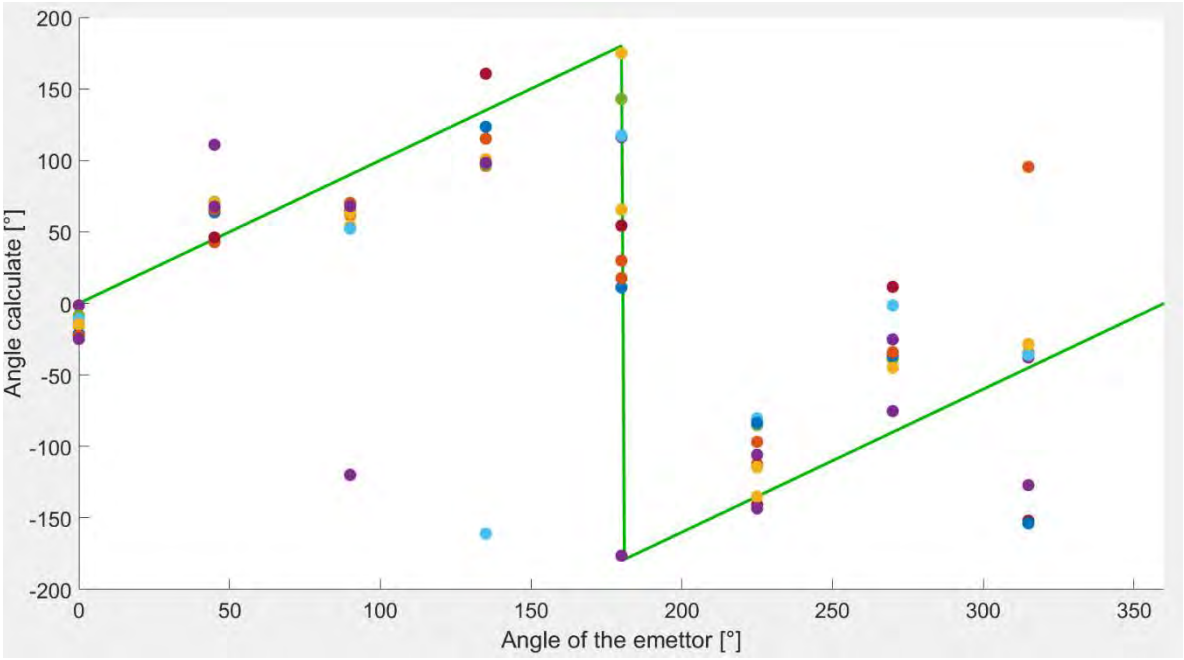


Figure 96 : Distance 5 mètres : Comparaison angles θ obtenus Vs angles θ attendus

Il y a beaucoup d'erreurs sur les mesures faites. La Figure 97 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

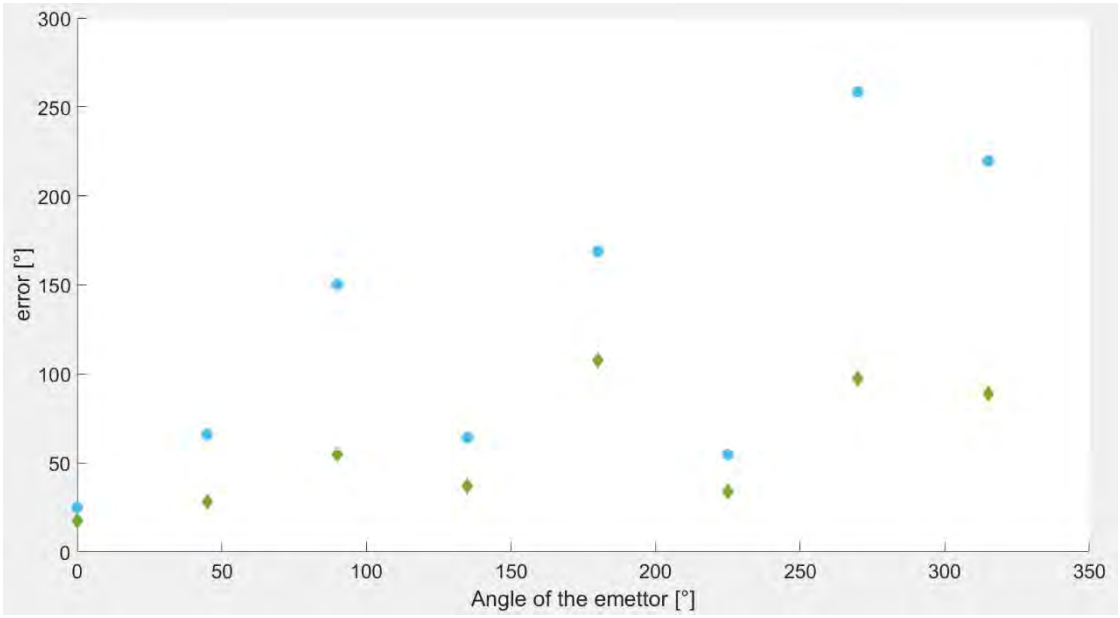


Figure 97 : Distance 5 mètres : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Les erreurs de mesures sont très importantes. Avec de tels erreurs de mesures, le système est inutilisable à cette distance.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

16.3 Émetteur ultrasonique UT-1640K-TT-2-R

Ce chapitre traites des tests effectué avec l'émetteur ultrasonique UT-1640K-TT-2-R. Ces tests servent à connaitre la distance maximale d'émission. Les microphones ont été placés à différentes distances de l'émetteur ultrasonique. L'amplitude des signaux est calculée et les résultats sont présentés dans le Tableau 38.

FIGURE	DISTANCE ENTRE LES MICROPHONES ET L'ÉMTTEUR ULTRASONIQUE	AMPLITUDES DES SIGNAUX
FIGURE 98	7 mètres	0.1
FIGURE 89	7.5 mètres	0.08
FIGURE 100	10 mètres	0.04
FIGURE 101	12 mètres	0.01

Tableau 38 : Amplitude des signaux des microphones pour des distances différentes

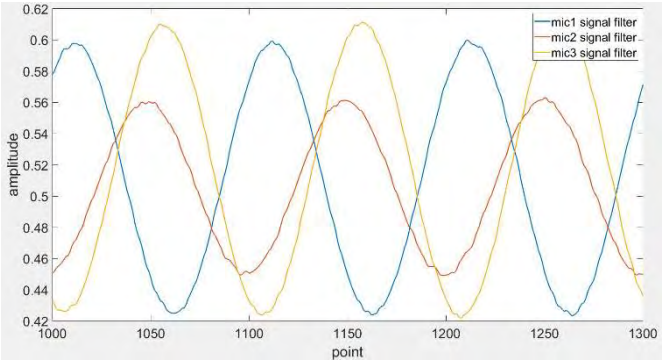


Figure 98 : Signaux microphones à une distance de 5 mètres

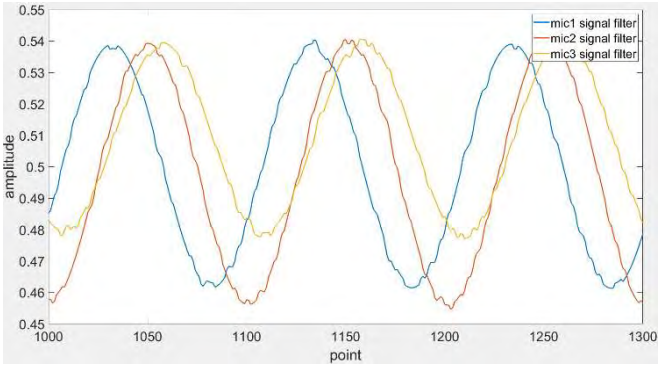


Figure 99 : Signaux microphones à une distance de 7.5 mètres

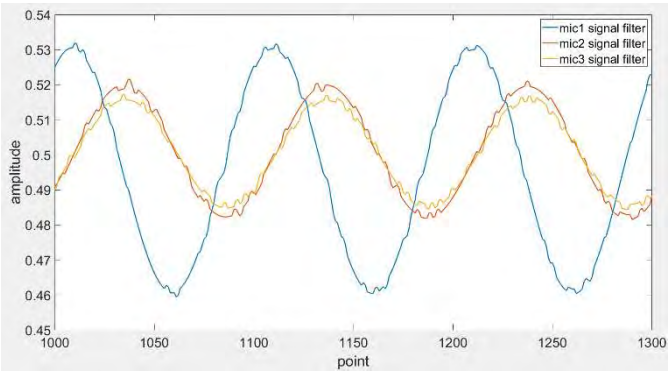


Figure 100 : Signaux microphones à une distance de 10 mètres

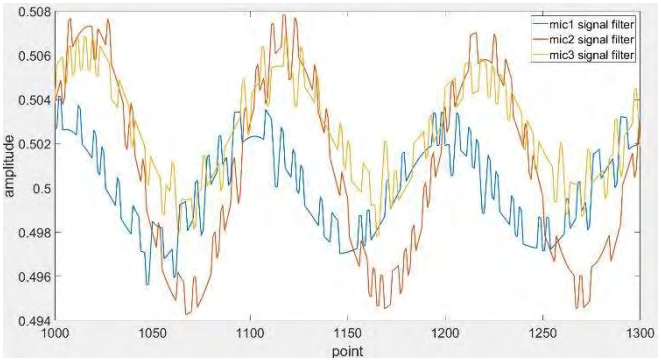


Figure 101 : Signaux microphones à une distance de 12 mètres

Ce qu'on aperçoit c'est que plus la distance entre l'émetteur ultrasonique et les microphones est grande, plus l'amplitude du son capté par le microphones diminue.

Lorsque la distance entre l'émetteur ultrasonique et les microphones est plus petite ou égale à 10 mètres, il devrait être possible de récupérer l'angle d'arrivée du son. En effet, le test pésenté en chapitre 15 avait l'amplitude des signaux des microphones de 0.03 et ce test a montré qu'il était possible de récupérer l'angle d'arrivée du son avec de tels amplitudes des signaux des microphones.

Lorsque la distance entre l'émetteur ultrasonique et les microphones est plus grande ou égale à 12 mètres, l'amplitude des signaux captés est quasiment nulle.

Il faut noter que la distance maximale entre l'émetteur ultrasonique et les microphones peut être impactée par la tension d'alimentation de l'émetteur ultrasonique. En effet, la tension d'alimention de cet émetteur ultrasonique est de 20[Vrms], mais dans ce test il a été alimenté avec une tension d'alimentation de 17.7[Vrms]. Il est donc probable que la distance maximale entre les microphones et l'émetteur ultrasonique puisse être plus grande si l'émetteur ultrasonique est alimenté en 20[Vrms].

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

16.3.1 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON À UNE DISTANCE DE 3 MÈTRES

La distance entre les microphones et l'émetteur ultrasonique est de 3 mètres.

La Figure 102 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

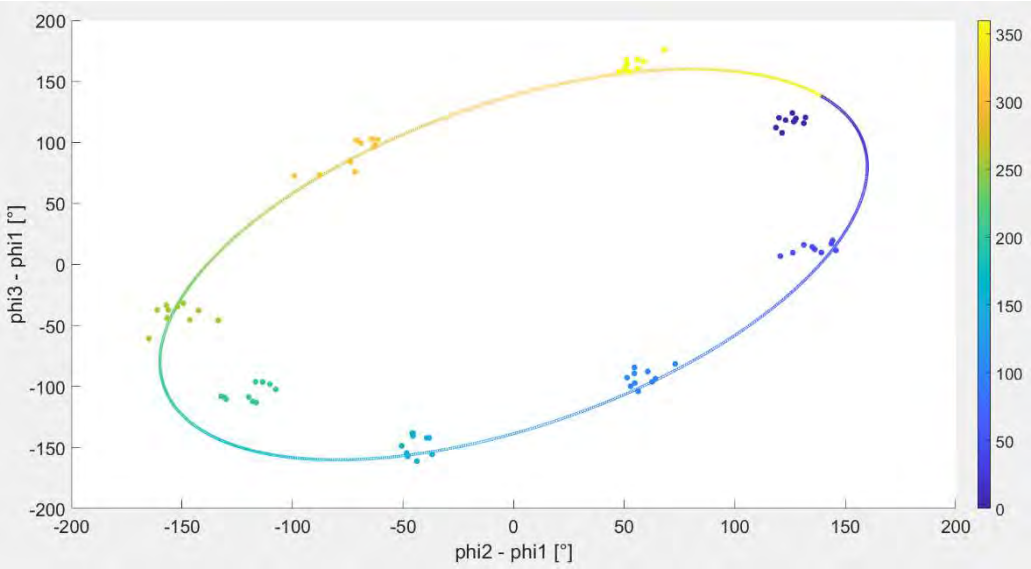


Figure 102 Distance 3 mètres : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont peu dispersées.

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 103 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

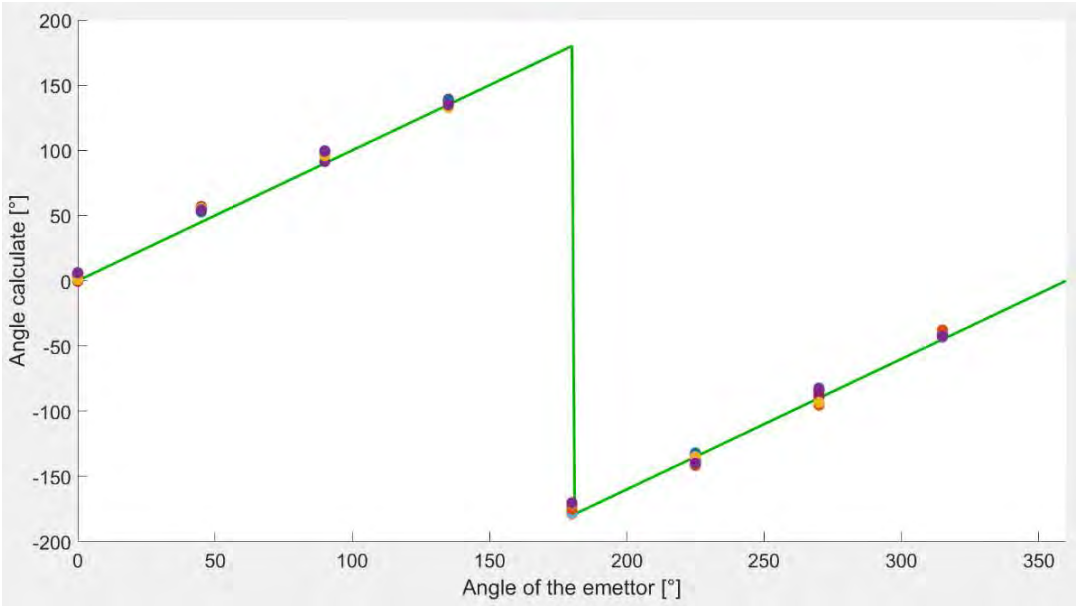


Figure 103 : Distance 3 mètres : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les mesures faites. La Figure 97 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

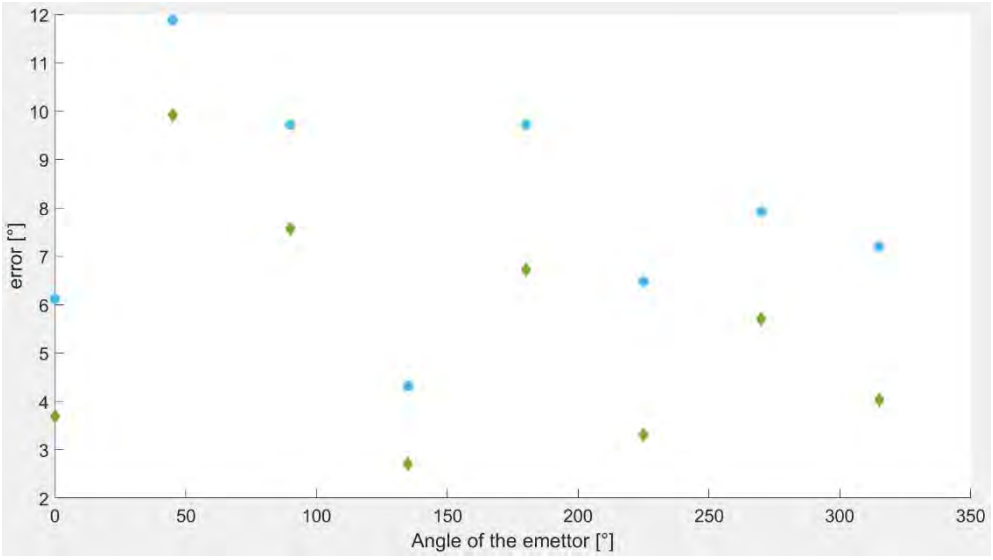


Figure 104 : Distance 3 mètres : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 104, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 3° et 10°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 4° et 12°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont bien inférieures à l'erreur de mesure acceptable de son projet.

16.3.2 Récupération de l'angle d'arrivée du son à une distance de 5 mètres

La distance entre les microphones et l'émetteur ultrasonique est de 5 mètres.

La Figure 102 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

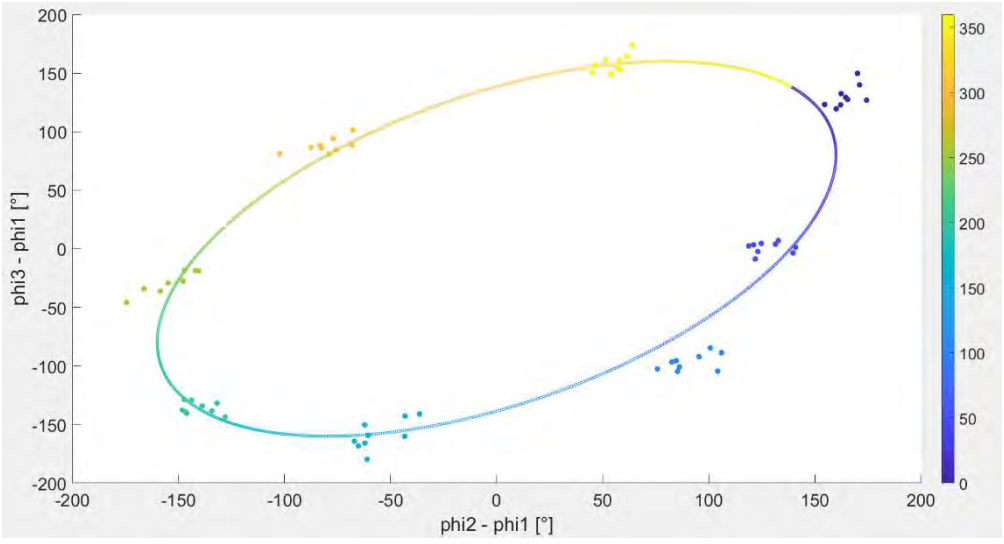


Figure 105 : Distance 5 mètres : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont un peu dispersées. Les résultats obtenus sont plus dispersés qu'à une distance de 3 mètres.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 103 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

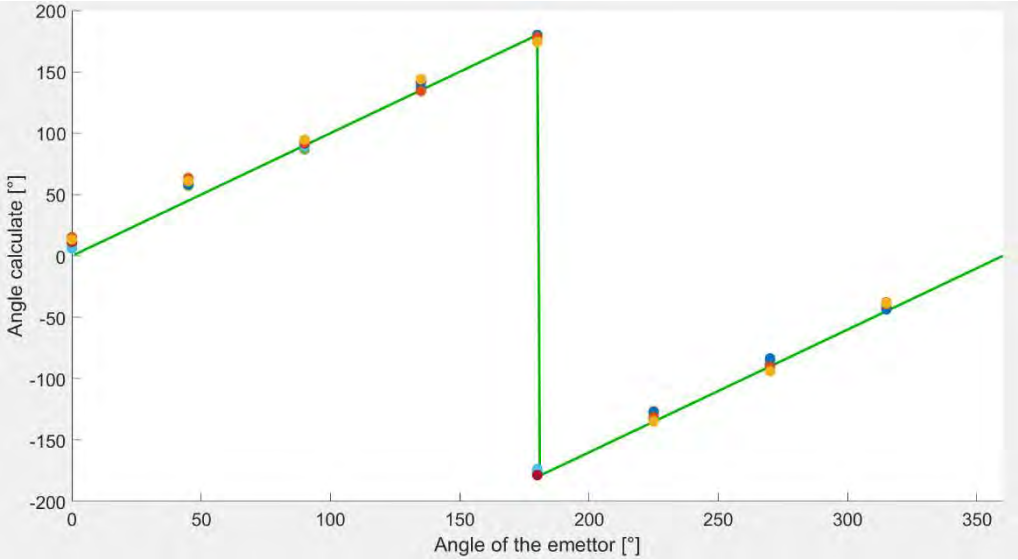


Figure 106 : Distance 5 mètres : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les mesures faites. La Figure 107 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

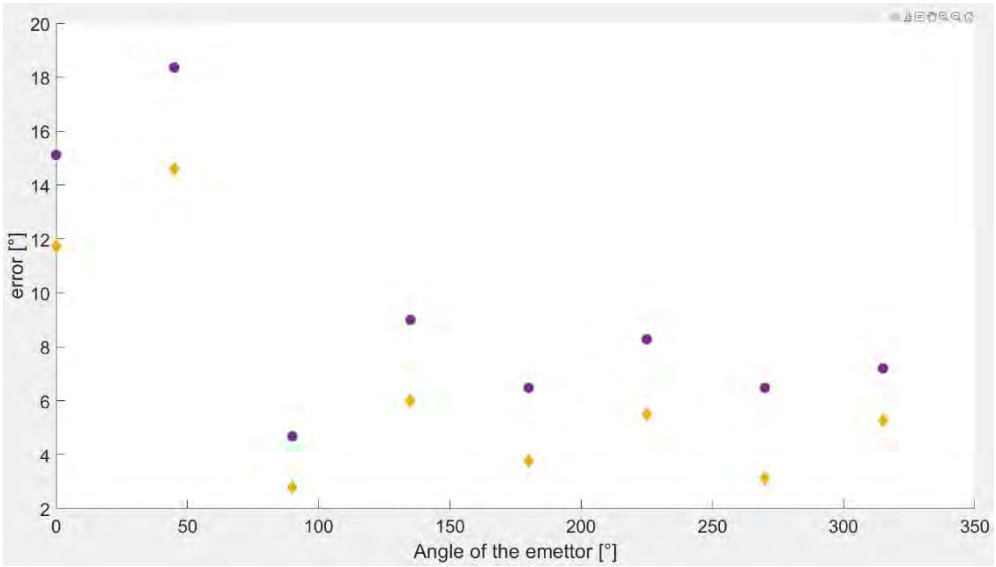


Figure 107 : Distance 5 mètres : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 107, les losanges jaunes représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques violets représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 3° et 14°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 5° et 19°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont inférieures à l'erreur de mesure acceptable de son projet.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

16.3.3 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON À UNE DISTANCE DE 7.5 MÈTRES

La distance entre les microphones et l'émetteur ultrasonique est de 7.5 mètres.

La Figure 108 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

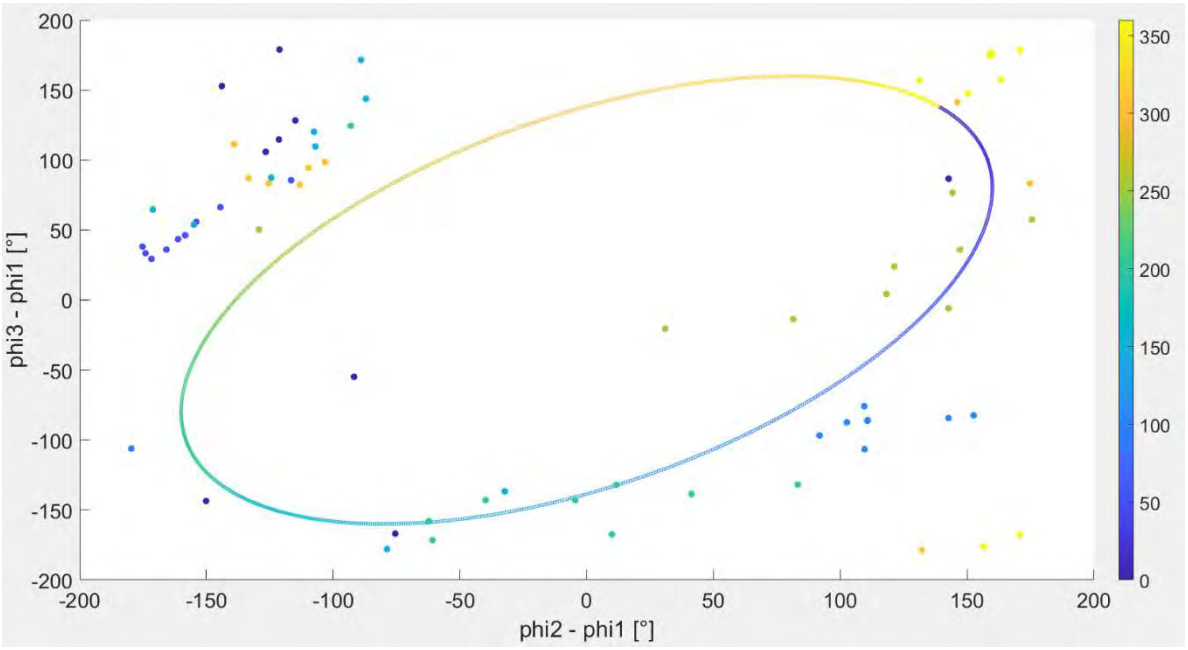


Figure 108 : Distance 7.5 mètres : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont un très dispersées. Les résultats obtenus sont beaucoup plus dispersés qu'à une distance de 5 mètres.

Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La Figure 108 compare les angles θ obtenus aux angles θ attendus. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

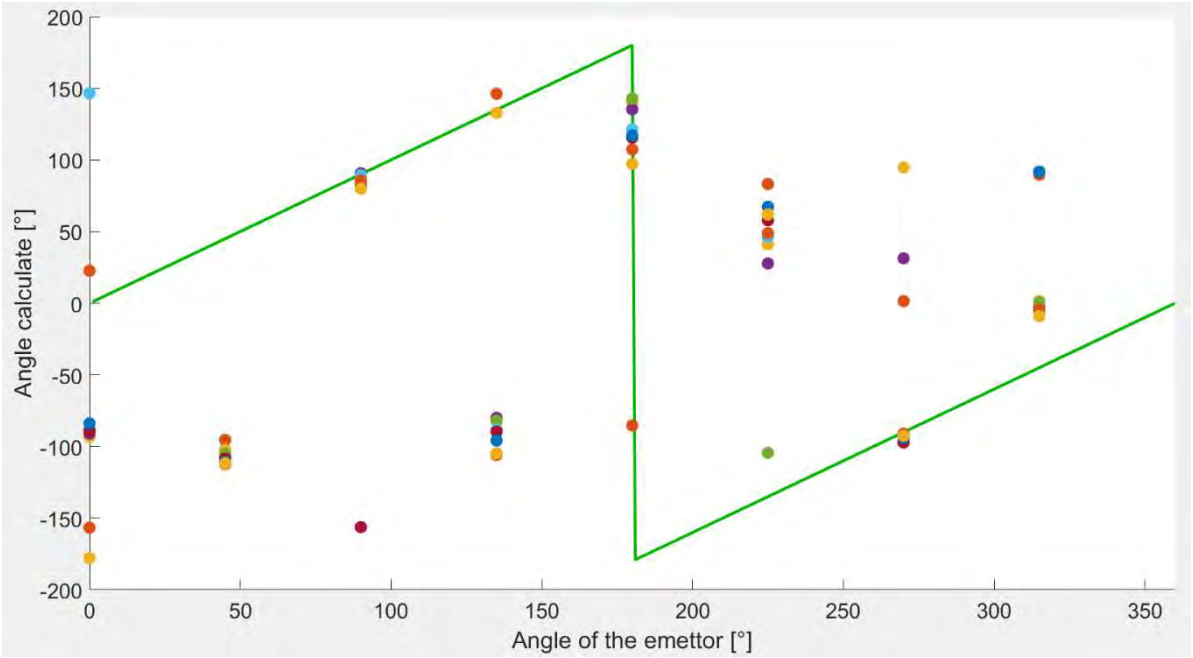


Figure 109 : Distance 7.5 mètres : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a beaucoup d'erreurs sur les mesures faites. La Figure 110 illustre les erreurs sur les angles θ obtenus par rapport aux angles θ attendus.

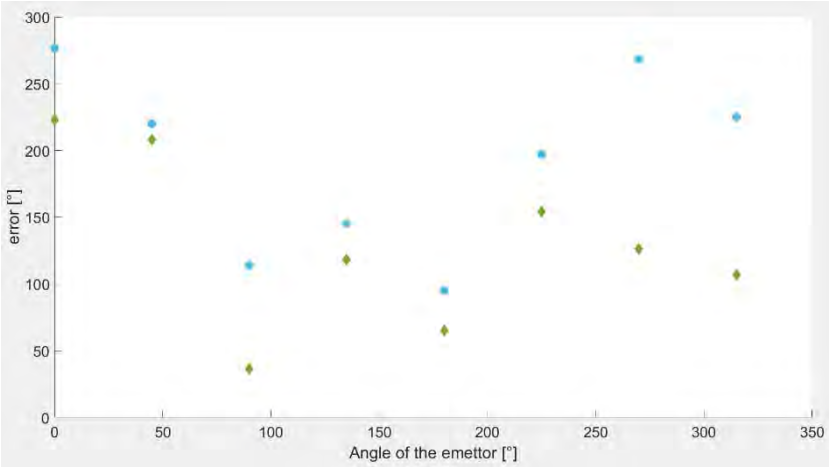


Figure 110 : Distance 7.5 mètres : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Les erreurs de mesures sont très importantes. Avec de tel erreurs de mesures, le système est inutilisable à cette distance.

16.4 Synthèse résultats sur la distance d'émission

Le Tableau 39 présente les synthèses des résultats sur les distances d'émissions des émetteurs ultrasoniques, avec une tension d'alimentation de 50[Vpp].

TEST	RÉSULTATS
ÉMETTEUR ULTRASONIQUE UT-1240K-TT-R, DISTANCE 3 MÈTRES	✓
ÉMETTEUR ULTRASONIQUE UT-1240K-TT-R, DISTANCE 5 MÈTRES	✗
ÉMETTEUR ULTRASONIQUE UT-1240K-TT-R, DISTANCE 3 MÈTRES	✓
ÉMETTEUR ULTRASONIQUE UT-1640K-TT-2-R, DISTANCE 5 MÈTRES	✓
ÉMETTEUR ULTRASONIQUE UT-1640K-TT-2-R, DISTANCE 7.5 MÈTRES	✗

Tableau 39 : Synthèse des résultats sur les tests de la distance d'émissions des émetteurs ultrasoniques

L'émetteur ultrasonique UT-1240K-TT-R a une distance maximale entre l'émetteur ultrasonique et les microphones de 3 mètres. Cependant, la distance maximale attendue entre l'émetteur ultrasonique et les microphones de 7.5 mètres (voir chapitre 16.2). Cette différence peut s'expliquer car le signal émis par l'émetteur ultrasonique rebondis contre les murs et cela perturbe le signal reçu.

L'émetteur ultrasonique UT-1640K-TT-2-R a une distance maximale entre l'émetteur ultrasonique et les microphones de 5 mètres. Cependant, la distance maximale attendue entre l'émetteur ultrasonique et les microphones de 10 mètres (voir chapitre 16.3). Cette différence peut s'expliquer car le signal émis par l'émetteur ultrasonique rebondis contre les murs et cela perturbe le signal reçu.

Lorsque la distance entre les microphones et l'émetteur ultrasonique est grande, le signal reçu par les microphones est sensible aux rebonds du signal sonore contre les murs. Cela impacte les résultats obtenus et diminue considérablement la distance maximale entre les microphones et l'émetteur ultrasonique.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

La tension d'alimentation des émetteurs ultrasoniques est inférieure à la tension d'alimentation maximale des émetteurs ultrasoniques. Ce test a montré que la tension d'alimentation impacte la pression acoustique du signal émis. En effet, les 2 composants comportent la même pression acoustique maximale. La pression acoustique des émetteurs ultrasonique est présentée dans le Tableau 40.

ÉMETTEUR ULTRASONIQUE	PRESSION ACOUSTIQUE
UT-1240K-TT-R	115[dB]
UT-1640K-TT-2-R	115[dB]

Tableau 40 : Pression acoustiques des émetteurs ultrasoniques. (Voir références 1 et 2)

Les 2 émetteurs ultrasoniques ont la même pression acoustique, cependant ils n'ont pas la même tension d'alimentation maximale. La tension d'alimentation maximale est présentée dans le Tableau 41.

ÉMETTEUR ULTRASONIQUE	TENSION D'ALIMENTATION MAXIMALE
UT-1240K-TT-R[1]	30[Vrms]
UT-1640K-TT-2-R[2]	20[Vrms]

Tableau 41 : Tension d'alimentation des émetteurs ultrasoniques

Les émetteurs ultrasoniques ont été alimenté avec une tension de 17.7[Vrms]. L'émetteur UT-1640K-TT-2-R à une tension d'alimentation maximale proche de la tension d'alimentation. Tandis que l'émetteur UT-1240K-TT-R à une tension d'alimentation maximale éloignée de la tension d'alimentation.

Les 2 émetteurs ont la même pression acoustique. Dans les tests, l'émetteur ultrasonique UT-1640K-TT-2-R à une distance maximale d'émission attendue de 10 mètres, tandis que l'émetteur ultrasonique UT-1240K-TT-R à une distance maximale d'émission attendue de 7.5 mètres. C'est pourquoi, on peut conclure que la tension d'alimentation a une influence sur la distance maximale d'émission.

En conclusion, ce test a montré que le signal sonore émis rebondit et cela impacte le signal reçu par les microphones. De plus, la tension d'alimentation des microphones à une influence sur la pression acoustique du signal émis.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

17 ANGLE D'INCIDENCE DU SON

Ce test a pour but de tester le cas où l'un des 2 plongeurs est plus haut que son partenaire de plongée. La Figure 111 illustre le principe de la mesure. L'angle β est modifié entre 45° et 75° dans les mesures suivantes.

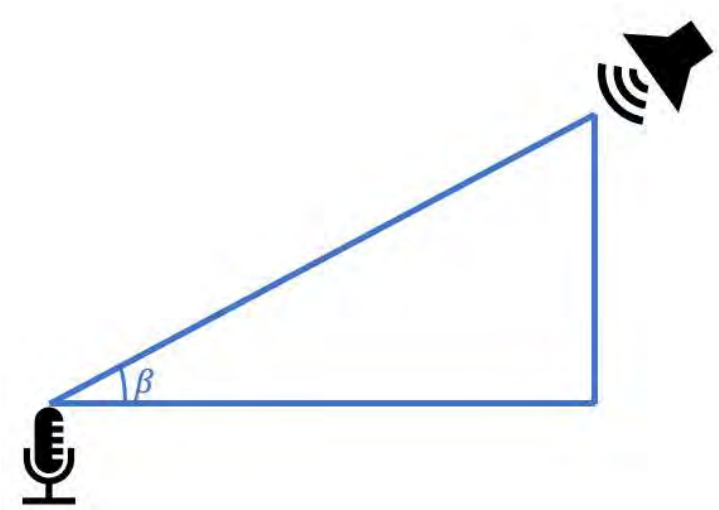


Figure 111 : Illustration des mesures

Le matériel utilisé pour ce test est illustré dans le Tableau 42.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
ÉMETTEUR ULTRASONIQUE	UT-1240K-TT-R
CIRCUIT IMPRIMÉ CARTE MICROPHONE	V1.1

Tableau 42 : Émetteur ultrasonique plus haut que les microphones : Matériel utilisé

La tension d'alimentation du l'émetteur ultrasonique est de 20[Vpp].

17.1 Protocole de test

Ce test utilise le montage présenté dans chapitre 13.

Une série de mesure a été prise pour chaque angles β différents. Des mesures sur 3 angles β ont été faites (45°,60° et 75°).

Pour chaque série de mesures, l'émetteur ultrasonique était fixe, tandis que les microphones ont été tournés entre chaque mesure. Les mesures ont été faites sur 8 angles équitablement réparti entre 0 et 360°. A chaque angle, 10 mesures ont été prises.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

17.2 Mesure avec l'angle β de 45°

Dans cette série de mesure, l'angle β est de 45°.

La Figure 112 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

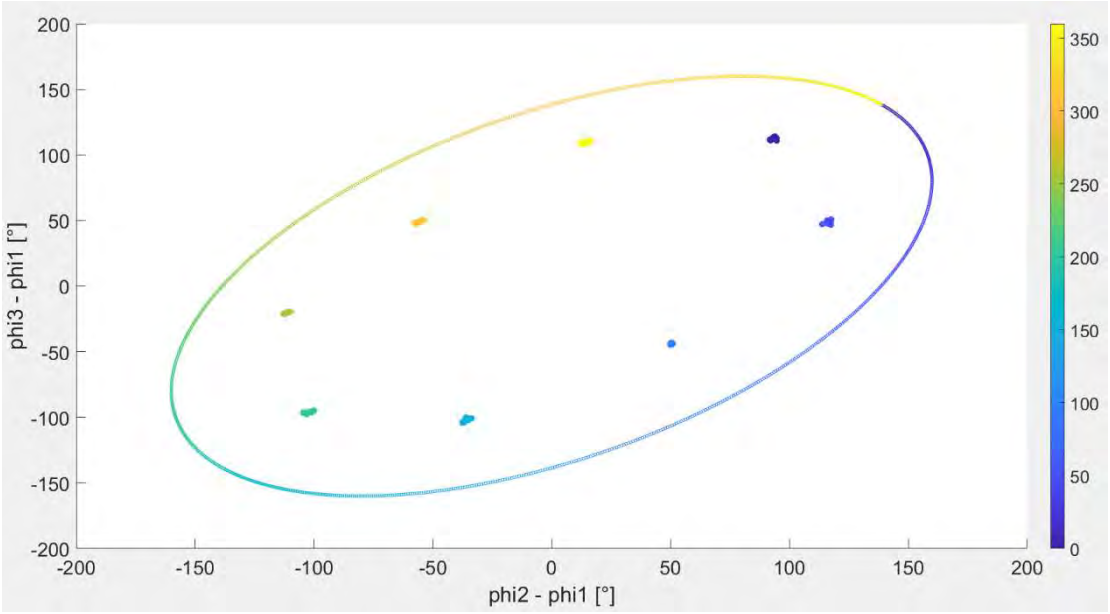


Figure 112 : Angle β de 45° : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées. De plus, l'ellipse formée par les mesures est bien plus petite que l'ellipse des résultats attendus pour un angle β de 0°. Ce résultat était attendus comme expliqué au chapitre 9.6.

La Figure 113 compare les résultats obtenus avec les résultats théoriques. Pour calculer l'angle d'arrivée du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

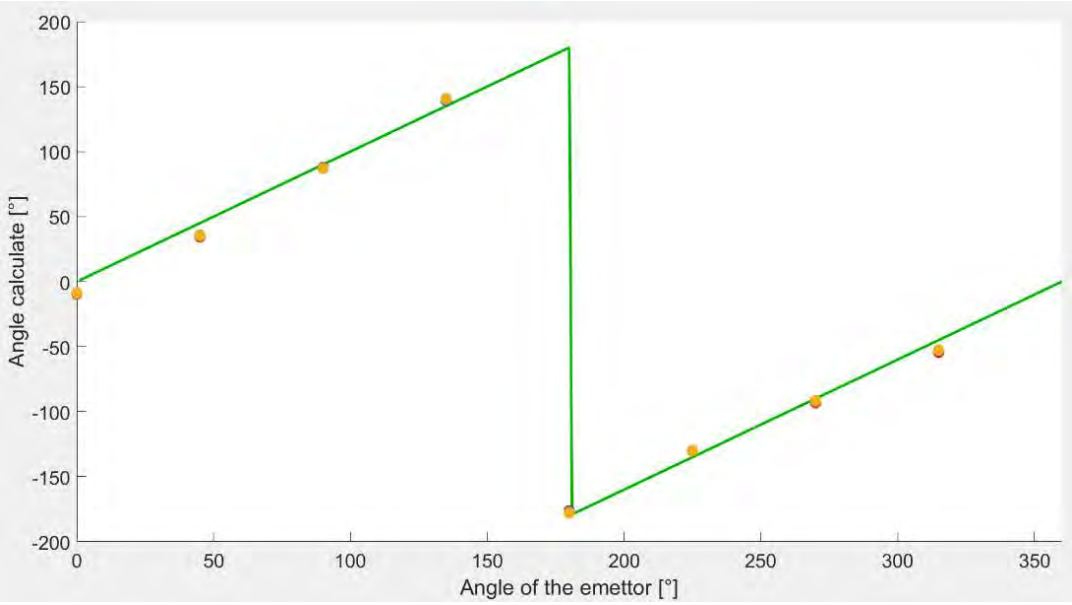


Figure 113 : Angle β de 45° : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a de l'erreur sur les angles θ obtenus. La Figure 114 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

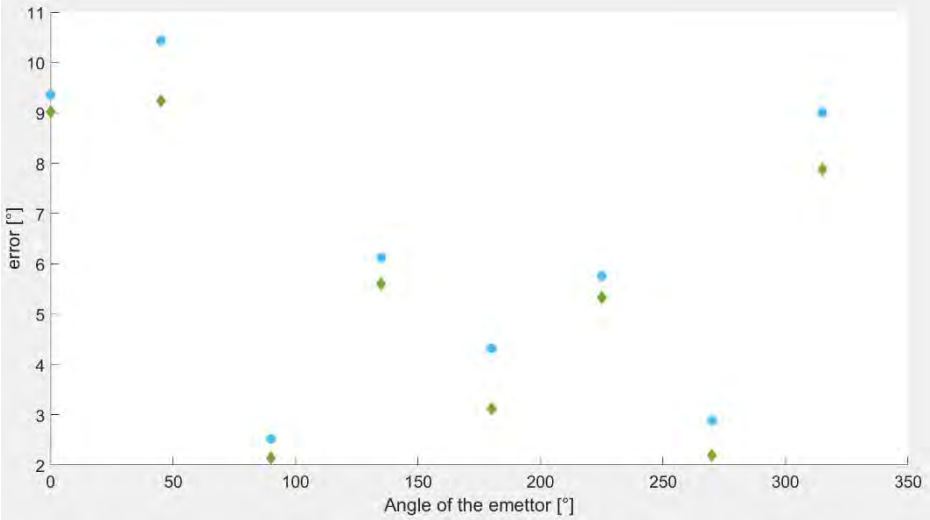


Figure 114 : Angle β de 45° : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 114, les losanges jaunes représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques violets représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 3° et 10°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 3° et 11°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont bien inférieures à l'erreur de mesure acceptable de ce projet.

17.3 Mesure avec l'angle β de 60°

Dans cette série de mesure, l'angle β est de 60°.

La Figure 115 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

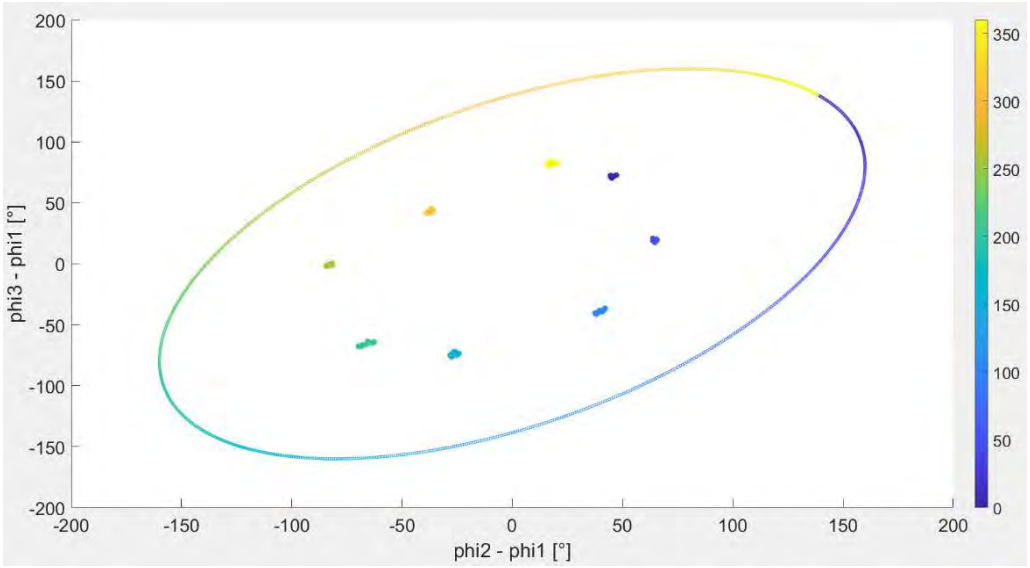


Figure 115 : Angle β de 60° : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées. De plus, l'ellipse formée par les mesures est bien plus petite que l'ellipse des résultats attendus pour un angle β de 0°. Lorsque l'angle β est de 60°, l'ellipse formée par les mesures est plus petite que l'ellipse formée par les mesures lorsque l'angle β est de 45°. Ces résultats étaient attendus comme expliqué au chapitre 9.6

La Figure 116 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

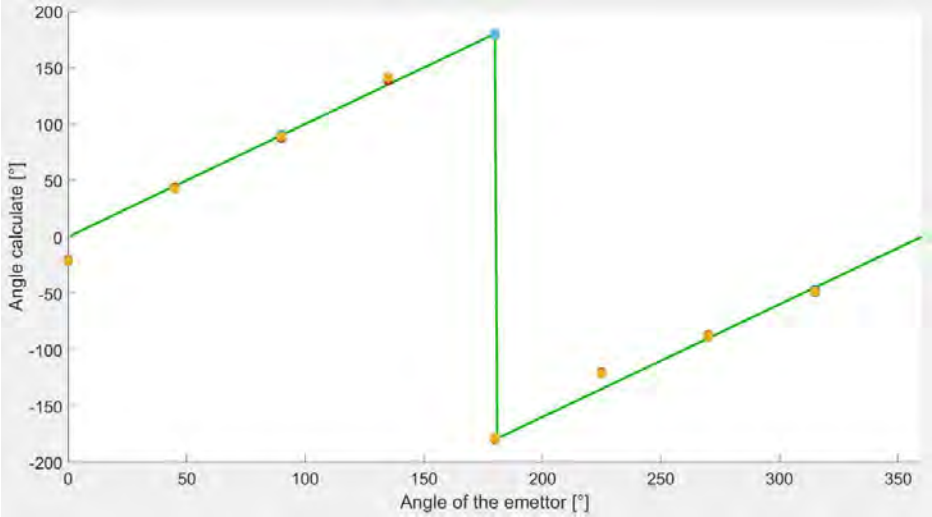


Figure 116 : Angle β de 60° : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les angles θ obtenus. La Figure 117 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

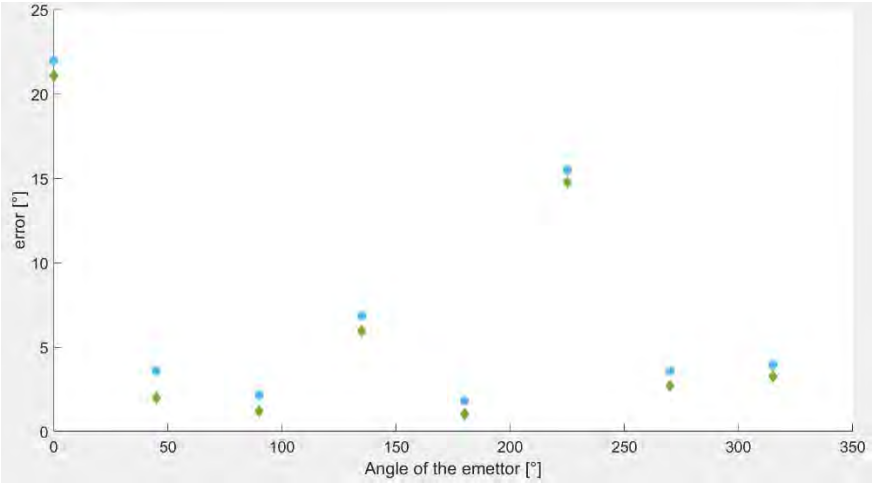


Figure 117 : Angle β de 60° : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 117, les losanges jaunes représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques violets représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 2° et 21°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 3° et 22°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont inférieures à l'erreur de mesure acceptable de ce projet.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

17.4 Mesure avec l'angle β de 75°

Dans cette série de mesure, l'angle β est de 75°.

La Figure 116 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 16 angles différents sont affichés sur le graphique.

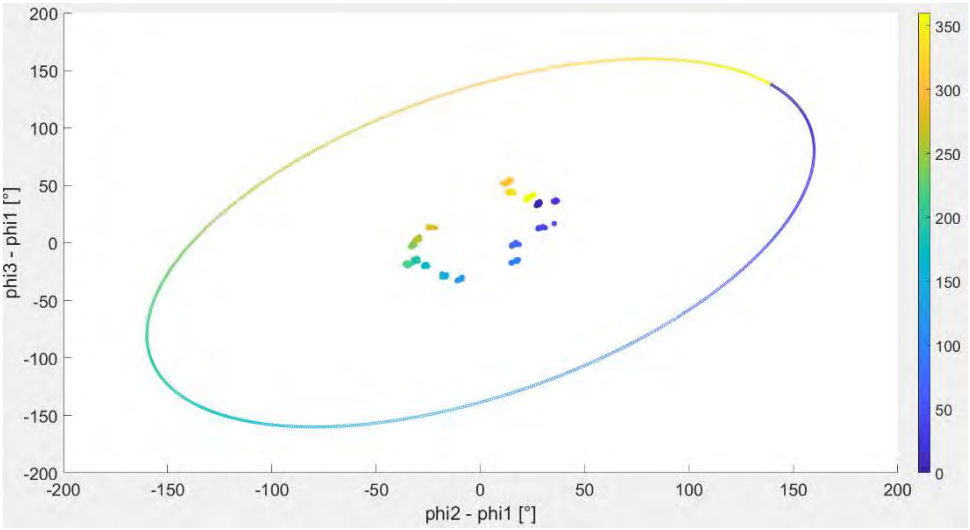


Figure 118 : Angle β de 75° : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées. De plus, l'ellipse formée par les mesures est bien plus petite que l'ellipse des résultats attendus pour un angle β de 0°. Lorsque l'angle β est de 70°, l'ellipse formée par les mesures est plus que l'ellipse formée par les mesures lorsque l'angle β est de 60°. Ces résultats étaient attendus comme expliqué au chapitre 9.6

La Figure 119 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisés. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

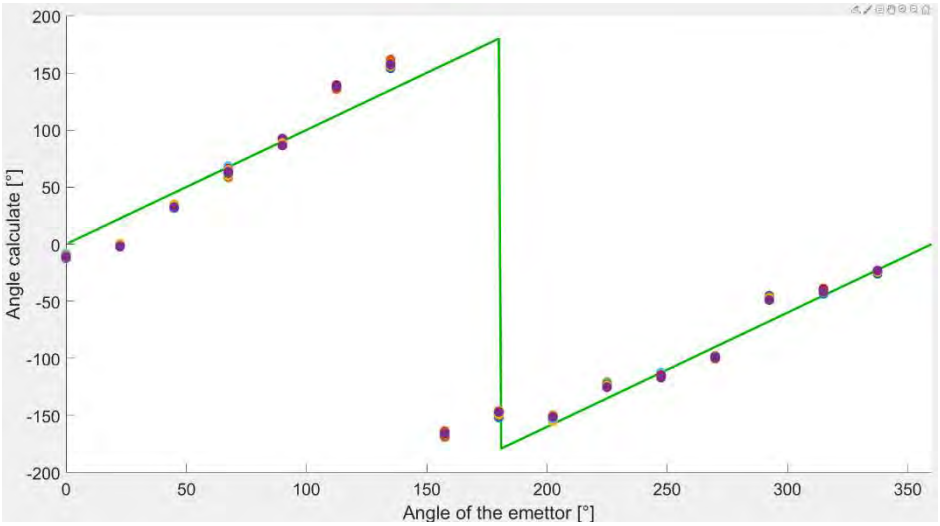


Figure 119 : Angle β de 75° : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les angles θ obtenus. La Figure 120 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

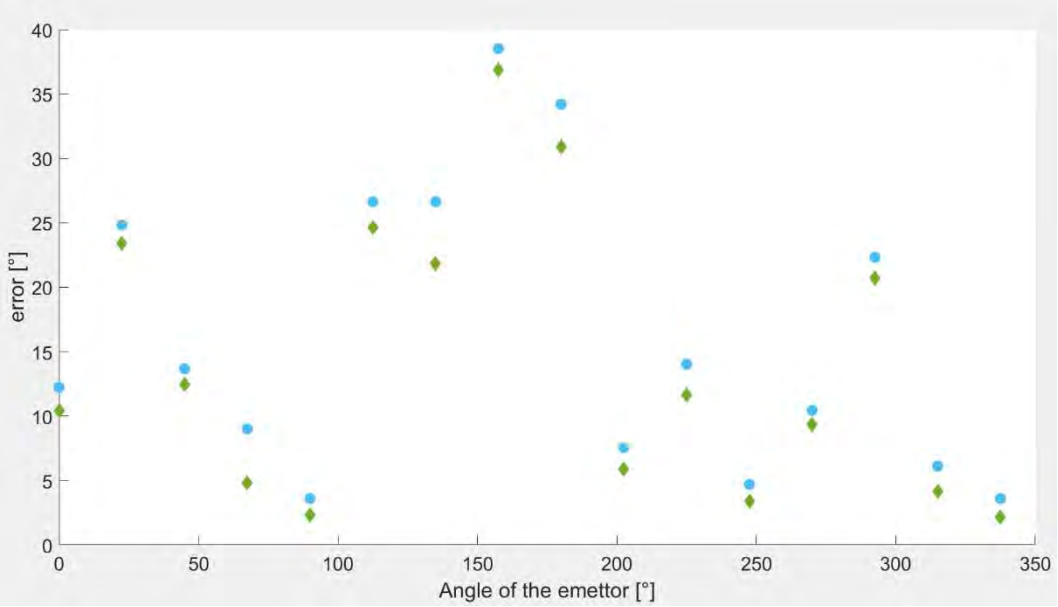


Figure 120 : Angle β de 75° : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 120, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 3° et 37°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures. Sur la Figure 118, on peut voir que l'ellipse formée par les mesures est très petite, et par conséquent une petite erreur de mesure se traduit par une grande différence sur l'angle d'arrivée du son trouvé.

L'erreur maximale des mesures varie entre 3° et 39°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont supérieurs aux erreurs de mesures acceptables.

17.5 Synthèse des résultats

Le Tableau 43 présente la synthèse des résultats lorsque l'un des 2 plongeurs est plus haut que son partenaire de plongée.

TEST	RÉSULTATS
ANGLE B DE 45°	✓
ANGLE B DE 60°	✓
ANGLE B DE 75°	✗

Tableau 43 : Synthèse des résultats sur les tests effectués lorsque l'émetteur ultrasonique est plus haut que les microphones

Il est possible de récupérer l'angle d'arrivée du son, même si l'angle d'incidence du son n'est pas nul. En effet, ces mesures ont montré qu'il est possible de récupérer l'angle d'arrivée du son jusqu'à un angle d'incidence de 60°.

À 70°, il y a trop d'erreurs de mesures. Cependant, les séries de mesures pour chaque angle présentent des résultats équivalents. Sur la Figure 118, on peut voir que l'ellipse formée par les mesures est très petite, et par conséquent une petite erreur de mesure se traduit par une grande différence sur l'angle d'arrivée du son trouvé.

Les mesures ont montré que lorsque l'angle d'incidence du son est de 60°, l'angle d'arrivée du son peut être récupéré. Au-delà d'un angle d'incidence de 70°, il est délicat de récupérer l'angle d'arrivée du son. En effet, la moindre erreur de mesure se traduit par une grande différence de l'angle d'arrivée du son trouvé.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

18 DIRECTIVITÉ DE L'ÉMETTEUR ULTRASONIQUE

Les émetteurs ultrasoniques possèdent une directivité. C'est-à-dire qu'ils n'émettent pas du son dans tout l'espace, mais seulement dans une certaine portion de l'espace. Les fiches techniques des composants UT-1240K-TT-R (voir note de fin 1) et UT-1640K-TT-2-R (voir note de fin 2) donnent la directivité des émetteurs ultrasoniques.

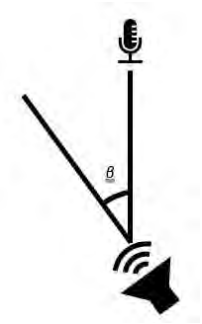


Figure 121 : angle d'émission de l'émetteur ultrasonique

La directivité des 2 émetteurs ultrasoniques est présentée dans le Tableau 44.

ÉMETTEUR ULTRASONIQUE	DIRECTIVITÉ
UT-1240K-TT-R	70[°]
UT-1640K-TT-2-R	80[°]

Tableau 44 : Directivités des émetteurs ultrasoniques utilisés

Lorsque l'angle β du test est égal à la directivité de l'émetteur ultrasonique, le signal reçu à une atténuation de 6dB. Au-delà de la directivité de l'émetteur ultrasonique, le signal reçu sera de plus en plus atténué.

Un test a été effectué afin de déterminer l'angle d'émission β jusqu'au quel il est possible de récupérer l'angle d'arrivée du son. Le matériel utilisé pour ce test est illustré dans le Tableau 45.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
ÉMETTEUR ULTRASONIQUE	UT-1240K-TT-R
ÉMETTEUR ULTRASONIQUE	UT-1640K-TT-2-R
CIRCUIT IMPRIMÉ CARTE MICROPHONE	V1.1

Tableau 45 : Test sur la directivité des émetteurs ultrasonique : Matériel utilisé

La tension d'alimentation des émetteurs ultrasoniques est de 20[Vpp].

18.1 Protocole de tests

Ce test utilise le montage présenté dans chapitre 13.

Premièrement, Les 2 émetteurs ultrasoniques ont été testés pour connaître leur angle d'émission β. Pour chaque microphone, des mesures ont été prises pour les angles β suivants :

0°	45°	80°
20°	70°	90°

L'angle d'arrivée du son est fixe pour toutes les mesures et vaut 0°.

Dans un deuxième temps, des séries de mesures ont été prises pour les angles β maximaux de chaque émetteur. Ce deuxième test utilise le montage présenté dans chapitre 13. Les mesures ont été faites sur 16 angles équitablement réparti entre 0 et 360°. A chaque angle, 10 mesures ont été prises.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

18.2 Microphone UT-1240K-TT-R

Ce chapitre traite des mesures effectuées sur le microphone UT-1240K-TT-R.

Pour rappel, la fiche technique de cet émetteur ultrasonique indique qu'il a un angle d'émission de 70°. Lorsque l'angle d'émission est de 70°, le signal émis à une atténuation de 6[dB].

Des tests avec des angles d'émission β différents, pour tester le composant.

La Figure 122 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 0°. L'amplitude des signaux est de 1.

La Figure 123 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 70°. L'amplitude des signaux est d'environ 0.6.

La Figure 124 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 80°. L'amplitude des signaux est d'envrion 0.3.

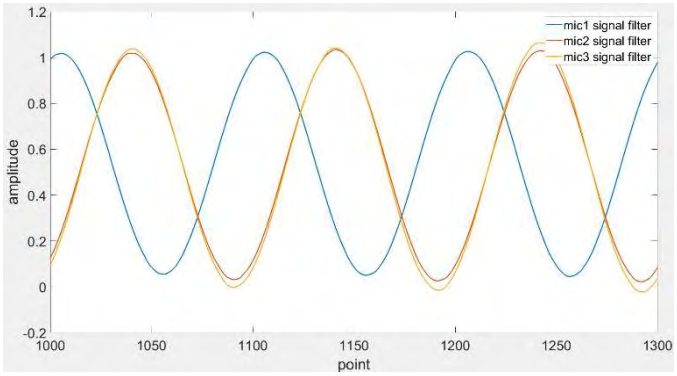


Figure 122 : Signaux microphones avec angle β = 0°

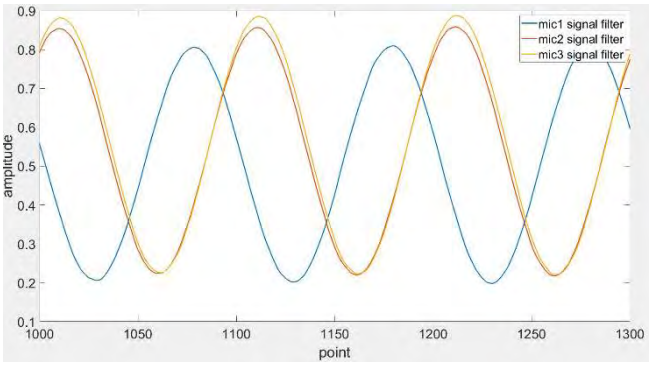


Figure 123 : Signaux microphones avec angle β = 70°

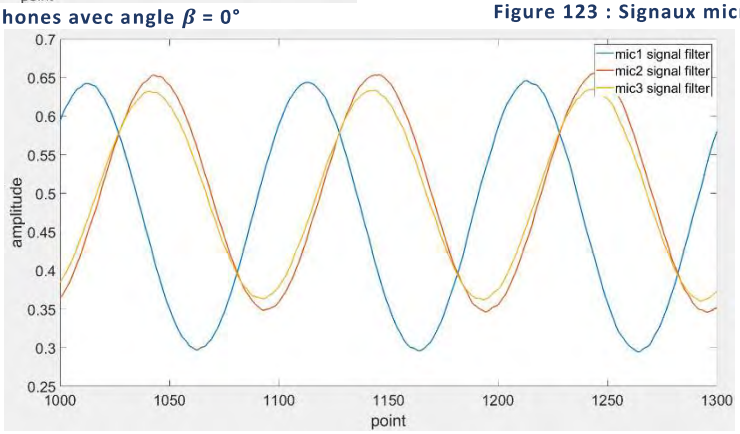


Figure 124 : Signaux microphones avec angle β = 80°

Ce qu'on aperçoit c'est que plus l'angle β est grand, plus l'amplitude du son captés par le microphones diminue.

Lorsque l'angle d'émission β est plus petit au égal à 80°, il est possible de récupérer l'angle d'arrivée du son. Lorsque l'angle d'émission β est plus grand que 80°, il n'est alors plus possible de récupérer l'angle d'arrivée du son. En effet, le signal capté par les microphones à une amplitude très faible. Cepedant, avec le test effectué en chapitre 15, l'amplitude des signaux à 80° est suffisante pour récupérer l'angle d'arrivée du son. En effet, le est effectué en chapitre 15 a montré qu'il était possible de récupérer l'angle d'arrivée du son avec une ampltiude des signaux de 0.03.

Ici le problème à partir de 80° c'est que, lors d'une prise de mesures, le déphasage entre les différents signaux n'ai jamais le même.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

18.2.1 ANGLE D'ARRIVÉE DU SON AVEC UN ANGLE B DE 70°

Dans cette série de mesure, l'angle β est de 70°.

La Figure 125 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 16 angles différents sont affichés sur le graphique.

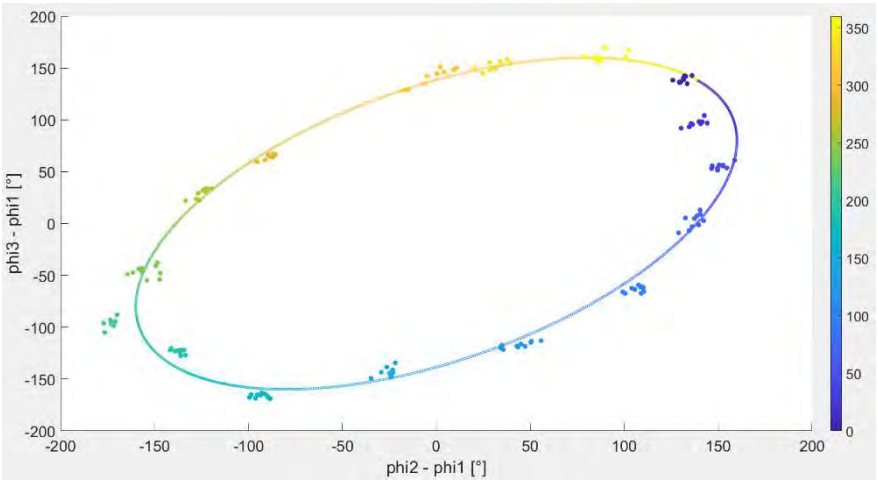


Figure 125 : Angle β de 70° : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées.

La Figure 126 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

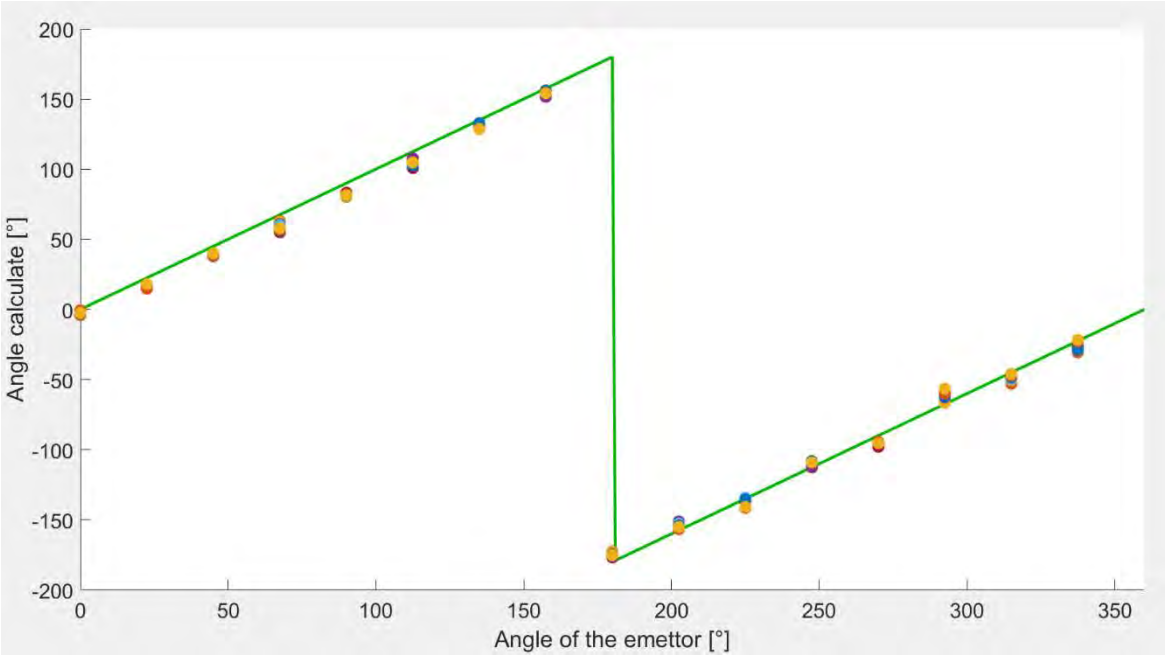


Figure 126 : Angle β de 70° : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les angles θ obtenus. La Figure 127 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

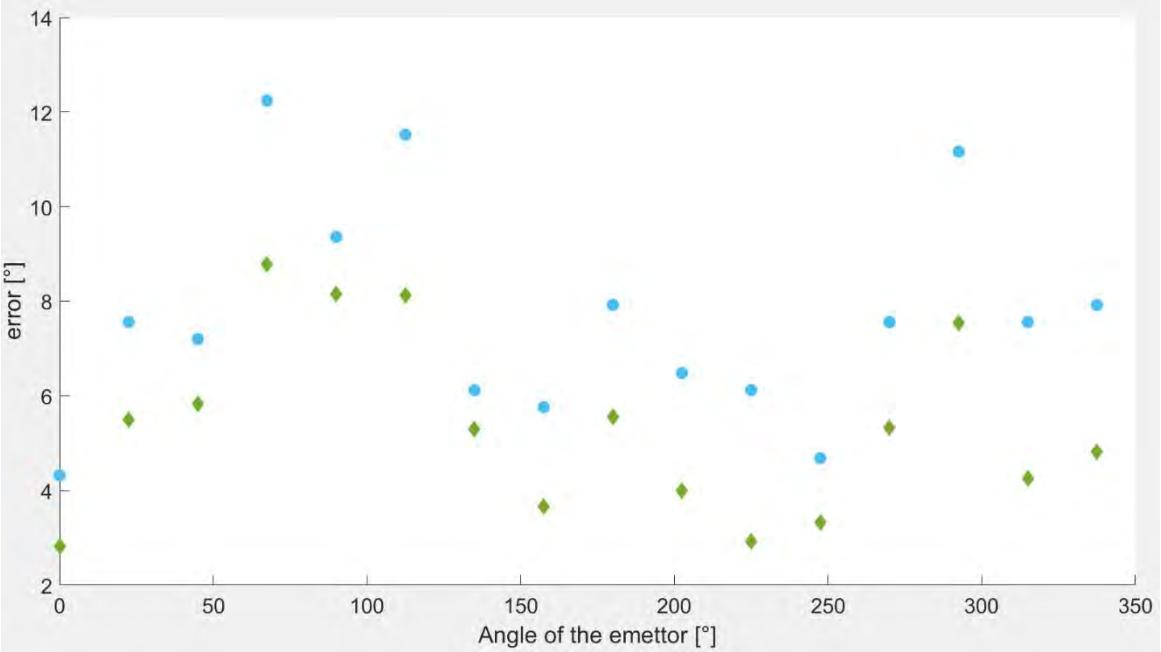


Figure 127 : Angle β de 70° : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 127, les losanges verts représentes l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 3° et 9°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 4° et 13°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures obtenues sont inférieures à l'erreur de mesure acceptable de ce projet.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

18.3 Microphone UT-1640K-TT-2-R

Ce chapitre traite des mesures effectuées sur le microphone UT-1640K-TT-2-R.

Pour rappel, la fiche technique de cet émetteur ultrasonique indique qu'il a un angle d'émission de 80°. Lorsque l'angle d'émission est de 80°, le signal émis à une atténuation de 6[dB].

Des tests avec des angles d'émission β différents, pour tester le composant.

La Figure 128 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 0°. L'amplitude des signaux est de 1.

La Figure 129 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 45°. L'amplitude des signaux est d'environ 0.2.

La Figure 130 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 60°. L'amplitude des signaux est d'envrion 0.06.

La Figure 131 représente les signaux filtré captés par les 3 microphones, lorsque l'angle β vaut 60°. L'amplitude des signaux est d'envrion 0.04.

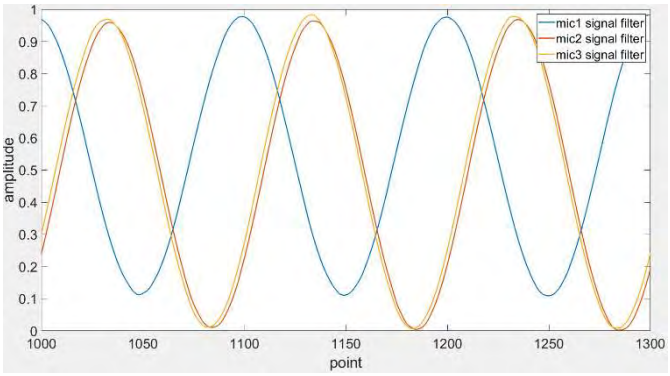


Figure 128 : Signaux microphones avec angle β = 0°

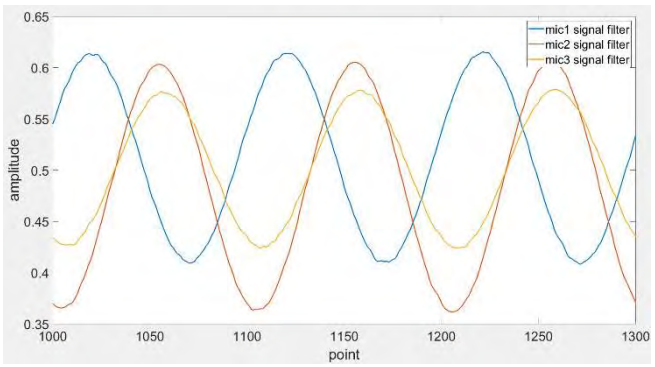


Figure 129 : Signaux microphones avec angle β = 45°

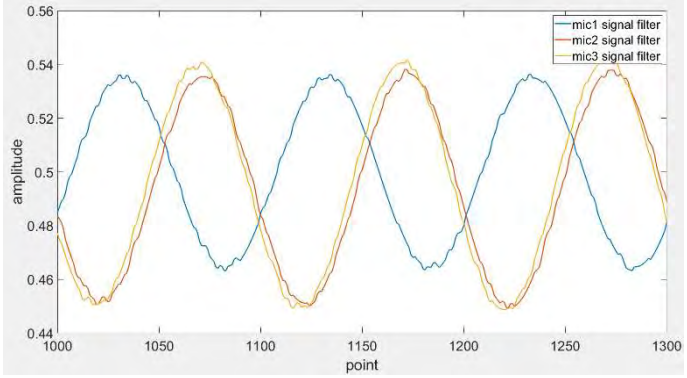


Figure 130 : Signaux microphones avec angle β = 60°

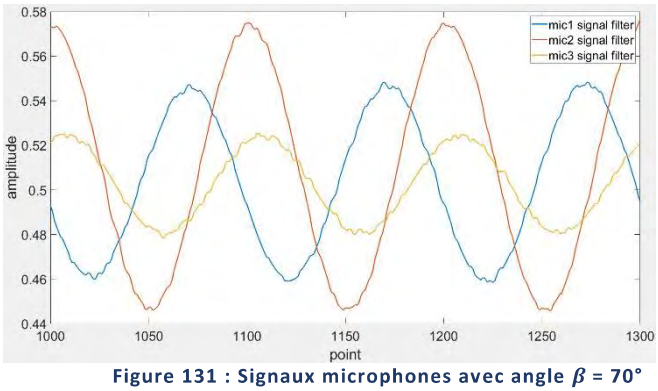


Figure 131 : Signaux microphones avec angle β = 70°

Ce qu'on aperçoit c'est que plus l'angle β est grand, plus l'amplitude du son captés par le microphones diminue.

Lorsque l'angle d'émission β est plus petit au égal à 70°, il est possible de récupérer l'angle d'arrivée du son. Lorsque l'angle d'émission β est plus grand que 70°, il n'est alors plus possible de récupérer l'angle d'arrivée du son. En effet, le signal capté par les microphones à une amplitude très faible. Cepedant, avec le test effectué en chapitre 15, l'amplitude des signaux à 70° est suffisante pour récupérer l'angle d'arrivée du son. En effet, le est effectué en chapitre 15 a montré qu'il était possible de récupérer l'angle d'arrivée du son avec une ampltiude des signaux de 0.03.

Ici le problème à partir de 70° c'est que, lors d'une prise de mesures, le déphasage entre les différents signaux n'ai jamais le même.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

18.3.1 ANGLE D'ARRIVÉE DU SON AVEC UN ANGLE B DE 60°

Dans cette série de mesure, l'angle β est de 60°.

La Figure 132 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 16 angles différents sont affichés sur le graphique.

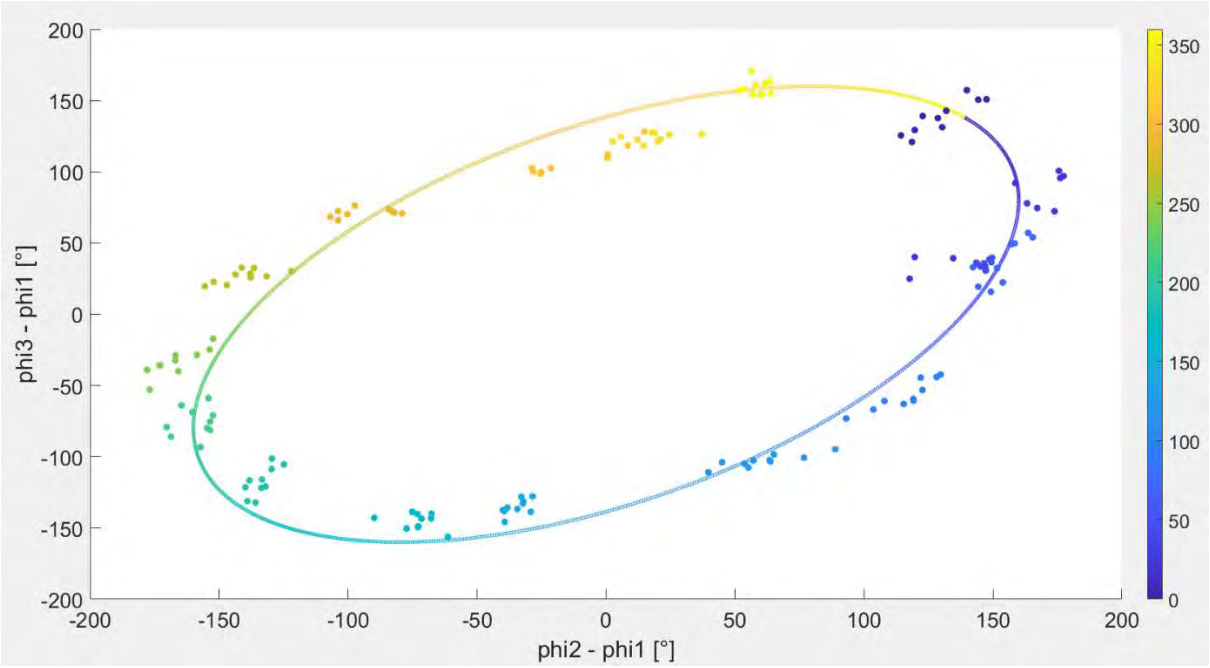


Figure 132 : Angle β de 45° : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont assez dispersées.

La Figure 133 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

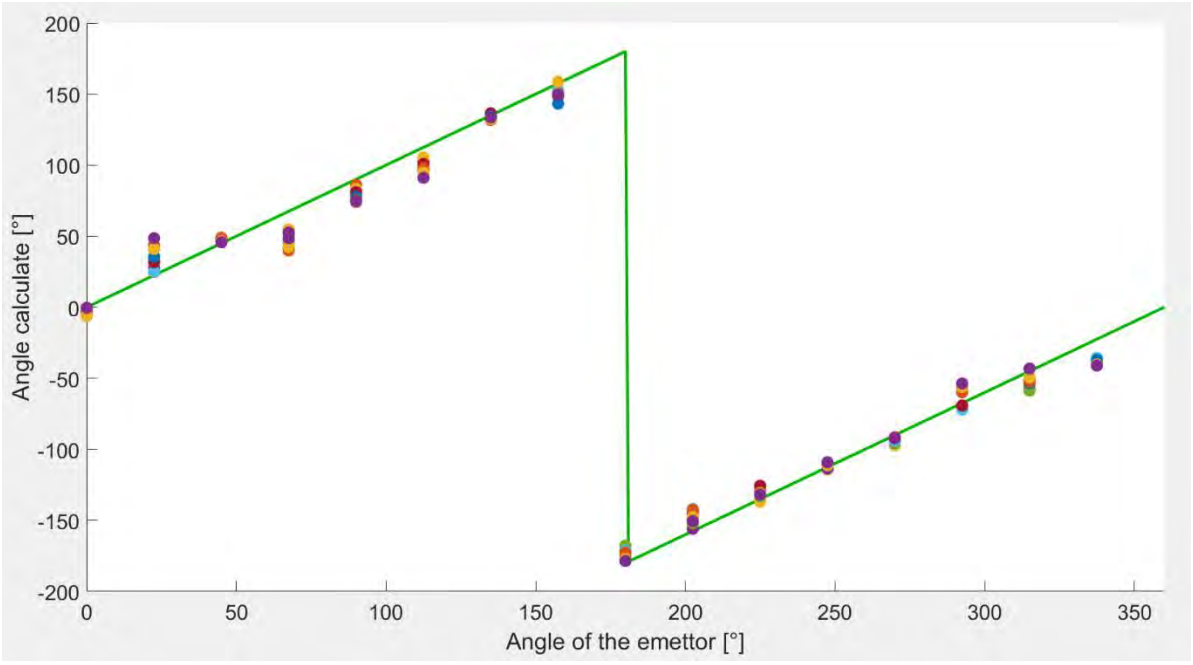


Figure 133 : Angle β de 45° : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a de l'erreur sur les angles θ obtenus. La Figure 134 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

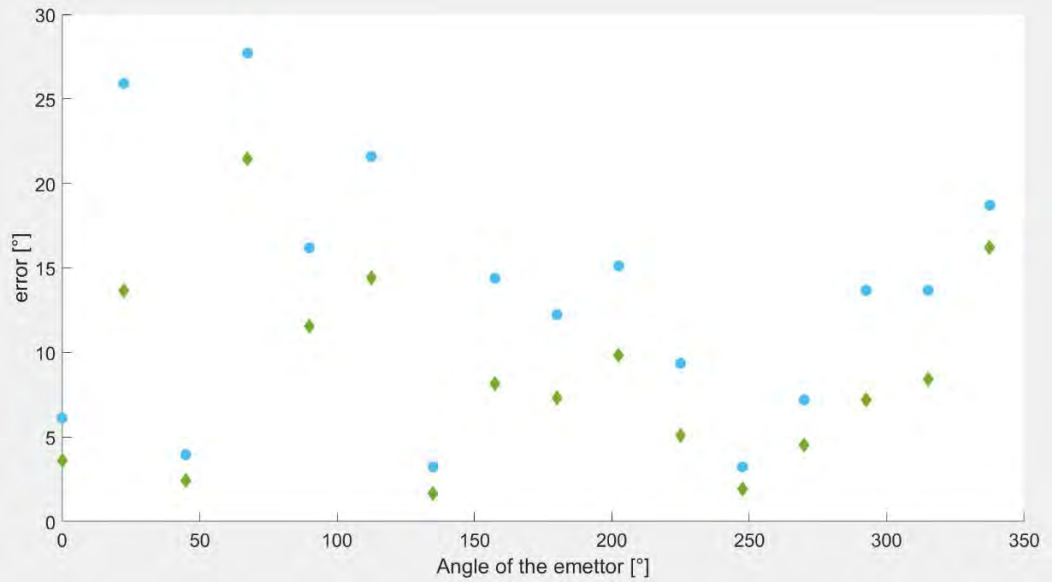


Figure 134 : Angle β de 45° : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 134, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 5° et 22°. Une partie de cette erreur vient de placement de la carte comportant les microphones qui n'est pas placée au degré près lors des mesures.

L'erreur maximale des mesures varie entre 5° et 27°.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5°. Les erreurs de mesures moyennes obtenues sont inférieures à l'erreur de mesure acceptable de ce projet. Il y a 2 erreurs maximales qui dépasse l'erreur acceptable. On est donc à la limite de ce qui est acceptable.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

18.4 Synthèse des résultats

Ce chapitre traite de la synthèse des résultats sur les tests de la directivité de l'émetteur ultrasonique.

18.4.1 Émetteur ultrasonique UT-1240K-TT-R

Les tests effectués ont démontré qu'il est possible de récupérer l'angle d'arrivée du son avec un angle β de 70°. Cela signifie que cet émetteur ultrasonique peut émetteur à 140°. Il en faudrait par conséquent 3 au minimum pour couvrir les 360° nécessaire.

Les tests ont montré que lorsque l'angle β est de 70°, l'amplitude du signal reçu par les microphones est plus faible que lorsque l'angle β est de 0°. Lorsque l'amplitude du signal reçu est plus faible, cela va impacter la distance maximale d'émission.

Il est donc préférable d'utiliser plus que 3 émetteurs ultrasoniques afin d'avoir une émission homogène et forte à 360° autour du plongeur.

18.4.2 Émetteur ultrasonique UT-1640K-TT-2-R

Les tests effectués ont démontré qu'il est possible de récupérer l'angle d'arrivée du son avec un angle β de 60°. Cela signifie que cet émetteur ultrasonique peut émetteur à 120°. Il en faudrait par conséquent 3 au minimum pour couvrir les 360° nécessaire.

Les tests ont montré que lorsque l'angle β est de 60°, l'amplitude du signal reçu par les microphones est plus faible que lorsque l'angle β est de 0°. Lorsque l'amplitude du signal reçu est plus faible, cela va impacter la distance maximale d'émission.

Il est donc préférable d'utiliser plus que 3 émetteurs ultrasoniques afin d'avoir une émission homogène et forte à 360° autour du plongeur.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

19 TEST BOÎTIER

Un boîtier doit être développé pour que le système puisse aller sous l'eau. Plusieurs matériaux pour ce boîtier ont été testés.

Le matériel utilisé pour ce test est illustré dans le Tableau 46.

MATÉRIEL	RÉFÉRENCE
GÉNÉRATEUR DE FONCTIONS	Agilent 33220A
AMPLIFICATEUR DE TENSION	TOELLNER TOE 7607
ÉMETTEUR ULTRASONIQUE	UT-1640K-TT-2-R
CIRCUIT IMPRIMÉ CARTE MICROPHONE	V1.1

Tableau 46 : Test avec un boîtier : Matériel utilisé

La tension d'alimentation des émetteurs ultrasoniques est de 50[Vpp].

19.1 Test avec des différents matériaux

Plusieurs matériaux ont été testés. Ces matériaux sont listés dans le Tableau 47.

MATÉRIAUX	FIGURE	AMPLITUDE
PLASTIQUE FIN	Figure 135	0.15
SCOTCH	Figure 136	0.01
SILICONE	Figure 137	0.08
AUTOCOLLANT	Figure 138	0.15
PLEXIGLAS	Figure 139	0.01

Tableau 47 : Liste des matériaux testés et l'amplitude des signaux mesurés

Les figures suivantes présentent les résultats des mesures effectués avec divers matériaux.

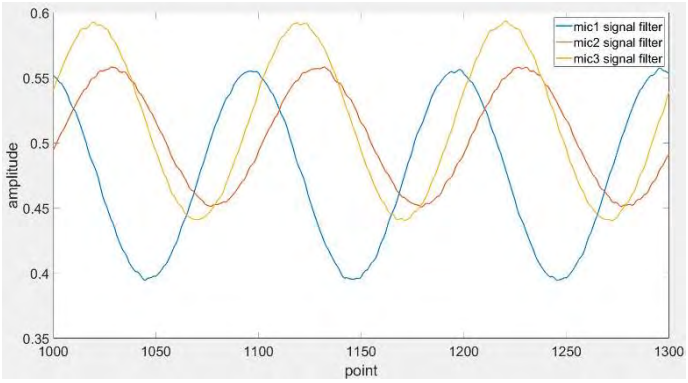


Figure 135: Signaux microphones plastique fin sur les microphones

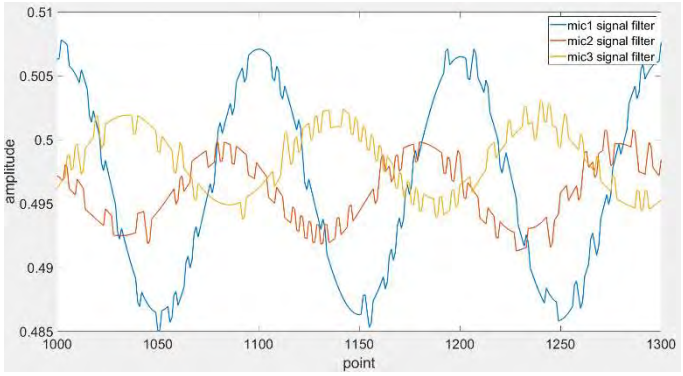


Figure 136 : Signaux microphones scotch sur les microphones

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Les figures suivantes présentent les résultats des mesures effectués avec divers matériaux.

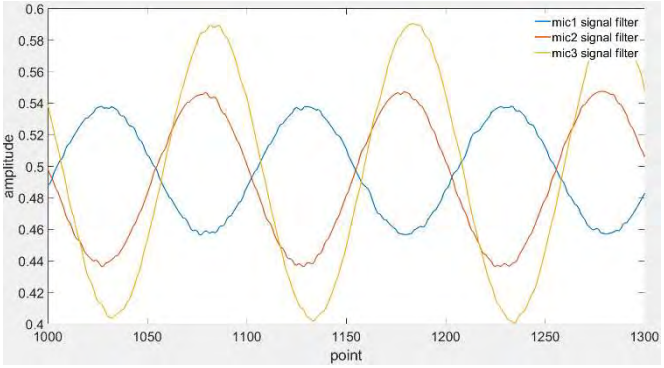


Figure 137 : Signaux microphones silicone sur les microphones

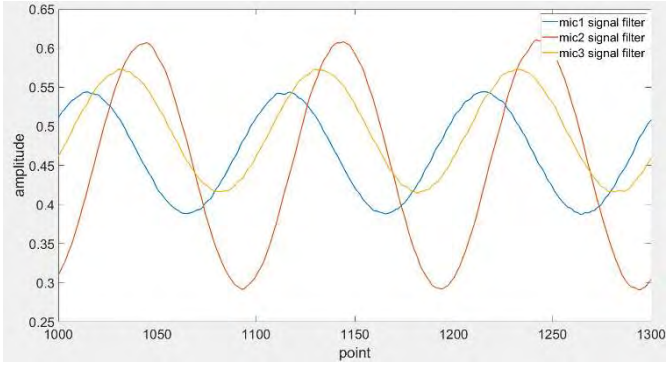


Figure 138 : Signaux microphones autocollant sur les microphones

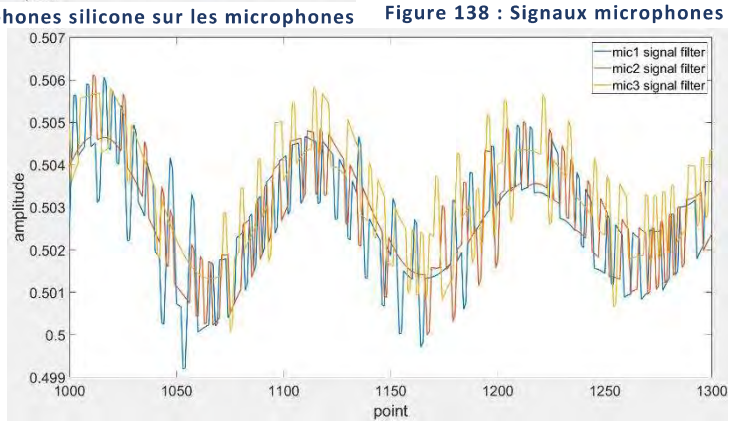


Figure 139 : Signaux microphones plexiglas sur les microphones

L'amplitude des signaux reçus par les microphones dépends des matériaux testés. Les amplitudes varinte entre 0.1 et 0.15.

Le test pésenté en chapitre 15 avait l'amplitude des signaux des microphones de 0.03 et ce test a montré qu'il était possible de récupérer l'angle d'arrivée du son avec de tels amplitudes des signaux des microphones.

Seul les tests avec le scotch et le Plexiglas comporte des amplitudes inférieures à 0.03. En effet, les signaux des microphones ont une amplitude de 0.01 lorsque du scotch ou du Plexiglas est déposé sur les microphones. Ces matériaux seront donc mis de côté, car les amplitudes des signaux reçus sont beucoup trop faible.

Outre l'amplitude, la phase des signaux est aussi importante. Dans ces tests l'angle d'arrivée du son est fixe et vaut 0°. Les signaux du micophone numéro 2 et du microphone numéro 3 doivent être en phases.

Les signaux du microphones numéro 2 et du microphones numéro 3 sont en phases dans les tests avec les matériaux suivants : plastique fin et silicone. Les signaux du microphone numéro 2 et du micophone numéro 3 ne sonpt pas en phases lorsqu'un autocollant est collé sur les microphones.

Il ne reste donc uniquement le silicone et le plastique fin.

Des tests supplémentaires ont été fait avec le plastique en chapitre 19.2.

Dest tests supplémentaire ont été fait avec le silicone en chapitre 19.3.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

19.2 Test avec du plastique fin

Des tests supplémentaires ont été effectués avec le plastique fin.

Le montage suivant a été utilisé pour construire le boîtier.

Figure 140 : Anneau pour les tests

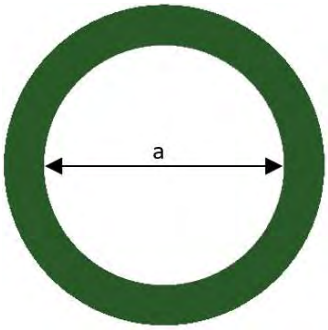


Figure 141 : Paramètre a

Figure 142 : Paramètre b

Dans les tests, les paramètre a et b présenté aux Figure 141 et Figure 142 ont été variés. La liste de toutes les configurations des paramètres a et b est présentée dans le Tableau 48.

TEST	VALEUR DU PARAMÈTRE A	VALEUR DU PARAMÈTRE B
1	10[mm]	2[mm]
2	10[mm]	3[mm]
3	10[mm]	5[mm]
4	15[mm]	2[mm]
5	15[mm]	3[mm]
6	15[mm]	5[mm]
7	20[mm]	2[mm]
8	20[mm]	3[mm]
9	20[mm]	5[mm]

Tableau 48 : Liste des tests avec les paramètres a et b

La feuille de plastique a été collée sur l'anneau présenté en Figure 142 et ce montage a été collé sur le circuit imprimé. Le montage résultant est illustré en Figure 143.

Figure 143 : Montage pour les tests

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Les figures suivantes présentes les signaux des microphones obtenus en testant les 9 configurations différentes des paramètres a et b.

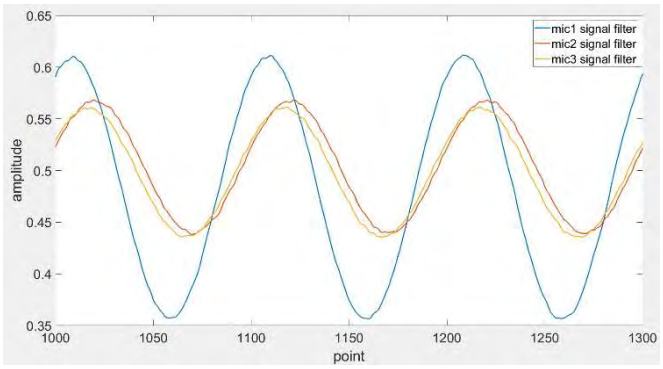


Figure 144: Test 1 : a = 10[mm] et b = 2[mm]

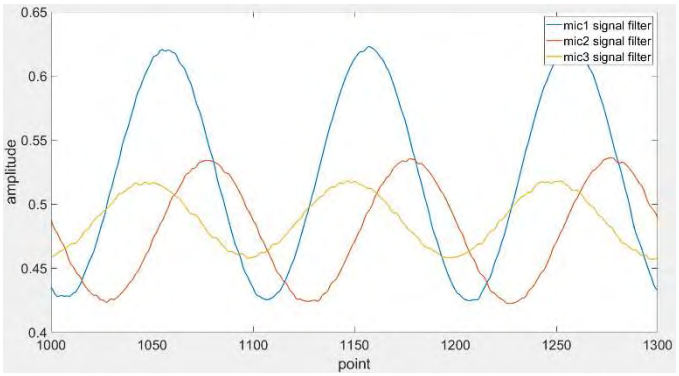


Figure 145 : Test 2 : a = 10[mm] et b = 3[mm]

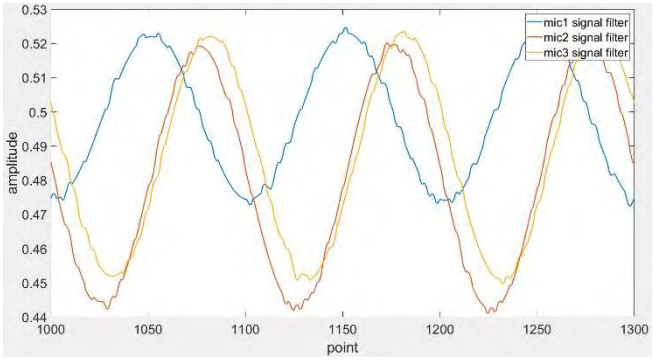


Figure 146 : Test 3 : a = 10[mm] et b = 5[mm]

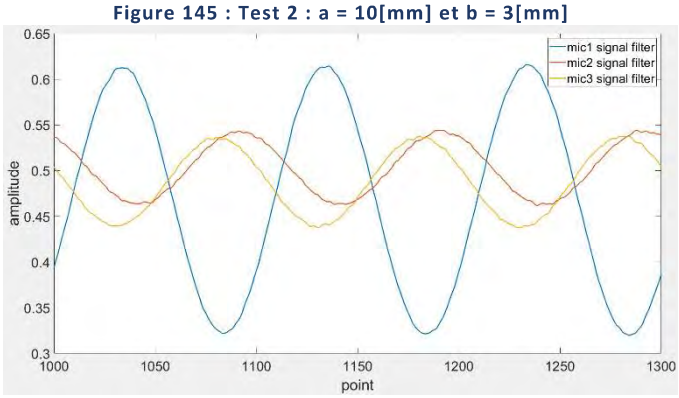


Figure 147 : Test 4 : a = 15[mm] et b = 2[mm]

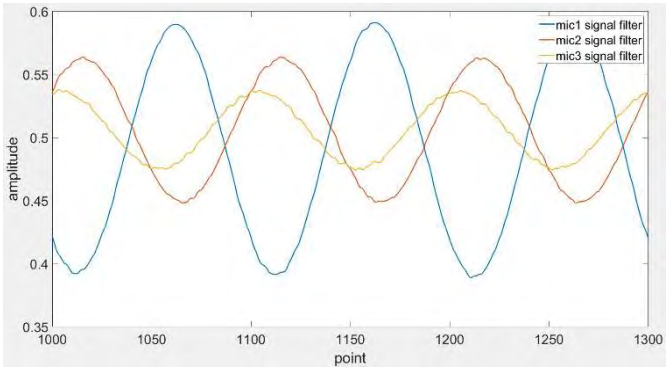


Figure 148 : Test 5 : a = 15[mm] et b = 3[mm]

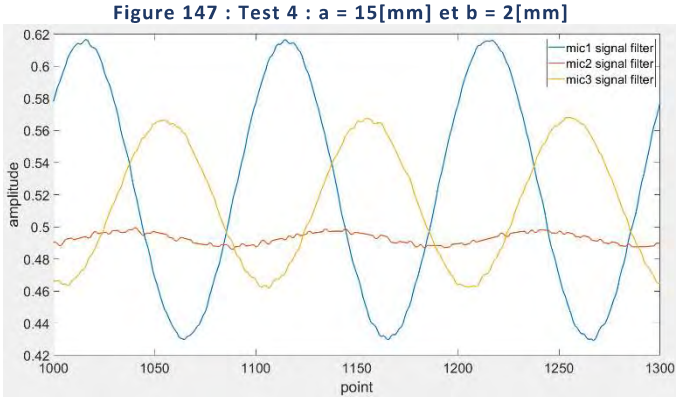


Figure 149 : Test 6 : a = 15[mm] et b = 5[mm]

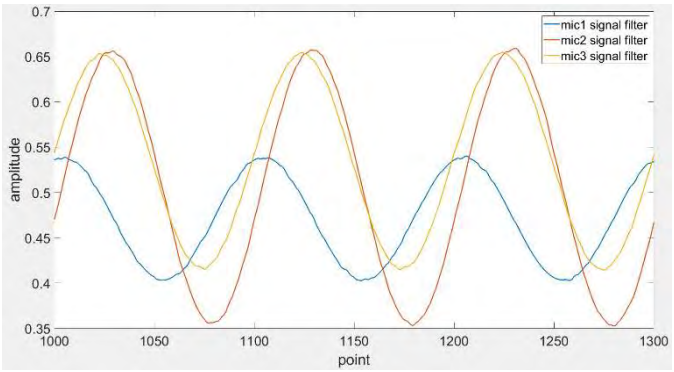


Figure 150 : Test 7 : a = 20[mm] et b = 2[mm]

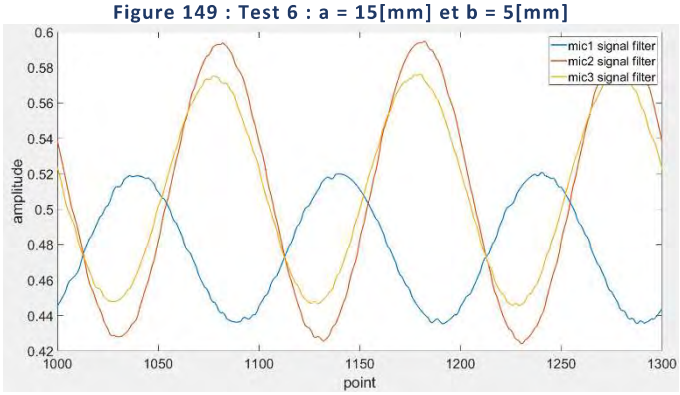


Figure 151: Test 8 : a = 20[mm] et b = 3[mm]

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

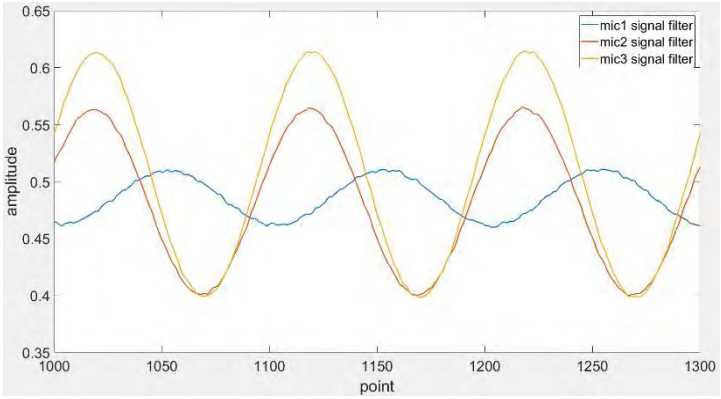


Figure 152 : Test 9 : a = 20[mm] et b = 5[mm]

Les amplitudes des signaux reçus par les microphones est présenté dans le Tableau 49.

TEST	PARAMÈTRE A	PARAMÈTRE B	FIGURE	AMPLITUDE
1	10[mm]	2[mm]	Figure 144	0.12
2	10[mm]	3[mm]	Figure 145	0.07
3	10[mm]	5[mm]	Figure 146	0.04
4	15[mm]	2[mm]	Figure 147	0.08
5	15[mm]	3[mm]	Figure 148	0.05
6	15[mm]	5[mm]	Figure 149	0.01
7	20[mm]	2[mm]	Figure 150	0.15
8	20[mm]	3[mm]	Figure 151	0.08
9	20[mm]	5[mm]	Figure 152	0.04

Tableau 49 : Amplitudes des signaux reçus pour les 9 tests effectués

On observe dans le Tableau 49 que plus le paramètre b est grand, plus l'amplitude du signal reçu est faible.

Le test pésenté en chapitre 15 avait l'amplitude des signaux des microphones de 0.03 et ce test a montré qu'il était possible de récupérer l'angle d'arrivée du son avec de tels amplitudes des signaux des microphones.

Dans les tests effectués, les signaux captés par les microphones ont une amplitude se situant entre 0.01 et 0.12. Dans un seul test, les signaux captés par les microphones ont une ampltiude inférieure à 0.03. C'est le test numéro 6.

Outre l'amplitude, la phase des signaux est aussi importante. Dans ces tests l'angle d'arrivée du son est fixe et vaut 0°. Les signaux du micophone numéro 2 et du microphone numéro 3 doivent être en phases.

Les signaux du microphones numéro 2 et du microphones numéro 3 sont en phases dans tout les tests à l'exceptions des tests numéro 2,4 et 6. En effet, dans les tests numéro 2,4 et 6, les signaux du microphones numéro 2 et du microphones numéro 3 ne sont pas en phases.

Les meilleurs résultats ont été obtenus avec le montage avec les paramètre a et b suivant :

- a = 20[mm]
- b = 2[mm]

Des tests ont été effectués avec cette configuration pour déterminer s'il est possible de récupérer l'angle d'arrivée du son. Ces tests sont présentés dans le chapitre 19.2.1.

Des tests ont aussi été effectués pour déterminer s'il est possible de récupérer l'angle d'arrivée du son, lorsque la feuille de plastique est directement appliquée sur les microphone. Ces tests sont présentés dans le chapitre 19.2.2.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

19.2.1 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON

Un test a été effectué pour récupérer l'angle d'arrivée du son. L'anneau avec le paramètre a = 20[mm] et le paramètre b = 2 [mm] a été testé. Ces paramètres ont été choisi, car ils donnent les meilleurs résultats sur les amplitudes des signaux reçus par les microphones.

La Figure 153 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

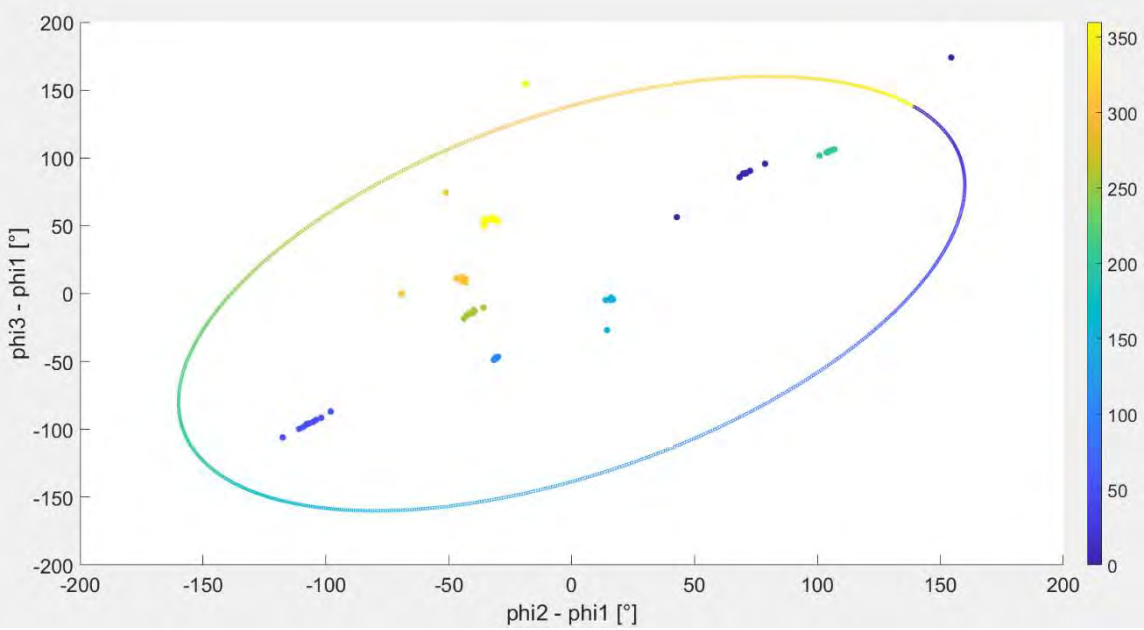


Figure 153 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont peu regroupées. De plus, les mesures faites à chaque angle sont placées de manière très aléatoire sur le graphe.

La Figure 154 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

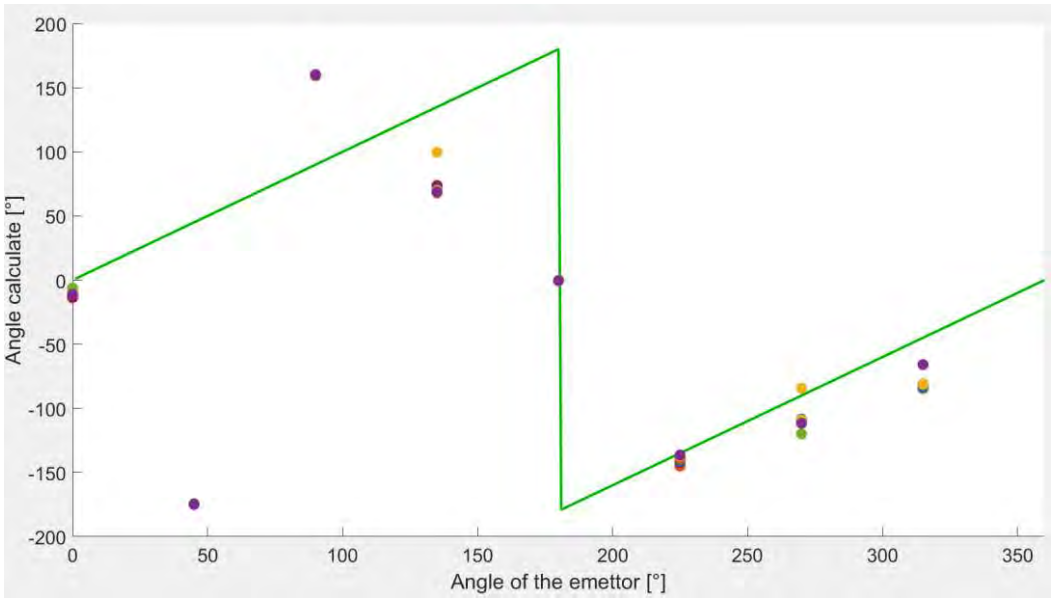


Figure 154 : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a beaucoup d'erreurs sur les angles θ obtenus. La Figure 155 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

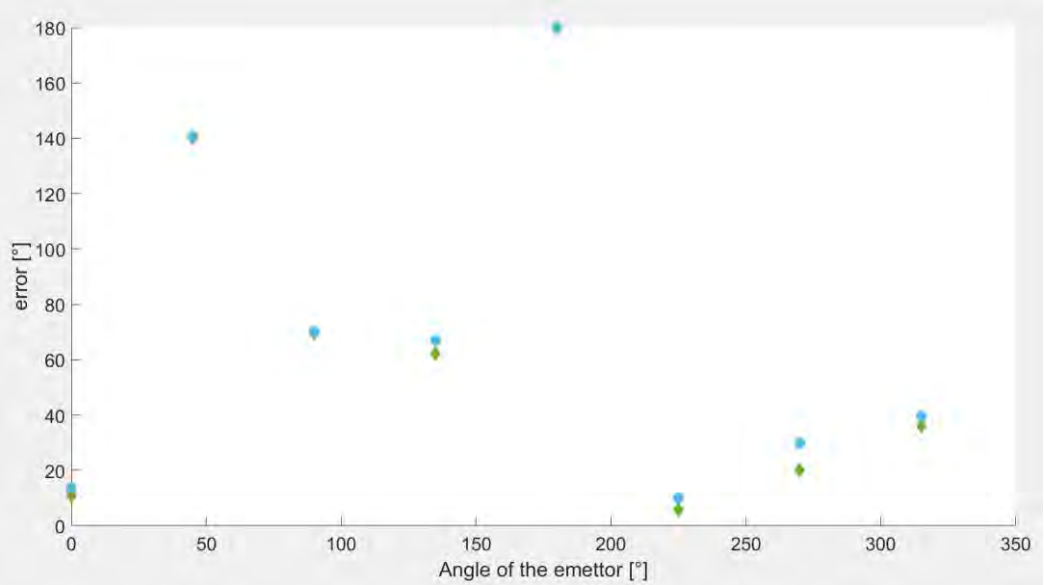


Figure 155 : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 155, les losanges verts présentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur obtenue est beaucoup trop grande. Le système est inutilisable avec de telles erreurs de mesures.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

19.2.2 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON

Un test a été effectué pour récupérer l'angle d'arrivée du son. La feuille de plastique a été directement posée sur les microphones dans ce test.

La Figure 156 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les compare à la théorie. Les mesures faites aux 16 angles différents sont affichés sur le graphique.

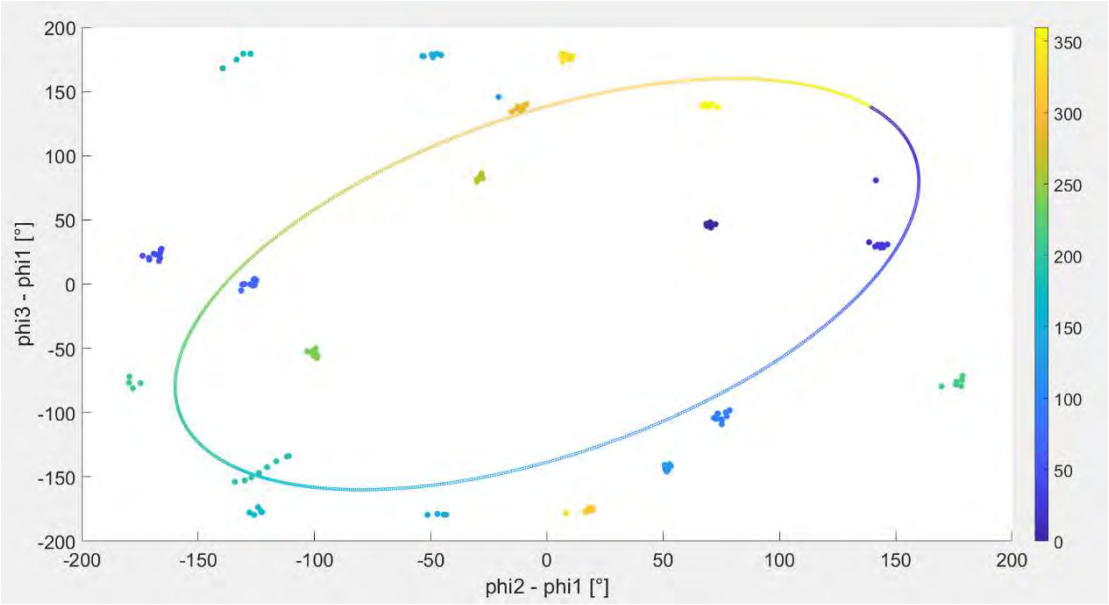


Figure 156: Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont assez regroupées.

La Figure 157 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

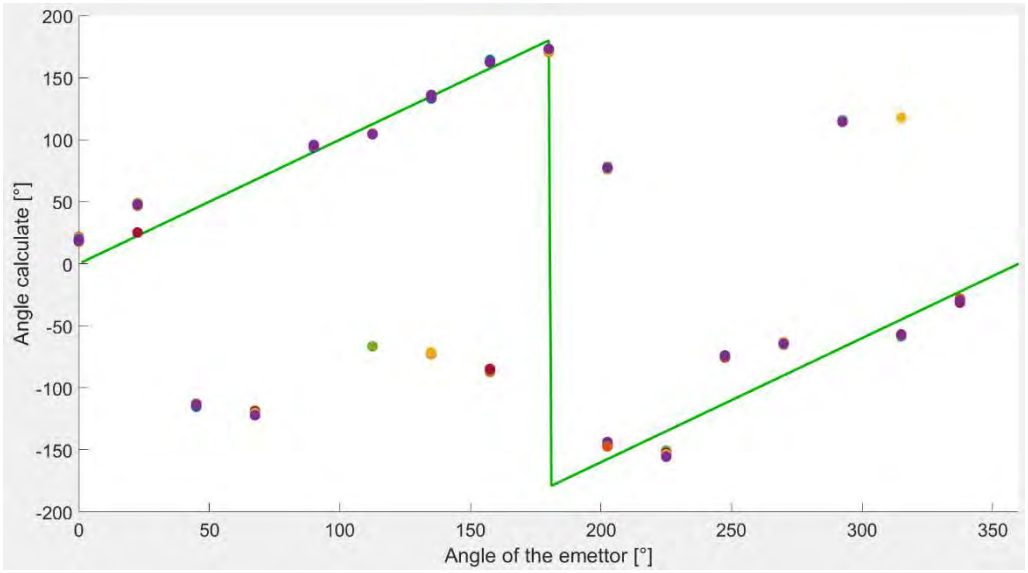


Figure 157 : Comparaison angles θ obtenus Vs angles θ attendus

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Il y a de l'erreur sur les angles θ obtenus. La Figure 158 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

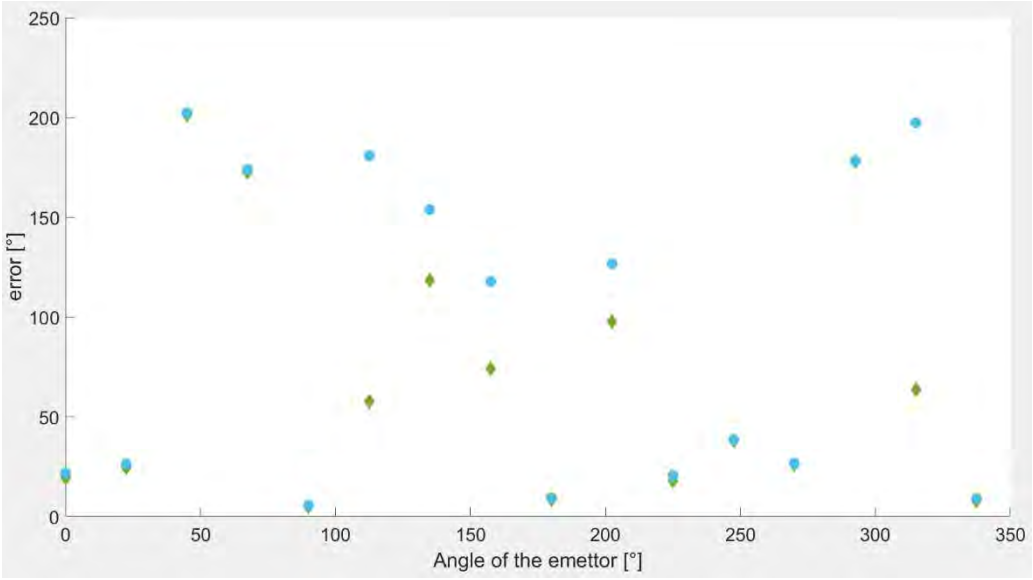


Figure 158 : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 158, les losanges verts présentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur moyenne varie entre 5° et 200°. Une partie de l'erreur peut être expliquée. En effet, en regardant le graphe à la Figure 159, certains angles ne sont pas corrects. Sur la Figure 159, un cercle rouge entour une série de mesure, et sur la Figure 160, un cercle rouge entour la même série de mesure. La différence entre les 2 séries de mesures est le fait que sur la Figure 160, la valeur de 2π ont été soustrait au déphasage $\varphi_3 - \varphi_1$. On observe que la série de mesure entourée par un cercle juste est plus correcte sur la Figure 160 que sur la Figure 159.

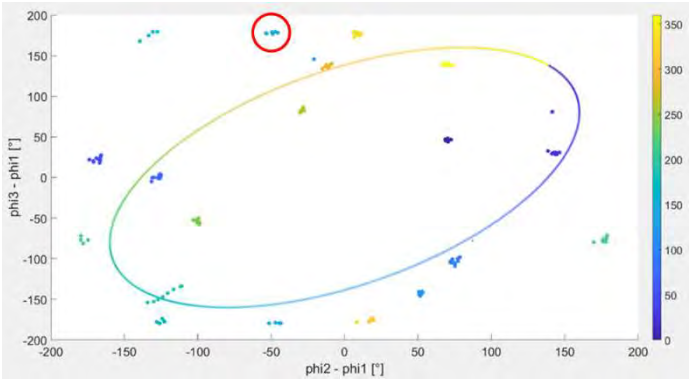


Figure 159 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

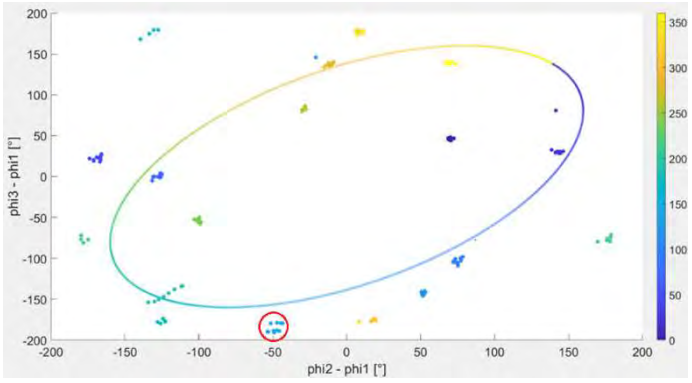


Figure 160 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ avec corrections Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Plusieurs autres séries de mesures présente le même problème. Cependant, une majorité des séries de mesures sont à leur place, mais elles possèdent tout de même beaucoup d'erreurs.

Le chapitre 10.1 présente une discussion sur l'erreur de mesure acceptable dans ce projet. L'erreur de mesure acceptable est de ±22.5. Dans les mesures effectuées, l'erreur est supérieure à l'erreur maximale.

Il y a trop d'erreur pour utiliser du plastique sur les microphones.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

19.3 Silicone

Ce chapitre traite des test effectués avec le silicone déposés sir les microphones.

19.3.1 MONTAGE BOÎTIER

La Figure 161 illustre le silicone déposé sur les microphones.

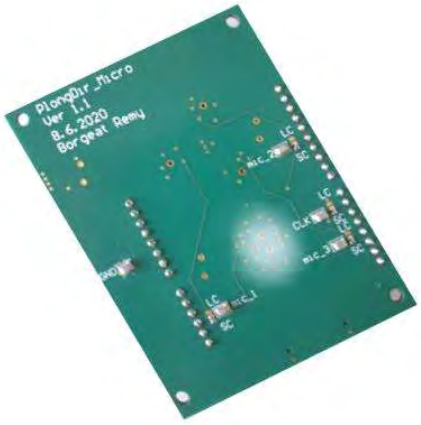


Figure 161 : Boîtier en silicone

19.3.2 RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON

La Figure 162 présente les mesures obtenues des déphasages $\varphi_2 - \varphi_1$ *et* $\varphi_3 - \varphi_1$ et les comparent à la théorie. Les mesures faites aux 8 angles différents sont affichés sur le graphique.

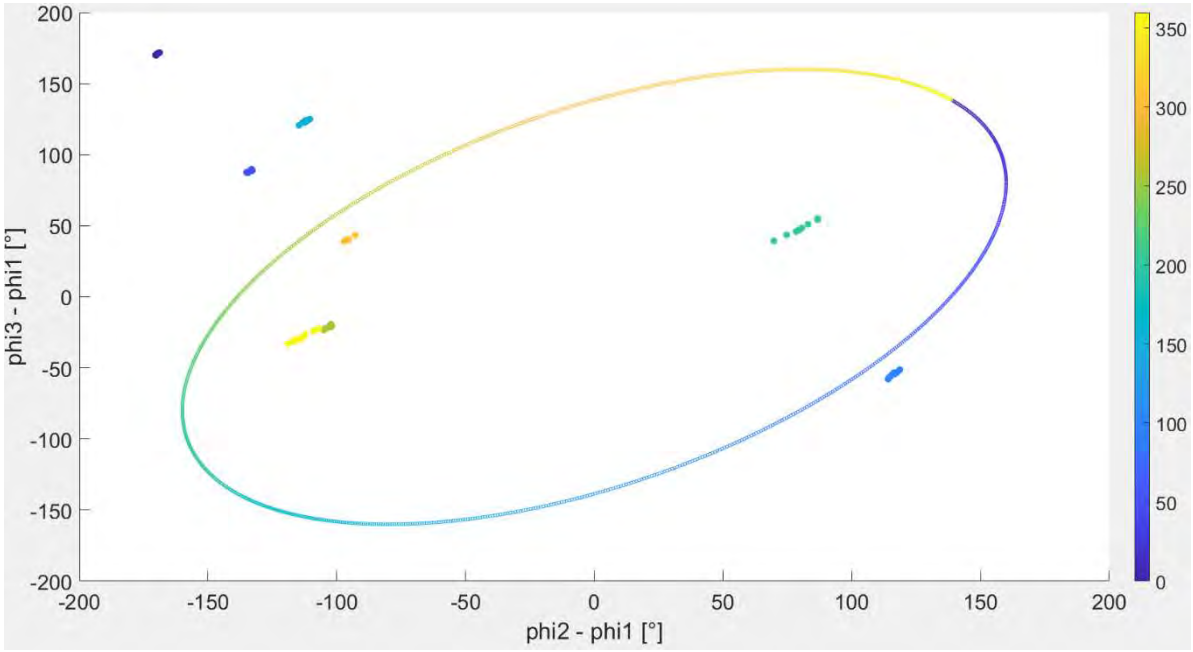


Figure 162 : Mesure de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$

Pour chaque angle, 10 mesures ont été faites. On peut observer que les mesures pour chaque angle sont regroupées. Cependant, les mesures faites à chaque angle sont placées de manière très aléatoire sur le graphe.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

La Figure 163 compare les résultats obtenus avec les résultats théoriques. Pour calculer la direction du son, l'algorithme présenté en chapitre 9 a été utilisé. La courbe verte représente les résultats attendus et les points représentent les résultats obtenus.

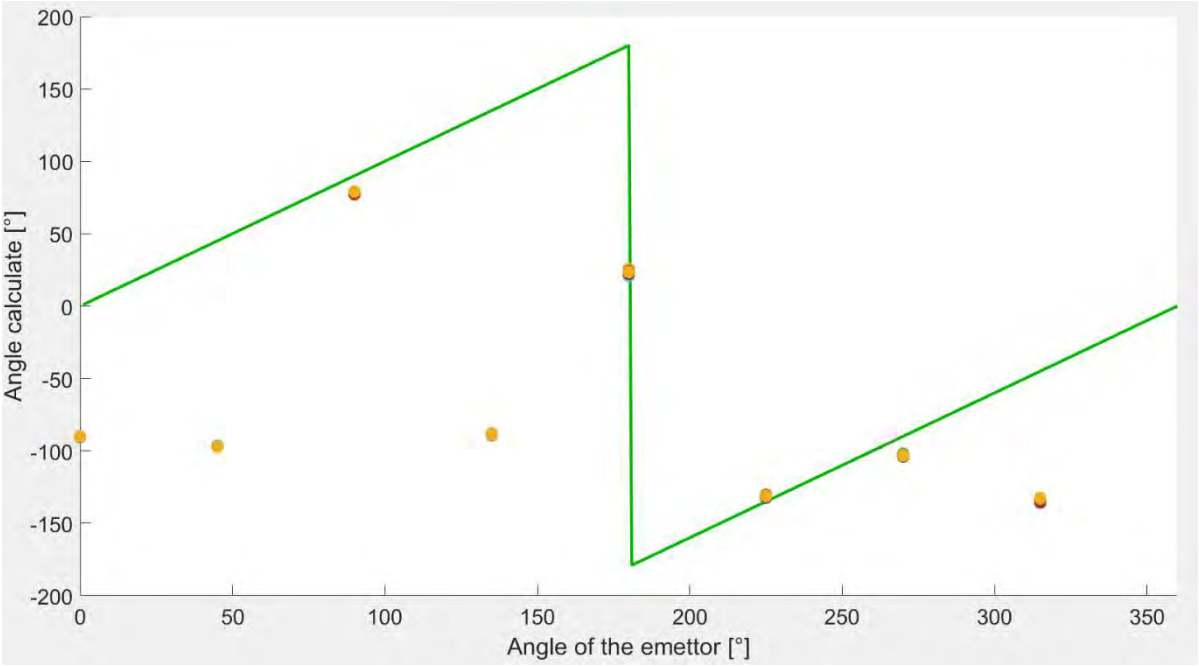


Figure 163 : Comparaison angles θ obtenus Vs angles θ attendus

Il y a de l'erreur sur les angles θ obtenus. La Figure 164 représente les erreurs de mesures effectuées par rapport aux angles θ attendus.

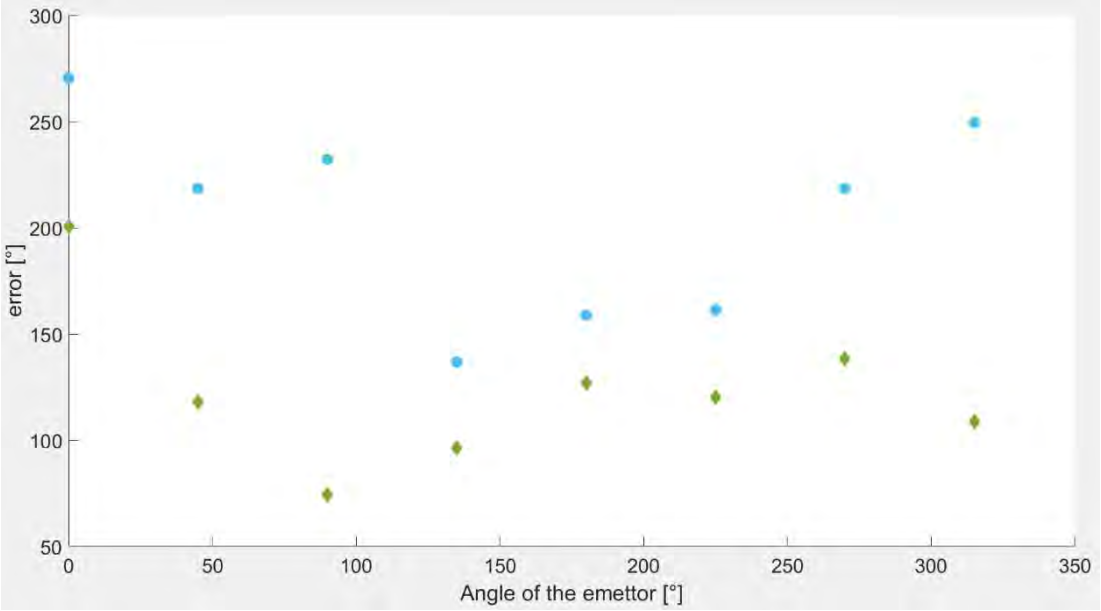


Figure 164 : Erreur de mesures de l'angle θ obtenus par rapport à l'angle θ attendus

Sur la Figure 164, les losanges verts représentent l'erreur moyenne des 10 mesures faites à chaque angle. Les disques bleus représentent l'erreur maximale des mesures faites à chaque angle.

L'erreur obtenue est beaucoup trop grande. Le système est inutilisable avec de telles erreurs de mesures.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

19.4 Synthèse des résultats

Le Tableau 50 présente la synthèse des résultats obtenus sur l'amplitude des signaux captés par les microphones avec différents matériaux.

TEST : AMPLITUDE DES SIGNAUX	RÉSULTATS
SCOTCH SUR LES MICROPHONES	✗
PLEXIGLAS SUR LES MICROPHONES	✗
AUTOCOLLANT SUR LES MICROPHONES	✓
FEUILLE EN PLASTIQUE SUR LES MICROPHONES	✓
SILICONE SUR LES MICROPHONES	✓

Tableau 50 : Synthèse des résultats du test d'amplitudes avec les différents matériaux

L'amplitude des signaux captés par les microphones lorsque du scotch ou du Plexiglas est utilisé, est quasiment nulle et par conséquent les signaux ne sont pas exploitables.

Le Tableau 51 présente la synthèse des résultats obtenus sur le déphasage des signaux captés par les microphones avec différents matériaux.

TEST : DÉPHASAGE DES SIGNAUX	RÉSULTATS
AUTOCOLLANT SUR LES MICROPHONES	✗
FEUILLE EN PLASTIQUE SUR LES MICROPHONES	✓
SILICONE SUR LES MICROPHONES	✓

Tableau 51 : Synthèse des résultats du test de déphasage avec les différents matériaux

Le déphasage entre les signaux captés par les microphones lorsqu'un autocollant est collé sur les microphones n'est pas équivalente à la valeur attendue. Il reste 2 matériaux qui ont passés le 2 premiers tests : Le silicone et la feuille en plastique.

Plusieurs montages différents avec la feuille de plastiques ont été testé. Ces montages sont présentés en chapitre 18.4. Le montage présentant les meilleurs résultats a été utilisés pour déterminer s'il est possible de récupérer l'angle d'arrivée du son.

Le Tableau 52 présente les résultats obtenus des tests fait sur la récupération de l'angle d'arrivée du son.

TEST : RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON	RÉSULTATS
RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON : PLASTIQUE FIN AVEC LE MONTAGE PRÉSENTÉ EN CHAPITRE 18.4	✗
RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON : PLASTIQUE FIN DIRECTEMENT SUR LES MICROPHONES	✗
RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON : SILICONE SUR LES MICROPHONES	✗

Tableau 52 : Synthèse des résultats du test de récupération de l'angle d'arrivée du son avec les différents matériaux

Aucun des matériaux testés n'as permis de retrouver l'angle d'arrivée du son. Pourtant les amplitudes des signaux étaient suffisantes pour récupérer l'angle d'arrivée du son. En effet, le test présenté en chapitre 15 à monter qu'il est possible de récupérer l'angle d'arrivée du son avec une amplitude de 0.03 et dans notre cas, les amplitudes des signaux sont supérieures à 0.03.

La conclusion de ces tests est qu'il n'est pas si facile de construire un boîtier autour des microphones. Il ne suffit pas de juste poser du silicone ou une feuille en plastique sur les microphones, mais il faut une réflexion beaucoup plus importante pour déterminer le matériau idéal et la méthode de récupération de l'angle. Cette réflexion est surtout basée sur des signaux audios et il faudrait une personne spécialisée dans ce domaine pour apporter les solutions nécessaires au problèmes rencontrés.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

20 CALCUL SUR UCONTRÔLEUR

Ce test se concentre sur les calculs effectués sur le uContrôleur. Les calculs faits sur Matlab ont été portés sur le uContrôleur. Ce chapitre présente les tests effectués sur les différentes parties des calculs effectuées sur le uContrôleur. Le Tableau 53 présente la liste de test effectués pour vérifier le fonctionnement des calculs sur uContrôleur.

TEST	CHAPITRE
CALCULS THÉORIQUES	Chapitre 20.1
ALGORITHME "MOVMEAN"	Chapitre 20.2
ALGORITHME DE DÉMODULATION I-Q	Chapitre 20.3
FILTRAGE DIGITAL	Chapitre 20.4
MISE EN COMMUN	Chapitre 20.5

Tableau 53 : Liste des tests effectués sur uContrôleur

Une synthèse des résultats est présentée dans le chapitre 20.6.

20.1 Calcul Théorique

Le calcul théorique sert à calculer la courbe $\theta = f(\alpha)$. La Figure 165 présente les résultats obtenus sur Matlab comparés aux résultats obtenus sur le contrôleur. La courbe bleue représente les calculs fait sur Matlab et la courbe orange représente les calculs faits sur les uContrôleur.

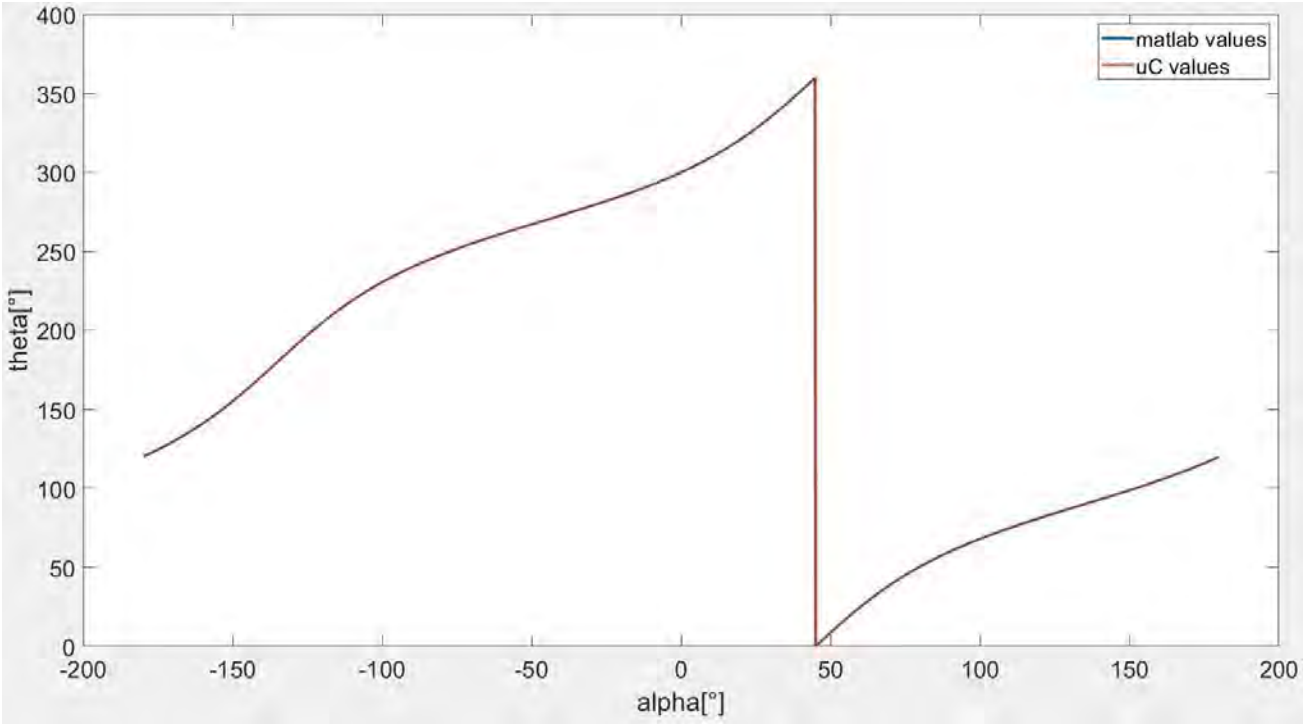


Figure 165 : Courbe $\theta = f(\alpha)$

Les résultats obtenus sur Matlab et sur le uContrôleur sont équivalent. En effet, les 2 courbes sont superposées. Les calculs théoriques fait sur uContrôleur sont corrects.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

20.2 MOVMEAN

La fonction "movmean" a dû être écrite pour le langage de programmation C++, car celle-ci n'existait pas. La fonction "movmean" est expliquée au chapitre 0

20.2.1 MOVMEAN(K,4)

Le premier test est de faire movmean(k,4) avec k étant des valeurs entières de 1 à 10. Les résultats attendus sont illustrés en Figure 166, ces résultats ont été calculés avec Matlab. Les résultats obtenus sont illustrés en Figure 167, ces résultats ont été calculés en C++.

```
>> k = 1:1:10

k =

     1     2     3     4     5     6     7     8     9    10

>> movmean(k,4)

ans =

    1.5000    2.0000    2.5000    3.5000    4.5000    5.5000    6.5000    7.5000    8.5000    9.0000
```

Figure 166 : movmean(k,4) sur Matlab

```
k :
1       2       3       4       5       6       7       8       9       10

results :
1.5     2       2.5     3.5     4.5     5.5     6.5     7.5     8.5     9
```

Figure 167 : movmean(k,4) en C++

Les résultats attendus correspondent aux résultats attendus.

20.2.2 MOVMEAN(K,5)

Le deuxième test est de faire movmean(k,5) avec k étant des valeurs entières de 1 à 10. Les résultats attendus sont illustrés en Figure 168, ces résultats ont été calculés avec Matlab. Les résultats obtenus sont illustrés en Figure 169, ces résultats ont été calculés en C++.

```
k =

     1     2     3     4     5     6     7     8     9    10

>> movmean(k,5)

ans =

    2.0000    2.5000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    8.5000    9.0000
```

Figure 168 : movmean(k,5) sur Matlab

```
k :
1       2       3       4       5       6       7       8       9       10

results :
1.5     2       3       4       5       6       7       8       6.8     9
```

Figure 169 : movmean(k,5) en C++

Les résultats attendus correspondent aux résultats attendus.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

20.2.3 MOVMEAN(K,6)

Le troisème test est de faire movmean(k,6), avec k étant des valeurs entières de 1 à 10. Les résultats attendus sont illustrés en Figure 170, ces résultats ont été calculés avec Matlab. Les résultats obtenus sont illustrés en Figure 171, ces résultats ont été calculés en C++.

```
k =

     1     2     3     4     5     6     7     8     9    10

>> movmean(k,6)

ans =

    2.0000    2.5000    3.0000    3.5000    4.5000    5.5000    6.5000    7.5000    8.0000    8.5000
```

Figure 170 : movmean(k,6) sur Matlab

```
k :
1       2       3       4       5       6       7       8       9       10

results :
2       2.5     3       3.5     4.5     5.5     6.5     7.5     8       8.5
```

Figure 171 : movmean(k,6) en C++

Les résultats attendus correspondent aux résultats attendus.

20.2.4 CONCLUSION

Avec les tests précédents, on peut conclure que l'algorithme movmean() implémenté sur le uContrôleur est fonctionnel.

20.3 Démodulation IQ

Afin de tester la démodulation IQ qui permet de trouver les déphasages entre des sinus, 3 sinus ont été générés. Chaque sinus à la même période, mais une phase différente. La phase est calculée de la même façon qu'au chapitre 8.2. Le sinus généré passe ensuite dans le démodulateur IQ et les déphasages $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$ sont obtenus.

Ces étapes ont été répétées pour des angles θ variant entre 0 et 360°. La Figure 172 présente les différences de phases $\varphi_2 - \varphi_1$ et $\varphi_3 - \varphi_1$obtenues et les différences de phases théoriques.

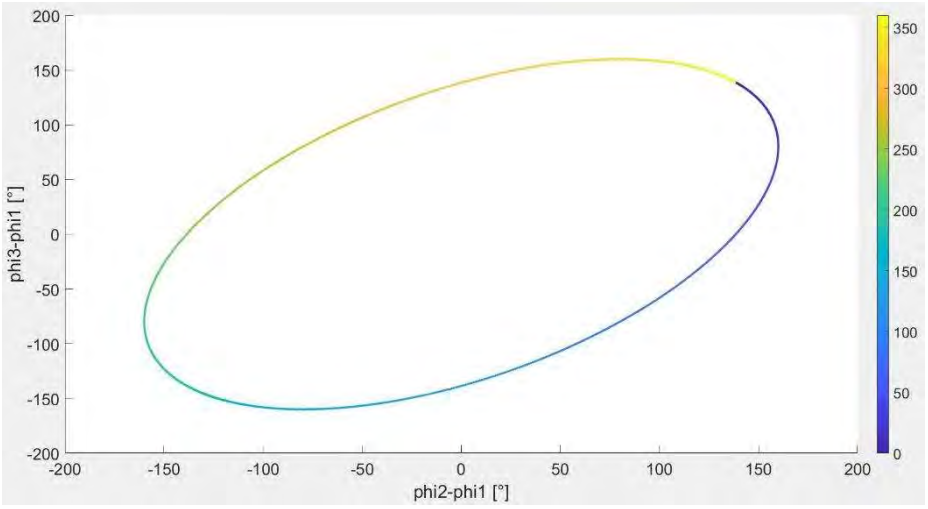


Figure 172 : Calcul de $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ sur uContrôleur Vs Calculs $(\varphi_2 - \varphi_1, \varphi_3 - \varphi_1)$ sur Matlab.

Les calculs fait sur uContrôleur sont égal aux calculs fait sur Matlab.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

20.4 Filtrage

Le filtre numérique implémenté dans le uContrôleur a été testé. La Figure 173 représente le signal appliqué à l'entrée du filtre passe-bas numérique. Ce signal est la sortie de l'un des microphones.

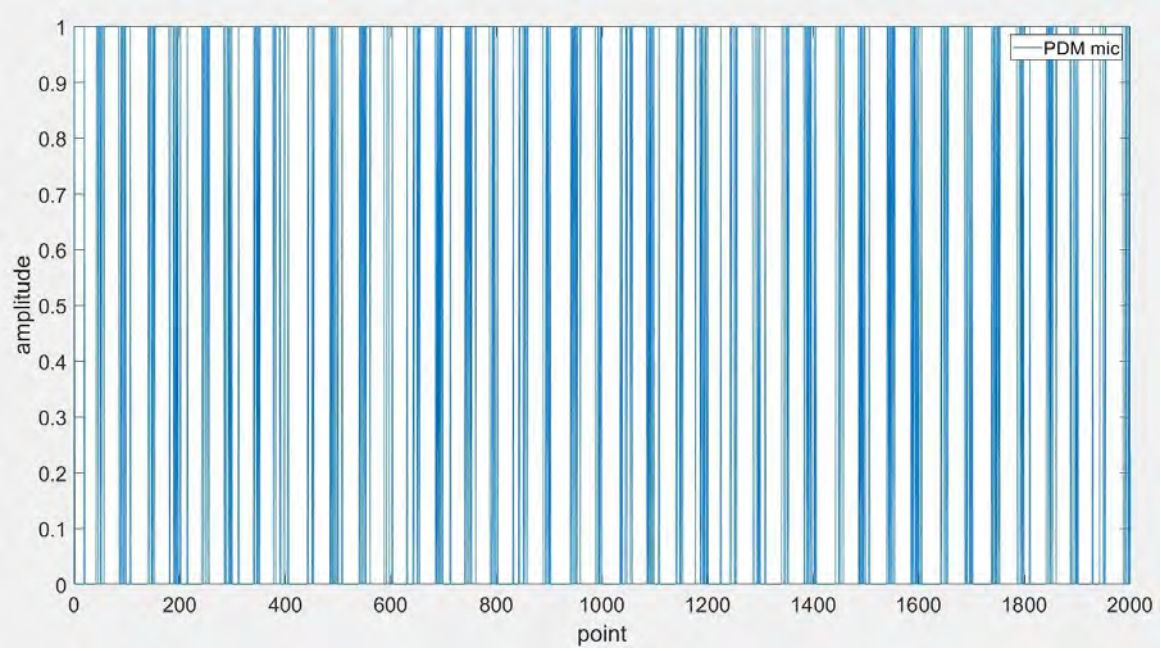


Figure 173 : Signal d'entrée du filtre passe-bas numérique

Le signal d'entrée est filtré une fois sur Matlab et une autre fois avec le filtre implémenté sur le uContrôleur (voir le chapitre 9.2). Une comparaison entre les résultats obtenus est illustrée sur la Figure 174.

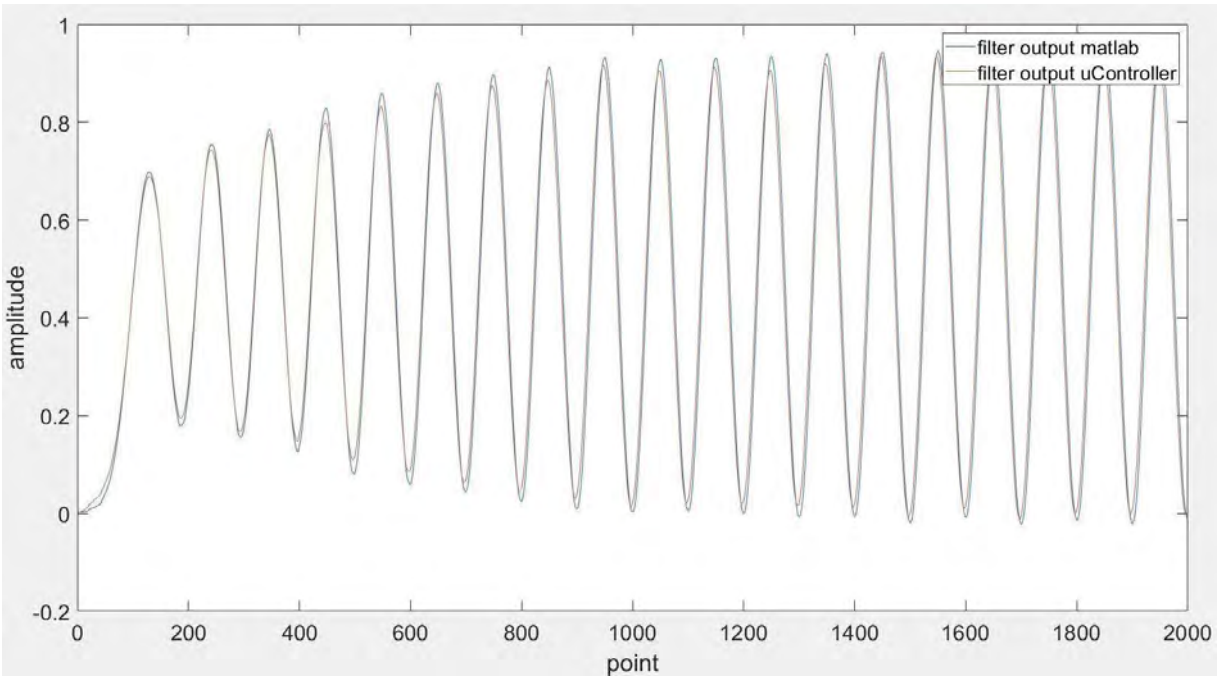


Figure 174 : Comparaison entre les filtrages fait sur Matlab et sur le uContrôleur

Les résultats obtenus avec l'algorithme implémenté sur le uContrôleur est équivalent aux résultats obtenus sur Matlab.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

20.5 Mise en commun

Cette partie met en commun toute le code écrit. Les mesures prises par les microphones ont été enregistrées. Le calcul de l'angle d'arrivée du son a été effectué une fois sur Matlab et une fois sur uContrôleur.

La Figure 175 présente les résultats des calculs fait sur Matlab. La Figure 176 présente les résultats des calculs fait sur uContrôleur. À noter que les résultats sont en radians.

```
phi12Tab =

   -2.3467   -2.3534   -2.3601   -2.3596   -2.3492   -2.3540   -2.3456   -2.3278   -2.3405

phi13Tab =

   -1.9243   -1.9254   -1.9196   -1.9171   -1.9294   -1.9420   -1.9264   -1.9006   -1.9150

result =

   -0.1759   -0.1759   -0.1822   -0.1822   -0.1696   -0.1696   -0.1759   -0.1759   -0.1759
```

Figure 175 : Calcul fait sur Matlab

```
============================
       Measure 1
============================
phi12 : -2.34001
phi13 : -1.92259
result : -0.169646
============================
       Measure 2
============================
phi12 : -2.34061
phi13 : -1.91669
result : -0.175929
============================
       Measure 3
============================
phi12 : -2.35022
phi13 : -1.91203
result : -0.182212
============================
       Measure 4
============================
phi12 : -2.34649
phi13 : -1.91102
result : -0.175929
============================
       Measure 5
============================
phi12 : -2.33679
phi13 : -1.92442
result : -0.169646
============================
       Measure 6
============================
phi12 : -2.29692
phi13 : -1.88887
result : -0.169646
============================
       Measure 7
============================
phi12 : -2.30009
phi13 : -1.88132
result : -0.175929
============================
       Measure 8
============================
phi12 : -2.28341
phi13 : -1.85725
result : -0.182212
============================
       Measure 9
============================
phi12 : -2.30418
phi13 : -1.87575
result : -0.175929
```

Figure 176 : Calcul fait sur uContrôleur

Les résultats obtenus sur uContrôleur sont équivalent aux résultats obtenus sur Matlab. L'erreur maximal sur le résultat obtenus est d'environ 0.07 radians (4°). Cette erreur est entièrement acceptable.

20.6 Conclusion des calculs sur uContrôleur

Le Tableau 54 présente la synthèse des résultats. Les résultats sont les calculs fait sur uContrôleur.

TEST	RÉSULTATS
CALCUL THÉORIQUES	✓
MOVMEAN	✓
DÉMODULATION IQ	✓
FILTRE PASSE-BAS	✓
MISE EN COMMUN	✓

Tableau 54 : Synthèse des résultats sur uContrôleur

Tout le calcul fait sur uContrôleur sont fonctionnels.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

21 SYNTHÈSES DES RÉSULTATS

Plusieurs tests ont été effectués. Le Tableau 55 présente la liste des tests effectués avec un lien vers le chapitre où les tests sont présentés et le chapitre où les synthèses des résultats sont faites

TEST	CHAPITRE	CHAPITRE DE LA SYNTHÈSE DES RÉSULTATS
CONSOMMATION DES ÉMETTEURS ULTRASONIQUES	12	12.3
RÉCUPÉRATION DE L'ANGLE D'ARRIVÉE DU SON DANS L'AIR	14	14.4
ÉMETTEUR ULTRASONIQUE COUVERT PAR DU SILICONE	15	15.4
DISTANCE MAXIMALE D'ÉMISSION	16	16.4
ANGLE D'INCIDENCE DU SON	17	17.5
DIRECTIVITÉ DES ÉMETTEURS ULTRASONIQUE	18	18.4
BOÎTER	19	19.4
CALCULS SUR UCONTRÔLEUR	20	20.6

Tableau 55 : Liste des tests effectués.

21.1 Émetteur ultrasonique

Premièrement, le test sur la consommation des émetteurs ultrasoniques a testé la consommation des émetteurs ultrasoniques. Le résultat pour une tension de 20[Vpp] est présentée dans le Tableau 56.

COMPOSANTS	COURANT CONSOMMÉ À LA FRÉQUENCE DE RÉSONNANCE
UT-1240K-TT-R	16.9 [mA]
UT-1640K-TT-2-R	14.1 [mA]

Tableau 56 : Consommation des émetteurs ultrasoniques

Cependant, le test sur la distance maximale d'émission a montré que la tension d'alimentation des émetteurs ultrasonique impacte la pression acoustique du signal émis par les émetteurs ultrasoniques. Si l'on désire que la pression acoustique du signal émis par les émetteurs ultrasoniques soit maximale, il faut alimenter les émetteurs ultrasoniques avec les tensions présenté dans le Tableau 57.

ÉMETTEUR ULTRASONIQUE	TENSION D'ALIMENTATION MAXIMALE
UT-1240K-TT-R[1]	30[Vrms]
UT-1640K-TT-2-R[2]	20[Vrms]

Tableau 57 : Tension d'alimentation des émetteurs ultrasonique

Le courant consommé par les émetteurs ultrasoniques alimenté avec leur tension efficaces maximales est plus important. Ces courants sont présentés en Tableau 58.

COMPOSANTS	COURANT CONSOMMÉ À LA FRÉQUENCE DE RÉSONNANCE
UT-1240K-TT-R	81.8[mA]
UT-1640K-TT-2-R	44.4[mA]

Tableau 58 : Courant consommé par les émetteurs ultrasonique à la fréquence de résonnance

Les tensions des émetteurs ultrasoniques sont élevées. Il faut prévoir un prévoir un circuit électronique permettant créer un signal sinusoïdal de 20[Vrms] ou 30[Vrms] dépendant de l'émetteur ultrasonique choisi.

Ensuite, un test a été effectué sur la directivité de l'émetteur ultrasonique. Le résultat de ce test est présenté dans le Tableau 59.

COMPOSANTS	DIRECTIVITÉ
UT-1240K-TT-R	70°
UT-1640K-TT-2-R	60°

Tableau 59 : Directivité des émetteurs ultrasoniques

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Avec cette directivité, on peut déterminer le nombre minimal d'émetteur ultrasonique nécessaire afin de localiser le plongeur à 360°. Le Tableau 60 présente le nombre d'émetteur ultrasonique minimal nécessaire dans ce projet.

COMPOSANTS	NOMBRE DE COMPOSANTS MINIMAL
UT-1240K-TT-R	3
UT-1640K-TT-2-R	3

Tableau 60 : Nombre de composants

Avec ce nombre de composants, le courant consommé s'en trouvent multiplié. En multipliant le nombre de composants à la consommation d'un émetteur ultrasonique on trouve les résultats présentés dans le Tableau 61.

COMPOSANTS	COURANT CONSOMMÉ PAR TOUS LES ÉMETTEURS ULTRASONIQUES
UT-1240K-TT-R	245.4[mA]
UT-1640K-TT-2-R	133.2[mA]

Tableau 61 : Consommation de tous les émetteurs ultrasoniques

Les courants consommés sont importants, mais ce n'est pas le seul problème auquel on est confronté. En effet, le système va dans l'eau est doit par conséquent être alimenté par une batterie. La tension de la batterie sera inférieure à la tension d'alimentation des émetteurs ultrasoniques. Il faut donc prévoir une électronique d'élévation de tension. Le courant qui sera tiré sur la batterie sera plus important que le courant consommé par les émetteurs ultrasoniques. Par conséquent, il faut prévoir une batterie avec beaucoup de stockage énergétique pour assurer l'alimentation électrique du système pendant une plongée.

L'électronique d'alimentation des émetteurs ultrasonique n'était pas le sujet de ce travail de bachelor et n'a donc pas été abordée. Cependant, il faut être conscient de problématiques soulevées.

21.2 Microphones

Les microphones ont premièrement été testés dans l'air, dans un environnement contrôlé. Concrètement, il n'avait pas d'obstacle, de murs ou d'objets à moins de 2 mètres autour de l'émetteurs ultrasoniques et des microphones. Ce premier test a eu de très bons résultats. En effet, il est possible de récupérer l'angle d'arrivée du son dans ces conditions-là. Ensuite, les microphones ont été testés dans l'air, dans un environnement non-contrôlé. Concrètement, il y avait des obstacles à environ de 1 mètres autour de l'émetteurs ultrasoniques et des microphones. Ce deuxième test a eu de bons résultats. En effet, il est possible de récupérer l'angle d'arrivée du son dans ces conditions-là. Cependant, ce test a aussi démontré que les microphones sont sensibles aux rebonds du signal ultrasonique sur des obstacles.

Ensuite, un test a été fait en déposant du silicone sur l'émetteur ultrasonique. Le signal reçu par les microphones était de très faible amplitude, mais il est tout de même possible de récupérer l'angle d'arrivée du son.

21.2.1 DISTANCES MAXIMALES

Un test portant sur la distance maximale entre les 2 plongeurs a été effectués. Autrement dit, c'est la distance maximale entre l'émetteur ultrasonique et les microphones qui a été testée. Ce test a été réalisé au milieu d'un couloir d'environ 2.5 mètres de large et très long. Ce test a été effectué avec les 2 émetteurs ultrasoniques. Ces émetteurs ultrasoniques étaient alimentés en 17.7[Vrms].

Le test précédant a montré qu'il est possible de récupérer l'angle d'arrivée du son même si l'amplitude des signaux captés par les microphones sont très faibles. Donc, pour les tests portants sur la distance maximale, les amplitudes des signaux reçus par les microphones ont été mesurées pour plusieurs distances entre l'émetteur ultrasonique et les microphones. On peut alors comparer les amplitudes des signaux mesurées aux différentes distances à l'amplitude du test précédent pour avoir une idée sur la distance maximale entre l'émetteur ultrasonique et les microphones. Les distances maximales attendues entre les émetteurs ultrasoniques et les microphones sont présentés dans le Tableau 62.

COMPOSANTS	DISTANCE MAXIMALES ATTENDUES
UT-1240K-TT-R	7.5 mètres
UT-1640K-TT-2-R	10 mètres

Tableau 62 : distances maximales attendues entre les émetteurs ultrasoniques et les microphones

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Le Tableau 63 présente les distances maximales obtenues entre les émetteurs ultrasonique et les microphones.

COMPOSANTS	DISTANCE MAXIMALES ATTENDUES
UT-1240K-TT-R	5 mètres
UT-1640K-TT-2-R	7.5 mètres

Tableau 63 : distances maximales obtenues entre les émetteurs ultrasoniques et les microphones

Les distances maximales obtenues entre les émetteurs ultrasoniques et les microphones sont inférieures aux distances maximales attendues entre les émetteurs ultrasoniques et les microphones. Cette différence peut s'expliquer car le signal émis par les émetteurs ultrasoniques rebondit contre les murs et vient perturber le signal reçu par les microphones. Ce test montre aussi que les microphones sont sensibles aux réflexions du signal émis par les émetteurs ultrasoniques.

Cela est un problème, car lors d'une plongée sous-marine, il peut y avoir des rochers, des collines et des obstacles sous l'eau sur lesquelles le signal émis par les émetteurs ultrasoniques peut rebondir et venir perturber le signal reçu. D'une fois le signal reçu perturbé, il est difficile de déterminer la direction de son partenaire de plongée.

21.2.2 Boîtier autour des microphones

Des tests ont été effectués pour construire un boîtier autour de l'émetteur ultrasonique. Du silicone a été déposé sur l'émetteur ultrasonique. L'amplitude des signaux reçus est alors très faible et par conséquent, la distance maximale d'émission est très faible.

Des tests ont été effectués pour construire un boîtier autour des microphones. Plusieurs matériaux ont été testés. Certains matériaux sont meilleurs que d'autres. Des tests ont été effectués avec du silicone sur les microphones et avec une feuille en plastique sur les microphones pour récupérer l'angle d'arrivée du son. Du silicone et du plastique ont été utilisés, car ce sont les matériaux qui donnaient les meilleurs résultats. Cependant, aucun de ces tests n'a permis de retrouver correctement l'angle d'arrivée du son. Pourtant les amplitudes des signaux était suffisant pour récupérer l'angle d'arrivée du son. Il n'est donc pas si facile de construire un boîtier autour des microphones. Il ne suffit pas de poser du silicone ou une feuille en plastique sur les microphones, mais il faut une réflexion beaucoup plus importante pour déterminer le matériau idéal et la méthode de récupération de l'angle. Cette réflexion est surtout basée sur des signaux audios et il faudrait une personne spécialisée dans ce domaine pour apporter les solutions nécessaires aux problèmes rencontrés. De plus, la distance maximale entre les 2 microphones est très faible lorsque des boîtiers sont utilisés. Il y a 2 options possibles pour augmenter la distance maximale entre les 2 plongeurs. La première option consiste à utiliser un matériau étanche, résistant à la pression et atténuant très peu le signal sonore. La deuxième option consiste à remplacer les microphones digitaux utilisés par des microphones analogiques. Un traitement du signal analogique devra être effectué. De cette manière, il est probable d'obtenir un signal avec une meilleure amplitude.

21.2.3 Les 2 plongeurs ne sont pas à la même profondeurs

Le cas où l'un des plongeurs se situe plus profondément que son partenaire de plongée a été testé. Ce test a eu de très bons résultats. En effet, il est possible de récupérer l'angle d'arrivée du son dans ces conditions-là. L'angle d'incidence du son jusqu'au quel il est possible de récupérer la direction de son partenaire de plongée est de 60°.

21.3 Calcul sur uContrôleur

Tous les calculs présentés ont été faits sur Matlab. Cependant pour aller dans l'eau, il est nécessaire d'avoir un système embarqué comportant un uContrôleur. Les calculs faits sur Matlab ont été portés sur le uContrôleur. Les calculs faits sur uContrôleur sont entièrement fonctionnels et correspondent aux résultats obtenus sur Matlab.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

V. CONCLUSION

22 CONCLUSION

Les buts de ce projet n'ont pas été atteints. Il n'est donc actuellement pas possible de déterminer la direction de son partenaire de plongée sous l'eau.

Dans l'air, il a été démontré qu'il est possible de retrouver la direction de son partenaire de plongée, cependant le système est sensible aux rebonds du signal contre les murs, les objets et les obstacles.

Un problème majeur rencontré dans ce projet est la construction d'un boîtier autour de l'émetteur ultrasonique et autour des microphones. Plusieurs matériaux ont été testés, mais aucun n'a donné de bons résultats. Les amplitudes des signaux reçus sont faibles, mais suffisant pour récupérer la direction de son partenaire de plongée. Cependant, la direction de son partenaire de plongée n'a pas pu être retrouvé. Il n'est, par conséquent, pas si facile de construire un boîter autour des microphones et de l'émetteur ultrasonique. Il faudrait une personne spécialisée dans le domaine audio pour comprendre les phénomènes qui se passent lorsque le son entre dans un boîtier et de connaître le matériau idéal pour construire ce boîter. De plus, les signaux reçus ont une amplitude faible, la distance maximale entre les 2 plongeurs sera donc impactée. Pour résoudre ce problème, les microphones digitaux employés devraient être remplacés par des microphones analogiques et un traitement analogique du signal devra être effectué. De cette manière, il est probable d'obtenir des signaux avec de meilleures amplitudes.

Un autre problème est l'alimentation des émetteurs ultrasoniques. Une batterie doit être utilisée pour alimenter les émetteurs ultrasoniques. Il faut prévoir une électronique permettant de convertir la tension continue de la batterie en tension alternative afin d'alimenter les émetteurs ultrasoniques. De plus, le courant consommé n'est pas négligeable.

En conclusion, ce travail a montré qu'il y a plusieurs problèmes différents à résoudre avant d'obtenir un produit fini et fonctionnel. Beaucoup de développement devra encore être réalisé avant de possiblement obtenir un système fonctionnel. Il faut noter que ce développement supplémentaire n'est pas garanti de succès.

23 REMERCIEMENTS

- Je tiens à remercier Mme. Alexandra Andersson de m'avoir suivi sur ce projet dans ce travail de bachelor.
- Je tiens à remercier M. Steve Gallay pour le montage des microphones sur les PCBs.
- Je tiens à remercier M. Pascal Sartoretti pour son aide sur le uContrôleur.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

VI. ANNEXES

24 ANNEXES

1. Planification
2. Script Matlab
3. Programme V1.0
    a. Code source
    b. Documentation UML
    c. Documentation du code
4. Programme V2.0
    a. Code source
    b. Documentation UML
    c. Documentation du code
5. PCB V1.1
    a. Routage face avant
    b. Routage face arrière
    c. Plan de perçage
    d. Implémentation face avant
    e. Implémentation face arrière
6. PCB V2.0
    a. Routage face avant
    b. Routage face arrière
    c. Plan de perçage
    d. Implémentation face avant
    e. Implémentation face arrière

Date : _____

Signature : _____

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

VII. RÉFÉRENCE

[1] Datasheet de l'émetteur ultrasonique UT-1240K-TT-R : https://www.mouser.ch/datasheet/2/334/UT-1240K-TT-R-957964.pdf
Date : 6/1/2016
Consulté la dernière fois le 12.08.2020

[2] Datasheet de l'émetteur ultrasonique UT-1640K-TT-2-R : https://www.mouser.ch/datasheet/2/334/UT-1640K-TT-2-R-953211.pdf
Date : 1/6/2020
Consulté la dernière fois le 12.08.2020

[3] Datasheet du microphone SPH0641LU4H-1 : https://www.mouser.ch/datasheet/2/218/-746191.pdf
Révision : A
Date : 5/16/2014.
Consulté la dernière fois le 12.08.2020

[4] Source de l'image : https://en.wikipedia.org/wiki/Pulse-density_modulation#/media/File:Pulse_density_modulation.svg
Consulté la dernière fois le 12.08.2020

[5] STM32F746G-Discovery : https://www.st.com/en/evaluation-tools/32f746gdiscovery.html
Consulté la dernière fois le 12.08.2020

[6] Source de l'image : https://www.digikey.ch/product-detail/fr/stmicroelectronics/STM32F746G-DISCO/497-15680-5-ND/5267791
Consulté la dernière fois le 12.08.2020

[7] Source de la propriété mathématiques : 3ème propriété :
http://uel.unisciel.fr/physique/outils_nancy/outils_nancy_ch06/co/apprendre_02_05.html
Consulté la dernière fois le 12.08.2020

[8] Source de la propriété mathématiques soustraction 1ère et 2ème propriété :
http://uel.unisciel.fr/physique/outils_nancy/outils_nancy_ch06/co/apprendre_02_05.html
Consulté la dernière fois le 12.08.2020

[9] Source de la propriété mathématiques addition 1ère et 2ème propriété :
http://uel.unisciel.fr/physique/outils_nancy/outils_nancy_ch06/co/apprendre_02_05.html
Consulté la dernière fois le 12.08.2020

[10] Source de l'image : https://fr.wikipedia.org/wiki/Atan2#/media/Fichier:Arctangent2.svg
Consulté la dernière fois le 12.08.2020

[11] Source de l'image : https://fr.wikipedia.org/wiki/Arc_tangente#/media/Fichier:Arctangent.svg
Consulté la dernière fois le 12.08.2020

[12] Source de la méthode "movmean" : https://ch.mathworks.com/help/matlab/ref/movmean.html.
Consulté la dernière fois le 12.08.2020

[13] Source de la méthode "scatter" : https://ch.mathworks.com/help/matlab/ref/scatter.html
Consulté la dernière fois le 12.08.2020

[14] Source de l'image :
https://www.st.com/resource/en/user_manual/dm00190424-discovery-kit-for-stm32f7-series-with-stm32f746ng-mcu-stmicroelectronics.pdf
Figure 4,
Consulté la dernière fois le 12.08.2020,
Révision : UM1907 Rev 5.
Date : Juin 2020

[15] Application note AN5027 :
https://www.st.com/resource/en/application_note/dm00380469-interfacing-pdm-digital-microphones-using-stm32-mcus-and-mpus-stmicroelectronics.pdf
Consulté la dernière fois le 12.08.2020,
Révision : AN5027 Rev 2,
Date : Juillet 2019.

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

[16] Source de l'image :
https://www.st.com/resource/en/reference_manual/dm00124865-stm32f75xxx-and-stm32f74xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
Figure 357,
Consulté la dernière fois le 12.08.2020,
Révision : RM0385 Rev 8,
Date : Juin 2018.

[17] Plan mécanique de la carte STM32F746G-Discovery :
https://www.st.com/resource/en/user_manual/dm00190424-discovery-kit-for-stm32f7-series-with-stm32f746ng-mcu-stmicroelectronics.pdf
Figure 5,
Consulté la dernière fois le 12.08.2020,
Révision : UM1907 Rev 5.
Date: Juin 2020

[18] Libraire graphique ugfx : https://ugfx.io/index
Consulté la dernière fois le 12.08.2020,

HES-SO Valais-Wallis • rte du Rawil 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

Annexe 1 :

Planification

Tâche	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Démarrage du projet															
Prise de connaissance															
Dévellopement circuit imprimé + programmation															
Schématique + PCB															
Programmation récupération des données des micros + sauvegarde SD															
Programmation calcul sur STM32															
Montage du PCB															
Test du PCB															
Algorithme Matlab															
Implémentation algorithme Matlab															
Simulation des micros avec Matlab + calcul direction émetteur															
TESTS															
Test algorithme matlab dans l'air															
Test algorithme matlab avec l'émetteur ultrasonique															
Test algorithme matlab sur la distance maximale															
Test algorithme matlab avec boîtier															
Test alogrithme uControlleur dans l'air															
Documentation															
Rédaction rapport															
Présentation															
Présentation															

Annexe 2 :

Scripts Matlab

```
%---------------------------------
% Get Phi12 and Phi13 output data of
% getPhase.m and calculate the
% sound angle arrival
%---------------------------------
clear
close all

index = 0;
numberDataPerAngle = 10;

phi12 = 16:numberDataPerAngle;
phi13 = 16:numberDataPerAngle;

phi12Mean = 16:1;
phi13Mean = 16:1;
hold('all')
index = 0;

angleID = 15
phi12acc = [];
phi13acc = [];
measId =20;


angleSoundStep = 22.5;
angleSoundNMeasure = 16;
%---------------------------------
% Get Phi12 and Phi13 from text file
%---------------------------------
for meas = measId:measId
    for angleSound = 0:angleSoundStep:(angleSoundNMeasure-1)*angleSoundStep
        index = index + 1;

        phi1= fscanf(fopen(sprintf('../Result/meas%d/phi12/angle%0.1f.txt',meas,↙
angleSound),'r'),'%f');
        phi2= fscanf(fopen(sprintf('../Result/meas%d/phi13/angle%0.1f.txt',meas,↙
angleSound),'r'),'%f');

        for i = 1:numberDataPerAngle
            phi12(index,i) = phi1(i);
            phi13(index,i) = phi2(i);
        end


    end
    index = 0;
end

%---------------------------------
% Calcul theorical Phi12 Phi13
%---------------------------------
f = 40e3;
vp = 343;  %[m/s] air
%vp = 1475;  %[m/s] water
```

```
%vp = 2730;  %[m/s] acrylic

lambda = vp/f;
k = 2*pi*f/vp;

r = 0.0022;

R = 3;
x1 = r;
y1 = 0;
x2 = -r/2;
y2 = sqrt(3)*r/2;
x3 = -r/2;
y3 = -sqrt(3)*r/2;

theta = (0:2*pi/1000:2*pi)';
theta_deg = (0:360/1000:360)';

x0 = R*cos(theta);
y0 = R*sin(theta);

d1 = sqrt((x1-x0).^2+(y1-y0).^2);
d2 = sqrt((x2-x0).^2+(y2-y0).^2);
d3 = sqrt((x3-x0).^2+(y3-y0).^2);

phi1 = k*d1;
phi2 = k*d2;
phi3 = k*d3;

phi1 = angle(exp(1j*phi1));
phi2 = angle(exp(1j*phi2));
phi3 = angle(exp(1j*phi3));

phi12Calc = phi2-phi1;
phi13Calc = phi3-phi1;

%calcul alphaCalc value
alphaCalc = atan2(phi13Calc,phi12Calc);

%calcul alphaCalc value
alpha = atan2(-phi13,-phi12);

%----------------------------------
% Sort the alphaCalc array
%----------------------------------
alphaCalcSort = alphaCalc;
thetaSort = theta;

for j = 0 : length(alphaCalcSort)-1
    for i = 1: length(alphaCalcSort)-j-1
        if alphaCalcSort(i)>alphaCalcSort(i+1)
            %switch first array
            temp = alphaCalcSort(i);
            alphaCalcSort(i) = alphaCalcSort(i+1);
            alphaCalcSort(i+1) = temp;
```

```
            %switch second array
            temp = thetaSort(i);
            thetaSort(i) = thetaSort(i+1);
            thetaSort(i+1) = temp;
        end
    end
end


%---------------------------------
% Get the angle sound arrival and
% measurement error
%---------------------------------

error = zeros(size(alpha));
for angleID = 1:angleSoundNMeasure
    for measID = 1:numberDataPerAngle
        for x = 1:length(alphaCalcSort)
            result(angleID,measID) = thetaSort(x);

            if alpha(angleID,measID) < alphaCalcSort(x)
                break;
            end
        end

        error(angleID,measID) = result(angleID,measID) - theta(cast((angleID-1) *↙
angleSoundStep* 1000 / 360,'uint16')+1);
        error(angleID,measID) = error(angleID,measID)  * 180/pi;

        if error(angleID,measID)  < -300
            error(angleID,measID)  = error(angleID,measID)  + 360;
        elseif error(angleID,measID)  > 300
            error(angleID,measID)  = error(angleID,measID)  - 360;
        end
    end

end

%---------------------------------
% Display result
%---------------------------------
figure(1)
scatter(phi13Calc*180/pi,phi12Calc * 180/pi,10,theta_deg);
hold all
for k=1:numberDataPerAngle
    scatter(-angle(exp(1j*(phi13(:,k))))*180/pi,-angle(exp(1j*(phi12(:,k))))*180/pi,↙
50,360*(0:angleSoundNMeasure-1)'/(angleSoundNMeasure-1),'filled');
end

ylabel('phi3 - phi1 [°]')
xlabel('phi2 - phi1 [°]')
set(gca,'FontSize',22)
figure(2)
angleSound = 0:angleSoundStep:(angleSoundNMeasure-1)*angleSoundStep;
```

```
hold all
resultWaiting = 360:1;
for i=1:360
    if i <= 180
        resultWaiting(i) = i;
    else
        resultWaiting(i) = -360 + i;
    end
end
plot(resultWaiting,'Linewidth',3,'color',[0, 0.7, 0]);

for i = 1:numberDataPerAngle
    h1 = plot(angleSound,angle(exp(1j*result(:,i)))*180/pi,'o','MarkerSize',12)
    set(h1, 'markerfacecolor', get(h1, 'color')); % Use same color to fill in markers
end

ylabel('Angle calculate [°]')
xlabel('Angle of the emettor [°]')
xlim([0 360])
set(gca,'FontSize',22)
figure(3)

hold all

for i = 1:angleSoundNMeasure
    errorRms(i) =  sqrt(mean(error(i,:).^2));
    errorMax(i) =  max(abs(error(i,:)));
end


for i = 1:numberDataPerAngle
    h1 = plot(angleSound,errorRms,'d','MarkerSize',12);
    set(h1, 'markerfacecolor', get(h1, 'color')); % Use same color to fill in markers
    h1 = plot(angleSound,errorMax,'o','MarkerSize',12);
    set(h1, 'markerfacecolor', get(h1, 'color')); % Use same color to fill in markers
end
ylabel('error [°]')
xlabel('Angle of the emettor [°]')
set(gca,'FontSize',22)
```

```
%---------------------------------
% take in input the 3 PDM from the micro
% filter the PDM to get 3 sinus
% calculate the phase shift bewtween the singals
% save the phases shift into text file
%---------------------------------
f = 40e3;
vp = 343;  %[m/s] air
%vp = 1475;  %[m/s] water
%vp = 2730;  %[m/s] acrylic

lambda = vp/f;
k = 2*pi*f/vp;
r = 0.002;

frequency = 40e3;
sampling = 100;
tSampling = 1 / (frequency * sampling);

Fs = 1/tSampling;
nPeriod = 240;
t = 0:tSampling:(nPeriod/frequency);
t = t(1:nPeriod*sampling);

%---------------------------------
% LOW PASS FILTER,FC = 40[khz]
%---------------------------------

f1_ech = 40000;
f2_ech = 60000;
fs = 4000000;

%fréqeunce du filtre annalogique
f1 = fs/pi * tan(f1_ech*pi/fs);
f2 = fs/pi * tan(f2_ech*pi/fs);
H0 = 0;
H1 = -1;
H2 = -40;

%approxime filtre num
Wp = f1_ech/(fs/2);
Ws = f2_ech/(fs/2);
Rp = H0-H1;
Rs = -H2;

[n,Wp] = ellipord(Wp,Ws,Rp,Rs);
[b,a] = ellip(n,Rp,Rs,Wp,'low');



%---------------------------------
% Get data from measurement
%---------------------------------

nBits = 16;
measureId = 28;
dist = 100;

angleSoundStep = 45.0;
angleSoundNMeasure = 8;
for angleSound = 0:angleSoundStep:(angleSoundNMeasure-1)*angleSoundStep
    for index=1:10

        path = sprintf('../Measure/MeasurementAngle%d/%0.1f/',measureId,angleSound);

        %---------------------------------
        % READ SD CARD
        %---------------------------------
        dataMic1 = fscanf( fopen([path sprintf('MIC%d.txt',(index-1)*3 +
        1)],'r'),'%d');
        dataMic2 = fscanf( fopen([path sprintf('MIC%d.txt',(index-1)*3 +
        2)],'r'),'%d');
        dataMic3 = fscanf( fopen([path sprintf('MIC%d.txt',(index-1)*3 +
```

```
3)],'r'),'%d');

%----------------------------------
% get PDM
%----------------------------------

%bit pattern give the bit of a signal from the LSB to MSB
%SPI give 16 bits where MSB is the older bits
for i=1:length(dataMic1)
    bitpattern = bitget(dataMic1(i), 1:nBits, 'uint16')';
    for y = 1:nBits
        pdmMic1(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
    end
end

for i=1:length(dataMic2)
    bitpattern = bitget(dataMic2(i), 1:nBits, 'uint16')';
    for y = 1:nBits
        pdmMic2(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
    end
end

for i=1:length(dataMic3)
    bitpattern = bitget(dataMic3(i), 1:nBits, 'uint16')';
    for y = 1:nBits
        pdmMic3(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
    end
end

%----------------------------------
% Filter
%----------------------------------
signalFilter1 =  filter(b,a,pdmMic1);
signalFilter2 =  filter(b,a,pdmMic2);
signalFilter3 =  filter(b,a,pdmMic3);

% ----------------------------------
% Phase detection with IQ demodulation
% ----------------------------------
sinI = (sin(2 * pi * frequency * t));
cosQ = (cos(2 * pi * frequency * t));

I_sin1 = sinI .* (signalFilter1-mean(signalFilter1));
Q_sin1 = cosQ .* (signalFilter1-mean(signalFilter1));

I_sin1 = I_sin1(1500:length(signalFilter1));
Q_sin1 = Q_sin1(1500:length(signalFilter1));

I_sin2 = sinI .* (signalFilter2-mean(signalFilter2));
Q_sin2 = cosQ .* (signalFilter2-mean(signalFilter2));
I_sin2 = I_sin2(1500:length(signalFilter2));
Q_sin2 = Q_sin2(1500:length(signalFilter2));

I_sin3 = sinI .* (signalFilter3-mean(signalFilter3));
Q_sin3 = cosQ .* (signalFilter3-mean(signalFilter3));
I_sin3 = I_sin3(1500:length(signalFilter3));
Q_sin3 = Q_sin3(1500:length(signalFilter3));

Imean_signal1 = movmean(I_sin1,50);
Qmean_signal1 = movmean(Q_sin1,50);

Imean_signal2 = movmean(I_sin2,50);
Qmean_signal2 = movmean(Q_sin2,50);

Imean_signal3 = movmean(I_sin3,50);
Qmean_signal3 = movmean(Q_sin3,50);

phiSin1 = angle(mean(exp(1j * atan2(Qmean_signal1,Imean_signal1))));
phiSin2 = angle(mean(exp(1j * atan2(Qmean_signal2,Imean_signal2))));
phiSin3 = angle(mean(exp(1j * atan2(Qmean_signal3,Imean_signal3))));

phi12 = phiSin2 - phiSin1;
phi13 = phiSin3 - phiSin1;
```

```
144
145             phi12 = wrapToPi(phi12);
146             phi13 = wrapToPi(phi13);
147             phi12Tab(index) = phi12;
148             phi13Tab(index) = phi13;
149         end
150
151         %----------------------------------
152         % save phi12 and phi13 to text file
153         %----------------------------------
154         pathResultPhi12 =
            sprintf('../Result/meas%d/phi12/angle%0.1f.txt',measureId,angleSound);
155         pathResultPhi13 =
            sprintf('../Result/meas%d/phi13/angle%0.1f.txt',measureId,angleSound);
156
157         fprintf(fopen(pathResultPhi12,'w'),'%0.3f\n',phi12Tab);
158         fprintf(fopen(pathResultPhi13,'w'),'%0.3f\n',phi13Tab);
159         fclose('all');
160
161     end
162     fclose('all');
```

```
%---------------------------------
% Get Phi12 and Phi13 output data of
% getPhase_V2.m and calculate the
% sound angle arrival
%---------------------------------
index = 0;
numberDataPerAngle = 10;

phi12 = numberDataPerAngle:1;
phi13 = numberDataPerAngle:1;

dist = 250;
angleTransducer = 80;
% pathResultPhi12 =
sprintf('C:/Users/rebor/Desktop/testDistance/Result/%dcm/phi12.txt',dist);
% pathResultPhi13 =
sprintf('C:/Users/rebor/Desktop/testDistance/Result/%dcm/phi13.txt',dist);

% pathResultPhi12 =
sprintf('C:/Users/rebor/Desktop/testAngleEmissionGrosTransducer/Result/%d°/phi12.txt',
angleTransducer);
% pathResultPhi13 =
sprintf('C:/Users/rebor/Desktop/testAngleEmissionGrosTransducer/Result/%d°/phi13.txt',
angleTransducer);

pathResultPhi12 =
'C:/Users/rebor/Documents/GitHub/PlongDir/03_Software/00_Matlab/Test/EmetteurSillicone
/Result/phi12.txt';
pathResultPhi13 =
'C:/Users/rebor/Documents/GitHub/PlongDir/03_Software/00_Matlab/Test/EmetteurSillicone
/Result/phi13.txt';

phi12= fscanf(fopen(pathResultPhi12,'r'),'%f');
phi13= fscanf(fopen(pathResultPhi13,'r'),'%f');

%---------------------------------
% Calcul theorical Phi12 Phi13
%---------------------------------
f = 40e3;
vp = 343;  %[m/s] air
%vp = 1475;  %[m/s] water
%vp = 2730;  %[m/s] acrylic

lambda = vp/f;
k = 2*pi*f/vp;

r = 0.002;

R = 1;
x1 = r;
y1 = 0;
x2 = -r/2;
y2 = sqrt(3)*r/2;
x3 = -r/2;
y3 = -sqrt(3)*r/2;

theta = (0:2*pi/1000:2*pi)';
theta_deg = (0:360/1000:360)';

x0 = R*cos(theta);
y0 = R*sin(theta);

d1 = sqrt((x1-x0).^2+(y1-y0).^2);
d2 = sqrt((x2-x0).^2+(y2-y0).^2);
d3 = sqrt((x3-x0).^2+(y3-y0).^2);

phi1 = k*d1;
phi2 = k*d2;
phi3 = k*d3;

phi1 = angle(exp(1j*phi1));
phi2 = angle(exp(1j*phi2));
phi3 = angle(exp(1j*phi3));
```

```
64
65  phi12Calc = phi2-phi1;
66  phi13Calc = phi3-phi1;
67
68  phi12Calc = wrapToPi(phi12Calc);
69  phi13Calc = wrapToPi(phi13Calc);
70
71  alphaCalc = atan2(phi12Calc,phi13Calc);
72  alpha = atan2(-phi12,-phi13);
73
74  %---------------------------------
75  % sort alphaCalc array
76  %---------------------------------
77  alphaCalcSort = alphaCalc;
78  thetaSort = theta;
79
80  for j = 0 : length(alphaCalcSort)-1
81      for i = 1: length(alphaCalcSort)-j-1
82          if alphaCalcSort(i)>alphaCalcSort(i+1)
83              %switch first array
84              temp = alphaCalcSort(i);
85              alphaCalcSort(i) = alphaCalcSort(i+1);
86              alphaCalcSort(i+1) = temp;
87
88              %switch second array
89              temp = thetaSort(i);
90              thetaSort(i) = thetaSort(i+1);
91              thetaSort(i+1) = temp;
92          end
93      end
94  end
95
96  %---------------------------------
97  % calculate result
98  %---------------------------------
99  for i = 1:length(alpha)
100     for x = 1:length(alphaCalcSort)
101         if alpha(i) < alphaCalcSort(x)
102             result(i) = thetaSort(x);
103             break;
104         end
105     end
106 end
107 %---------------------------------
108 % Display result
109 %---------------------------------
110 figure(1)
111 scatter(phi12Calc*180/pi,phi13Calc*180/pi,10,theta_deg);
112 hold all
113 scatter(-phi12*180/pi,-phi13*180/pi,8,[0 0 0]);
114 xlabel('phi2-phi1[°]')
115 ylabel('phi3-phi1[°]')
116
117 figure(2)
118 plot(result*180/pi)
119
```

```
%---------------------------------
% take in input the 3 PDM from the micro
% filter the PDM to get 3 sinus
% calculate the phase shift bewtween the signals
% save the phases shift into text file
%---------------------------------
f = 40e3;
vp = 343;  %[m/s] air
%vp = 1475;  %[m/s] water
%vp = 2730;  %[m/s] acrylic

lambda = vp/f;
k = 2*pi*f/vp;
r = 0.002;

frequency = 40e3;
sampling = 100;
tSampling = 1 / (frequency * sampling);

Fs = 1/tSampling;
nPeriod = 240;
t = 0:tSampling:(nPeriod/frequency);
t = t(1:nPeriod*sampling);
%---------------------------------
% LOW PASS FILTER, fc = 40[khz]
%---------------------------------

f1_ech = 40000;
f2_ech = 60000;
fs = 4000000;

%fréqeunce du filtre annalogique
f1 = fs/pi * tan(f1_ech*pi/fs);
f2 = fs/pi * tan(f2_ech*pi/fs);
H0 = 0;
H1 = -1;
H2 = -40;

%approxime filtre num
Wp = f1_ech/(fs/2);
Ws = f2_ech/(fs/2);
Rp = H0-H1;
Rs = -H2;

[n,Wp] = ellipord(Wp,Ws,Rp,Rs);
[b,a] = ellip(n,Rp,Rs,Wp,'low');

%---------------------------------
% Get data from measurement
%---------------------------------

nBits = 16;
measureId = 15;
dist = 300;
angleTransducer = 70;
```

```
for index=1:10
    path = '../Test/EmetteurSillicone/Measure/';
    %---------------------------------
    % READ SD CARD
    %---------------------------------
    dataMic1 = fscanf( fopen([path sprintf('MIC%d.txt',(index-1)*3 + 1)],'r'),'%d');
    dataMic2 = fscanf( fopen([path sprintf('MIC%d.txt',(index-1)*3 + 2)],'r'),'%d');
    dataMic3 = fscanf( fopen([path sprintf('MIC%d.txt',(index-1)*3 + 3)],'r'),'%d');

    %---------------------------------
    % get PDM
    %---------------------------------

    %bit pattern give the bit of a signal from the LSB to MSB
    %SPI give 16 bits where MSB is the older bits
    for i=1:length(dataMic1)
        bitpattern = bitget(dataMic1(i), 1:nBits, 'uint16')';
        for y = 1:nBits
            pdmMic1(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
        end
    end

    for i=1:length(dataMic2)
        bitpattern = bitget(dataMic2(i), 1:nBits, 'uint16')';
        for y = 1:nBits
            pdmMic2(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
        end
    end

    for i=1:length(dataMic3)
        bitpattern = bitget(dataMic3(i), 1:nBits, 'uint16')';
        for y = 1:nBits
            pdmMic3(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
        end
    end

    %---------------------------------
    % Filter
    %---------------------------------
    signalFilter1 =  filter(b,a,pdmMic1);
    signalFilter2 =  filter(b,a,pdmMic2);
    signalFilter3 =  filter(b,a,pdmMic3);

    % ---------------------------------
    % Phase detection
    % ---------------------------------
    sinI = (sin(2 * pi * frequency * t));
    cosQ = (cos(2 * pi * frequency * t));

    I_sin1 = sinI .* (signalFilter1-mean(signalFilter1));
    Q_sin1 = cosQ .* (signalFilter1-mean(signalFilter1));

    I_sin1 = I_sin1(1500:length(signalFilter1));
    Q_sin1 = Q_sin1(1500:length(signalFilter1));
```

```
    I_sin2 = sinI .* (signalFilter2-mean(signalFilter2));
    Q_sin2 = cosQ .* (signalFilter2-mean(signalFilter2));
    I_sin2 = I_sin2(1500:length(signalFilter2));
    Q_sin2 = Q_sin2(1500:length(signalFilter2));

    I_sin3 = sinI .* (signalFilter3-mean(signalFilter3));
    Q_sin3 = cosQ .* (signalFilter3-mean(signalFilter3));
    I_sin3 = I_sin3(1500:length(signalFilter3));
    Q_sin3 = Q_sin3(1500:length(signalFilter3));

    Imean_signal1 = movmean(I_sin1,50);
    Qmean_signal1 = movmean(Q_sin1,50);

    Imean_signal2 = movmean(I_sin2,50);
    Qmean_signal2 = movmean(Q_sin2,50);

    Imean_signal3 = movmean(I_sin3,50);
    Qmean_signal3 = movmean(Q_sin3,50);

    phiSin1 = angle(mean(exp(1j * atan2(Qmean_signal1,Imean_signal1))));
    phiSin2 = angle(mean(exp(1j * atan2(Qmean_signal2,Imean_signal2))));
    phiSin3 = angle(mean(exp(1j * atan2(Qmean_signal3,Imean_signal3))));

    phi12 = phiSin2 - phiSin1;
    phi13 = phiSin3 - phiSin1;

    phi12 = wrapToPi(phi12);
    phi13 = wrapToPi(phi13);

    phi12Tab(index) = phi12;
    phi13Tab(index) = phi13;


end

%----------------------------------
% save phi12 and phi13 to text file
%----------------------------------
pathResultPhi12 = '../Test/EmetteurSillicone/Result/phi12.txt';
pathResultPhi13 = '../Test/EmetteurSillicone/Result/phi13.txt';

fprintf(fopen(pathResultPhi12,'w'),'%0.3f\n',phi12Tab);
fprintf(fopen(pathResultPhi13,'w'),'%0.3f\n',phi13Tab);
fclose('all');
```

```
%---------------------------------
%
% Triangulation of position script
% display soutput singal filtered of
% microphones
%---------------------------------

%---------------------------------
%Sinus generation
%---------------------------------
f = 40e3;
vp = 343;  %[m/s] air
%vp = 1475;  %[m/s] water
%vp = 2730;  %[m/s] acrylic

lambda = vp/f;
k = 2*pi*f/vp;
r = 0.002;

frequency = 40e3;
sampling = 100;
tSampling = 1 / (frequency * sampling);

Fs = 1/tSampling;
nPeriod = 240;
t = 0:tSampling:(nPeriod/frequency);
t = t(1:nPeriod*sampling);
nBits = 16;


for index=1:10

    %---------------------------------
    % READ SD CARD
    %---------------------------------
    path = '../Test/AngleEmissionUT-1640K-TT-2-R/Measure/70°/';

    dataMic1 = fscanf( fopen(sprintf([path 'MIC%d.txt'],(index-1)*3 + 1),'r'),'%d');
    dataMic2 = fscanf( fopen(sprintf([path 'MIC%d.txt'],(index-1)*3 + 2),'r'),'%d');
    dataMic3 = fscanf( fopen(sprintf([path 'MIC%d.txt'],(index-1)*3 + 3),'r'),'%d');


    %bit pattern give the bit of a signal from the LSB to MSB
    %SPI give 8 bits where MSB is the older bits
    for i=1:length(dataMic1)
        bitpattern = bitget(dataMic1(i), 1:nBits, 'uint16')';
        for y = 1:nBits
            pdmMic1(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
        end
    end

    for i=1:length(dataMic2)
        bitpattern = bitget(dataMic2(i), 1:nBits, 'uint16')';
        for y = 1:nBits
            pdmMic2(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
```

```
    end
end

for i=1:length(dataMic3)
    bitpattern = bitget(dataMic3(i), 1:nBits, 'uint16')';
    for y = 1:nBits
        pdmMic3(nBits*(i-1) + y) = bitpattern(nBits + 1 - y);
    end
end

%----------------------------------
% LOW PASS FILTER
%----------------------------------

f1_ech = 40000;
f2_ech = 60000;
fs = 4000000;

%fréqeunce du filtre annalogique
f1 = fs/pi * tan(f1_ech*pi/fs);
f2 = fs/pi * tan(f2_ech*pi/fs);
H0 = 0;
H1 = -1;
H2 = -40;

%approxime filtre num
Wp = f1_ech/(fs/2);
Ws = f2_ech/(fs/2);
Rp = H0-H1;
Rs = -H2;

[n,Wp] = ellipord(Wp,Ws,Rp,Rs);
[b,a] = ellip(n,Rp,Rs,Wp,'low');

%----------------------------------
% Filter
%----------------------------------
signalFilter1 =  filter(b,a,pdmMic1);
signalFilter2 =  filter(b,a,pdmMic2);
signalFilter3 =  filter(b,a,pdmMic3);

% ----------------------------------
% Phase detection
% ----------------------------------
sinI = (sin(2 * pi * frequency * t));
cosQ = (cos(2 * pi * frequency * t));

I_sin1 = sinI .* (signalFilter1-mean(signalFilter1));
Q_sin1 = cosQ .* (signalFilter1-mean(signalFilter1));

I_sin1 = I_sin1(1500:length(signalFilter1));
Q_sin1 = Q_sin1(1500:length(signalFilter1));

I_sin2 = sinI .* (signalFilter2-mean(signalFilter2));
Q_sin2 = cosQ .* (signalFilter2-mean(signalFilter2));
```

```
    I_sin2 = I_sin2(1500:length(signalFilter2));
    Q_sin2 = Q_sin2(1500:length(signalFilter2));

    I_sin3 = sinI .* (signalFilter3-mean(signalFilter3));
    Q_sin3 = cosQ .* (signalFilter3-mean(signalFilter3));
    I_sin3 = I_sin3(1500:length(signalFilter3));
    Q_sin3 = Q_sin3(1500:length(signalFilter3));

    Imean_signal1 = movmean(I_sin1,50);
    Qmean_signal1 = movmean(Q_sin1,50);

    Imean_signal2 = movmean(I_sin2,50);
    Qmean_signal2 = movmean(Q_sin2,50);

    Imean_signal3 = movmean(I_sin3,50);
    Qmean_signal3 = movmean(Q_sin3,50);

    phiSin1 = angle(mean(exp(1j*atan2(Qmean_signal1,Imean_signal1))));
    phiSin2 = angle(mean(exp(1j*atan2(Qmean_signal2,Imean_signal2))));
    phiSin3 = angle(mean(exp(1j*atan2(Qmean_signal3,Imean_signal3))));

    phi12 = phiSin2 - phiSin1;
    phi13 = phiSin3 - phiSin1;
    %---------------------------------
    % Calculate direction
    %---------------------------------

    if phi12 > pi
        phi12 = phi12 - 2 * pi;
    elseif phi12 < -pi
         phi12 = phi12 + 2 * pi;
    end

    if phi13 > pi
        phi13 = phi13 - 2 * pi;
    elseif phi13 < -pi
         phi13 = phi13 + 2 * pi;
    end

    disp(sprintf('phi12 value %d : %f',index,phi12*180/pi));
    disp(sprintf('phi13 value %d : %f',index,phi13*180/pi));

    phi12Tab(index) = phi12*180/pi;
    phi13Tab(index) = phi13*180/pi;

    figure(index);
    plot([signalFilter1' signalFilter2' signalFilter3']);
    xlabel('point')
    ylabel('amplitude')
    legend('mic1 signal filter','mic2 signal filter','mic3 signal filter')
    set(gca,'FontSize',20)
    xlim([1000 1300])

end
phi12Tab
```

```
phi13Tab
fclose('all');
```

```
%---------------------------------
%
% Triangulation of position script
% Test the algorithm to detect sound
% arrival
%---------------------------------

%---------------------------------
%Sinus generation
%---------------------------------
f = 40e3;
vp = 343;  %[m/s] air
%vp = 1475;  %[m/s] water
%vp = 2730;  %[m/s] acrylic

lambda = vp/f;
k = 2*pi*f/vp;

coeffDist = 0.9;

r = 0.002;

R = 3;
x1 = r;
y1 = 0;
x2 = -r/2;
y2 = sqrt(3)*r/2;
x3 = -r/2;
y3 = -sqrt(3)*r/2;

theta = (0:2*pi/1000:2*pi)';
x0 = R*cos(theta);
y0 = R*sin(theta);

d1 = sqrt((x1-x0).^2+(y1-y0).^2);
d2 = sqrt((x2-x0).^2+(y2-y0).^2);
d3 = sqrt((x3-x0).^2+(y3-y0).^2);

phi1 = k*d1;
phi2 = k*d2;
phi3 = k*d3;

phi1 = angle(exp(1j*phi1));
phi2 = angle(exp(1j*phi2));
phi3 = angle(exp(1j*phi3));

%--------------------add some error---------------------------
degreeErrorMax = 25;

%create random error bewteen 0 and 10 degrees
randomVector1 = rand(1001,1) * degreeErrorMax * pi/180;
randomVector2 = rand(1001,1) * degreeErrorMax * pi/180;
randomVector3 = rand(1001,1) * degreeErrorMax * pi/180;

phi12Calc = phi2-phi1;
```

```
phi13Calc = phi3-phi1;

%add the error to the phase
phi1 = phi1 + randomVector1;
phi2 = phi2 + randomVector2;
phi3 = phi3 + randomVector3;

%renormalize the phse
phi1 = angle(exp(1j*phi1));
phi2 = angle(exp(1j*phi2));
phi3 = angle(exp(1j*phi3));


DSM = DeltaSigmaModulator('Oversampling',1);
%----------------------------------
% LOW PASS FILTER
%----------------------------------

f1_ech = 40000;
f2_ech = 60000;
fs = 4000000;

%fréqeunce du filtre annalogique
f1 = fs/pi * tan(f1_ech*pi/fs);
f2 = fs/pi * tan(f2_ech*pi/fs);
H0 = 0;
H1 = -1;
H2 = -40;

%approxime filtre num
Wp = f1_ech/(fs/2);
Ws = f2_ech/(fs/2);
Rp = H0-H1;
Rs = -H2;

[n,Wp] = ellipord(Wp,Ws,Rp,Rs);
[b,a] = ellip(n,Rp,Rs,Wp,'low');

% figure(1)
% %sys = tf(b,a,1/fs);
% options = bodeoptions;
% options.FreqUnits = 'Hz';
% bode(sys,options)

result = length(theta):1;
error = length(theta):1;
phi12 = length(theta):1;
phi13 = length(theta):1;

for x = 1:length(theta)

    frequency = 40e3;
    sampling = 100;
    tSampling = 1 / (frequency * sampling);
    Fs = 1/tSampling;
```

```
nPeriod = 50;

t = 0:tSampling:(nPeriod/frequency);

%the 3 sinus
sinus1 = (sin(2 * pi * frequency * t + phi1(x,1)) + 1) /2;
sinus2 = (sin(2 * pi * frequency * t + phi2(x,1)) + 1) /2;
sinus3 = (sin(2 * pi * frequency * t + phi3(x,1)) + 1) /2;

%----------------------------------
% A/D converter sigma delta
%----------------------------------

[Signal1,SignalDS1] = DSM.update(sinus1);
[Signal2,SignalDS2] = DSM.update(sinus2);
[Signal3,SignalDS3] = DSM.update(sinus3);

%----------------------------------
% Filter
%----------------------------------

signalFilter1 =  filter(b,a,SignalDS1);
signalFilter2 =  filter(b,a,SignalDS2);
signalFilter3 =  filter(b,a,SignalDS3);

%----------------------------------
% Phase detection
%----------------------------------

sinI = (sin(2 * pi * frequency * t));
cosQ = (cos(2 * pi * frequency * t));

I_sin1 = sinI .* (signalFilter1-mean(signalFilter1));
Q_sin1 = cosQ .* (signalFilter1-mean(signalFilter1));

I_sin1 = I_sin1(1500:5000);
Q_sin1 = Q_sin1(1500:5000);

I_sin2 = sinI .* (signalFilter2-mean(signalFilter2));
Q_sin2 = cosQ .* (signalFilter2-mean(signalFilter2));
I_sin2 = I_sin2(1500:5000);
Q_sin2 = Q_sin2(1500:5000);

I_sin3 = sinI .* (signalFilter3-mean(signalFilter3));
Q_sin3 = cosQ .* (signalFilter3-mean(signalFilter3));
I_sin3 = I_sin3(1500:5000);
Q_sin3 = Q_sin3(1500:5000);

Imean_signal1 = movmean(I_sin1,50);
Qmean_signal1 = movmean(Q_sin1,50);

Imean_signal2 = movmean(I_sin2,50);
Qmean_signal2 = movmean(Q_sin2,50);
```

```
    Imean_signal3 = movmean(I_sin3,50);
    Qmean_signal3 = movmean(Q_sin3,50);

    phiSin1 = angle(mean(exp(1j*(atan2(Qmean_signal1,Imean_signal1)))));
    phiSin2 = angle(mean(exp(1j*(atan2(Qmean_signal2,Imean_signal2)))));
    phiSin3 = angle(mean(exp(1j*(atan2(Qmean_signal3,Imean_signal3)))));

    phi12(x) = phiSin2 - phiSin1;
    phi13(x) = phiSin3 - phiSin1;
end

alphaCalc = atan2(phi13Calc,phi12Calc);
alpha = atan2(phi13,phi12);

%---------------------------------
% sort alphaCalc array
%---------------------------------
alphaCalcSort = alphaCalc;
thetaSort = theta;

for j = 0 : length(alphaCalcSort)-1
    for i = 1: length(alphaCalcSort)-j-1
        if alphaCalcSort(i)>alphaCalcSort(i+1)
            %switch first array
            temp = alphaCalcSort(i);
            alphaCalcSort(i) = alphaCalcSort(i+1);
            alphaCalcSort(i+1) = temp;

            %switch second array
            temp = thetaSort(i);
            thetaSort(i) = thetaSort(i+1);
            thetaSort(i+1) = temp;
        end
    end
end

error = zeros(size(alpha));
%---------------------------------
% calcul result and error
%---------------------------------
for i = 1:length(alpha)
    for x = 1:length(alphaCalcSort)
        result(i) = thetaSort(x);
        if alpha(i) < alphaCalcSort(x)
            break;
        end
    end

    error(i) = (result(i) - theta(i));

    error(i) = error(i) * 180/pi;
    if error(i) < -300
        error(i) = error(i) + 360;
    elseif error(i) > 300
        error(i) = error(i) - 360;
```

```
    end
end

%----------------------------------
% display scatter
%----------------------------------
figure(2)
hold all
scatter(phi13Calc*180/pi,phi12Calc*180/pi,10,theta*180/pi)
scatter(phi13*180/pi,phi12*180/pi,10,360*(0:1000)'/1000)
set(gca,'FontSize',22)
ylabel('phi3-phi1 [°]')
xlabel('phi2-phi1 [°]')

%----------------------------------
% display result
%----------------------------------
figure(3)
hold all

resultWaiting = length(result):1;
stepAngle = 180/500;
for i=1:length(result)
    if i <= 501
        resultWaiting(i) = (i-1) * stepAngle;
    else
        resultWaiting(i) = -180 + (i-502) * stepAngle;
    end
end
plot(angle(exp(1j*result))*180/pi)
plot(resultWaiting,'LineWidth',2)
set(gca,'FontSize',22)
ylabel('Angle of the emettor [°]')
xlabel('Point number')

%----------------------------------
% display error
%----------------------------------
figure(4)
plot(error)
ylabel('Angle Error in [°]')
xlabel('Point number')
set(gca,'FontSize',22)
%----------------------------------
% display angleCalcSort
%----------------------------------
figure(5)
plot(alphaCalcSort*180/pi,thetaSort*180/pi);
ylabel('theta[°]')
xlabel('alpha[°]')
set(gca,'FontSize',22)
figure(6)
plot(alphaCalc*180/pi,theta*180/pi);
ylabel('theta[°]')
xlabel('alpha[°]')
```

```
set(gca,'FontSize',22)
```

Annexe 3 :

Programme
V1.0

Annexe 3.1 :

Code source

```
/*
 * Controller.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef CONTROLLER_H_
#define CONTROLLER_H_

//list
#include <event/evStartMeasure.h>
#include <list>

//for sstream
#include <sstream>
#include <string.h>
#include <stdio.h>

//interface
#include "xf/behavior.h"

//events
#include "event/evMeasureMicsFinish.h"

//relations
#include "app/SpiController.h"
#include "app/SdCard.h"

#define WAIT_TIME 1000
#define START_MIC_TIME 20
/**
 * @brief this class is a singleton
 * it contains a state machine
 * the state machine is started and each second the measure is started
 * then it will take Measurement on the micro for 10 ms
 * Then it wil save the data on a sd card and go back to the initial state
 */
class Controller : public XFBehavior
{

    friend void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef* hspi);

    public:
        /**
        * @brief getInstance of the singleton class
        * @return the only one instance of the singleton class
        */
        static Controller& getInstance();
        void startBehavior();                                   ///< @brief start the state machine

        /**
        * @brief return the only one instance of the singleton class
        * @param theSdCard is a pointer tothe instance of the SdCard class
        */
        void initRelations(SdCard* theSdCard);

        /**
        * @brief register a new SpiController instance
        * @param theSPIController is a pointer to the instance of the SpiController class
        * @param micIndex is the micro index in the PCB
        */
        void registerSpiController(SpiController* theSPIController, uint8_t micIndex);

    protected:

        XFEventStatus processEvent();            ///< @brief State machine

        void dma_Finish();                        ///< @brief called when one dma transfer is finish
    private:

        Controller();
        ~Controller();
        Controller(const Controller&){}
        Controller& operator=(const Controller&){}

        /**
        *@brief start the microphone clock
        * microphone take 10[ms] to startup
        */
        void startMicrophones();

        void saveMeasure();     ///< @brief Save measure to SdCard
```

```
84          void startMeasure(); ///< @brief Launch a PWM to start mics measure
85          void stopMeasure();  ///< @brief Stop the PWM Stop mics measure
86
87          /**
88           * @brief Enumeration used to have a unique identifier for every
89           * state in the state machine.
90           */
91          typedef enum
92          {
93              STATE_NULL = 0,         ///< @brief null state
94              WAIT,                ///< @brief Wait timer
95              START_MICS,            ///< @brief start taking measure from the micros
96              START_MEASURE,         ///< @brief start taking measure from the micros
97              SAVE_MEASURE,        ///< @brief store measure value into SD Card
98          } eMainState;
99
100         eMainState _currentState;            ///< @brief Attribute indicating currently active state
101         eMainState _oldState;                ///< @brief Attribute indicating old state
102
103         /**
104          * @brief Struct who contains micIndex
105          * and theSpiController for the mic with that
106          * micIndex
107          */
108         typedef struct{
109             SpiController* theSPIController;    ///< @brief the SPIController who controller the mic with mic index
110             uint8_t micIndex;                    ///< @brief index of the mics controlled by the spi controller
111         }MicController;
112
113         std::list<MicController*> micControllerList;     ///< @brief list of callback
114         XFMutexDefault* mutexMicControllerList;         ///< @brief mutex for the list
115         SdCard* theSdCard;                                ///< @brief the sdCardController
116 };
117
118 #endif /* CONTROLLER_H_ */
```

```cpp
/*
 * Controller.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/Controller.h"
#include "main.h"

extern "C" TIM_HandleTypeDef htim2;
extern "C" TIM_HandleTypeDef htim3;

Controller::Controller() {
    // TODO Auto-generated constructor stub
    this->_currentState = STATE_NULL;
    this->_oldState = STATE_NULL;

    this->mutexMicControllerList = nullptr;

    this->theSdCard = nullptr;
}

Controller::~Controller() {
    // TODO Auto-generated destructor stub

    //delete mutexSpiControllerList if it exist
    if(this->mutexMicControllerList)
        delete this->mutexMicControllerList;
}

Controller& Controller::getInstance()
{
    //create a static object
    static Controller theController;
    //return the static object
    return theController;
}


void Controller::initRelations(SdCard* theSdCard)
{
    //create mutex
    this->mutexMicControllerList = (XFMutexDefault*) interface::XFMutex::create();

    this->theSdCard = theSdCard;
}

void Controller::startBehavior()
{
    //start the state machine
    XFBehavior::startBehavior();

    //start the spi in receive mode
    for(std::list<MicController*>::iterator it = this->micControllerList.begin();it!=this->micControllerList.end(); it++)
    {
        (*it)->theSPIController->start();
    }
}

void Controller::registerSpiController(SpiController* theSpiController, uint8_t micIndex)
{
    bool found = false;

    //enter critical section
    this->mutexMicControllerList->lock();

    for(std::list<MicController*>::iterator it = this->micControllerList.begin();it!=this->micControllerList.end(); it++)
    {
        if((*it)->theSPIController  == theSpiController)
        {
            found = true;
            break;
        }
    }
    //if the spiController is not in the list --> add to the list
    if(!found)
    {
        this->micControllerList.push_back(new MicController({theSpiController,micIndex}));
    }

    //leave critical section
    this->mutexMicControllerList->unlock();
}

void Controller::dma_Finish()
{
    static uint8_t nMeasureFinish = 0;
    nMeasureFinish++;

```

```
91      //when the 3 dma transfer are finish, generate an event
92      if(nMeasureFinish ==  3 && this->_currentState == START_MEASURE)
93      {
94          nMeasureFinish = 0;
95          GEN(evMeasureMicsFinish);
96      }
97      else if(this->_currentState != START_MEASURE)
98      {
99          nMeasureFinish = 0;
100     }
101 }
102
103 void Controller::saveMeasure()
104 {
105     std::list<MicController*>::iterator it;
106
107     char fileName[9];
108     bool measureOk = false;
109
110     //uint8_t* ptrData;
111
112     //stingsstream : contains the message to write in sd card
113     stringstream messageToWrite;
114     static uint8_t index = 0;
115
116     // CAN'T PROTECT WITH MUTEX HERE, some methode to manage sd card use hal_delay --> if a mutex is done(interrupt stopped)
117     //the hal_delay()
118     for( it = this->micControllerList.begin();
119             it!=this->micControllerList.end(); it++)
120     {
121         sprintf(fileName,"mic%d.txt",(*it)->micIndex+index*3);
122
123         //open file
124         if(!this->theSdCard->openFile(fileName))
125         {
126             measureOk = false;
127             break;
128         }
129
130         //The microphone take 10 ms to start measurement(change mode time)
131         //the clock is at 4[MHz] --> 40kbits during 10[ms] --> 2500 * 16 bits
132         //dont take the first 3500 to take a little margin
133         for(uint16_t i = 3500; i < DATA_SIZE ; i++)
134         {
135             messageToWrite << to_string(((*it)->theSPIController->getData()[i])) << "\n";
136         }
137
138         if(!this->theSdCard->writeFile(messageToWrite.str().c_str(), messageToWrite.str().size()))
139         {
140             measureOk = false;
141             break;
142         }
143
144         messageToWrite=std::stringstream();
145         messageToWrite.str(std::string());
146
147         //close file
148         if(!this->theSdCard->closeFile(fileName))
149         {
150             measureOk = false;
151             break;
152         }
153
154         measureOk = true;
155     }
156
157     if(measureOk)
158         index++;
159
160 }
161
162
163 void Controller::startMeasure()
164 {
165     HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
166
167 }
168 void Controller::startMicrophones()
169 {
170     //launch microphone
171     HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
172 }
173
174
175 void Controller::stopMeasure()
176 {
177     //stop the clock of the SPI
178     HAL_TIM_PWM_Stop(&htim3,TIM_CHANNEL_1);
179     HAL_TIM_PWM_Stop(&htim2,TIM_CHANNEL_1);
180
181     //flush spi fifo
```

```
182     std::list<MicController*>::iterator it;
183 
184     for( it = this->micControllerList.begin();
185             it!=this->micControllerList.end(); it++)
186     {
187         HAL_SPIEx_FlushRxFifo((*it)->theSPIController->getHSPI());
188     }
189 }
190 
191 
192 XFEventStatus Controller::processEvent()
193 {
194 
195     eEventStatus eventStatus = XFEventStatus::Unknown;
196 
197     //get current event
198     const XFEvent * event  = getCurrentEvent();
199 
200     _oldState = _currentState;
201 
202     switch (_currentState)
203     {
204     case STATE_NULL:
205         if(event->getEventType() == XFEvent::Initial)
206         {
207             //intial event --> go to WAIT_BUTTON_PRESSED state
208             _currentState = WAIT;
209             eventStatus = XFEventStatus::Consumed;
210         }
211         break;
212     case WAIT: //wait timeout occur
213         if(event->getEventType() == XFEvent::Timeout &&
214                 event->getId() == TimeoutIds::timeoutWaitId)
215         {
216             //timeout occures--> go to start measure state
217             _currentState = START_MICS;
218             eventStatus = XFEventStatus::Consumed;
219         }
220         break;
221     case START_MICS://start measure
222         if (event->getEventType() == XFEvent::Timeout
223                 && event->getId() == TimeoutIds::timeoutStartMeasureId)
224         {
225             //measure are finished--> ggo to save measure state
226             _currentState = START_MEASURE;
227             eventStatus = XFEventStatus::Consumed;
228         }
229         break;
230     case START_MEASURE://start measure
231         if (event->getEventType() == XFEvent::Event
232                 && event->getId() == EventIds::evMeasureMicsFinishId)
233         {
234             //measure are finished--> ggo to save measure state
235             _currentState = SAVE_MEASURE;
236             eventStatus = XFEventStatus::Consumed;
237         }
238         break;
239     case SAVE_MEASURE: //save measure
240         if(event->getEventType() == XFEvent::NullTransition)
241         {
242             //go back to WAITstate
243             _currentState = WAIT;
244             eventStatus = XFEventStatus::Consumed;
245         }
246         break;
247     default:
248         break;
249     }
250 
251     if(_oldState != _currentState)
252     {
253         switch(_currentState)
254         {
255         case WAIT:
256             if(this->_oldState == STATE_NULL || this->_oldState == SAVE_MEASURE)
257             {
258                 scheduleTimeout(timeoutWaitId,WAIT_TIME);
259             }
260             break;
261         case START_MICS:
262             if(this->_oldState == WAIT)
263             {
264                 this->startMicrophones();
265                 scheduleTimeout(timeoutStartMeasureId,START_MIC_TIME);
266             }
267             break;
268         case START_MEASURE:
269             if(this->_oldState == START_MICS)
270             {
271                 this->startMeasure();
272                 Trace::out("start measure");
```

```
273             }
274             break;
275         case SAVE_MEASURE:
276             this->stopMeasure();
277             this->saveMeasure();
278             GEN(XFNullTransition);
279             Trace::out("save measure finish");
280             break;
281         default:
282             break;
283         }
284 
285     }
286     return eventStatus;
287 
288 }
```

```
#ifndef FACTORY_H
#define FACTORY_H

//
// What is seen only by the C++ compiler
//
#ifdef __cplusplus

// for the list
#include <list>

//class to init
#include "app/Controller.h"
#include "app/SpiController.h"
#include "app/SdCard.h"

extern "C" SPI_HandleTypeDef hspi1;
extern "C" SPI_HandleTypeDef hspi2;
extern "C" SPI_HandleTypeDef hspi5;

extern "C" DMA_HandleTypeDef hdma_spi1_rx;
extern "C" DMA_HandleTypeDef hdma_spi2_rx;
extern "C" DMA_HandleTypeDef hdma_spi5_rx;

extern "C" TIM_HandleTypeDef htim1;

/**
 * @brief This is the factory. The factory
 * creates the object of class and build
 * the relations between the class
 */
class Factory
{
public:
    Factory();

    static void initialize();                  ///< @brief Initializes the factory
    static void build();                      ///< @brief Creates components and initializes relations
protected:
    static SpiController spiController1;    ///< @brief SpiController for the micro number 2
    static SpiController spiController2;    ///< @brief SpiController for the micro number 3
    static SpiController spiController3;    ///< @brief SpiController for the micro number 1
    static SdCard sdCard;                    ///< @brief Sd card management
};

#endif // __cplusplus

//
// What is seen by the C and C++ compiler
//
#ifdef __cplusplus
extern "C" {
#endif // __cplusplus

void Factory_initialize();
void Factory_build();

#ifdef __cplusplus
}
#endif // __cplusplus

#endif // FACTORY_H
```

```cpp
#include "trace/trace.h"
#include "factory.h"


SpiController Factory::spiController1;
SpiController Factory::spiController2;
SpiController Factory::spiController3;
SdCard Factory::sdCard;

Factory::Factory()
{

}

void Factory::initialize()
{
    //init all spiController
    spiController1.init(&hspi1);
    spiController2.init(&hspi2);
    spiController3.init(&hspi5);

}

void Factory::build()
{
    //init relations
    Controller::getInstance().initRelations(&sdCard);

    //register all SPI into Controller class
    Controller::getInstance().registerSpiController(&spiController3,1);
    Controller::getInstance().registerSpiController(&spiController2,2);
    Controller::getInstance().registerSpiController(&spiController1,3);


    //launch state machine
    Controller::getInstance().startBehavior();
}

void Factory_initialize()
{
    Factory::initialize();
}

void Factory_build()
{
    Factory::build();
}
```

```
/*
 * SdCard.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef SDCARD_H_
#define SDCARD_H_

#include <string.h>

//for debug
#include "trace/trace.h"

//for sd card
#include "stm32f7xx_hal_sd.h"
#include "inc/fatfs.h"
#include "ff.h"


/**
 * @brief this class write data into SDCard
 */
class SdCard {
    public:
        SdCard();
        virtual ~SdCard();
        /**
        *@brief open a file in the sd card
        *@param fileName is the text file Name
        *@return true if the file has been created and opened
        */
        bool openFile(char* fileName);
        /**
        *@brief close a file in the sd card
        *@param fileName is the text file Name
        *@return true if the file has been closed
        */
        bool closeFile(char* fileName);
        /**
        *@brief write into the text file some data
        *@param data is a pointer to the data to be written
        *@param length is the length of the data to be write
        *@return true if the file has been writted correctly
        */
        bool writeFile(const char* data,uint32_t length);
    private:
        //variable for sd card write
        FRESULT res;            ///< @brief result of the action
        FIL fil;                ///< @brief file object structure
        FATFS fatSD;            ///< @brief file system object
        FILINFO fno;            ///< @brief file  information
        UINT bytesWritten;        ///< @brief bytes written in sd card
};


#endif /* SDCARD_H_ */
```

```cpp
/*
 * SdCard.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/SdCard.h"


SdCard::SdCard() {
    // TODO Auto-generated constructor stub

}

SdCard::~SdCard() {
    // TODO Auto-generated destructor stub
}

bool SdCard::openFile(char* fileName)
{
    //mount sd card
    res = f_mount(&fatSD,"",1);

    //res = FR_OK, if the card has been mounted
    if(res!=FR_OK)
    {
        //if not, unmount the card and leave the function
        f_mount(0, "",0);
        Trace::out("error");
        return false;
    }

    //open sd card
    res = f_open(&fil, fileName, FA_CREATE_ALWAYS | FA_WRITE);

    //res = FR_OK, if the card has been open
    if(res!=FR_OK)
    {
        //if not, unmount the card and leave the function
        f_mount(0, "",0);
        Trace::out("error");
        return false;
    }

    return true;
}


bool SdCard::closeFile(char* fileName)
{
    //close sd_card
    f_close(&fil);

    //res = FR_OK, if the card has been closed
    if(res!=FR_OK)
    {
        //if not, unmount the card and leave the function
        f_mount(0, "",0);
        Trace::out("error");
        return false;
    }

    //unmount the sd card
    f_mount(0, "",0);

    return true;
}

bool SdCard::writeFile(const char* dataPtr, uint32_t dataLength)
{
```

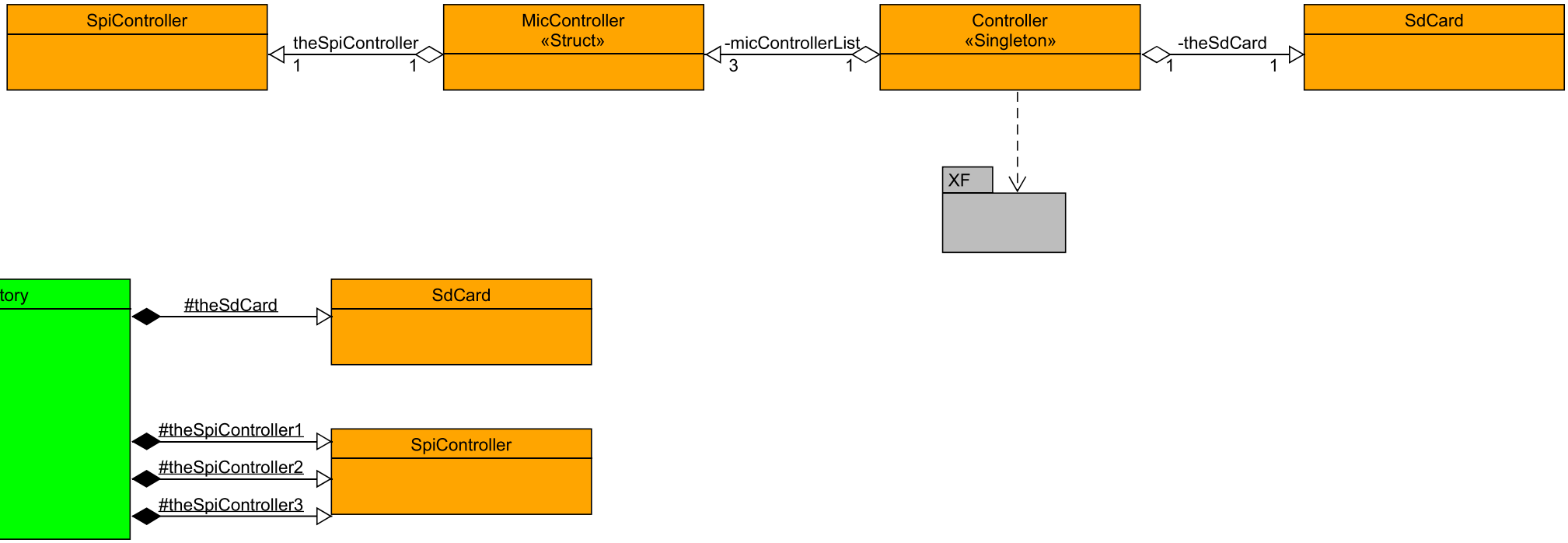
```
72      //put the pointer where we will write at the end of the file
73      res = f_lseek(&fil,f_size(&fil));
74
75      //res = FR_OK, if the pointer has been set to the end of the file
76      if(res!=FR_OK)
77      {
78          //if not, unmount the card and leave the function
79          f_mount(0, "",0);
80          Trace::out("error");
81          return false;
82      }
83
84      //write messageToWrite into sd card
85
86      res = f_write(&fil, dataPtr, dataLength, &bytesWritten);
87
88      //res = FR_OK, if the card has been writed
89      if(res!=FR_OK)
90      {
91          //if not, unmount the card and leave the function
92          f_mount(0, "",0);
93          Trace::out("error");
94          return false;
95      }
96
97      return true;
98  }
```

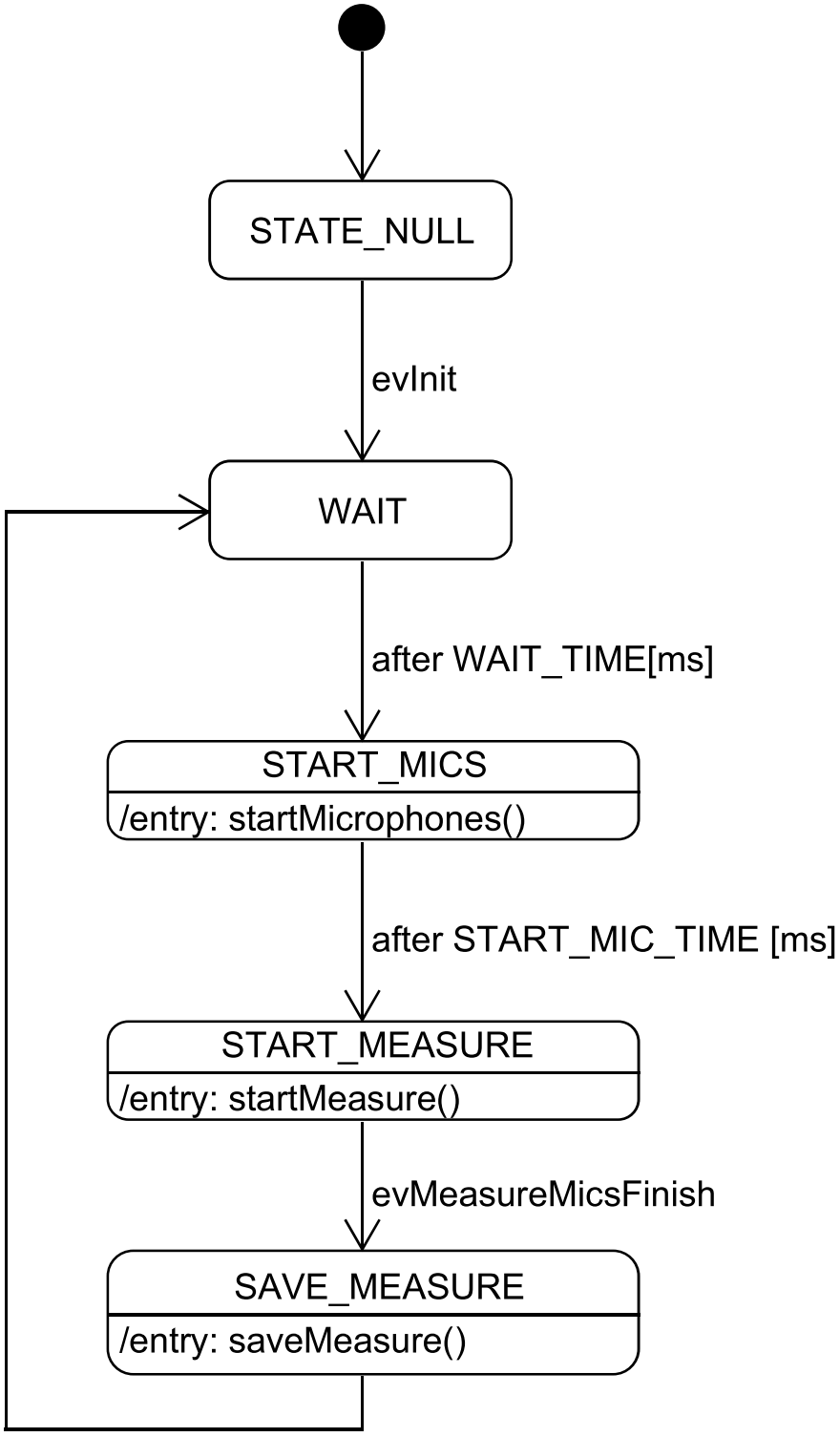
```
/*
 * SpiController.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef SPICONTROLLER_H_
#define SPICONTROLLER_H_

#include "main.h"

#define DATA_SIZE 5000 ///<@def @brief size of the data stored in memory

/**
 * @brief Manage the SPI Communcation
 */
class SpiController {

public:
        SpiController();
        virtual ~SpiController();


        /**
        *@brief initialization
        *@param theSPI is the configuration for the SPI communication
        */
        void init(SPI_HandleTypeDef* theSPI);

        void start();                       ///< @brief Start SPI in receive master only mode

        /**
        *@brief get the data pointer
        *@return the pointer to the SPI data
        */
        uint16_t* getData();            ///< @brief get the data pointer

        /**
        *@brief getter of hspi variable
        *@return theSPI communication configuration
        */
        SPI_HandleTypeDef* getHSPI();     ///< @brief getter of hspi variable

    private:
        SPI_HandleTypeDef* hspi;        ///< @brief SPI handle structure
        uint16_t* data;                   ///< @brief pointer to the data
};

#endif /* SPICONTROLLER_H_ */
```

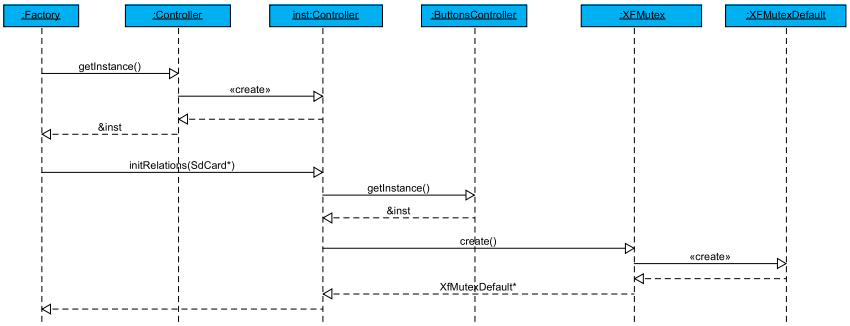
```cpp
/*
 * SpiController.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/SpiController.h"


SpiController::SpiController() {

    //create dynamical array
    this->data = new uint16_t[DATA_SIZE];

    //init dynamical array
    for(int i = 0; i<DATA_SIZE;i++)
    {
        this->data[i] = 0;
    }

    this->hspi = nullptr;
}

SpiController::~SpiController() {
    //free the dynmical memory used by data
    if(this->data)
        delete[] this->data;
}

void SpiController::init(SPI_HandleTypeDef* hspi)
{
    this->hspi = hspi;
}

void SpiController::start()
{
    //launch SPI in receive slave only mode with dma transfert
    HAL_SPI_Receive_DMA(this->hspi,(uint8_t*)this->data,DATA_SIZE);
}
uint16_t* SpiController::getData()
{
    return this->data;
}
SPI_HandleTypeDef* SpiController::getHSPI()
{
    return this->hspi;
}
```

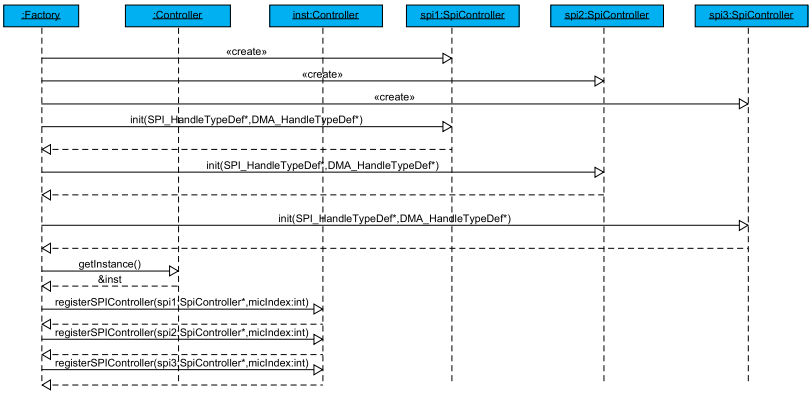
Annexe 3.2 :

Documentation UML

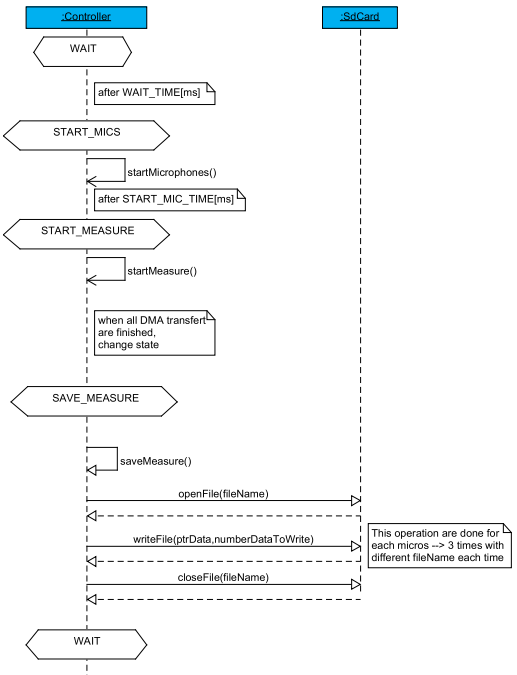

SpiController
theSpiController
1
1
MicController
«Struct»
-micControllerList
3
1
Controller
«Singleton»
-theSdCard
1
1
SdCard
XF
Factory
+build()
+initialize()
#theSdCard
SdCard
#theSpiController1
#theSpiController2
#theSpiController3
SpiController

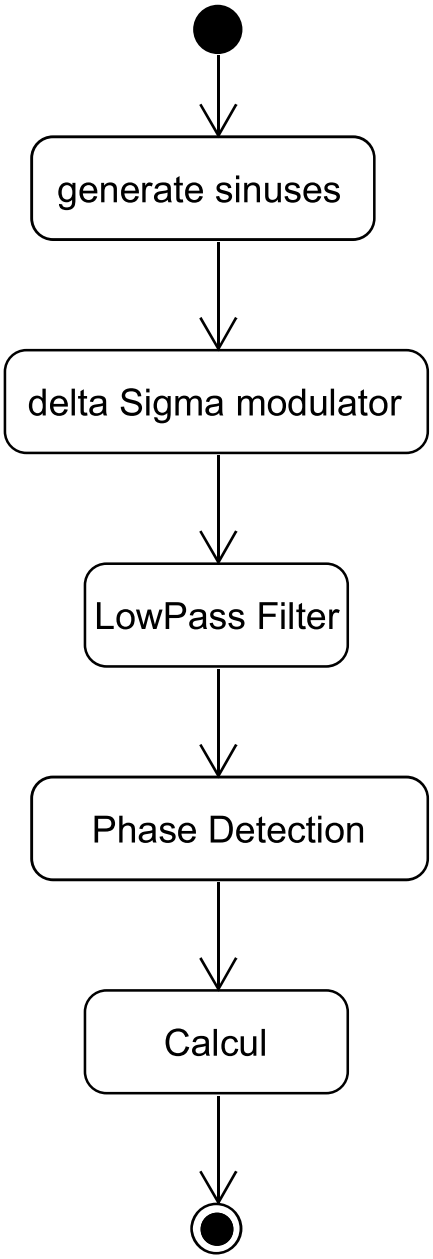

STATE_NULL
evInit
WAIT
after WAIT_TIME[ms]
START_MICS
/entry: startMicrophones()
after START_MIC_TIME [ms]
START_MEASURE
/entry: startMeasure()
evMeasureMicsFinish
SAVE_MEASURE
/entry: saveMeasure()

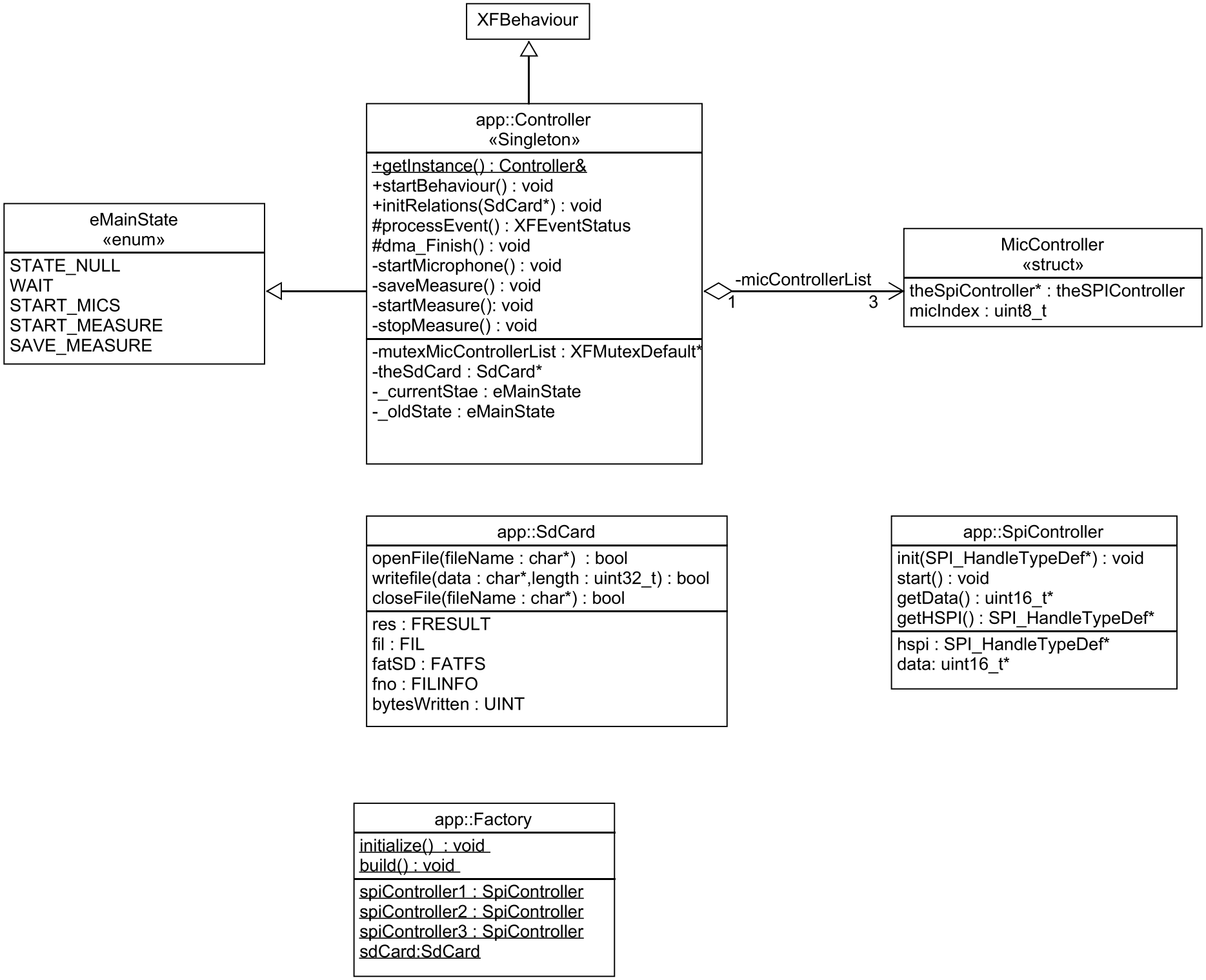


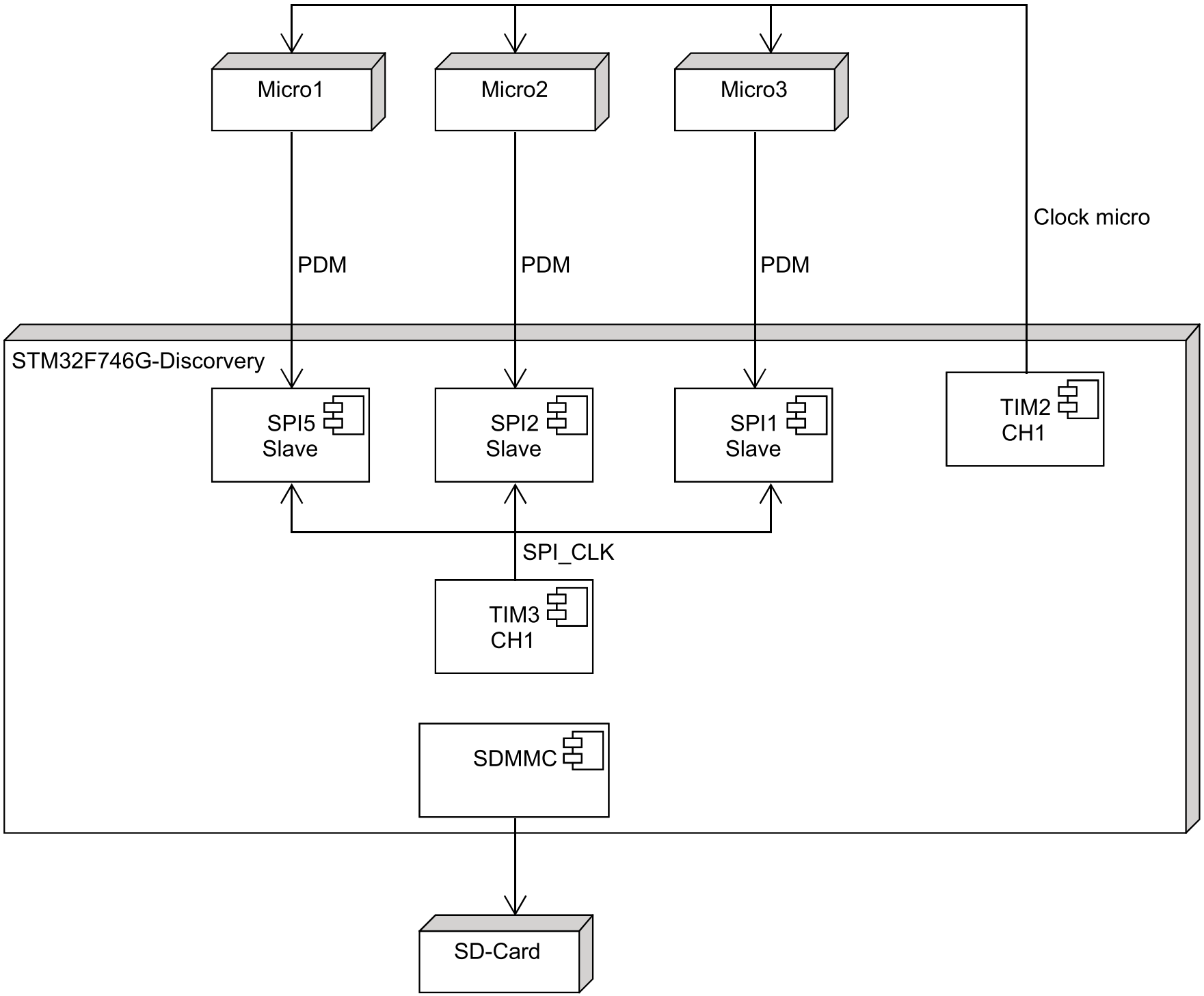


Take measure + save measure




generate sinuses
delta Sigma modulator
LowPass Filter
Phase Detection
Calcul


XFBehaviour
app::Controller
«Singleton»
+getInstance() : Controller&
+startBehaviour() : void
+initRelations(SdCard*) : void
#processEvent() : XFEventStatus
#dma_Finish() : void
-startMicrophone() : void
-saveMeasure() : void
-startMeasure(): void
-stopMeasure() : void
-mutexMicControllerList : XFMutexDefault*
-theSdCard : SdCard*
-_currentStae : eMainState
-_oldState : eMainState
eMainState
«enum»
STATE_NULL
WAIT
START_MICS
START_MEASURE
SAVE_MEASURE
-micControllerList
1
3
MicController
«struct»
theSpiController* : theSPIController
micIndex : uint8_t
app::SdCard
openFile(fileName : char*) : bool
writefile(data : char*,length : uint32_t) : bool
closeFile(fileName : char*) : bool
res : FRESULT
fil : FIL
fatSD : FATFS
fno : FILINFO
bytesWritten : UINT
app::SpiController
init(SPI_HandleTypeDef*) : void
start() : void
getData() : uint16_t*
getHSPI() : SPI_HandleTypeDef*
hspi : SPI_HandleTypeDef*
data: uint16_t*
app::Factory
initialize() : void
build() : void
spiController1 : SpiController
spiController2 : SpiController
spiController3 : SpiController
sdCard:SdCard


Micro1
Micro2
Micro3
Clock micro
PDM
PDM
PDM
STM32F746G-Discorvery
SPI5
Slave
SPI2
Slave
SPI1
Slave
TIM2
CH1
SPI_CLK
TIM3
CH1
SDMMC
SD-Card

Annexe 3.3 :

Documentation du code

SoundDirectionDetection

Generated by Doxygen 1.8.16

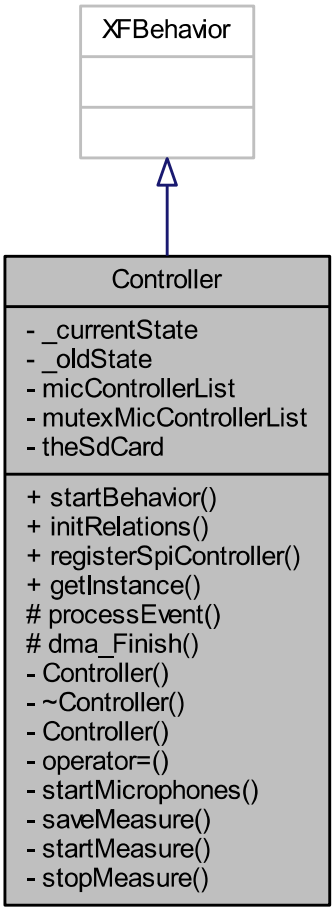
1 Class Documentation

1.1 Controller Class Reference

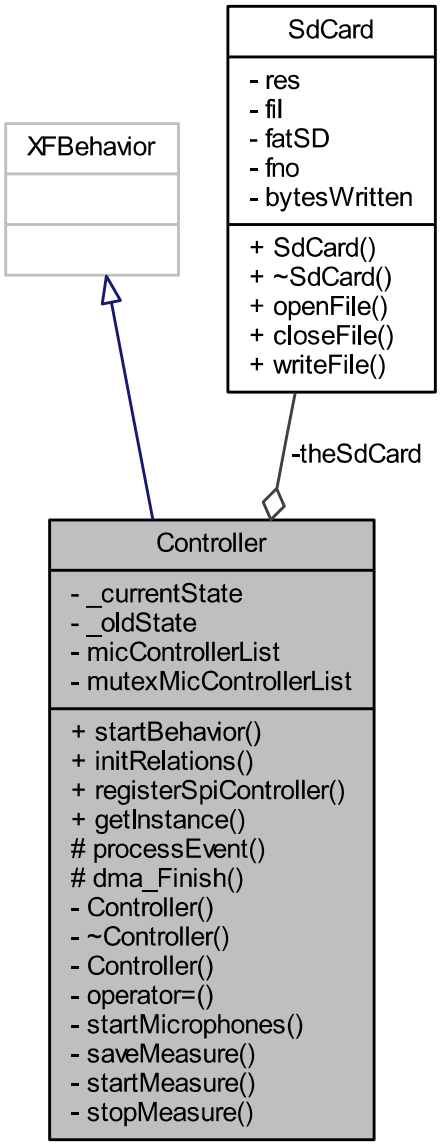
this class is a singleton it contains a state machine the state machine is started and each second the measure is started then it will take Measurement on the micro for 10 ms Then it wil save the data on a sd card and go back to the initial state

```
#include <Controller.h>
```

Inheritance diagram for Controller:



Collaboration diagram for Controller:



Classes

- struct **MicController**

  Struct who contains micIndex and theSpiController for the mic with that micIndex.

Public Member Functions

- void **startBehavior** ()

  start the state machine

- void **initRelations** (**SdCard** * **theSdCard**)

  return the only one instance of the singleton class
- void **registerSpiController** (**SpiController** *theSPIController, uint8_t micIndex)

  *register a new **SpiController*** (p. 15) *instance*

Static Public Member Functions

- static **Controller** & **getInstance** ()

  getInstance of the singleton class

Protected Member Functions

- XFEventStatus **processEvent** ()

  State machine.
- void **dma_Finish** ()

  called when one dma transfer is finish

Private Types

- enum **eMainState** {
  STATE_NULL = 0, **WAIT**, **START_MICS**, **START_MEASURE**,
  SAVE_MEASURE }

  Enumeration used to have a unique identifier for every state in the state machine.

Private Member Functions

- **Controller** ()
- ~**Controller** ()
- **Controller** (const **Controller** &)
- **Controller** & **operator=** (const **Controller** &)
- void **startMicrophones** ()

  start the microphone clock microphone take 10[ms] to startup
- void **saveMeasure** ()

  *Save measure to **SdCard*** (p. 11)*.*
- void **startMeasure** ()

  Launch a PWM to start mics measure.
- void **stopMeasure** ()

  Stop the PWM Stop mics measure.

Private Attributes

- **eMainState** **_currentState**

  Attribute indicating currently active state.
- **eMainState** **_oldState**

  Attribute indicating old state.
- std::list< **MicController** * > **micControllerList**

  list of callback
- XFMutexDefault * **mutexMicControllerList**

  mutex for the list
- **SdCard** * **theSdCard**

  the sdCardController

Friends

- void **HAL_SPI_RxCpltCallback** (SPI_HandleTypeDef *hspi)

1.1.1 Detailed Description

this class is a singleton it contains a state machine the state machine is started and each second the measure is started then it will take Measurement on the micro for 10 ms Then it wil save the data on a sd card and go back to the initial state

1.1.2 Member Enumeration Documentation

1.1.2.1 eMainState `enum Controller::eMainState [private]`

Enumeration used to have a unique identifier for every state in the state machine.

Enumerator

STATE_NULL	null state
WAIT	Wait timer.
START_MICS	start taking measure from the micros
START_MEASURE	start taking measure from the micros
SAVE_MEASURE	store measure value into SD Card

1.1.3 Constructor & Destructor Documentation

1.1.3.1 Controller() [1/2] `Controller::Controller ( ) [private]`

1.1.3.2 ~Controller() `Controller::~Controller ( ) [private]`

1.1.3.3 Controller() [2/2] `Controller::Controller ( const Controller & ) [inline], [private]`

1.1.4 Member Function Documentation

1.1.4.1 dma_Finish() `void Controller::dma_Finish ( ) [protected]`

called when one dma transfer is finish

1.1.4.2 getInstance() `Controller & Controller::getInstance ( ) [static]`

getInstance of the singleton class

Returns

the only one instance of the singleton class

1.1.4.3 initRelations() `void Controller::initRelations ( SdCard * theSdCard )`

return the only one instance of the singleton class

Parameters

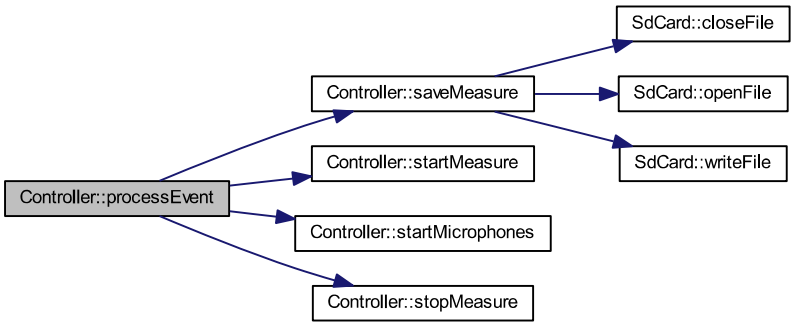
theSdCard	is a pointer tothe instance of the **SdCard** (p. 11) class

1.1.4.4 operator=() `Controller& Controller::operator= ( const Controller & ) [inline], [private]`

1.1.4.5 processEvent() `XFEventStatus Controller::processEvent ( ) [protected]`

State machine.

Here is the call graph for this function:



1.1.4.6 registerSpiController() `void Controller::registerSpiController ( SpiController * theSPIController, uint8_t micIndex )`

register a new **SpiController** (p. 15) instance

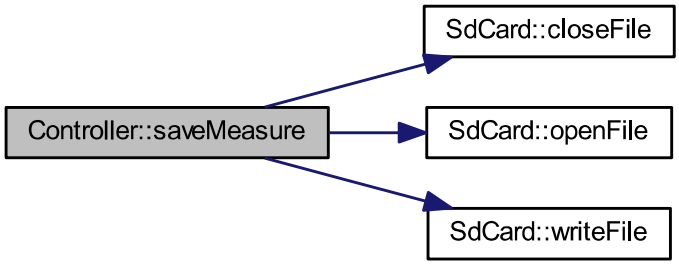
Parameters

theSPIController	is a pointer to the instance of the **SpiController** (p. 15) class
micIndex	is the micro index in the PCB

1.1.4.7 saveMeasure() `void Controller::saveMeasure ( ) [private]`

Save measure to **SdCard** (p. 11).

Here is the call graph for this function:



Here is the caller graph for this function:

1.1.4.8 startBehavior() `void Controller::startBehavior ( )`

start the state machine

1.1.4.9 startMeasure() `void Controller::startMeasure ( ) [private]`

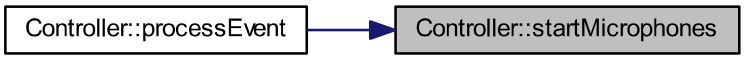
Launch a PWM to start mics measure.

Here is the caller graph for this function:

1.1.4.10 startMicrophones() `void Controller::startMicrophones ( ) [private]`

start the microphone clock microphone take 10[ms] to startup

Here is the caller graph for this function:



1.1.4.11 stopMeasure() `void Controller::stopMeasure ( ) [private]`

Stop the PWM Stop mics measure.

Here is the caller graph for this function:

1.1.5 Friends And Related Function Documentation

1.1.5.1 HAL_SPI_RxCpltCallback `void HAL_SPI_RxCpltCallback ( SPI_HandleTypeDef * hspi ) [friend]`

1.1.6 Member Data Documentation

1.1.6.1 _currentState `eMainState Controller::_currentState [private]`

Attribute indicating currently active state.

1.1.6.2 _oldState `eMainState Controller::_oldState [private]`

Attribute indicating old state.

1.1.6.3 micControllerList `std::list< MicController* > Controller::micControllerList [private]`

list of callback

1.1.6.4 mutexMicControllerList `XFMutexDefault* Controller::mutexMicControllerList [private]`

mutex for the list

1.1.6.5 theSdCard `SdCard* Controller::theSdCard [private]`

the sdCardController

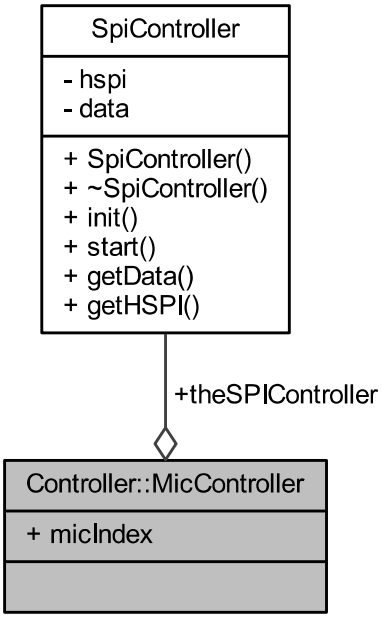
The documentation for this class was generated from the following files:

- C:/Users/rebor/Desktop/doxygen/app/ **Controller.h**
- C:/Users/rebor/Desktop/doxygen/app/ **Controller.cpp**

1.2 Controller::MicController Struct Reference

Struct who contains micIndex and theSpiController for the mic with that micIndex.

Collaboration diagram for Controller::MicController:



Public Attributes

- **SpiController** * **theSPIController**
  the SPIController who controller the mic with mic index
- uint8_t **micIndex**
  index of the mics controlled by the spi controller

1.2.1 Detailed Description

Struct who contains micIndex and theSpiController for the mic with that micIndex.

1.2.2 Member Data Documentation

1.2.2.1 micIndex `uint8_t Controller::MicController::micIndex`

index of the mics controlled by the spi controller

1.2.2.2 theSPIController `SpiController* Controller::MicController::theSPIController`

the SPIController who controller the mic with mic index

The documentation for this struct was generated from the following file:

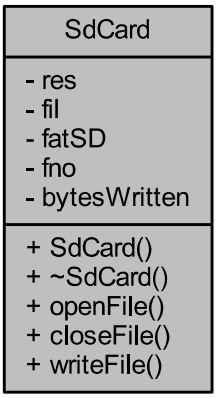
- C:/Users/rebor/Desktop/doxygen/app/ **Controller.h**

1.3 SdCard Class Reference

this class write data into SDCard

```
#include <SdCard.h>
```

Collaboration diagram for SdCard:



Public Member Functions

- **SdCard** ()
- virtual ~**SdCard** ()
- bool **openFile** (char *fileName)

  open a file in the sd card
- bool **closeFile** (char *fileName)

  close a file in the sd card
- bool **writeFile** (const char *data, uint32_t length)

  write into the text file some data

Private Attributes

- FRESULT **res**
  result of the action
- FIL **fil**
  file object structure
- FATFS **fatSD**
  file system object
- FILINFO **fno**
  file information
- UINT **bytesWritten**
  bytes written in sd card

1.3.1 Detailed Description

this class write data into SDCard

1.3.2 Constructor & Destructor Documentation

1.3.2.1 SdCard() `SdCard::SdCard ( )`

1.3.2.2 ~SdCard() `SdCard::~SdCard ( ) [virtual]`

1.3.3 Member Function Documentation

1.3.3.1 closeFile() `bool SdCard::closeFile ( char * fileName )`

close a file in the sd card

Parameters

fileName	is the text file Name

Returns

true if the file has been closed

Here is the caller graph for this function:

1.3.3.2 openFile()

```
bool SdCard::openFile (
        char * fileName )
```

open a file in the sd card

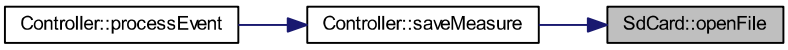
Parameters

fileName	is the text file Name

Returns

true if the file has been created and opened

Here is the caller graph for this function:



1.3.3.3 writeFile()

```
bool SdCard::writeFile (
        const char * data,
        uint32_t length )
```

write into the text file some data

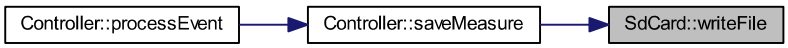
Parameters

data	is a pointer to the data to be written
length	is the length of the data to be write

Returns

true if the file has been writted correctly

Here is the caller graph for this function:



1.3.4 Member Data Documentation

1.3.4.1 bytesWritten `UINT SdCard::bytesWritten [private]`

bytes written in sd card

1.3.4.2 fatSD `FATFS SdCard::fatSD [private]`

file system object

1.3.4.3 fil `FIL SdCard::fil [private]`

file object structure

1.3.4.4 fno `FILINFO SdCard::fno [private]`

file information

1.3.4.5 res `FRESULT SdCard::res [private]`

result of the action

The documentation for this class was generated from the following files:

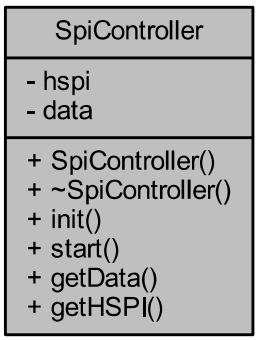
- C:/Users/rebor/Desktop/doxygen/app/ **SdCard.h**
- C:/Users/rebor/Desktop/doxygen/app/ **SdCard.cpp**

1.4 SpiController Class Reference

Manage the SPI Communcation.

`#include <SpiController.h>`

Collaboration diagram for SpiController:



Public Member Functions

- **SpiController** ()
- virtual ~**SpiController** ()
- void **init** (SPI_HandleTypeDef *theSPI)

  initialization
- void **start** ()

  Start SPI in receive master only mode.
- uint16_t * **getData** ()

  get the data pointer
- SPI_HandleTypeDef * **getHSPI** ()

  getter of hspi variable

Private Attributes

- SPI_HandleTypeDef * **hspi**

  SPI handle structure.
- uint16_t * **data**

  pointer to the data

1.4.1 Detailed Description

Manage the SPI Communcation.

1.4.2 Constructor & Destructor Documentation

1.4.2.1 SpiController() `SpiController::SpiController ( )`

1.4.2.2 ~SpiController() `SpiController::~SpiController ( ) [virtual]`

1.4.3 Member Function Documentation

1.4.3.1 getData() `uint16_t * SpiController::getData ( )`

get the data pointer

Returns

the pointer to the SPI data

get the data pointer

1.4.3.2 getHSPI() `SPI_HandleTypeDef * SpiController::getHSPI ( )`

getter of hspi variable

Returns

theSPI communication configuration

getter of hspi variable

1.4.3.3 init() `void SpiController::init ( SPI_HandleTypeDef * theSPI )`

initialization

Parameters

the↩ SPI	is the configuration for the SPI communication

1.4.3.4 start() void SpiController::start ( )

Start SPI in receive master only mode.

1.4.4 Member Data Documentation

1.4.4.1 data uint16_t* SpiController::data [private]

pointer to the data

1.4.4.2 hspi SPI_HandleTypeDef* SpiController::hspi [private]

SPI handle structure.

The documentation for this class was generated from the following files:

- C:/Users/rebor/Desktop/doxygen/app/ **SpiController.h**
- C:/Users/rebor/Desktop/doxygen/app/ **SpiController.cpp**

Annexe 4 :

Programme
V2.0

Annexe 4.1 :

Code source

```
/*
 * Controller.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef CONTROLLER_H_
#define CONTROLLER_H_

//list
#include <list>
#include <math.h>

//for sstream
#include <sstream>
#include <string.h>
#include <stdio.h>

//relations with others class
#include "app/LowPassFilter.h"
#include "app/SpiController.h"
#include "app/Point.h"
#include "app/SdCard.h"
#include "app/gui.h"

//state machine and events
#include "xf/behavior.h"
#include "event/events.h"
#include "event/evMeasureMicsFinish.h"


#define SAMPLE_PER_PERIOD 100         ///<@def @brief the number of sample per period of the sinus
#define FREQUENCY 40000                ///<@def @brief frequency of the sound
#define VP_AIR 343                    ///<@def @brief propogation speed of the sound in the air
#define BIT_NUMBER_SPI 16            ///<@def @brief number of bits of the spi communication
#define MIC_RADIUS 0.0022            ///<@def @brief The radius of the circle where the mics are placed
#define NUMBER_THEORICAL_POINT 1000    ///<@def @brief Distance of the transmitter
#define TRANSMITTER_RADIUS 3


#define CIRCLE_RADIUS 10    ///<@def @brief radius of the circle to display
#define BLACK_COLOR 0        ///<@def @brief balck color value = 0x000000
#define LINE_LENGTH 50        ///<@def @brief Length of the line to draw

#define WAIT_TIME 500            ///<@def @brief time bewteen 2 measurements
#define START_MIC_TIME 20        ///<@def @brief time to wait for the mics changed mode
#define ELEMENT_TO_IGNORE 500    ///<@def @brief Element to ignore after the digital filter. (first period of the sinus are bad)

using namespace std;
/**
 * @brief this class contains a state machine
 * the state machine is started when the users push on the start button
 * Then it will save the data on a sd card and go back to the initial state
 */
class Controller : public XFBehavior
{
    friend void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef* hspi);
    friend void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim);

public:
    /**
    * @brief return the only one instance of the singleton class
    * @return the only one instance of the Controller class
    */
    static Controller& getInstance();
    void startBehavior();                  ///< @brief start the state machine

    /**
    * @brief init relations
    * @param theLowPassFilter is a pointer to the LowPassFilter class instance
    * @param theSdCard is a pointer to the SdCard class instance
    * @param theGUI is a pointer to the Gui class instance
    */
    void initRelations(LowPassFilter* theLowPassFilter, SdCard* theSdCard,Gui* theGUI);    ///< @brief init relations

    /**
    * @brief register a new SPI communication
    * @param theSPIController is a pointer to the SpiController class instance
    * @param micIndex is the index of the microphone in the PCB
    */
    void registerSpiController(SpiController* theSPIController, uint8_t micIndex);

    /**
    * @brief calcul the theta and alpha angle
    * this two angle are needed to found the direction
    * of the others divers.
    */
    void calculTheorical();
protected:

    /**
    * @brief State machine
```

```
94      * @return XFEventStatus    return the events status : Consummed or unknow
95      */
96      XFEventStatus processEvent();
97      void dma_Finish();        ///< @brief called when dma transfert is finish
98
99  private:
100
101     Controller();
102     ~Controller();
103     Controller(const Controller&){}
104     Controller& operator=(const Controller&){}
105
106     void startMeasure();     ///< @brief Start mics measure
107     void stopMeasure();     ///< @brief Stop mics measure
108
109     /**
110     *@brief normalize a angle between -Pi and Pi
111     *@param angle : the angle to normalize
112     *@return the normalized angle
113     */
114     float normAngle(float angle);
115
116     /**
117     *@brief calculate the phases of a sinus, using IQ demodulation
118     *@param sinus : sinus to calculate the phases
119     *@param length : length of the sinus array
120     *@return the phases of the sinus value
121     */
122     float IQdemodulation(float* sinus,uint32_t length);
123
124     /**
125     *@brief calculate the mean value of a signal
126     *@param signal : signal to calculate the mean value
127     *@param indexStart : index in the signal array to start the mean value
128     *@param numberElement : numberElement of the mean value
129     *@return the mean value of the array
130     */
131     float mean(float* signal, uint32_t indexStart, uint32_t numberElement);
132
133     /**
134     *@brief calculate the moving mean of a signal
135     *@param signal : signal to calculate the moving mean
136     *@param meanSize : size of the moving mean
137     *@param length : length of the signal array
138     *@return the moving mean array pointer
139     */
140     float* movMean(float* signal, int32_t meanSize,uint32_t length);
141
142     /**
143     *@brief start the microphone clock
144     * microphone take 10[ms] to startup
145     */
146     void startMicrophones();
147
148     /**
149     *@brief draw a arrow on the LCD screen to indicate the sound direction
150     *@param angle : the sound direction
151     */
152     void drawResult(float angle);
153
154
155     void saveData(int32_t*,uint32_t,std::string);
156     void saveData(uint16_t*,uint32_t,std::string);
157     void saveData(float*,uint32_t,std::string);
158
159
160     /**
161     *@brief calcul the direction of the sound
162     * @return angle of the sound direction
163     */
164     float calculSoundDirection();
165     /**
166      * @brief Enumeration used to have a unique identifier for every
167      * state in the state machine.
168      */
169     typedef enum
170     {
171         STATE_NULL = 0,         ///< @brief null state
172         STATE_WAIT,             ///< @brief wait state
173         START_MICS,             ///< @brief start microphoen clock
174         START_MEASURE,         ///< @brief start taking measure from the micros
175         CALCUL_DIRECTION,     ///< @brief calcul direction of the sound
176     } eMainState;
177
178     eMainState _currentState;            ///< @brief Attribute indicating currently active state
179     eMainState _oldState;                ///< @brief Attribute indicating old state
180
181     /**
182      * @brief Struct who contains micIndex
183      * and theSpiController for the mic with that
184      * micIndex
185      */
186     typedef struct{
187         SpiController* theSPIController;    ///< @brief the SPIController who controller the mic with mic index
```

```
188         uint8_t micIndex;                   ///< @brief index of the mics controlled by the spi controller
189     }MicController;
190 
191     std::list<MicController*> micControllerList;     ///< @brief list of callback
192     XFMutexDefault* mutexMicControllerList;         ///< @brief mutex for the list
193     LowPassFilter* theLowPassFilter;                ///< @brief pointer to the LowPassFilter class
194     float* theta;         ///< @brief pointer to the theta array, this array contains theta angle values
195     float* alpha;         ///< @brief pointer to the alpha array, this array contains alpha angle values
196     SdCard* theSdCard;    ///< @brief pointer to the SdCard class
197     Gui* theGui;        ///< @brief pointer to the Gui class
198 
199 
200 
201 };
202 
203 #endif /* CONTROLLER_H_ */
```

```cpp
/*
 * Controller.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/Controller.h"
#include "main.h"

extern "C" TIM_HandleTypeDef htim2;
extern "C" TIM_HandleTypeDef htim3;
extern "C" TIM_HandleTypeDef htim5;

Controller::Controller() {
    // TODO Auto-generated constructor stub
    this->_currentState = STATE_NULL;
    this->_oldState = STATE_NULL;

    this->mutexMicControllerList = nullptr;

    this->theSdCard = nullptr;

    this->alpha = nullptr;
    this->theta = nullptr;
    this->theGui = nullptr;
}

Controller::~Controller() {
    // TODO Auto-generated destructor stub

    //delete mutexSpiControllerList if it exist
    if(this->mutexMicControllerList)
        delete this->mutexMicControllerList;
}

Controller& Controller::getInstance()
{
    //create a static object
    static Controller theController;
    //return the static object
    return theController;
}


void Controller::initRelations(LowPassFilter* theLowPassFilter, SdCard* theSdCard, Gui* theGui)
{
    //create mutex
    this->mutexMicControllerList = (XFMutexDefault*) interface::XFMutex::create();

    this->theSdCard = theSdCard;
    this->theLowPassFilter = theLowPassFilter;
    this->theGui = theGui;
}

void Controller::startBehavior()
{
    //start the state machine
    XFBehavior::startBehavior();

    //start the spi in receive mode
    for(std::list<MicController*>::iterator it = this->micControllerList.begin();it!=this->micControllerList.end(); it++)
    {
        (*it)->theSPIController->start();
    }
}

void Controller::registerSpiController(SpiController* theSpiController, uint8_t micIndex)
{
    bool found = false;

    //enter critical section
    this->mutexMicControllerList->lock();

    for(std::list<MicController*>::iterator it = this->micControllerList.begin();it!=this->micControllerList.end(); it++)
    {
        if((*it)->theSPIController  == theSpiController)
        {
            found = true;
            break;
        }
    }
    //if the spiController is not in the list --> add to the list
    if(!found)
    {
        this->micControllerList.push_back(new MicController({theSpiController,micIndex}));
    }

    //leave critical section
    this->mutexMicControllerList->unlock();
}

void Controller::dma_Finish()
{
    static uint8_t nMeasureFinish = 0;
    nMeasureFinish++;

    //when the 3 dma transfer are finish, generate an event
    if(nMeasureFinish ==  3 && this->_currentState == START_MEASURE)
    {
        nMeasureFinish = 0;
        GEN(evMeasureMicsFinish);
    }
    else if(this->_currentState != START_MEASURE)
    {
        nMeasureFinish = 0;
    }
}
```

```
109 void Controller::startMeasure()
110 {
111     //start spi
112     HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
113 }
114
115 void Controller::startMicrophones()
116 {
117     //launch microphone
118     HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
119 }
120
121 void Controller::stopMeasure()
122 {
123     //stop the clock of the SPI
124     HAL_TIM_PWM_Stop(&htim3,TIM_CHANNEL_1);
125     HAL_TIM_PWM_Stop(&htim2,TIM_CHANNEL_1);
126
127     //flush spi fifo
128     std::list<MicController*>::iterator it;
129
130     for( it = this->micControllerList.begin();
131             it!=this->micControllerList.end(); it++)
132     {
133         HAL_SPIEx_FlushRxFifo((*it)->theSPIController->getHSPI());
134     }
135 }
136
137
138 XFEventStatus Controller::processEvent()
139 {
140
141     float angle = 0.f;
142     eEventStatus eventStatus = XFEventStatus::Unknown;
143
144     //get current event
145     const XFEvent * event  = getCurrentEvent();
146
147     _oldState = _currentState;
148
149     switch (_currentState)
150     {
151     case STATE_NULL:
152         if(event->getEventType() == XFEvent::Initial)
153         {
154             //intial event --> go to START_MEASURE state
155             _currentState = STATE_WAIT;
156             eventStatus = XFEventStatus::Consumed;
157         }
158         break;
159     case STATE_WAIT:
160         if(event->getEventType() == XFEvent::Timeout &&
161                 event->getId() == TimeoutIds::timeoutWaitId)
162         {
163             //timeout occures--> go to start measure state
164             _currentState = START_MICS;
165             eventStatus = XFEventStatus::Consumed;
166         }
167         break;
168     case START_MICS: //start measure
169         if (event->getEventType() == XFEvent::Timeout //one measuremnent is finish
170                 && event->getId() == TimeoutIds::timeoutStartMeasureId)
171         {
172             _currentState = START_MEASURE;
173             eventStatus = XFEventStatus::Consumed;
174         }
175         break;
176     case START_MEASURE: //start measure
177         if (event->getEventType() == XFEvent::Event //one measuremnent is finish
178                 && event->getId() == EventIds::evMeasureMicsFinishId)
179         {
180             _currentState = CALCUL_DIRECTION;
181             eventStatus = XFEventStatus::Consumed;
182         }
183         break;
184     case CALCUL_DIRECTION: //calcul direction of the sound
185         if(event->getEventType() == XFEvent::NullTransition)
186         {
187             //go back to WAIT_BUTTON_PRESSED state
188             _currentState = STATE_WAIT;
189             eventStatus = XFEventStatus::Consumed;
190         }
191         break;
192     default:
193         break;
194     }
195
196     if(_oldState != _currentState)
197     {
198         switch(_currentState){
199         case STATE_WAIT:
200             if(this->_oldState == STATE_NULL || this->_oldState == CALCUL_DIRECTION)
201             {
202                 scheduleTimeout(timeoutWaitId,WAIT_TIME);
203             }
204             break;
205         case START_MICS:
206             if(this->_oldState == STATE_WAIT || this->_oldState == CALCUL_DIRECTION)
207             {
208                 this->startMicrophones();
209                 scheduleTimeout(timeoutStartMeasureId,START_MIC_TIME);
210             }
211             break;
212         case START_MEASURE:
213             if(this->_oldState == START_MICS)
214             {
215                 this->startMeasure();
216             }
217             break;
```

```
218         case CALCUL_DIRECTION:
219             if(this->_oldState == START_MEASURE)
220             {
221                 this->stopMeasure();
222                 //Trace::out("calcul direction");
223                 angle = this->calculSoundDirection();
224                 this->drawResult(this->normAngle(angle+M_PI));
225                 Trace::out("%4d\n",(int32_t)(angle * 1000));
226 
227                 GEN(XFNullTransition);
228             }
229             break;
230         default:
231             break;
232         }
233     }
234     return eventStatus;
235 }
236 void Controller::calculTheorical()
237 {
238     float k = 2*M_PI*FREQUENCY/VP_AIR;
239 
240     Point m1;
241     Point m2;
242     Point m3;
243     Point* t = nullptr;
244 
245     float* phi1 = nullptr;
246     float* phi2 = nullptr;
247     float* phi3 = nullptr;
248 
249 
250     float* phi12 = nullptr;
251     float* phi13 = nullptr;
252 
253     float temp = 0.0;
254 
255     //set coordonate of microphones
256     m1.setX(MIC_RADIUS);
257     m1.setY(0);
258 
259     m2.setX(-MIC_RADIUS/2);
260     m2.setY(sqrt(3) * MIC_RADIUS/2);
261 
262     m3.setX(-MIC_RADIUS/2);
263     m3.setY(-sqrt(3) * MIC_RADIUS/2);
264 
265     this->theta = new float[NUMBER_THEORICAL_POINT];
266 
267     for(int i = 0; i<NUMBER_THEORICAL_POINT;i++)
268     {
269         this->theta[i] = 2.f * i * M_PI/ NUMBER_THEORICAL_POINT;
270     }
271 
272     t = new Point[NUMBER_THEORICAL_POINT];
273 
274     //set postion of transmitter
275     for(int i =0;i<NUMBER_THEORICAL_POINT;i++)
276     {
277         t[i].setX(cos(theta[i]) * TRANSMITTER_RADIUS);
278         t[i].setY(sin(theta[i]) * TRANSMITTER_RADIUS);
279     }
280 
281     phi1 = new float[NUMBER_THEORICAL_POINT];
282     phi2 = new float[NUMBER_THEORICAL_POINT];
283     phi3 = new float[NUMBER_THEORICAL_POINT];
284 
285     //calculate phases of the 3 microphones
286     for(int i=0;i<NUMBER_THEORICAL_POINT;i++)
287     {
288         phi1[i] = normAngle(k * m1.dist(&t[i]));
289         phi2[i] = normAngle(k * m2.dist(&t[i]));
290         phi3[i] = normAngle(k * m3.dist(&t[i]));
291     }
292 
293     //free dynamical memory
294     delete[] t;
295 
296     phi12 = new float[NUMBER_THEORICAL_POINT];
297     phi13 = new float[NUMBER_THEORICAL_POINT];
298 
299     //calculate phase shift
300     for(int i=0;i<NUMBER_THEORICAL_POINT;i++)
301     {
302         phi12[i] = phi2[i] - phi1[i];
303         phi13[i] = phi3[i] - phi1[i];
304     }
305     //free dynamical memory
306     delete[] phi1;
307     delete[] phi2;
308     delete[] phi3;
309 
310     this->alpha = new float[NUMBER_THEORICAL_POINT];
311 
312     for(int i=0; i<NUMBER_THEORICAL_POINT;i++)
313     {
314         alpha[i] = atan2(phi13[i],phi12[i]);
315     }
316 
317     //sort alpha array. Bulbble sort is used
318     for(int i = 0; i<NUMBER_THEORICAL_POINT - 1;i++)
319     {
320         for(int j = 0; j<NUMBER_THEORICAL_POINT-i -1; j++)
321         {
322             if(alpha[j] > alpha[j+1])
323             {
324                 //switch both element
325                 temp = alpha[j];
326                 alpha[j] = alpha[j+1];
```

```
                alpha[j+1] = temp;

                //switch both element
                temp = theta[j];
                theta[j] = theta[j+1];
                theta[j+1] = temp;

            }
        }
    }
    delete[] phi12;
    delete[] phi13;
}

float Controller::normAngle(float angle)
{
    while(angle < -M_PI)
        angle += 2 * M_PI;

    while(angle > M_PI)
        angle -= 2* M_PI;

    return angle;
}
float Controller::calculSoundDirection()
{
    int index = 0;

    //sinus after low pass filter
    int32_t* sinus[this->micControllerList.size()];
    float* sinusFloat[this->micControllerList.size()];

    //iterator for the MicController list
    std::list<MicController*>::iterator it;

    static int indexTest = 0;

    float phi[this->micControllerList.size()];


    float phi12 = 0.f;
    float phi13 = 0.f;
    float alphaMeas = 0.f;
    float result = 0.f;

    std::string fileName;
    //create a array of pointer to the dynamical memory
    for(uint8_t i = 0; i < this->micControllerList.size();i++)
    {
        sinus[i] = new int32_t[DATA_SIZE * BIT_NUMBER_SPI];

        phi[i] = 0.f;

        //init dynamics array
        for(uint16_t y = 0; y<DATA_SIZE * BIT_NUMBER_SPI;y++)
        {
            sinus[i][y] = 0;
        }
    }
    int32_t* data;

    //=========================================
    //           filter PDM to get a sinus
    //=========================================
    for(it = this->micControllerList.begin();
            it!=this->micControllerList.end(); it++)
    {
        data = new int32_t[DATA_SIZE * BIT_NUMBER_SPI];

        //create PDM.
        for(uint16_t y = 0; y < DATA_SIZE; y++)
        {
            for(uint8_t z = 0; z<BIT_NUMBER_SPI;z++)
            {
                //spiController->data is a 8 bits ptr, we need a array of byte --> create it
                //put msb first in the array
                data[BIT_NUMBER_SPI*y + z] = 20000 * ((((*it)->theSPIController->getData()[y])  & (1<<(BIT_NUMBER_SPI-1-z))) >> (BIT_NUMBER_SPI-1-z));
            }
        }


        this->theLowPassFilter->filter(data,sinus[index],DATA_SIZE * BIT_NUMBER_SPI);

        index++;
        delete[] data;
    }


    for(int i=0; i<this->micControllerList.size();i++)
    {
        sinusFloat[i] = new float[DATA_SIZE * BIT_NUMBER_SPI];

        for(int y = 0; y<DATA_SIZE * BIT_NUMBER_SPI;y++)
        {
            sinusFloat[i][y] = (1.f/20000) * sinus[i][y];
        }
        delete[] sinus[i];
    }

    //=========================================
    //     Phase detection with IQ demodulation
    //=========================================

    for(int i=0; i<this->micControllerList.size();i++)
    {
        phi[i] = this->IQdemodulation(&sinusFloat[i][ELEMENT_TO_IGNORE],(DATA_SIZE * BIT_NUMBER_SPI) - ELEMENT_TO_IGNORE);
        phi[i] = this->normAngle(phi[i]);
        //phi[i] = this->IQdemodulation(sinusFloat[i],DATA_SIZE * BIT_NUMBER_SPI);
    }
```

```
436
437     // free dynamical memory
438     for(uint8_t i = 0; i<this->micControllerList.size();i++)
439     {
440         delete[] sinusFloat[i];
441     }
442
443     phi12 = normAngle(phi[1] - phi[0]);
444     phi13 = normAngle(phi[2] - phi[0]);
445
446     //========================================
447     //         calculate sound direction
448     //========================================
449     alphaMeas = atan2(-phi12,-phi13);
450 //    Trace::out("phi12 : %f", phi12);
451 //    Trace::out("phi13 : %f", phi13);
452
453
454     for(int i =0;i<NUMBER_THEORICAL_POINT;i++)
455     {
456         if(alphaMeas<alpha[i])
457         {
458             result = theta[i];
459             break;
460         }
461
462     }
463
464     this->normAngle(result);
465     return result;
466
467 }
468
469 float Controller::IQdemodulation(float* sinus,uint32_t length)
470 {
471     float* phi = new float[length];
472     float* I = new float[length];
473     float* Q = new float[length];
474     float* Imean = nullptr;
475     float* Qmean = nullptr;
476     float aMean = 0.f;
477     float bMean = 0.f;
478
479     float t = 0;
480     float result = 0;
481     float meanSinValue = 0;
482
483     float* a = nullptr;
484     float* b = nullptr;
485
486     meanSinValue = mean(sinus,0,length);
487     for(uint32_t i = 0; i<length;i++)
488     {
489         t = 1.0/(FREQUENCY * SAMPLE_PER_PERIOD) * i;
490         //calculate I and Q composant
491         I[i] = sin(2 * M_PI * FREQUENCY * t) * (sinus[i] - meanSinValue);
492         Q[i] = cos(2 * M_PI * FREQUENCY * t) * (sinus[i] - meanSinValue);
493     }
494
495     //perform an average at 80[khz]
496     Imean = this->movMean(I,50,length);
497     Qmean = this->movMean(Q,50,length);
498
499     a = new float[length];
500     b = new float[length];
501
502     //get phases
503     for(int i=0;i<length;i++)
504     {
505         a[i] = cos(atan2(Qmean[i],Imean[i]));
506         b[i] = sin(atan2(Qmean[i],Imean[i]));
507     }
508
509     aMean = mean(a,0,length);
510     bMean = mean(b,0,length);
511
512     result = atan2(bMean,aMean);
513
514     //free dynmaical memory
515     delete[] I;
516     delete[] Q;
517     delete[] Imean;
518     delete[] Qmean;
519     delete[] phi;
520     delete[] a;
521     delete[] b;
522
523     return result;
524 }
525
526 float* Controller::movMean(float* signal, int32_t meanSize,uint32_t length)
527 {
528     float* meanValue = new float[length];
529     for(int32_t i = 0; i<length;i++)
530     {
531         //start of the mean
532         if(i - meanSize/2 < 0)
533         {
534             meanValue[i] = this->mean(signal, 0, meanSize/2 + i);
535         }
536         //end of the mean
537         else if( i + (meanSize/2-1) >= length)
538         {
539             meanValue[i] = this->mean(signal,i-meanSize/2, length-(i-meanSize/2));
540         }
541         //middle of the mean
542         else
543         {
544             meanValue[i] = this->mean(signal,i-meanSize/2,meanSize);
```

```
        }
    }
    return meanValue;
}
float Controller::mean(float* signal, uint32_t indexStart, uint32_t numberElement)
{
    float result = 0;

    //caculate the mean
    for(int i=0; i<numberElement;i++)
    {
        result += signal[indexStart + i];
    }

    return result/numberElement;
}

void Controller::drawResult(float angle)
{

    this->theGui->clearDisplay();
    this->theGui->drawLine(LCD_WIDTH/2,LCD_HEIGHT/2,LCD_WIDTH/2 + LINE_LENGTH*cos(angle),
            LCD_HEIGHT/2 + LINE_LENGTH*sin(angle),BLACK_COLOR);
    this->theGui->drawCircle(LCD_WIDTH/2,LCD_HEIGHT/2,CIRCLE_RADIUS,BLACK_COLOR);
}

void Controller::saveData(int32_t* dataToSave,uint32_t size,std::string fileName)
{

    //stingsstream : contains the message to write in sd card
    stringstream messageToWrite;

    //open file
    this->theSdCard->openFile(fileName.c_str()) ;

    for(uint16_t i = 0; i < size ; i++)
    {
        messageToWrite << to_string(dataToSave[i]) << "\n";
    }

    this->theSdCard->writeFile(messageToWrite.str().c_str(), messageToWrite.str().size());


    messageToWrite=std::stringstream();
    messageToWrite.str(std::string());

    //close file
    this->theSdCard->closeFile(fileName.c_str());
}
void Controller::saveData(uint16_t* dataToSave,uint32_t size,std::string fileName)
{

    //stingsstream : contains the message to write in sd card
    stringstream messageToWrite;

    //open file
    this->theSdCard->openFile(fileName.c_str()) ;

    //The microphone take 10 ms to start measurement(change mode time)
    //the clock is at 4[MHz] --> 40kbits during 10[ms] --> 2500 * 16 bits
    //dont take the first 3500 to take a little margin
    for(uint16_t i = 0; i < size ; i++)
    {
        messageToWrite << to_string(dataToSave[i]) << "\n";
    }

    this->theSdCard->writeFile(messageToWrite.str().c_str(), messageToWrite.str().size());


    messageToWrite=std::stringstream();
    messageToWrite.str(std::string());

    //close file
    this->theSdCard->closeFile(fileName.c_str());
}
void Controller::saveData(float* dataToSave,uint32_t size,std::string fileName)
{

    //stingsstream : contains the message to write in sd card
    stringstream messageToWrite;

    //open file
    this->theSdCard->openFile(fileName.c_str()) ;

    //The microphone take 10 ms to start measurement(change mode time)
    //the clock is at 4[MHz] --> 40kbits during 10[ms] --> 2500 * 16 bits
    //dont take the first 3500 to take a little margin
    for(uint16_t i = 0; i < size ; i++)
    {
        messageToWrite << to_string((int32_t)floor(100000.f * dataToSave[i])) << "\n";

    }

    this->theSdCard->writeFile(messageToWrite.str().c_str(), messageToWrite.str().size());


    messageToWrite=std::stringstream();
    messageToWrite.str(std::string());

    //close file
    this->theSdCard->closeFile(fileName.c_str());
}
```

```
#ifndef FACTORY_H
#define FACTORY_H

//
// What is seen only by the C++ compiler
//
#ifdef __cplusplus

// for the list
#include <list>

//class to init
#include "app/Controller.h"
#include "app/SpiController.h"
#include "app/LowPassFilter.h"
#include "app/SdCard.h"
#include "app/gui.h"

extern "C" SPI_HandleTypeDef hspi1;
extern "C" SPI_HandleTypeDef hspi2;
extern "C" SPI_HandleTypeDef hspi5;

extern "C" DMA_HandleTypeDef hdma_spi2_rx;
extern "C" DMA_HandleTypeDef hdma_spi3_rx;
extern "C" DMA_HandleTypeDef hdma_spi5_rx;

/**
 * @brief This is the factory. The factory
 * creates the object of class and build
 * the relations between the class
 */
class Factory
{
public:
    Factory();

    static void initialize();                  ///< @brief Initializes the factory
    static void build();                      ///< @brief Creates components and initializes relations

protected:
    static SpiController spiController1;    ///< @brief SpiController for the micro number 2
    static SpiController spiController2;    ///< @brief SpiController for the micro number 3
    static SpiController spiController3;    ///< @brief SpiController for the micro number 1
    static LowPassFilter lowPass;///<@brief LowPassFiler instance
    static SdCard theSdCard;    ///< @brief SdCard instance
    static Gui theGui;            ///< @brief Gui instance
};

#endif // __cplusplus

//
// What is seen by the C and C++ compiler
//
#ifdef __cplusplus
extern "C" {
#endif // __cplusplus

void Factory_initialize();
void Factory_build();

#ifdef __cplusplus
}
#endif // __cplusplus

#endif // FACTORY_H
```

```cpp
#include "trace/trace.h"
#include "factory.h"
#include "app/SdCard.h"


SpiController Factory::spiController1;
SpiController Factory::spiController2;
SpiController Factory::spiController3;
LowPassFilter Factory::lowPass;
SdCard Factory::theSdCard;
Gui Factory::theGui;

Factory::Factory()
{

}

void Factory::initialize()
{
    //init all spiController
    spiController1.init(&hspi1);
    spiController2.init(&hspi2);
    spiController3.init(&hspi5);
    theGui.initialize();
}

void Factory::build()
{
    //init relations
    Controller::getInstance().initRelations(&lowPass,&theSdCard,&theGui);

    //register all SPI into Controller class
    Controller::getInstance().registerSpiController(&spiController3,1);
    Controller::getInstance().registerSpiController(&spiController2,2);
    Controller::getInstance().registerSpiController(&spiController1,3);


    lowPass.initRelations();

    //coeff of the first stage
    lowPass.addStage(new LowPassFilter::Coeff_2Order({1,-0.9761,0,0.0014,0.0014,0}));
    //coeff of the second stage
    lowPass.addStage(new LowPassFilter::Coeff_2Order({1,-1.9705,0.9728,1,-1.9877,1}));
    //coeff of the third stage
    lowPass.addStage(new LowPassFilter::Coeff_2Order({1,-1.9898 ,0.9938,1,-1.9938,1}));

    Controller::getInstance().calculTheorical();
    //launch state machine
    Controller::getInstance().startBehavior();



    //theGui.setResultText("hello");


}


void Factory_initialize()
{
    Factory::initialize();
}

void Factory_build()
{
    Factory::build();
}
```

```cpp
#ifndef APP_GUI_H
#define APP_GUI_H

#include "ugfx/gfx.h"

#define LCD_WIDTH 480       ///<@def @brief LCD screen WIDTH
#define LCD_HEIGHT 272      ///<@def @brief LCD screen height
/**
 * @brief The Gui (Graphical User Interface) class display line and circle on the LCD screen.
 */
class Gui
{
public:
    Gui();


    void initialize(); ///< @brief initialize Gui class
    /**
    * @brief draw a line on the LCD screen
    * @param x0 x value of the first point of the line
    * @param y0 y value of the first point of the line
    * @param x1 x value of the second point of the line
    * @param y1 y value of the second point of the line
    * @param color color of the line
    */
    void drawLine(int16_t x0,int16_t y0,int16_t x1,int16_t y1,uint16_t color);

    /**
    * @brief draw a filled circle on the LCD screen
    * @param x0 x value of the center of the circle
    * @param y0 y value of the center of the circle
    * @param radius radius value of the circle
    * @param color color of the circle
    */

    void drawCircle(int16_t x0, int16_t y0, int16_t radius, uint16_t color);

    void clearDisplay(); ///< @brief clear the LCD screen

};

#endif // APP_GUI_H
```

```cpp
#include <string.h>
#include <assert.h>
#include "trace/trace.h"
#include "ui-gen/resources_manager.h"
#include "ui-gen/gui.h"
#include "gui.h"

Gui::Gui()
{

}

void Gui::initialize()
{
    // Initialize the uGFX library
    gfxInit();

    // Initialize the display
    guiInit();
    guiShowPage(DP_SOUND_DETECTION);

    // Some custom drawing options
    {
        // Set title background
        gwinSetBgColor(ghResultField, RGB2COLOR(12, 12, 12));
        gwinClear(ghResultField);
    }
}

void Gui::drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
        uint16_t color)
{
    gdispDrawLine(x0, y0, x1, y1,color);
}
void Gui::drawCircle(int16_t x0, int16_t y0, int16_t radius, uint16_t color)
{
    gdispFillCircle(x0,y0,radius,color);
}

void Gui::clearDisplay()
{
    gwinClear(ghPageContainerDp_soundDetection);
}
```

```cpp
/*
 * Point.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef Point_H_
#define Point_H_

#include "math.h"
/**
 * @brief this class represents a point with x and y
 * coordonate. Allows to calculate distance bewteen 2 points
 */
class Point {
    public:
        Point();
        virtual ~Point();

        /**
        * @brief calculate the distance between two points
        * @param secondPoint the 2nd point to calculate the distance
        * @return the distance between the 2 points
        */
        float dist(Point* secondPoint);

        /**
        * @brief setter of x point value
        * @param x new x value
        */
        void setX(float x);

        /**
        * @brief setter of y point value
        * @param y new y value
        */
        void setY(float y);

        /**
        * @brief getter of x point value
        * @return x point value
        */
        float getX();

        /**
        * @brief getter of y point value
        * @return y point value
        */
        float getY();
    private:
        float x;  ///< @brief x point value
        float y;  ///< @brief y point value

};


#endif /* Point_H_ */
```

```cpp
/*
 * Point.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/Point.h"


Point::Point()
{
    this->x = 0;
    this->y = 0;
}
Point::~Point()
{

}
float Point::dist(Point* secondPoint)
{
    return hypot((this->getX() - secondPoint->getX()),
        (this->getY() - secondPoint->getY()));
}

float Point::getX()
{
    return this->x;
}
float Point::getY()
{
    return this->y;
}

void Point::setX(float x)
{
    this->x = x;
}
void Point::setY(float y)
{
    this->y = y;
}
```

```
/*
 * LowPassFilter.h
 *
 *  Created on: 12 juin 2020
 *      Author: remyb
 */

#ifndef LOWPASSFILTER_H_
#define LOWPASSFILTER_H_

#include <list>
#include <stdint.h>

#include "xf/include/xf/behavior.h"

/**
 * @brief this class implements a low pass filter
 * the number of stage is choosen by the users
 * each stage is a second order IIR filter
 */
class LowPassFilter
{
public:
    LowPassFilter();
    virtual ~LowPassFilter();

    void initRelations(); ///< @brief initialize relations

    /**
     *@bried 2 order coeff of the filter
     */
    typedef struct
    {
        double a0;
        double a1;
        double a2;
        double b0;
        double b1;
        double b2;
    } Coeff_2Order;

    /**
    * @brief add a second order stage to the filter
    * @param coeff coefficient of the second order stage filter
    */
    void addStage(Coeff_2Order* coeff);

    /**
    * @bried execute the filter
    * @param input : input signal to be filtred, pointer to an array
    * @param output : output of the filter ,pointer to an array
    * @param size : size of the input array
    */
    void filter(int32_t* input,int32_t* output,uint16_t size);    ///< @brief execute the filter
private :

    /**
    * @brief one stage of the filter
    * @param coeff : Coefficient of the stage
    * @param old0 : oldValue of the digital filter
    * @param old0m1 : oldValue of the digital filter
    * @param Input : pointer to the the array that must be filtered
    * @param Output : pointer to the the array that has been filtered
    */
    void iir_biquad_SecondOrder_dir_s(Coeff_2Order Coeff, int32_t *old0, int32_t *old0m1,int32_t *Input, int32_t *Output);

    std::list<Coeff_2Order*>  coeffList;///< @brief lsit of all stage coefficient
    XFMutexDefault* mutexCoeffList;     ///< @brief mutex for the list
};

#endif /* LOWPASSFILTER_H_ */
```

```cpp
/*
 * LowPassFilter.cpp
 *
 *  Created on: 12 juin 2020
 *      Author: remyb
 */
#include "trace/trace.h"
#include <app/LowPassFilter.h>


LowPassFilter::LowPassFilter() {
    this->mutexCoeffList = nullptr;

}

LowPassFilter::~LowPassFilter() {
    //enter critical section
    this->mutexCoeffList->lock();

    std::list<Coeff_2Order*>::iterator it;

    //free dynmical memory and erase the list
    for(it = this->coeffList.begin();
            it!=this->coeffList.end();)
    {
        //delete dynmical object
        delete (*it);

        it++;

        //erase the previous element
        this->coeffList.erase(std::prev(it,1));
    }

    this->mutexCoeffList->unlock();

    //delete mutexSpiControllerList if it exist
    if(this->mutexCoeffList)
        delete this->mutexCoeffList;
}

void LowPassFilter::initRelations()
{
    //create mutex
    this->mutexCoeffList =
            (XFMutexDefault*) interface::XFMutex::create();
}
void LowPassFilter::addStage(Coeff_2Order* stageToAdd)
{
    //enter critical section
    this->mutexCoeffList->lock();

    this->coeffList.push_back(stageToAdd);

    //leave critical section
    this->mutexCoeffList->unlock();
}

void LowPassFilter::filter(int32_t* input, int32_t* output, uint16_t size)
{

    //first partial equation value stage 1
    int32_t oldo_s1 = 0;
    //second partial equation value stage 1
    int32_t oldom1_s1 = 0;
    //first partial equation value stage 2
    int32_t oldo_s2 = 0;
    //second partial equation value stage 2
    int32_t oldom1_s2 = 0;
    //first partial equation value stage 2
    int32_t oldo_s3 = 0;
    //second partial equation value stage 2
    int32_t oldom1_s3 = 0;
```

```
74
75      int i = 0;
76
77      //enter critical section
78      this->mutexCoeffList->lock();
79
80      int32_t* temp[coeffList.size()-1];
81
82      for(i = 0; i< coeffList.size() - 1; i++)
83      {
84          temp[i] = new int32_t[size];
85      }
86
87      i = 0;
88
89      std::list<Coeff_2Order*>::iterator it;
90
91      for(uint16_t index = 0; index <  size; index++)
92      {
93          //go through all the stage
94          for(it = this->coeffList.begin();it!=this->coeffList.end(); it++)
95          {
96              switch(i){
97              case 0://first stage
98                  iir_biquad_SecondOrder_dir_s(**it, &oldo_s1, &oldom1_s1,
99                          &input[index], temp[i]);
100                 break;
101             case 1://second stage
102                 iir_biquad_SecondOrder_dir_s(**it, &oldo_s2, &oldom1_s2,
103                         temp[i-1], temp[i]);
104                 break;
105             case 2://third stage
106                 iir_biquad_SecondOrder_dir_s(**it, &oldo_s3, &oldom1_s3,
107                         temp[i-1], &output[index]);
108                 break;
109             }
110             i++;
111         }
112         i = 0;
113     }
114
115     //leave critical section
116     this->mutexCoeffList->unlock();
117
118     for(i = 0; i< coeffList.size()-1; i++)
119     {
120         delete[] temp[i];
121     }
122 }
123
124 void LowPassFilter::iir_biquad_SecondOrder_dir_s(Coeff_2Order Coeff,
125         int32_t *oldO, int32_t *oldOm1,int32_t *Input, int32_t *Output)
126 /* Second order IIR filter : Canonical Direct form II (D-N) */
127
128 /*
129           -     b0         -a0
130      --->¦+¦---->-----+----<---->¦+¦--->
131          ¦            ^           ¦
132          ¦            ¦           ^
133          ¦           ---          ¦
134          ¦         ¦ -1¦          ¦
135          v         ¦z  ¦          ¦
136          ¦          ---          ¦
137          ¦     b1    ^   -a1     ¦
138          + ---->-----+----<----- +
139          ¦            ^           ¦
140          ¦           ---          ¦
141          ¦         ¦ -1¦          ¦
142          ¦         ¦z  ¦          ¦
143          ¦          ---          ¦
144          ¦           ^           ¦
145           ----->-----+-----<-----
146              b2          -a2
```

```
147
148     */
149
150   {
151       //new partial value 1
152       register int32_t newO;
153       //new partial value 2
154       register int32_t newOm1;
155
156       //calculate the output value
157       *Output = (int32_t)(*Input * Coeff.b0) + (int32_t)(*oldOm1);
158
159       //calculate the new partial value
160       newO = (int32_t)(*Input * Coeff.b2) -
161               (int32_t)(*Output *Coeff.a2);
162
163       newOm1 = (int32_t)(*Input * Coeff.b1) -
164               (int32_t)(*Output *Coeff.a1) + (int32_t)(*oldO);
165
166       //shift in time the value
167       *oldO = newO;
168       *oldOm1 = newOm1;
169   }
170
```

```
/*
 * SdCard.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef SDCARD_H_
#define SDCARD_H_

#include <string.h>

//for debug
#include "trace/trace.h"

//for sd card
#include "stm32f7xx_hal_sd.h"
#include "inc/fatfs.h"
#include "ff.h"


/**
 * @brief this class write data into SDCard
 */
class SdCard {
    public:
        SdCard();
        virtual ~SdCard();
        /**
        *@brief open a file in the sd card
        *@param fileName is the text file Name
        *@return true if the file has been created and opened
        */
        bool openFile(const char* fileName);
        /**
        *@brief close a file in the sd card
        *@param fileName is the text file Name
        *@return true if the file has been closed
        */
        bool closeFile(const char* fileName);
        /**
        *@brief write into the text file some data
        *@param data is a pointer to the data to be written
        *@param length is the length of the data to be write
        *@return true if the file has been writted correctly
        */
        bool writeFile(const char* data,uint32_t length);
    private:
        //variable for sd card write
        FRESULT res;            ///< @brief result of the action
        FIL fil;                ///< @brief file object structure
        FATFS fatSD;            ///< @brief file system object
        FILINFO fno;            ///< @brief file  information
        UINT bytesWritten;        ///< @brief bytes written in sd card
};


#endif /* SDCARD_H_ */
```

```cpp
/*
 * SdCard.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/SdCard.h"


SdCard::SdCard() {
    // TODO Auto-generated constructor stub

}

SdCard::~SdCard() {
    // TODO Auto-generated destructor stub
}

bool SdCard::openFile(const char* fileName)
{
    //mount sd card
    res = f_mount(&fatSD,"",1);

    //res = FR_OK, if the card has been mounted
    if(res!=FR_OK)
    {
        //if not, unmount the card and leave the function
        f_mount(0, "",0);
        Trace::out("error");
        return false;
    }

    //open sd card
    res = f_open(&fil, fileName, FA_CREATE_ALWAYS | FA_WRITE);

    //res = FR_OK, if the card has been open
    if(res!=FR_OK)
    {
        //if not, unmount the card and leave the function
        f_mount(0, "",0);
        Trace::out("error");
        return false;
    }

    return true;
}


bool SdCard::closeFile(const char* fileName)
{
    //close sd_card
    f_close(&fil);

    //res = FR_OK, if the card has been closed
    if(res!=FR_OK)
    {
        //if not, unmount the card and leave the function
        f_mount(0, "",0);
        Trace::out("error");
        return false;
    }

    //unmount the sd card
    f_mount(0, "",0);

    return true;
}

bool SdCard::writeFile(const char* dataPtr, uint32_t dataLength)
{
```

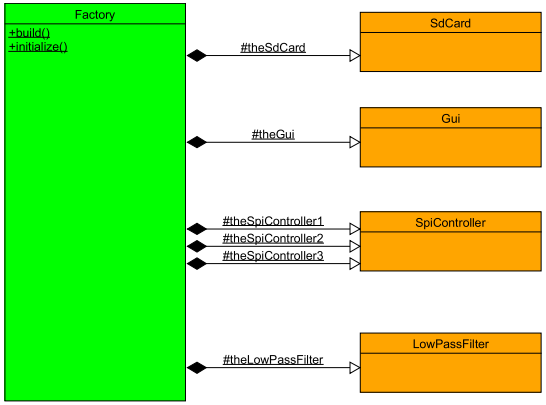
```
72      //put the pointer where we will write at the end of the file
73      res = f_lseek(&fil,f_size(&fil));
74
75      //res = FR_OK, if the pointer has been set to the end of the file
76      if(res!=FR_OK)
77      {
78          //if not, unmount the card and leave the function
79          f_mount(0, "",0);
80          Trace::out("error");
81          return false;
82      }
83
84      //write messageToWrite into sd card
85
86      res = f_write(&fil, dataPtr, dataLength, &bytesWritten);
87
88      //res = FR_OK, if the card has been writed
89      if(res!=FR_OK)
90      {
91          //if not, unmount the card and leave the function
92          f_mount(0, "",0);
93          Trace::out("error");
94          return false;
95      }
96
97      return true;
98  }
```

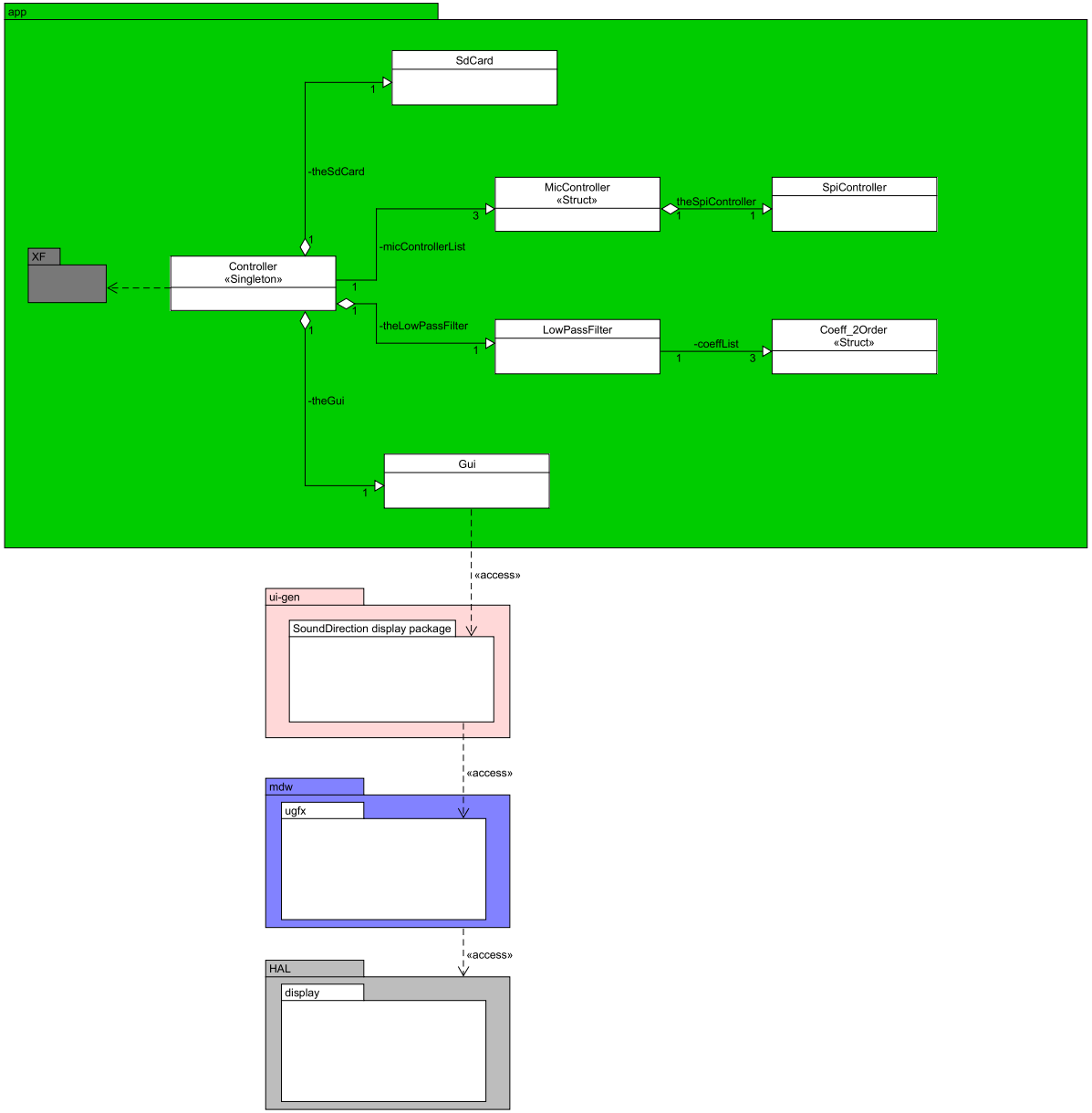
```cpp
/*
 * SpiController.h
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#ifndef SPICONTROLLER_H_
#define SPICONTROLLER_H_

#include "main.h"

#define DATA_SIZE 300 ///<@def @brief size of the data stored in memory

/**
 * @brief Manage the SPI Communcation
 */
class SpiController {

public:
        SpiController();
        virtual ~SpiController();


        /**
        *@brief initialization
        *@param theSPI is the configuration for the SPI communication
        */
        void init(SPI_HandleTypeDef* theSPI);

        void start();                     ///< @brief Start SPI in receive master only mode

        /**
        *@brief get the data pointer
        *@return the pointer to the SPI data
        */
        uint16_t* getData();            ///< @brief get the data pointer

        /**
        *@brief getter of hspi variable
        *@return theSPI communication configuration
        */
        SPI_HandleTypeDef* getHSPI();     ///< @brief getter of hspi variable

    private:
        SPI_HandleTypeDef* hspi;        ///< @brief SPI handle structure
        uint16_t* data;                   ///< @brief pointer to the data
};

#endif /* SPICONTROLLER_H_ */
```

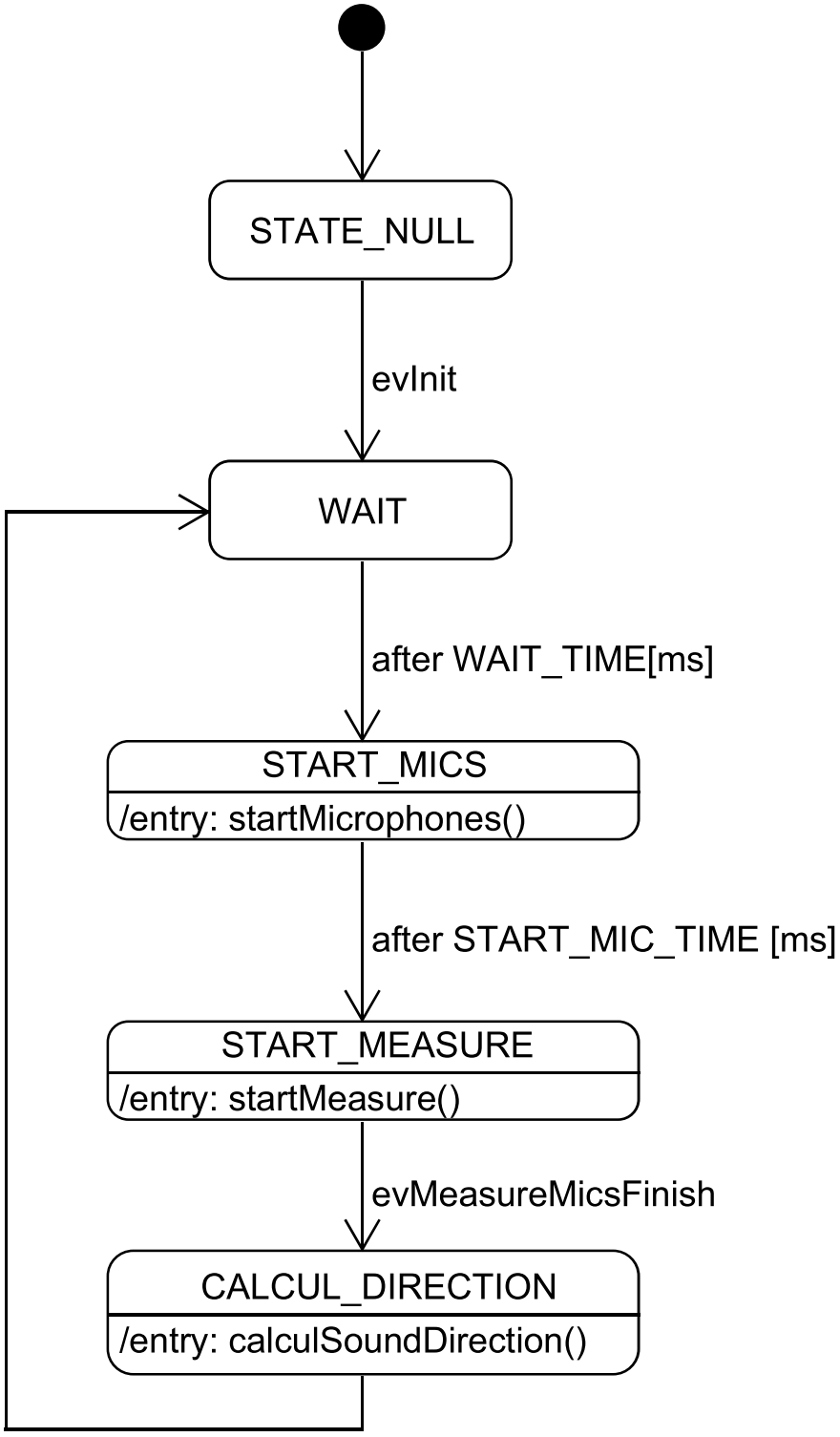
```cpp
/*
 * SpiController.cpp
 *
 *  Created on: 10 juin 2020
 *      Author: remyb
 */

#include "app/SpiController.h"
#include "trace/trace.h"

SpiController::SpiController() {

    //create dynamical array
    this->data = new uint16_t[DATA_SIZE];

    //init dynamical array
    for(int i = 0; i<DATA_SIZE;i++)
    {
        this->data[i] = 0;
    }

    this->hspi = nullptr;
}

SpiController::~SpiController() {
    //free the dynmical memory used by data
    if(this->data)
        delete[] this->data;
}

void SpiController::init(SPI_HandleTypeDef* hspi)
{
    this->hspi = hspi;
}

void SpiController::start()
{
    //launch SPI in receive master only mode with dma transfert
    HAL_SPI_Receive_DMA(this->hspi,(uint8_t*)this->data,DATA_SIZE);

}
uint16_t* SpiController::getData()
{
    return this->data;
}

SPI_HandleTypeDef* SpiController::getHSPI()
{
    return this->hspi;
}
```

Annexe 4.2:

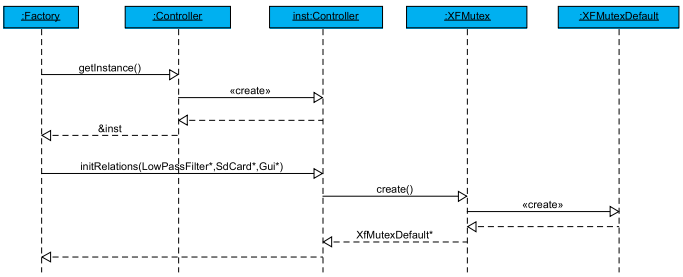
Documentation UML

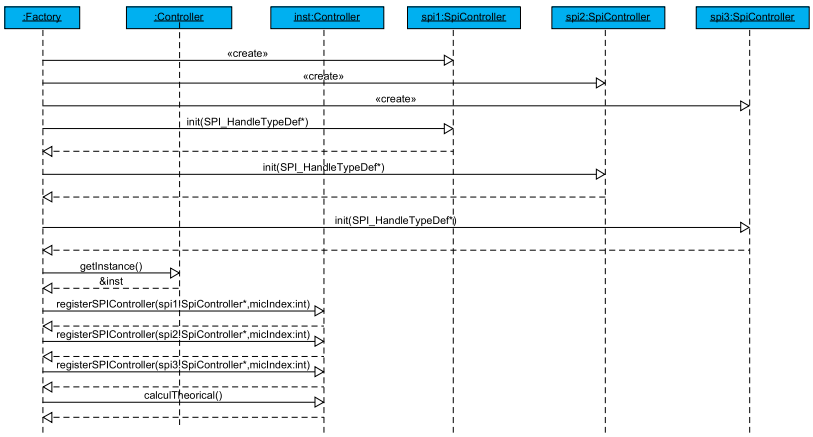

Factory
+build()
+initialize()
#theSdCard
SdCard
#theGui
Gui
#theSpiController1
#theSpiController2
#theSpiController3
SpiController
#theLowPassFilter
LowPassFilter


app
SdCard
1
-theSdCard
MicController
«Struct»
3
theSpiController
1
1
SpiController
XF
Controller
«Singleton»
1
-micControllerList
1
1
-theLowPassFilter
LowPassFilter
1
-coeffList
1
3
Coeff_2Order
«Struct»
1
-theGui
Gui
1
«access»
ui-gen
SoundDirection display package
«access»
mdw
ugfx
«access»
HAL
display

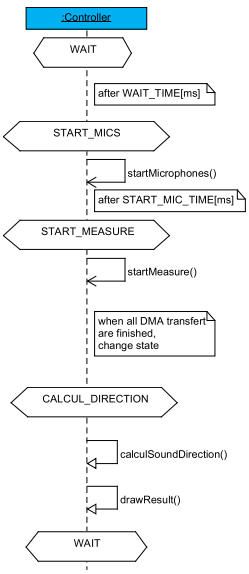

STATE_NULL
evInit
WAIT
after WAIT_TIME[ms]
START_MICS
/entry: startMicrophones()
after START_MIC_TIME [ms]
START_MEASURE
/entry: startMeasure()
evMeasureMicsFinish
CALCUL_DIRECTION
/entry: calculSoundDirection()

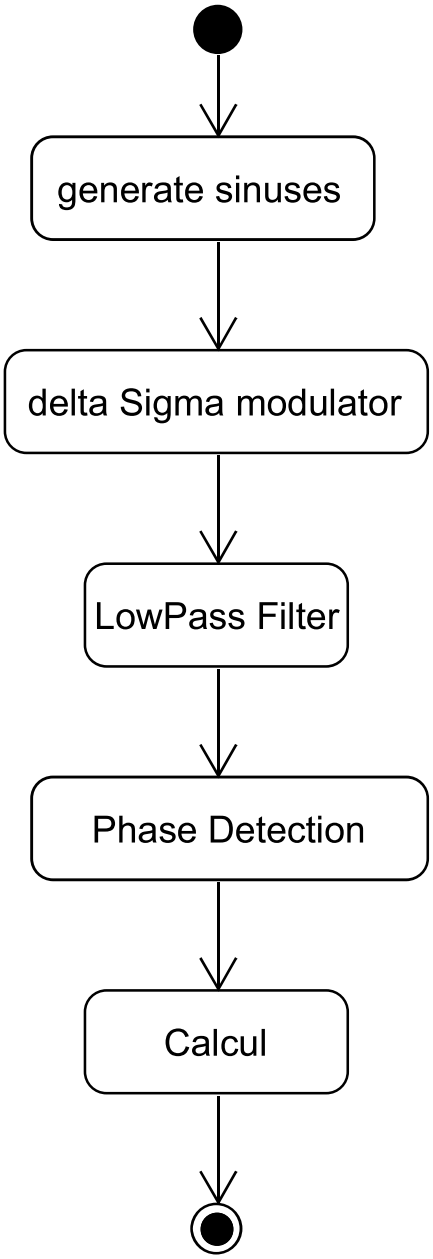
Initialisation Controller

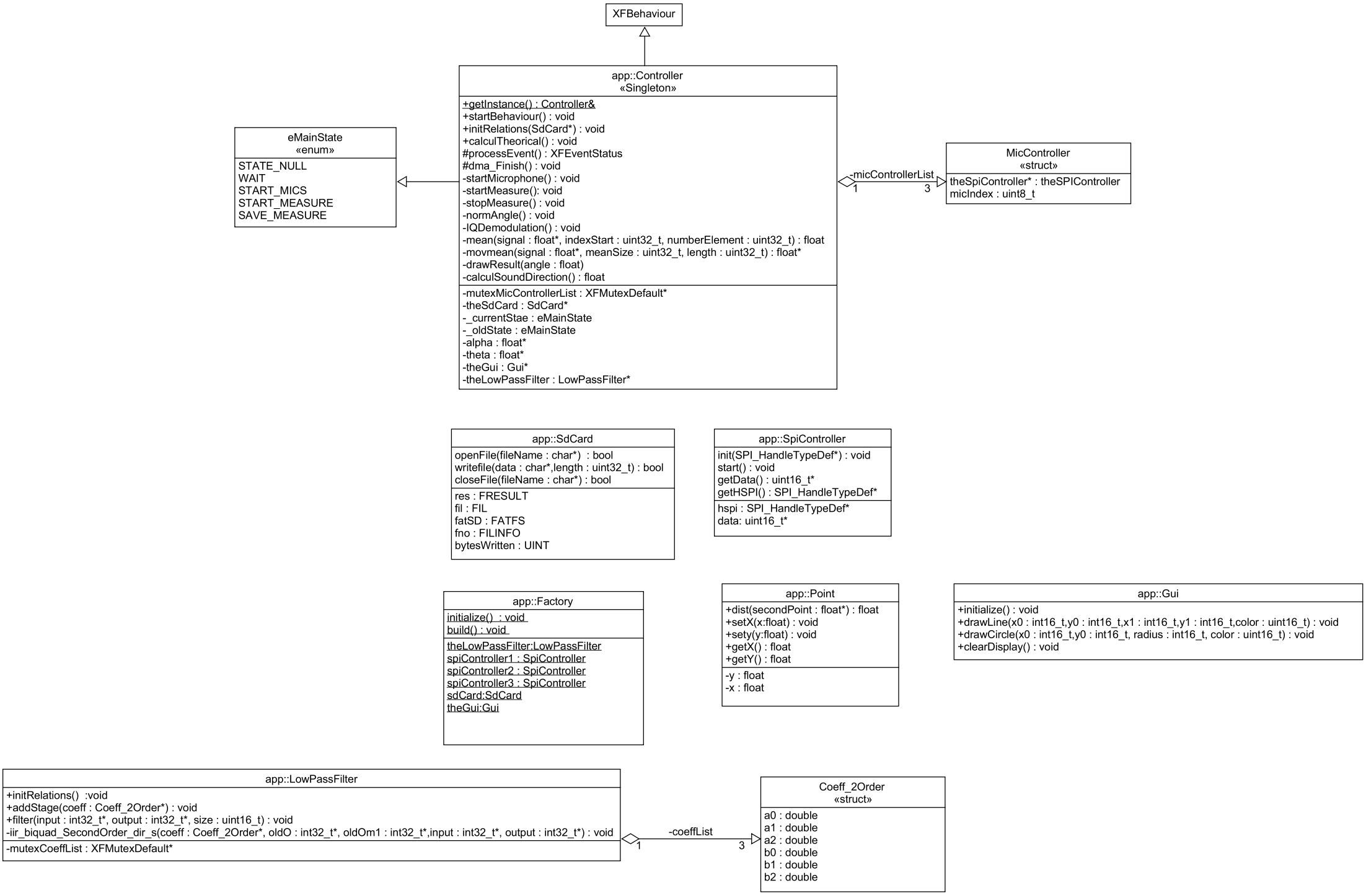


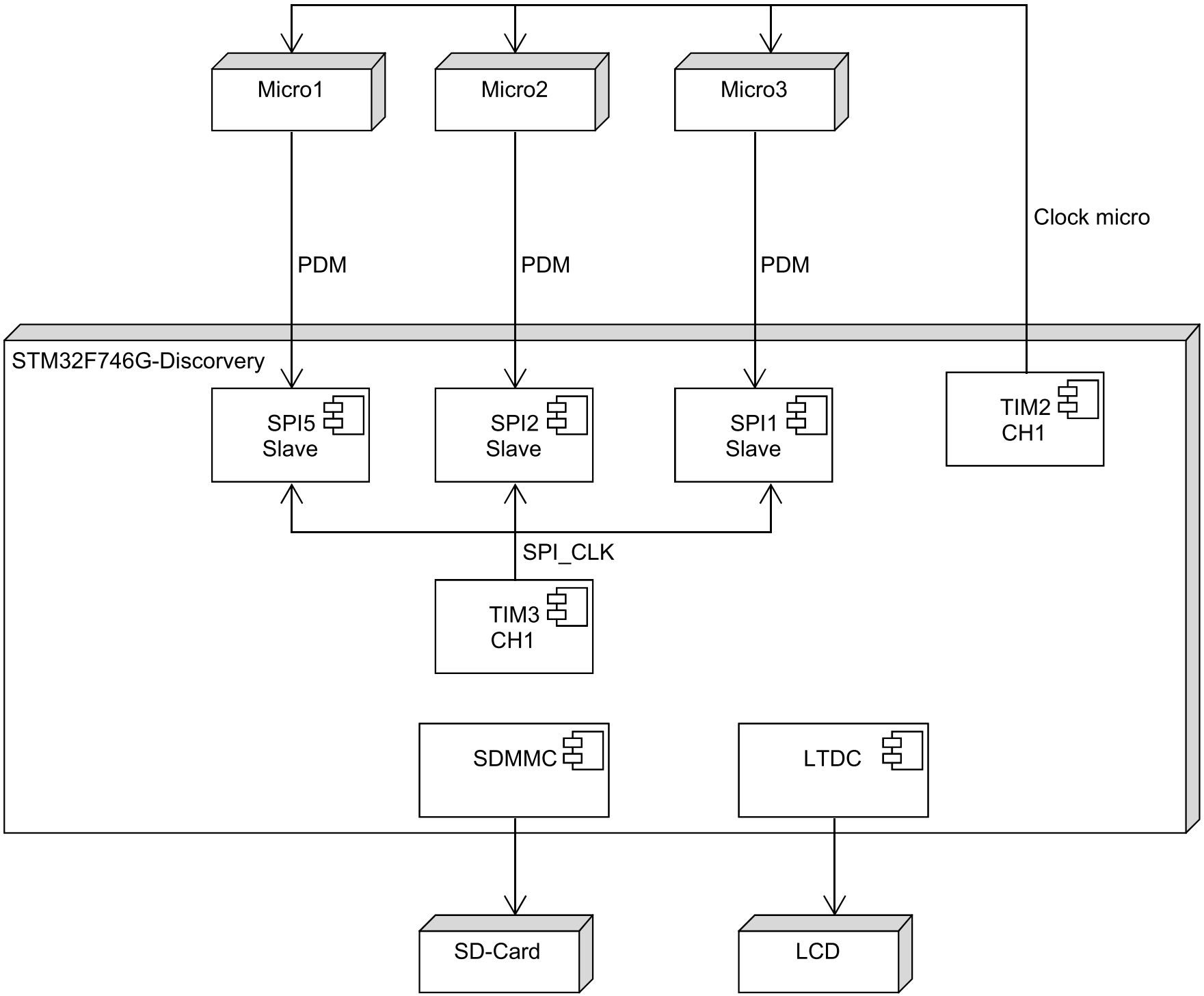


Take measure + calculSoundDirection




generate sinuses
delta Sigma modulator
LowPass Filter
Phase Detection
Calcul


XFBehaviour
app::Controller
«Singleton»
+getInstance() : Controller&
+startBehaviour() : void
+initRelations(SdCard*) : void
+calculTheorical() : void
#processEvent() : XFEventStatus
#dma_Finish() : void
-startMicrophone() : void
-startMeasure(): void
-stopMeasure() : void
-normAngle() : void
-IQDemodulation() : void
-mean(signal : float*, indexStart : uint32_t, numberElement : uint32_t) : float
-movmean(signal : float*, meanSize : uint32_t, length : uint32_t) : float*
-drawResult(angle : float)
-calculSoundDirection() : float
-mutexMicControllerList : XFMutexDefault*
-theSdCard : SdCard*
-_currentStae : eMainState
-_oldState : eMainState
-alpha : float*
-theta : float*
-theGui : Gui*
-theLowPassFilter : LowPassFilter*
eMainState
«enum»
STATE_NULL
WAIT
START_MICS
START_MEASURE
SAVE_MEASURE
-micControllerList
1
3
MicController
«struct»
theSpiController* : theSPIController
micIndex : uint8_t
app::SdCard
openFile(fileName : char*) : bool
writefile(data : char*,length : uint32_t) : bool
closeFile(fileName : char*) : bool
res : FRESULT
fil : FIL
fatSD : FATFS
fno : FILINFO
bytesWritten : UINT
app::SpiController
init(SPI_HandleTypeDef*) : void
start() : void
getData() : uint16_t*
getHSPI() : SPI_HandleTypeDef*
hspi : SPI_HandleTypeDef*
data: uint16_t*
app::Factory
initialize() : void
build() : void
theLowPassFilter:LowPassFilter
spiController1 : SpiController
spiController2 : SpiController
spiController3 : SpiController
sdCard:SdCard
theGui:Gui
app::Point
+dist(secondPoint : float*) : float
+setX(x:float) : void
+sety(y:float) : void
+getX() : float
+getY() : float
-y : float
-x : float
app::Gui
+initialize() : void
+drawLine(x0 : int16_t,y0 : int16_t,x1 : int16_t,y1 : int16_t,color : uint16_t) : void
+drawCircle(x0 : int16_t,y0 : int16_t, radius : int16_t, color : uint16_t) : void
+clearDisplay() : void
app::LowPassFilter
+initRelations() :void
+addStage(coeff : Coeff_2Order*) : void
+filter(input : int32_t*, output : int32_t*, size : uint16_t) : void
-iir_biquad_SecondOrder_dir_s(coeff : Coeff_2Order*, oldO : int32_t*, oldOm1 : int32_t*,input : int32_t*, output : int32_t*) : void
-mutexCoeffList : XFMutexDefault*
-coeffList
1
3
Coeff_2Order
«struct»
a0 : double
a1 : double
a2 : double
b0 : double
b1 : double
b2 : double


Micro1
Micro2
Micro3
PDM
PDM
PDM
Clock micro
STM32F746G-Discorvery
SPI5
Slave
SPI2
Slave
SPI1
Slave
TIM2
CH1
SPI_CLK
TIM3
CH1
SDMMC
LTDC
SD-Card
LCD

Annexe 4.3:

Documentation du code

SoundDirectionDetection

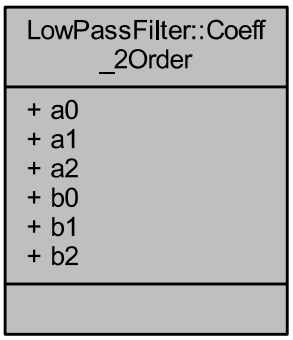
Generated by Doxygen 1.8.16

1 Class Documentation

1.1 LowPassFilter::Coeff_2Order Struct Reference

`#include <LowPassFilter.h>`

Collaboration diagram for LowPassFilter::Coeff_2Order:



Public Attributes

- double **a0**
- double **a1**
- double **a2**
- double **b0**
- double **b1**
- double **b2**

1.1.1 Detailed Description

@bried 2 order coeff of the filter

1.1.2 Member Data Documentation

1.1.2.1 a0 `double LowPassFilter::Coeff_2Order::a0`

1.1.2.2 a1 double LowPassFilter::Coeff_2Order::a1

1.1.2.3 a2 double LowPassFilter::Coeff_2Order::a2

1.1.2.4 b0 double LowPassFilter::Coeff_2Order::b0

1.1.2.5 b1 double LowPassFilter::Coeff_2Order::b1

1.1.2.6 b2 double LowPassFilter::Coeff_2Order::b2

The documentation for this struct was generated from the following file:

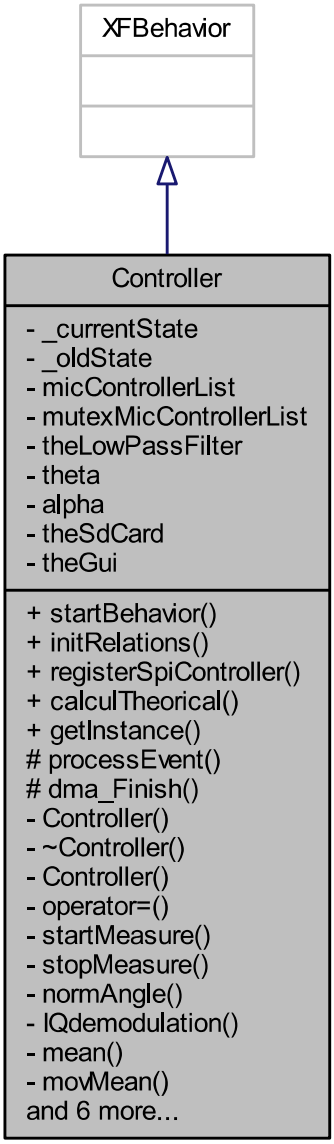
- C:/Users/rebor/Desktop/doxygen/app/ **LowPassFilter.h**

1.2 Controller Class Reference

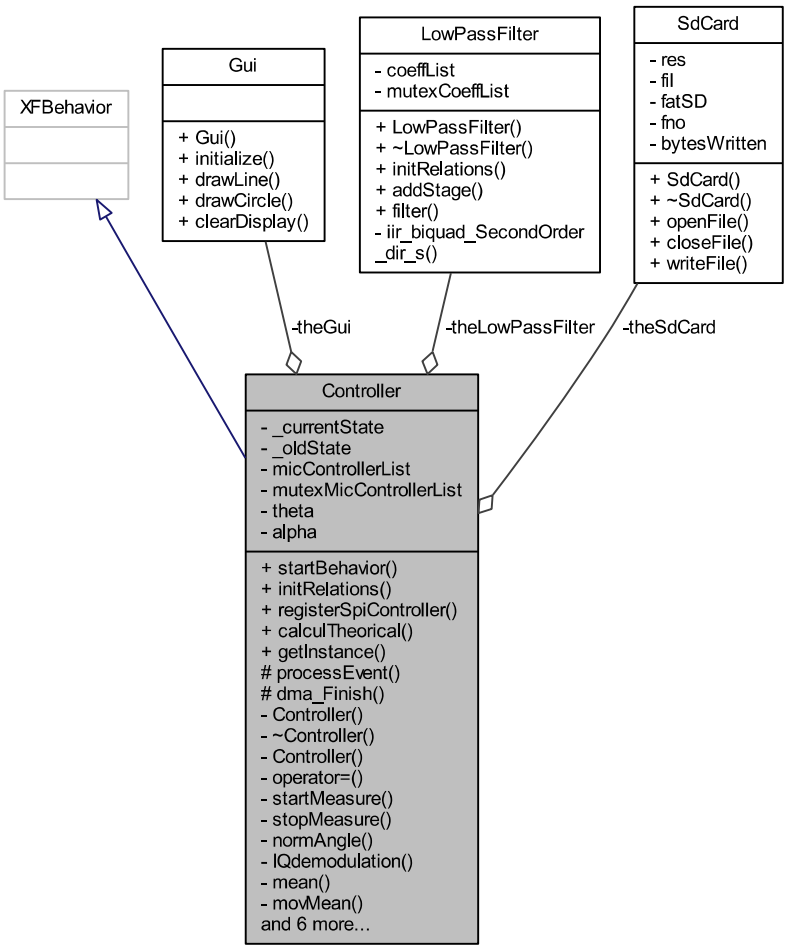
this class contains a state machine the state machine is started when the users push on the start button Then it will save the data on a sd card and go back to the initial state

```
#include <Controller.h>
```

Inheritance diagram for Controller:



Collaboration diagram for Controller:



Classes

- struct **MicController**

  Struct who contains micIndex and theSpiController for the mic with that micIndex.

Public Member Functions

- void **startBehavior** ()

  start the state machine
- void **initRelations** (**LowPassFilter** ∗ **theLowPassFilter**, **SdCard** ∗ **theSdCard**, **Gui** ∗theGUI)

  init relations
- void **registerSpiController** (**SpiController** ∗theSPIController, uint8_t micIndex)

  register a new SPI communication
- void **calculTheorical** ()

  calcul the theta and alpha angle this two angle are needed to found the direction of the others divers.

Static Public Member Functions

- static **Controller** & **getInstance** ()

  return the only one instance of the singleton class

Protected Member Functions

- XFEventStatus **processEvent** ()

  State machine.
- void **dma_Finish** ()

  called when dma transfert is finish

Private Types

- enum **eMainState** {
  STATE_NULL = 0, **STATE_WAIT**, **START_MICS**, **START_MEASURE**,
  CALCUL_DIRECTION }

  Enumeration used to have a unique identifier for every state in the state machine.

Private Member Functions

- **Controller** ()
- ~**Controller** ()
- **Controller** (const **Controller** &)
- **Controller** & **operator=** (const **Controller** &)
- void **startMeasure** ()

  Start mics measure.
- void **stopMeasure** ()

  Stop mics measure.
- float **normAngle** (float angle)

  normalize a angle between -Pi and Pi
- float **IQdemodulation** (float *sinus, uint32_t length)

  calculate the phases of a sinus, using IQ demodulation
- float **mean** (float *signal, uint32_t indexStart, uint32_t numberElement)

  calculate the mean value of a signal
- float * **movMean** (float *signal, int32_t meanSize, uint32_t length)

  calculate the moving mean of a signal
- void **startMicrophones** ()

  start the microphone clock microphone take 10[ms] to startup
- void **drawResult** (float angle)

  draw a arrow on the LCD screen to indicate the sound direction
- void **saveData** (int32_t *, uint32_t, std::string)
- void **saveData** (uint16_t *, uint32_t, std::string)
- void **saveData** (float *, uint32_t, std::string)
- float **calculSoundDirection** ()

  calcul the direction of the sound

Private Attributes

- **eMainState _currentState**

  Attribute indicating currently active state.
- **eMainState _oldState**

  Attribute indicating old state.
- std::list< **MicController** * > **micControllerList**

  list of callback
- XFMutexDefault * **mutexMicControllerList**

  mutex for the list
- **LowPassFilter** * **theLowPassFilter**

  *pointer to the **LowPassFilter*** (p. *21) class*
- float * **theta**

  pointer to the theta array, this array contains theta angle values
- float * **alpha**

  pointer to the alpha array, this array contains alpha angle values
- **SdCard** * **theSdCard**

  *pointer to the **SdCard*** (p. *30) class*
- **Gui** * **theGui**

  *pointer to the **Gui*** (p. *18) class*

Friends

- void **HAL_SPI_RxCpltCallback** (SPI_HandleTypeDef *hspi)
- void **HAL_TIM_PeriodElapsedCallback** (TIM_HandleTypeDef *htim)

1.2.1 Detailed Description

this class contains a state machine the state machine is started when the users push on the start button Then it will save the data on a sd card and go back to the initial state

1.2.2 Member Enumeration Documentation

1.2.2.1 eMainState `enum Controller::eMainState [private]`

Enumeration used to have a unique identifier for every state in the state machine.

Enumerator

STATE_NULL	null state
STATE_WAIT	wait state
START_MICS	start microphoen clock
START_MEASURE	start taking measure from the micros
CALCUL_DIRECTION	calcul direction of the sound

1.2.3 Constructor & Destructor Documentation

1.2.3.1 Controller() [1/2] Controller::Controller ( ) [private]

1.2.3.2 ~Controller() Controller::~Controller ( ) [private]

1.2.3.3 Controller() [2/2] Controller::Controller (const Controller &) [inline], [private]

1.2.4 Member Function Documentation

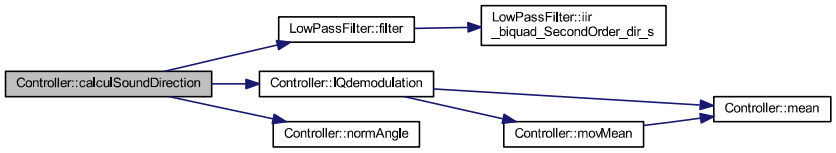
1.2.4.1 calculSoundDirection() float Controller::calculSoundDirection ( ) [private]

calcul the direction of the sound

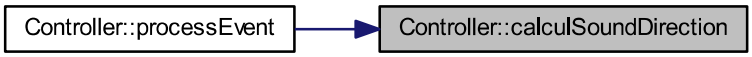
Returns

angle of the sound direction

Here is the call graph for this function:



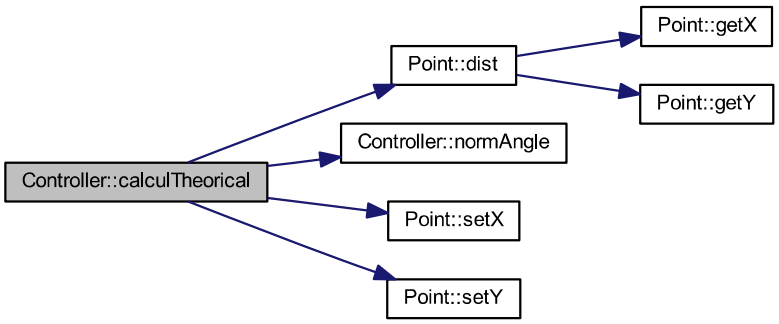
Here is the caller graph for this function:



1.2.4.2 calculTheorical() `void Controller::calculTheorical ( )`

calcul the theta and alpha angle this two angle are needed to found the direction of the others divers.

Here is the call graph for this function:



1.2.4.3 dma_Finish() `void Controller::dma_Finish ( ) [protected]`

called when dma transfert is finish

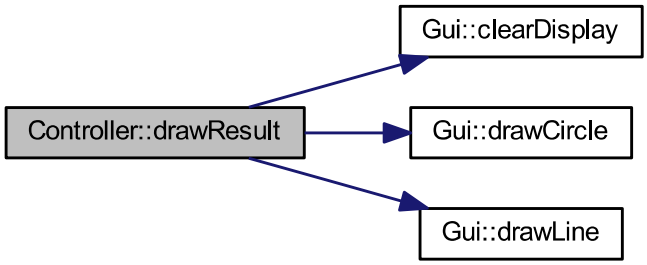
1.2.4.4 drawResult() `void Controller::drawResult ( float angle ) [private]`

draw a arrow on the LCD screen to indicate the sound direction

Parameters

angle	: the sound direction

Here is the call graph for this function:



Here is the caller graph for this function:

1.2.4.5 getInstance() `Controller & Controller::getInstance ( ) [static]`

return the only one instance of the singleton class

Returns

the only one instance of the **Controller** (p. 3) class

1.2.4.6 initRelations() `void Controller::initRelations (`

```
LowPassFilter * theLowPassFilter,
SdCard * theSdCard,
Gui * theGUI )
```

init relations

Parameters

theLowPassFilter	is a pointer to the **LowPassFilter** (p. 21) class instance
theSdCard	is a pointer to the **SdCard** (p. 30) class instance
theGUI	is a pointer to the **Gui** (p. 18) class instance

init relations

1.2.4.7 IQdemodulation()

```
float Controller::IQdemodulation (
        float * sinus,
        uint32_t length )  [private]
```

calculate the phases of a sinus, using IQ demodulation

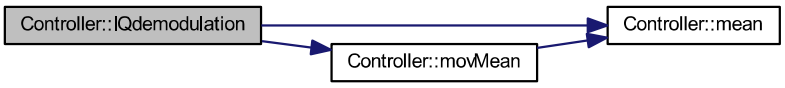
Parameters

sinus	: sinus to calculate the phases
length	: length of the sinus array

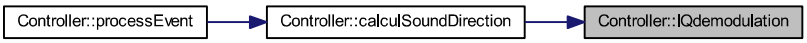
Returns

the phases of the sinus value

Here is the call graph for this function:



Here is the caller graph for this function:



1.2.4.8 mean()

```
float Controller::mean (
        float * signal,
        uint32_t indexStart,
        uint32_t numberElement )  [private]
```

calculate the mean value of a signal

Parameters

signal	: signal to calculate the mean value
indexStart	: index in the signal array to start the mean value
numberElement	: numberElement of the mean value

Returns

the mean value of the array

Here is the caller graph for this function:

1.2.4.9 movMean() `float * Controller::movMean ( float * signal, int32_t meanSize, uint32_t length ) [private]`

calculate the moving mean of a signal

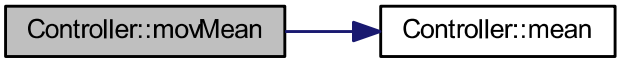
Parameters

signal	: signal to calculate the moving mean
meanSize	: size of the moving mean
length	: length of the signal array

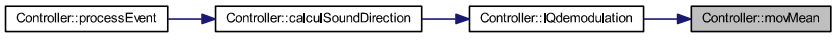
Returns

the moving mean array pointer

Here is the call graph for this function:



Here is the caller graph for this function:



1.2.4.10 normAngle() `float Controller::normAngle ( float angle ) [private]`

normalize a angle between -Pi and Pi

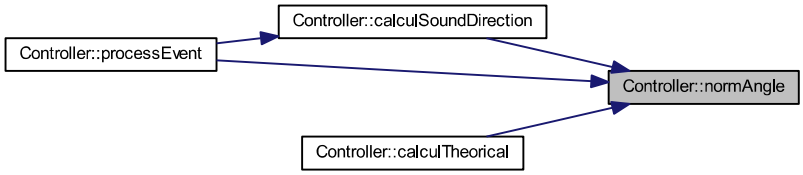
Parameters

angle	: the angle to normalize

Returns

the normalized angle

Here is the caller graph for this function:



1.2.4.11 operator=() `Controller& Controller::operator= ( const Controller & ) [inline], [private]`

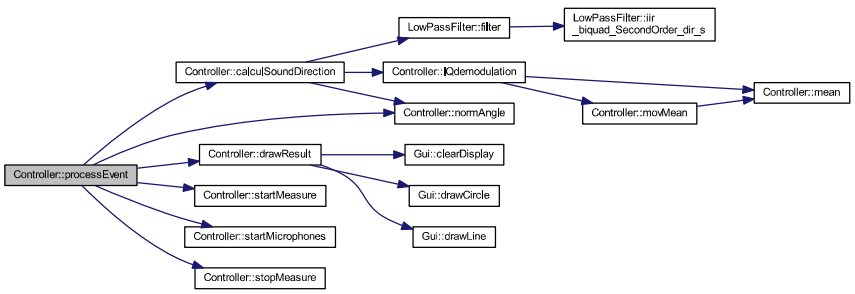
1.2.4.12 processEvent() `XFEventStatus Controller::processEvent ( ) [protected]`

State machine.

Returns

XFEventStatus return the events status : Consummed or unknow

Here is the call graph for this function:



1.2.4.13 registerSpiController()

```
void Controller::registerSpiController (
        SpiController * theSPIController,
        uint8_t micIndex )
```

register a new SPI communication

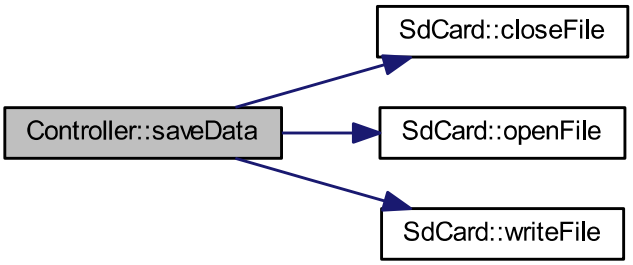
Parameters

theSPIController	is a pointer to the **SpiController** (p. 34) class instance
micIndex	is the index of the microphone in the PCB

1.2.4.14 saveData() [1/3]

```
void Controller::saveData (
        float * dataToSave,
        uint32_t size,
        std::string fileName )  [private]
```

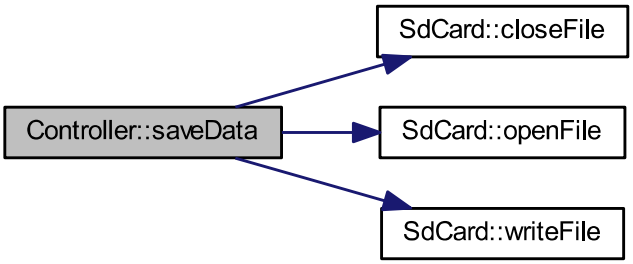
Here is the call graph for this function:



1.2.4.15 saveData() [2/3]

```
void Controller::saveData (
        int32_t * dataToSave,
        uint32_t size,
        std::string fileName )  [private]
```

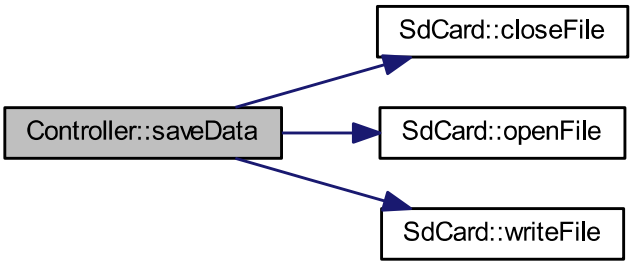
Here is the call graph for this function:



1.2.4.16 saveData() [3/3]

```
void Controller::saveData (
        uint16_t * dataToSave,
        uint32_t size,
        std::string fileName )  [private]
```

Here is the call graph for this function:



1.2.4.17 startBehavior()

```
void Controller::startBehavior ( )
```

start the state machine

1.2.4.18 startMeasure() `void Controller::startMeasure ( ) [private]`

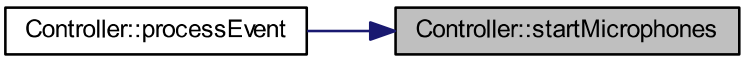
Start mics measure.

Here is the caller graph for this function:

1.2.4.19 startMicrophones() `void Controller::startMicrophones ( ) [private]`

start the microphone clock microphone take 10[ms] to startup

Here is the caller graph for this function:



1.2.4.20 stopMeasure() `void Controller::stopMeasure ( ) [private]`

Stop mics measure.

Here is the caller graph for this function:

1.2.5 Friends And Related Function Documentation

1.2.5.1 HAL_SPI_RxCpltCallback

```
void HAL_SPI_RxCpltCallback (
        SPI_HandleTypeDef * hspi ) [friend]
```

1.2.5.2 HAL_TIM_PeriodElapsedCallback

```
void HAL_TIM_PeriodElapsedCallback (
        TIM_HandleTypeDef * htim ) [friend]
```

1.2.6 Member Data Documentation

1.2.6.1 _currentState

```
eMainState Controller::_currentState [private]
```

Attribute indicating currently active state.

1.2.6.2 _oldState

```
eMainState Controller::_oldState [private]
```

Attribute indicating old state.

1.2.6.3 alpha

```
float* Controller::alpha [private]
```

pointer to the alpha array, this array contains alpha angle values

1.2.6.4 micControllerList

```
std::list< MicController*> Controller::micControllerList [private]
```

list of callback

1.2.6.5 mutexMicControllerList

```
XFMutexDefault* Controller::mutexMicControllerList [private]
```

mutex for the list

1.2.6.6 theGui

`Gui* Controller::theGui [private]`

pointer to the **Gui** (p. 18) class

1.2.6.7 theLowPassFilter

`LowPassFilter* Controller::theLowPassFilter [private]`

pointer to the **LowPassFilter** (p. 21) class

1.2.6.8 theSdCard

`SdCard* Controller::theSdCard [private]`

pointer to the **SdCard** (p. 30) class

1.2.6.9 theta

`float* Controller::theta [private]`

pointer to the theta array, this array contains theta angle values

The documentation for this class was generated from the following files:

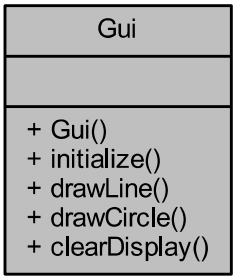
- C:/Users/rebor/Desktop/doxygen/app/ **Controller.h**
- C:/Users/rebor/Desktop/doxygen/app/ **Controller.cpp**

1.3 Gui Class Reference

The **Gui** (p. 18) (Graphical User Interface) class display line and circle on the LCD screen.

`#include <gui.h>`

Collaboration diagram for Gui:



Public Member Functions

- **Gui** ()
- void **initialize** ()

  *initialize **Gui** (p. 18) class*
- void **drawLine** (int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color)

  draw a line on the LCD screen
- void **drawCircle** (int16_t x0, int16_t y0, int16_t radius, uint16_t color)

  draw a filled circle on the LCD screen
- void **clearDisplay** ()

  clear the LCD screen

1.3.1 Detailed Description

The **Gui** (p. 18) (Graphical User Interface) class display line and circle on the LCD screen.

1.3.2 Constructor & Destructor Documentation

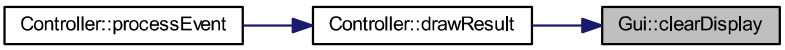
1.3.2.1 Gui() `Gui::Gui ( )`

1.3.3 Member Function Documentation

1.3.3.1 clearDisplay() `void Gui::clearDisplay ( )`

clear the LCD screen

Here is the caller graph for this function:



1.3.3.2 drawCircle() `void Gui::drawCircle (`

```
int16_t x0,
int16_t y0,
int16_t radius,
uint16_t color )
```

draw a filled circle on the LCD screen

Parameters

x0	x value of the center of the circle
y0	y value of the center of the circle
radius	radius value of the circle
color	color of the circle

Here is the caller graph for this function:

1.3.3.3 drawLine()

```
void Gui::drawLine (
        int16_t x0,
        int16_t y0,
        int16_t x1,
        int16_t y1,
        uint16_t color )
```

draw a line on the LCD screen

Parameters

x0	x value of the first point of the line
y0	y value of the first point of the line
x1	x value of the second point of the line
y1	y value of the second point of the line
color	color of the line

Here is the caller graph for this function:

1.3.3.4 initialize()

```
void Gui::initialize ( )
```

initialize **Gui** (p. 18) class

The documentation for this class was generated from the following files:

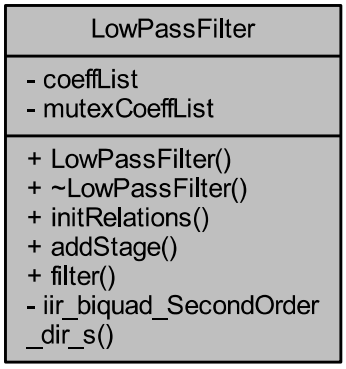
- C:/Users/rebor/Desktop/doxygen/app/ **gui.h**
- C:/Users/rebor/Desktop/doxygen/app/ **gui.cpp**

1.4 LowPassFilter Class Reference

this class implements a low pass filter the number of stage is choosen by the users each stage is a second order IIR filter

`#include <LowPassFilter.h>`

Collaboration diagram for LowPassFilter:



Classes

- struct **Coeff_2Order**

Public Member Functions

- **LowPassFilter** ()
- virtual ~**LowPassFilter** ()
- void **initRelations** ()

  initialize relations
- void **addStage** (**Coeff_2Order** ∗coeff)

  add a second order stage to the filter
- void **filter** (int32_t ∗input, int32_t ∗output, uint16_t size)

  execute the filter

Private Member Functions

- void **iir_biquad_SecondOrder_dir_s** (**Coeff_2Order** Coeff, int32_t ∗oldO, int32_t ∗oldOm1, int32_t ∗Input, int32_t ∗Output)

  one stage of the filter

Private Attributes

- std::list< **Coeff_2Order** ∗ > **coeffList**

  lsit of all stage coefficient
- XFMutexDefault ∗ **mutexCoeffList**

  mutex for the list

1.4.1 Detailed Description

this class implements a low pass filter the number of stage is choosen by the users each stage is a second order IIR filter

1.4.2 Constructor & Destructor Documentation

1.4.2.1 LowPassFilter() `LowPassFilter::LowPassFilter ( )`

1.4.2.2 ∼LowPassFilter() `LowPassFilter::∼LowPassFilter ( ) [virtual]`

1.4.3 Member Function Documentation

1.4.3.1 addStage() `void LowPassFilter::addStage ( Coeff_2Order * coeff )`

add a second order stage to the filter

Parameters

coeff	coefficient of the second order stage filter

1.4.3.2 filter() `void LowPassFilter::filter ( int32_t * input,`

```
            int32_t * output,
            uint16_t size )
```

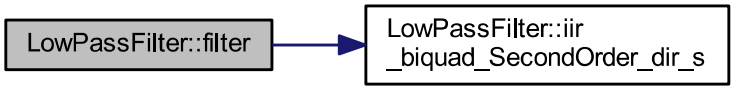
execute the filter

@bried execute the filter

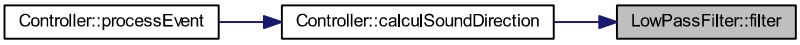
Parameters

input	: input signal to be filtred, pointer to an array
output	: output of the filter ,pointer to an array
size	: size of the input array

Here is the call graph for this function:



Here is the caller graph for this function:



1.4.3.3 iir_biquad_SecondOrder_dir_s()

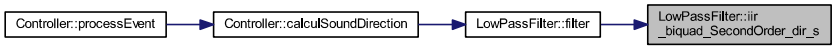
```
void LowPassFilter::iir_biquad_SecondOrder_dir_s (
            Coeff_2Order Coeff,
            int32_t * oldO,
            int32_t * oldOm1,
            int32_t * Input,
            int32_t * Output )  [private]
```

one stage of the filter

Parameters

coeff	: Coefficient of the stage
oldO	: oldValue of the digital filter
oldOm1	: oldValue of the digital filter
Input	: pointer to the the array that must be filtered
Output	: pointer to the the array that has been filtered

Here is the caller graph for this function:



1.4.3.4 initRelations() `void LowPassFilter::initRelations ( )`

initialize relations

1.4.4 Member Data Documentation

1.4.4.1 coeffList `std::list< Coeff_2Order* > LowPassFilter::coeffList [private]`

lsit of all stage coefficient

1.4.4.2 mutexCoeffList `XFMutexDefault* LowPassFilter::mutexCoeffList [private]`

mutex for the list

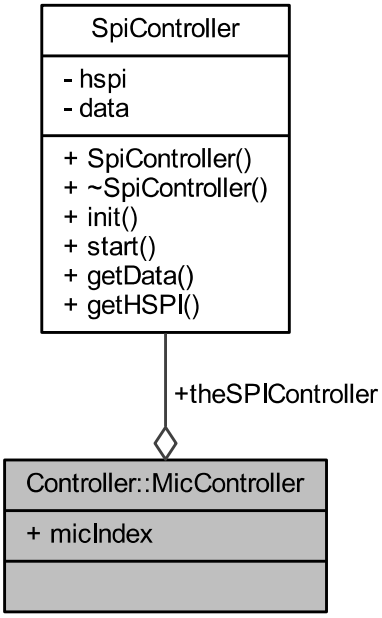
The documentation for this class was generated from the following files:

- C:/Users/rebor/Desktop/doxygen/app/ **LowPassFilter.h**
- C:/Users/rebor/Desktop/doxygen/app/ **LowPassFilter.cpp**

1.5 Controller::MicController Struct Reference

Struct who contains micIndex and theSpiController for the mic with that micIndex.

Collaboration diagram for Controller::MicController:



Public Attributes

- **SpiController** * **theSPIController**

  the SPIController who controller the mic with mic index
- uint8_t **micIndex**

  index of the mics controlled by the spi controller

1.5.1 Detailed Description

Struct who contains micIndex and theSpiController for the mic with that micIndex.

1.5.2 Member Data Documentation

1.5.2.1 micIndex `uint8_t Controller::MicController::micIndex`

index of the mics controlled by the spi controller

1.5.2.2 theSPIController `SpiController* Controller::MicController::theSPIController`

the SPIController who controller the mic with mic index

The documentation for this struct was generated from the following file:

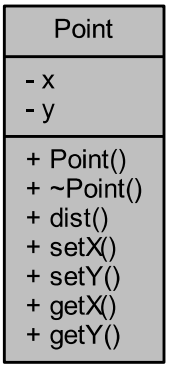
- C:/Users/rebor/Desktop/doxygen/app/ **Controller.h**

1.6 Point Class Reference

this class represents a point with x and y coordonate. Allows to calculate distance bewteen 2 points

`#include <Point.h>`

Collaboration diagram for Point:



Public Member Functions

- **Point** ()
- virtual ~**Point** ()
- float **dist** (**Point** *secondPoint)
  calculate the distance between two points
- void **setX** (float **x**)
  setter of x point value
- void **setY** (float **y**)
  setter of y point value
- float **getX** ()
  getter of x point value
- float **getY** ()
  getter of y point value

Private Attributes

- float **x**

  x point value
- float **y**

  y point value

1.6.1 Detailed Description

this class represents a point with x and y coordonate. Allows to calculate distance bewteen 2 points

1.6.2 Constructor & Destructor Documentation

1.6.2.1 Point() `Point::Point ( )`

1.6.2.2 ~Point() `Point::~Point ( ) [virtual]`

1.6.3 Member Function Documentation

1.6.3.1 dist()

```
float Point::dist (
    Point * secondPoint )
```

calculate the distance between two points

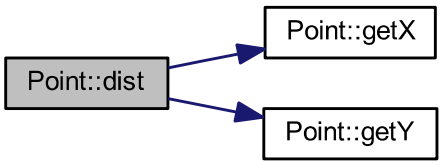
Parameters

secondPoint	the 2nd point to calculate the distance

Returns

the distance between the 2 points

Here is the call graph for this function:



Here is the caller graph for this function:

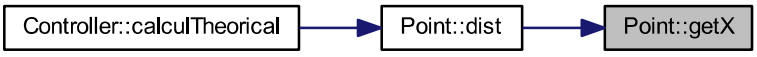
1.6.3.2 getX() `float Point::getX ( )`

getter of x point value

Returns

x point value

Here is the caller graph for this function:



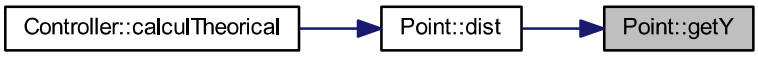
1.6.3.3 getY() `float Point::getY ( )`

getter of y point value

Returns

y point value

Here is the caller graph for this function:



1.6.3.4 setX() `void Point::setX ( float x )`

setter of x point value

Parameters

x	new x value

Here is the caller graph for this function:

1.6.3.5 setY() `void Point::setY ( float y )`

setter of y point value

Parameters

y	new y value

Here is the caller graph for this function:

1.6.4 Member Data Documentation

1.6.4.1 x `float Point::x [private]`

x point value

1.6.4.2 y `float Point::y [private]`

y point value

The documentation for this class was generated from the following files:

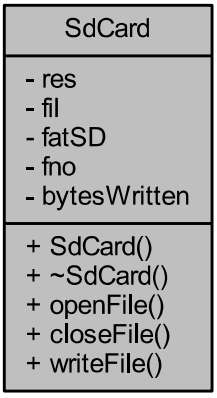
- C:/Users/rebor/Desktop/doxygen/app/ **Point.h**
- C:/Users/rebor/Desktop/doxygen/app/ **Point.cpp**

1.7 SdCard Class Reference

this class write data into SDCard

```
#include <SdCard.h>
```

Collaboration diagram for SdCard:



Public Member Functions

- **SdCard** ()
- virtual ~**SdCard** ()
- bool **openFile** (const char *fileName)

  open a file in the sd card
- bool **closeFile** (const char *fileName)

  close a file in the sd card
- bool **writeFile** (const char *data, uint32_t length)

  write into the text file some data

Private Attributes

- FRESULT **res**

  result of the action
- FIL **fil**

  file object structure
- FATFS **fatSD**

  file system object
- FILINFO **fno**

  file information
- UINT **bytesWritten**

  bytes written in sd card

1.7.1 Detailed Description

this class write data into SDCard

1.7.2 Constructor & Destructor Documentation

1.7.2.1 SdCard() `SdCard::SdCard ( )`

1.7.2.2 ~SdCard() `SdCard::~SdCard ( ) [virtual]`

1.7.3 Member Function Documentation

1.7.3.1 closeFile() `bool SdCard::closeFile ( const char * fileName )`

close a file in the sd card

Parameters

fileName	is the text file Name

Returns

true if the file has been closed

Here is the caller graph for this function:

1.7.3.2 openFile() `bool SdCard::openFile ( const char * fileName )`

open a file in the sd card

Parameters

fileName	is the text file Name

Returns

true if the file has been created and opened

Here is the caller graph for this function:

1.7.3.3 writeFile()

```
bool SdCard::writeFile (
        const char * data,
        uint32_t length )
```

write into the text file some data

Parameters

data	is a pointer to the data to be written
length	is the length of the data to be write

Returns

true if the file has been writted correctly

Here is the caller graph for this function:

1.7.4 Member Data Documentation

1.7.4.1 bytesWritten `UINT SdCard::bytesWritten [private]`

bytes written in sd card

1.7.4.2 fatSD `FATFS SdCard::fatSD [private]`

file system object

1.7.4.3 fil `FIL SdCard::fil [private]`

file object structure

1.7.4.4 fno `FILINFO SdCard::fno [private]`

file information

1.7.4.5 res `FRESULT SdCard::res [private]`

result of the action

The documentation for this class was generated from the following files:

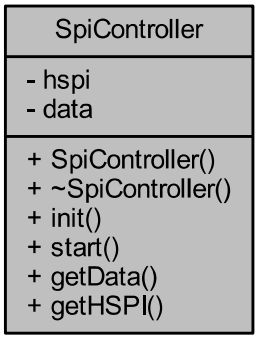
- C:/Users/rebor/Desktop/doxygen/app/ **SdCard.h**
- C:/Users/rebor/Desktop/doxygen/app/ **SdCard.cpp**

1.8 SpiController Class Reference

Manage the SPI Communcation.

`#include <SpiController.h>`

Collaboration diagram for SpiController:



Public Member Functions

- **SpiController** ()
- virtual ~**SpiController** ()
- void **init** (SPI_HandleTypeDef *theSPI)
  initialization
- void **start** ()
  Start SPI in receive master only mode.
- uint16_t * **getData** ()
  get the data pointer
- SPI_HandleTypeDef * **getHSPI** ()
  getter of hspi variable

Private Attributes

- SPI_HandleTypeDef * **hspi**
  SPI handle structure.
- uint16_t * **data**
  pointer to the data

1.8.1 Detailed Description

Manage the SPI Communcation.

1.8.2 Constructor & Destructor Documentation

1.8.2.1 SpiController() `SpiController::SpiController ( )`

1.8.2.2 ~SpiController() `SpiController::~SpiController ( ) [virtual]`

1.8.3 Member Function Documentation

1.8.3.1 getData() `uint16_t * SpiController::getData ( )`

get the data pointer

Returns

the pointer to the SPI data

get the data pointer

1.8.3.2 getHSPI() SPI_HandleTypeDef * SpiController::getHSPI ( )

getter of hspi variable

Returns

theSPI communication configuration

getter of hspi variable

1.8.3.3 init() void SpiController::init (SPI_HandleTypeDef * *theSPI*)

initialization

Parameters

the↩ SPI	is the configuration for the SPI communication

1.8.3.4 start() void SpiController::start ( )

Start SPI in receive master only mode.

1.8.4 Member Data Documentation

1.8.4.1 data uint16_t* SpiController::data [private]

pointer to the data

1.8.4.2 hspi SPI_HandleTypeDef* SpiController::hspi [private]

SPI handle structure.

The documentation for this class was generated from the following files:

- C:/Users/rebor/Desktop/doxygen/app/ **SpiController.h**
- C:/Users/rebor/Desktop/doxygen/app/ **SpiController.cpp**

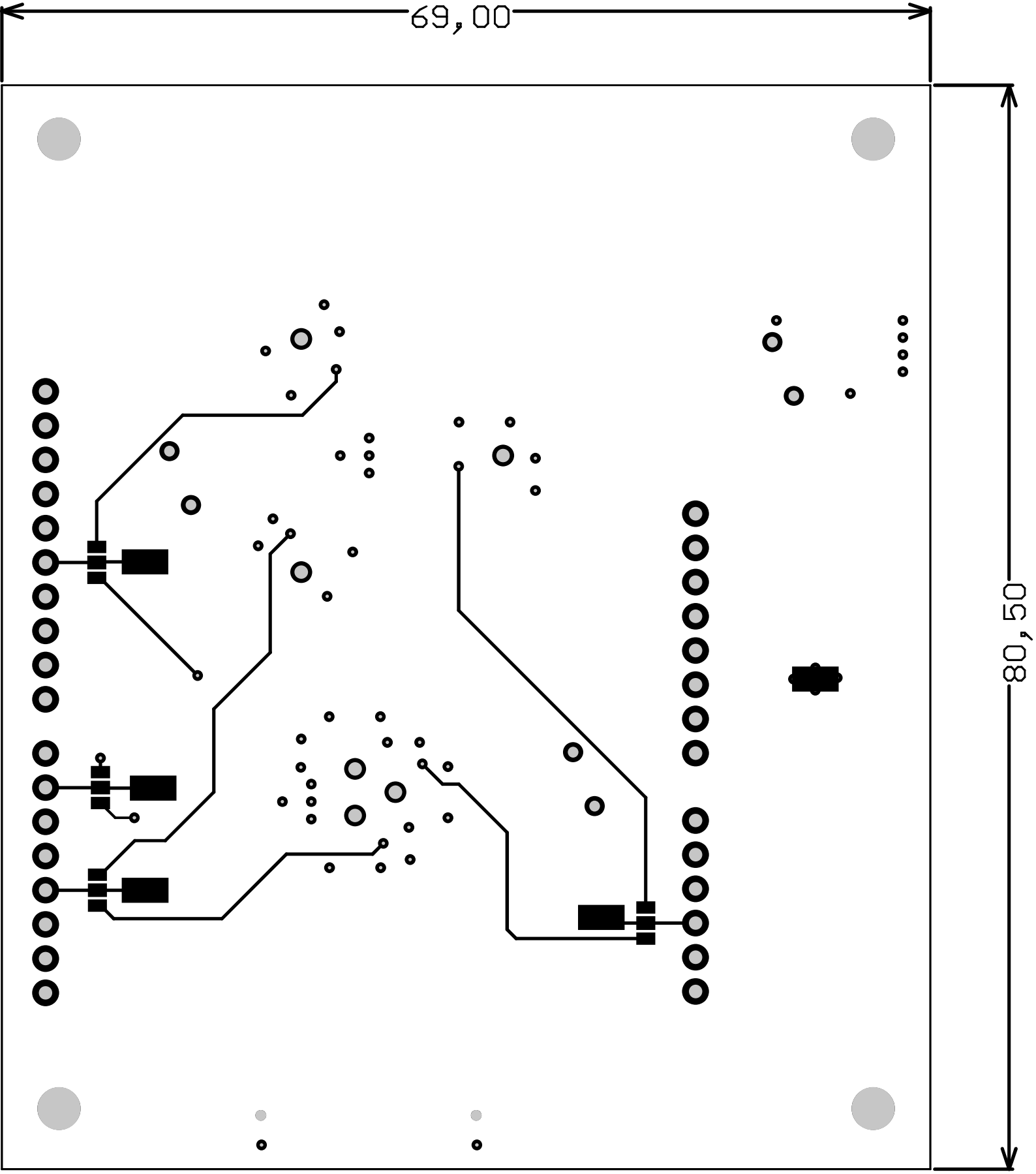
Annexe 5 :

PCB V1.1

Annexe 5.1 et 5.2 :

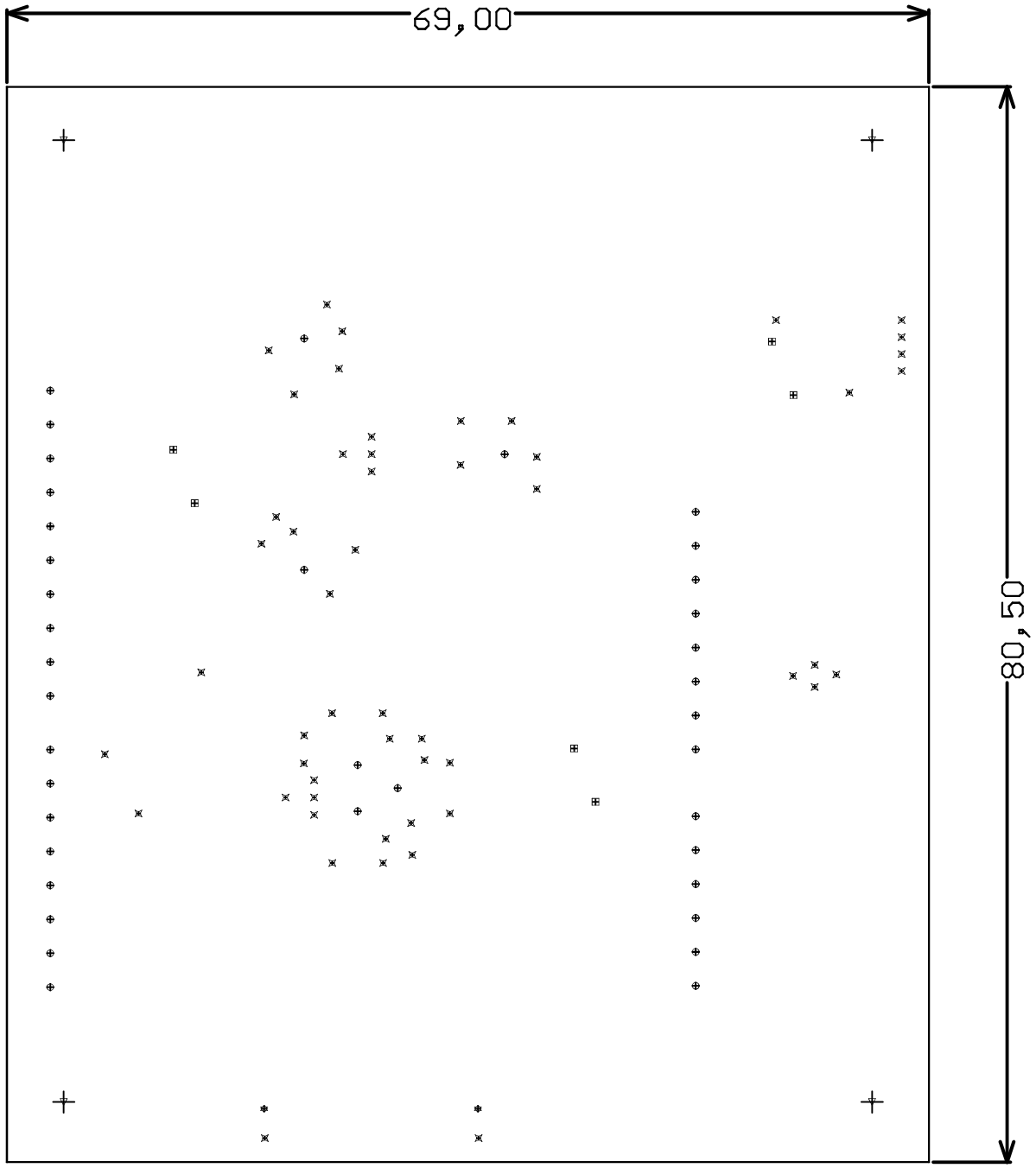
Routage face avant et routage face arrière

62,00
82,08

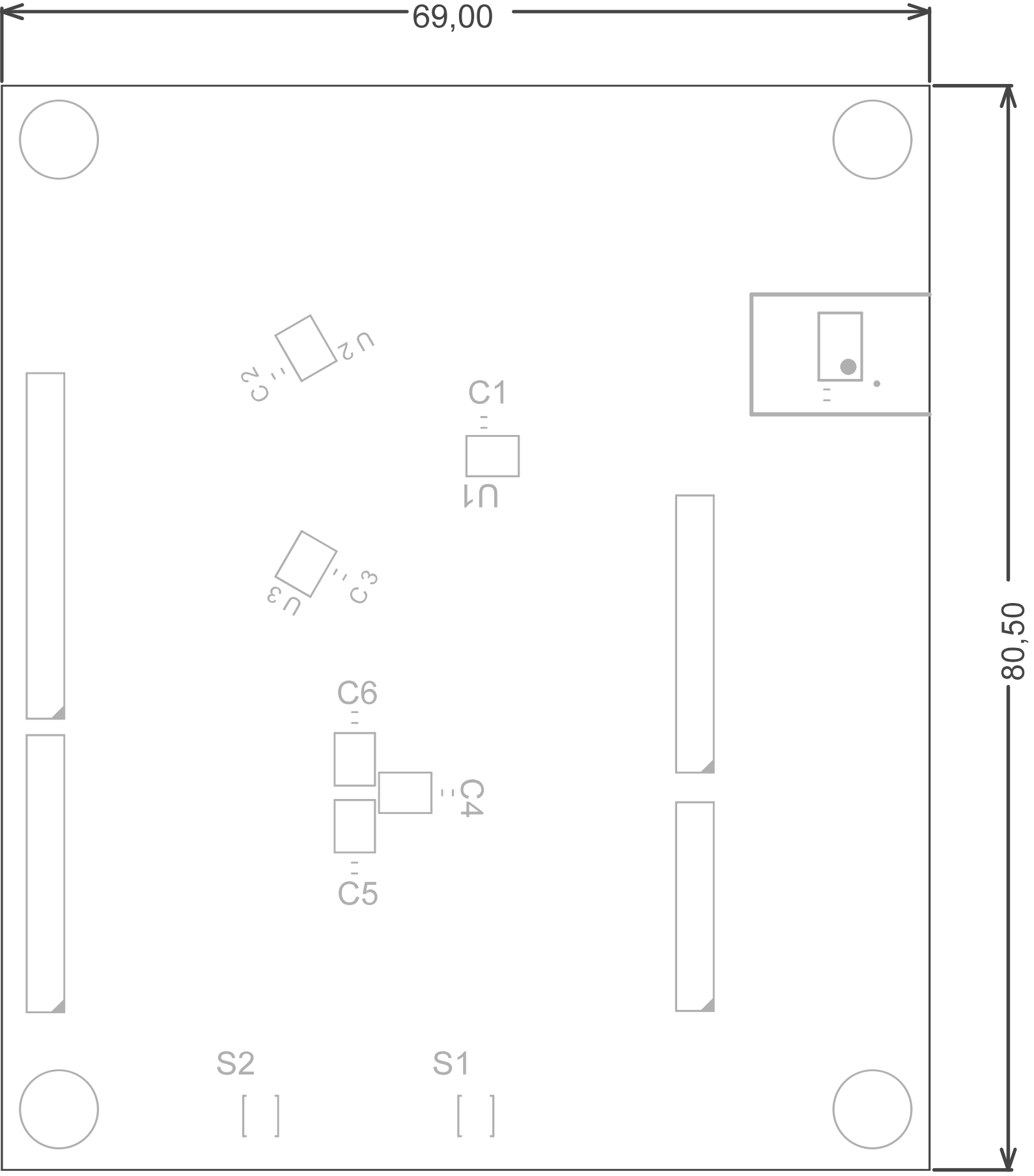

69,00
80,50

Annexe 5.3 et 5.4:

Plan de perçage et implémentation face arrière

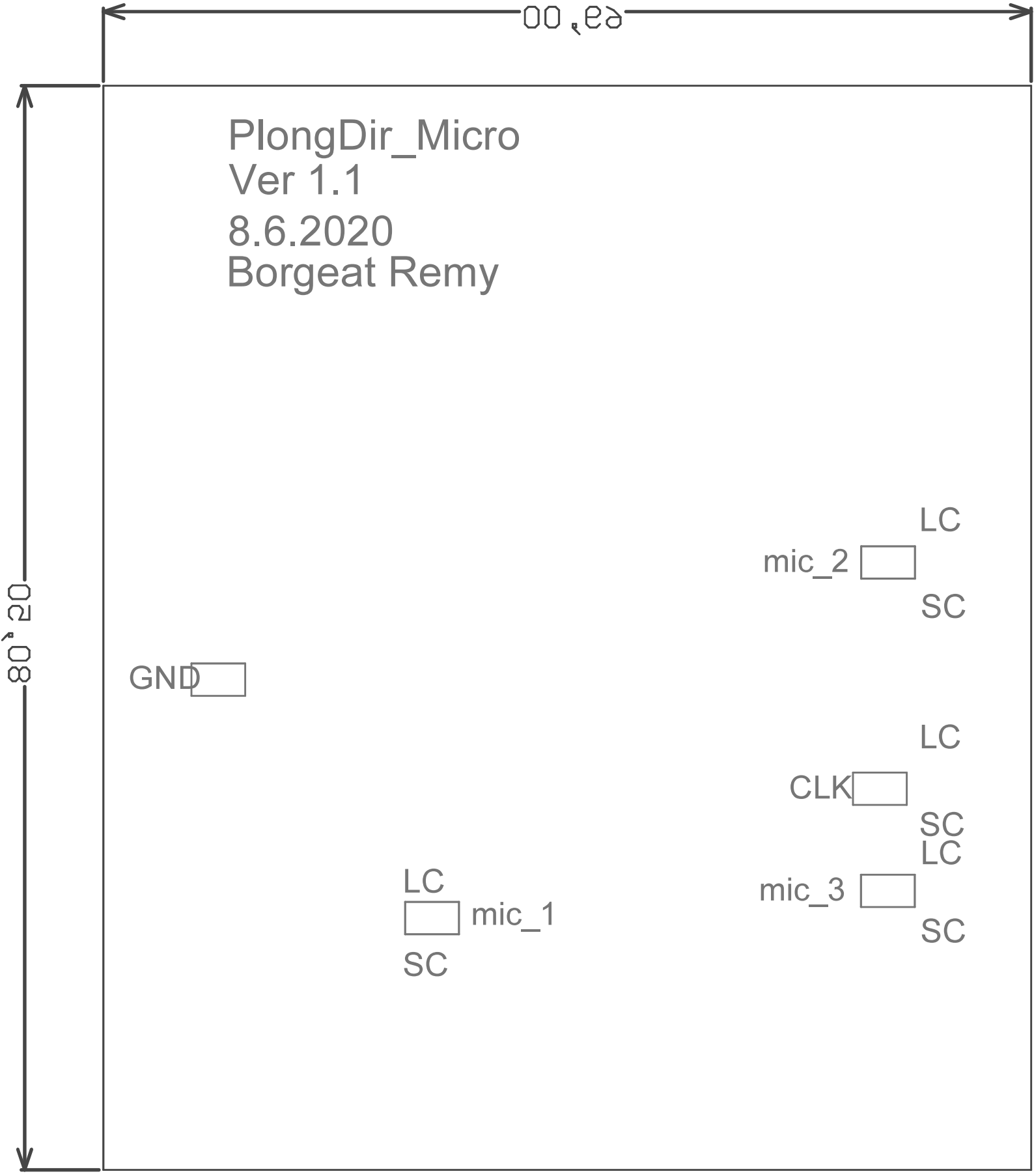


Symbol	Count	Hole Size
⊠	52	9,84mil (0,250mm)
✡	2	31,50mil (0,800mm)
□	6	31,50mil (0,800mm)
○	38	39,37mil (1,000mm)
▽	4	125,98mil (3,200mm)
	102 Total	


69,00
80,50
C1
C2
C3
C4
C5
C6
U1
U2
U3
S1
S2

Annexe 5.5:

Implémentation face avant

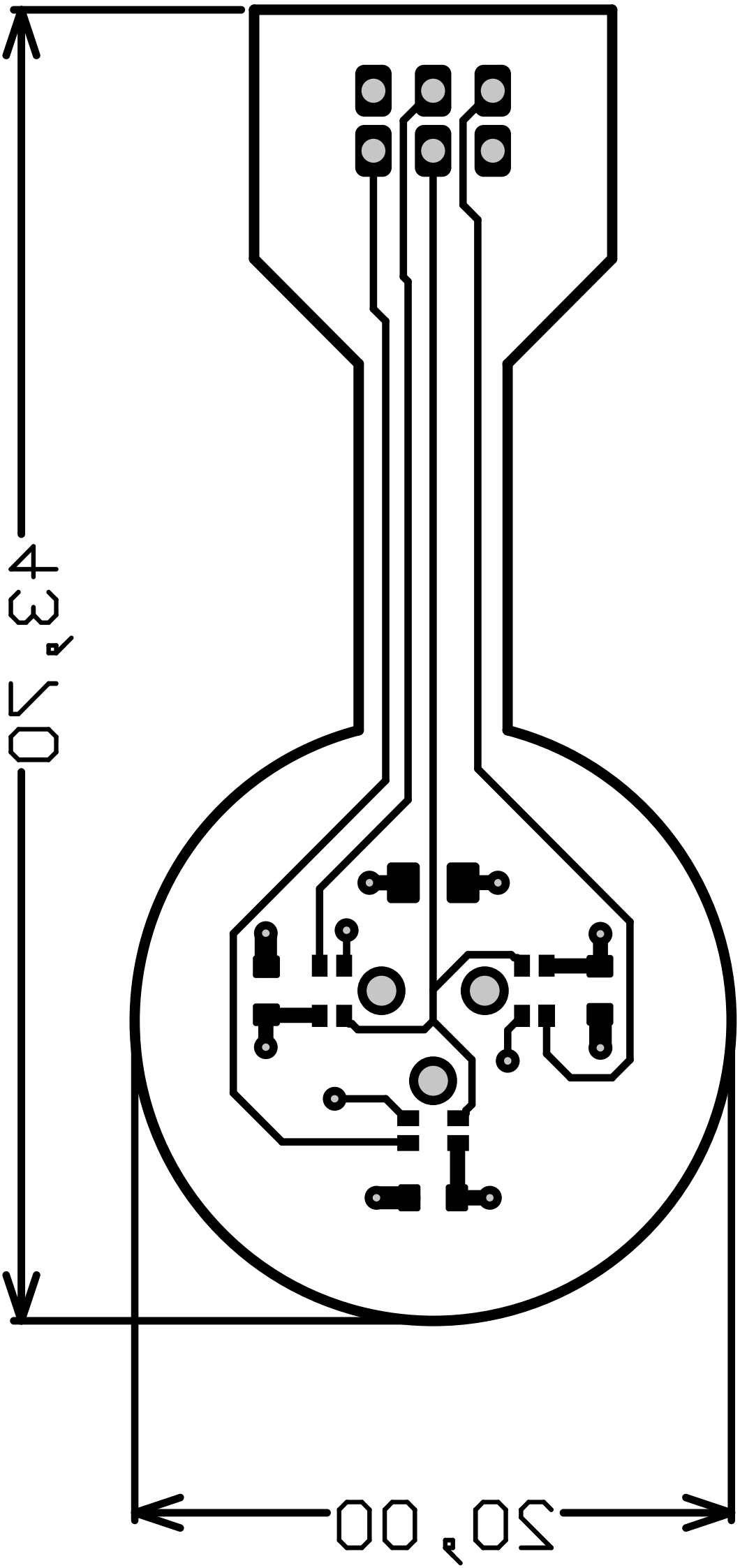

PlongDir_Micro
Ver 1.1
8.6.2020
Borgeat Remy
LC
mic_2
SC
GND
LC
CLK
SC
LC
mic_3
SC
LC
mic_1
SC

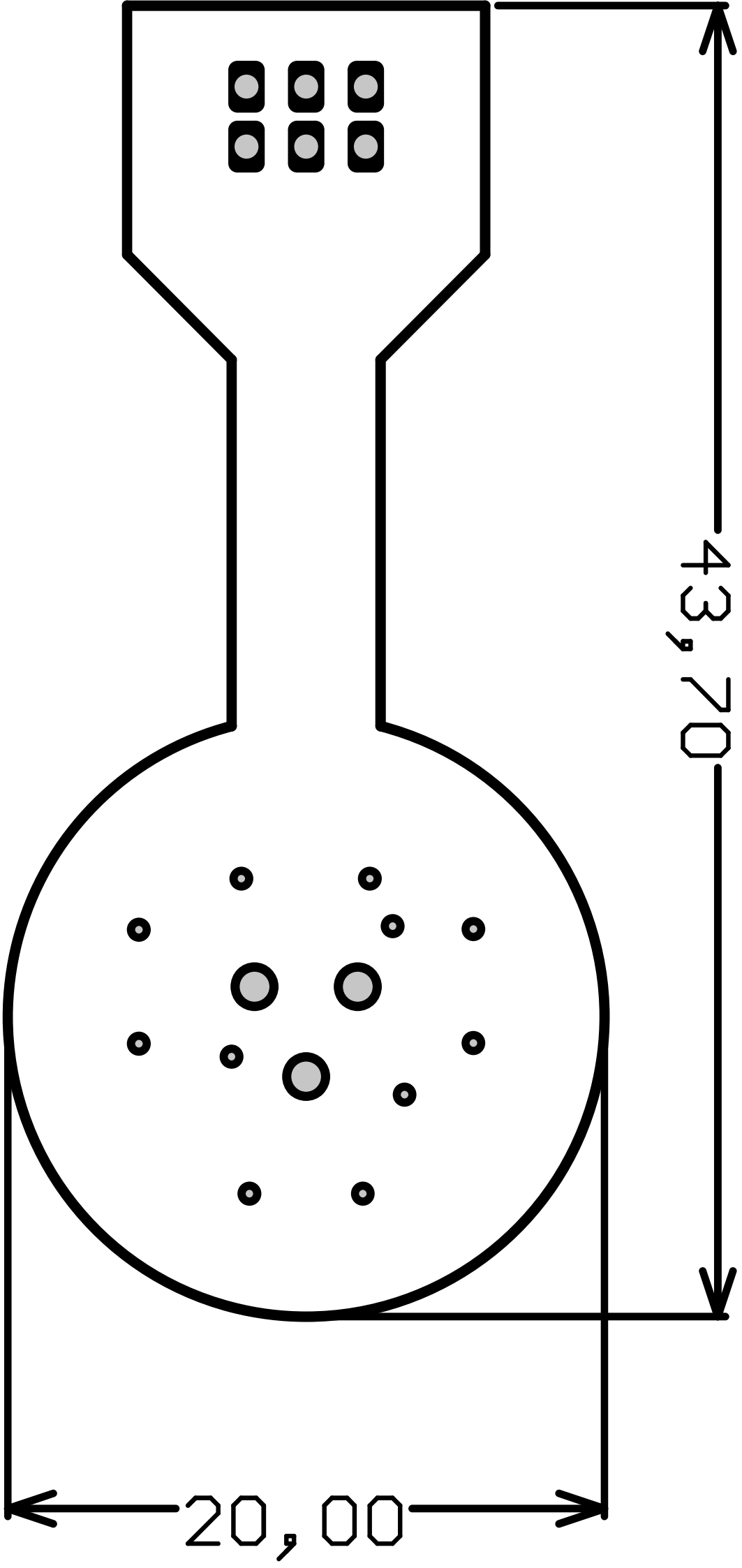
Annexe 6 :

PCB V2.0

Annexe 6.1 et 6.2 :

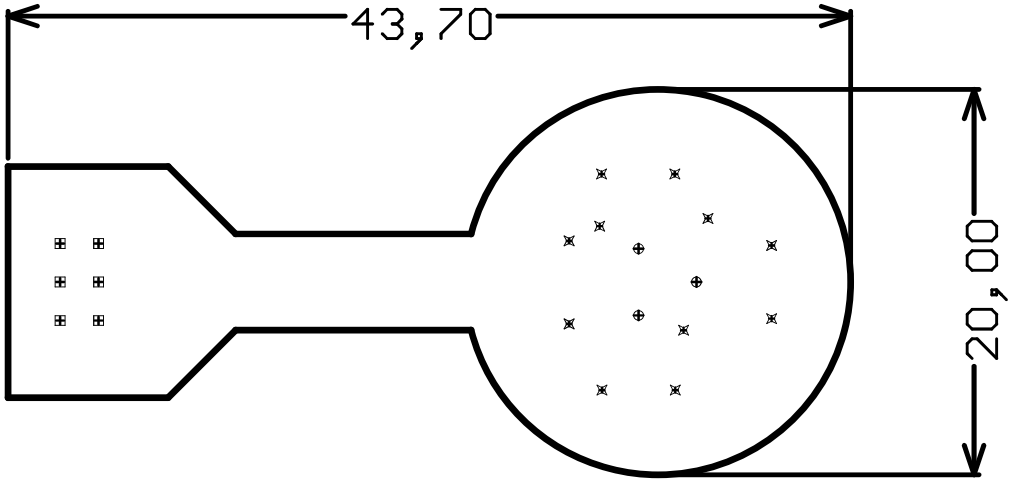
Routage face avant et routage face arrière


43,70
20,00

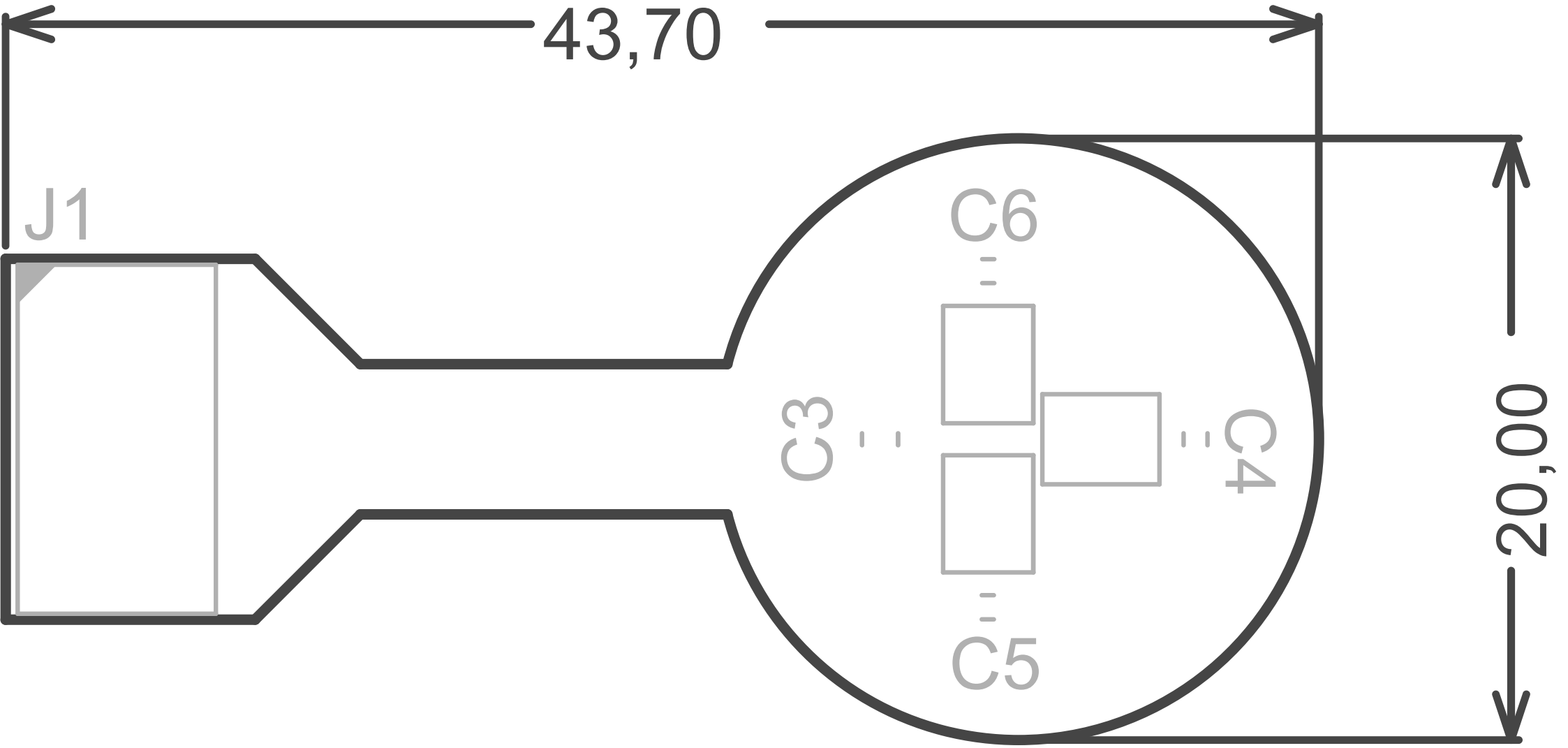

43,70
20,00

Annexe 6.3 et 6.4:

Plan de perçage et implémentation face arrière

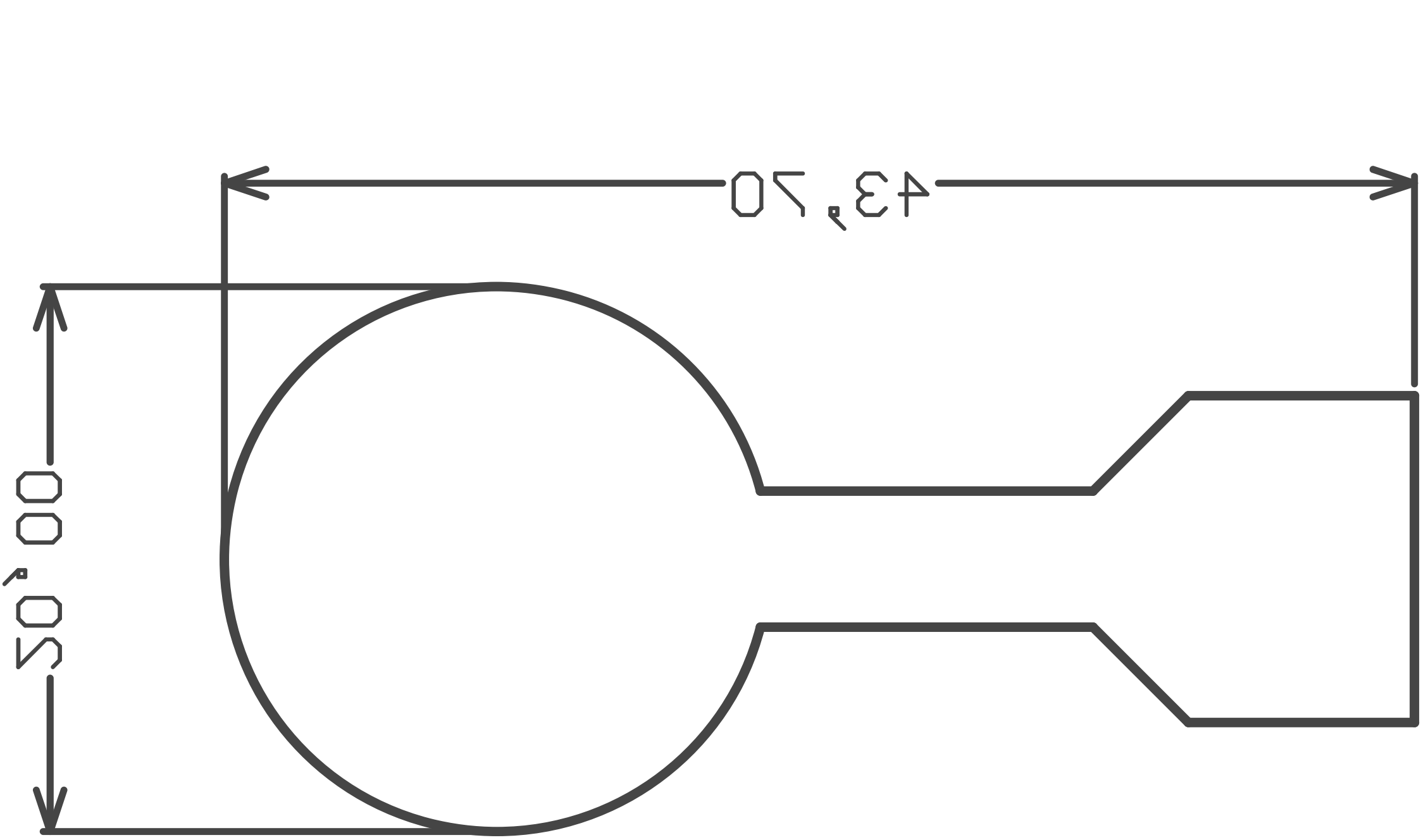


Symbol	Count	Hole Size
○	3	39,37mil (1,000mm)
□	6	31,50mil (0,800mm)
⌘	11	9,84mil (0,250mm)
	20 Total	


43,70
J1
C6
C3
C4
C5
20,00

Annexe 6.5:

Implémentation face avant


43,70
20,00