

Projet de fin d'études

*Imageur ultrasonore pour le contrôle non
destructif à l'aide du langage Python*

BILONGO Danielle, BINUANI Nicolas et ROBIN Gwendoline

Promotion Hall 2018

Table des matières

Introduction	3
1. Présentation générale.....	4
1.1. Présentation de l'entreprise	4
1.2. Contexte	4
1.2.1. Le Contrôle Non Destructif	4
1.2.2. Faisceau	6
1.2.3. Sonde.....	7
1.2.4. A-scan	10
1.2.5. B-scan	10
1.2.6. C-scan.....	11
1.2.7. Avantages et défauts des ultrasons.....	11
1.3. Valeur ajoutée	11
2. Méthodologie.....	13
2.1. Démarrage.....	13
2.2. Modélisation.....	14
2.3. Instrumentation.....	18
3. Résultat et analyse critique.....	25
3.1. Résultats.....	25
3.1.1. Modélisation.....	25
3.1.2. Instrumentation	26
3.2. Critères d'évaluation.....	28
3.2.1. FPS	28
3.2.2. Critère visuel	33
3.3. Analyse critique du projet.....	35
Conclusion.....	37
Annexes	38
Annexe 1 : Gestion de projet.....	38
1.1. Gestion générale de la répartition des tâches au sein du groupe.....	38
1.2. La matrice RACI.....	38
Annexe 2 : Planning	40

Annexe 3 : Difficultés rencontrées	41
3.1. Apprentissage d'un langage : Python	41
3.2. Installation des librairies	41
3.3. Choix de l'architecture	42
3.4. Débogage.....	43
3.5. Récupération des données issues de l'appareil	43
Annexe 4 : Remerciements.....	44
Annexe 5 : Glossaire	45
Annexe 6 : Table des illustrations.....	46

Introduction

Nous sommes trois étudiants en dernière année de cycle ingénieur, Gwendoline Robin, Danielle Bilongo et Nicolas Binuani qui dans le cadre de notre Projet de Fin d'Etudes, avons choisi le sujet d'un « Imageur Ultrasonore en langage Python ». L'entreprise DB-SAS et The Phased Array Company avec laquelle nous avons travaillé par l'intermédiaire de M. CARCREFF.

Ce projet a pour but d'implémenter une GUI (Graphic User Interface) et de s'interfacer à un générateur d'onde acoustique développé par l'entreprise. L'utilisation de cet appareil permet de vérifier la présence de défaut dans des matériaux traduit par l'apparition d'un écho ou perturbation dans un signal impulsionnel émis par le générateur.

Pour décomposer ce projet, nous avons à la suite du cours de gestion de projet pris pour parti d'effectuer dans un premier temps un prototypage avec des données fixes. Ensuite, nous avons implémenté la partie instrumentation avec les données en temps réel.

Ce projet a plusieurs objectifs, en effet, la même interface a déjà été implémentée sous d'autres langages comme Matlab, LabView, C++ mais pas en Python, ce qui fait une première nouveauté. Ensuite, nous cherchons à comparer la vitesse de rafraichissement des images entre les différents langages. De plus, Python étant un langage open-source et gratuit à télécharger contrairement à Matlab par exemple permet un nouvel environnement de développement pour le logiciel de l'entreprise.

Notre travail s'inscrit donc dans une durée de 4 mois (196 heures théoriques), le projet suppose d'être repris soit en stage ou dans un projet de fin d'études à venir afin de pouvoir être améliorer et d'ajouter de nouvelles fonctionnalités. Nous avons livré plusieurs exécutable fonctionnels et corrects au client.

Nous allons donc maintenant présenter notre projet en détails.

Tout d'abord, nous ferons une courte présentation de l'entreprise, puis nous poserons le contexte du projet ainsi que les notions importantes pour le comprendre avant de terminer sur la mise en avant de ce que notre travail a apporté à l'entreprise et dans notre formation.

Ensuite, nous présenterons notre méthodologie. Et pour finir, nous vous montrerons nos résultats, comment nous avons pu évaluer notre travail enfin, nous ferons une analyse critique de notre projet.

Puis, nous concluons sur d'éventuelles perspectives d'évolutions de notre projet ainsi que l'avis du client.

1. Présentation générale

1.1. Présentation de l'entreprise

Nous avons réalisé ce projet en collaboration avec une entreprise se nommant DB-SAS. Il s'agit d'une société qui se situe à Nantes en Loire Atlantique dans les Pays de la Loire (44). Celle-ci possède un bureau d'études composé d'une dizaine d'ingénieurs et docteurs en électronique et informatique. Son chiffre d'affaire est d'environ 1.5 M€ et est en progression chaque année.

Son domaine d'expertise est la recherche et le développement du contrôle non destructif couramment appelé CND par la méthode des ultrasons exclusivement.

Elle exerce son activité en France et à l'international dans divers domaines d'activités : les transports, la production industrielle, le transport de matières premières ou bien la génération d'énergie.

Pour finir, les activités de la société sont diverses et variées : la recherche et le développement pour le Contrôle Non Destructif, le développement de carte électronique ou bien même du conseil en inspection par ultrasons.

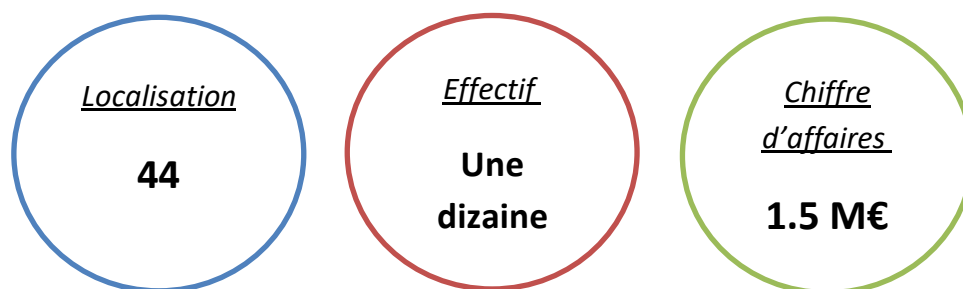


Figure 1 - Présentation rapide de l'entreprise cliente

1.2. Contexte

1.2.1. Le Contrôle Non Destructif

Depuis plus de 60 ans, il existe des appareils de recherche de défauts par ultrasons et sont utilisés dans les applications industrielles et même médicales. Dans les années 40, on connaît les mêmes lois fondamentales de la physique qui régissent la propagation des ondes sonores à haute fréquence pour rechercher les indications cachées, telles les fissures, les vides, la porosité et d'autres discontinuités internes dans les métaux, les composites, les plastiques et les céramiques et pour mesurer l'épaisseur des matériaux et analyser leurs propriétés. L'inspection par ultrasons est une méthode sûre et non destructive utilisée depuis longtemps dans un grand nombre d'industries de base de service, de processus et de fabrication, surtout dans les applications d'inspection de soudures et des charpentes métalliques.

L'expansion de l'inspection par ultrasons se compare facilement avec les développements dans le secteur des composants électroniques et plus tard dans le secteur de l'informatique. Les premières études en Europe et aux États-Unis dans les années 1930 ont montrées que les ondes sonores à haute fréquence se reflètent sur les indications cachées de façon prévisible, créant ainsi des échos caractéristiques qu'il était possible d'afficher sur les écrans des oscilloscopes. Le développement de sonars pendant la deuxième guerre mondiale a également contribué aux recherches dans le domaine des ultrasons. En 1945, le chercheur américain Floyd Firestone a breveté un appareil appelé Supersonic Reflectoscope, considéré comme le premier appareil de recherche de défauts par ultrasons utilisant la technique par réflexion fréquemment utilisée aujourd'hui. Cet appareil est à l'origine des nombreux appareils commerciaux présentés dans les années suivantes. À la fin des années 40, des chercheurs au Japon ont été les premiers à utiliser l'inspection par ultrasons dans le cadre de diagnostics médicaux. Ils ont utilisé un des premiers équipements de B-scan donnant une image à deux dimensions des couches des tissus. Dans les années 1960, on utilisait les premiers scanners médicaux pour rechercher et avoir un aperçu des tumeurs, des caillots sanguins et d'autres troubles comparables. Dans les années 70, le lancement de mesureurs d'épaisseur de précision a introduit l'inspection par ultrasons dans un grand nombre d'activités de fabrication nécessitant plus de sécurité et de test. (Exemple : dans l'aviation, le transport ferroviaire, le bâtiment et dans l'imagerie médicale non invasive).

Les dernières avancées des appareils à ultrasons reposent sur les techniques de traitement numérique du signal et les microprocesseurs bon marché offerts à partir des années 1980. Ces développements ont conduit à la dernière génération de petits appareils portables extrêmement fiables et de systèmes en ligne destinés à la recherche de défauts (Cf. Figure 2), à la mesure d'épaisseur et à la représentation acoustique.



Figure 2 - Exemple de contrôle non destructif sur une aile d'avion

1.2.2. Faisceau

Le temps de réponse d'un système d'inspection par ultrasons dépend de plusieurs facteurs. On a la sonde, le type d'appareil et ses réglages, ainsi que les propriétés acoustiques du matériau inspecté. La réponse des sondes multiéléments représenté sous un faisceau (Cf. Figure 3), dépend des paramètres de conception de la sonde que nous allons expliquer influençant l'allure des signaux reçus et donc le faisceau ultrasonore.

Quatre paramètres importants ont un certain nombre d'effets interdépendants sur le rendement de la sonde. Voici ces quatre paramètres et quelques explications.

Fréquence -- La fréquence d'inspection a un effet significatif sur la propagation du faisceau. Dans la pratique, les fréquences élevées peuvent offrir un meilleur rapport signal sur bruit que les basses fréquences, car ils ont un potentiel de focalisation plus grand et donc un point focal resserré et optimisé. Par contre, la pénétration diminue lors d'utilisation de fréquences élevées, du fait d'une augmentation de l'atténuation du matériau. Les applications comportant de très longs parcours sonores ou un matériau à forte atténuation ou diffusion nécessitent l'utilisation de basses fréquences. Généralement, les sondes multiéléments industrielles sont offertes dans des fréquences variant de 1 MHz à 15 MHz, dans notre cas elle sera de 5 MHz.

Taille de l'élément -- Au fur et à mesure que la taille des éléments d'un réseau diminue, la capacité de déflexion du faisceau augmente. Toutefois, si la taille des éléments est inférieure à une longueur d'onde, de puissants lobes secondaires indésirables se formeront (explication à la fin du chapitre).

Nombre d'éléments -- Au fur et à mesure que le nombre d'éléments augmente, la zone de couverture physique de la sonde et sa sensibilité, ainsi que sa capacité de focalisation et de déflexion, augmentent également. Il faut cependant souvent trouver un compromis entre l'utilisation de sondes avec un grand nombre d'éléments, permettant une plus grande liberté dans l'inspection, la complexité et les coûts du système.

Pas et ouverture -- Le pas correspond à la distance entre les éléments et l'ouverture correspond à la taille réelle d'un élément émetteur. Or, cette ouverture est habituellement composée d'un groupe d'éléments qui envoie des impulsions simultanément. Pour optimiser l'étendue de déflexion, le pas doit être de petite taille. Une grande ouverture est nécessaire pour obtenir une sensibilité optimale et une forte focalisation. Les appareils multiéléments modernes supportent habituellement des ouvertures de lois focales allant jusqu'à 16 éléments. Les systèmes les plus avancés supportent des ouvertures allant jusqu'à 32 éléments, voire même 64 éléments.

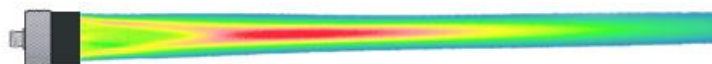


Figure 3 - Représentation d'un faisceau US (en rouge les zones de haute énergie et en bleu et vert les zones plus basse d'énergie)

1.2.3. Sonde

Ces sondes existent dans une grande variété de tailles, de fréquences et de boîtiers, mais la structure interne de la plupart d'entre elles reste identique. (Cf. Figure 4) En règle générale, l'élément actif de la sonde est constitué d'un disque mince, d'un carré ou d'un rectangle de céramique piézo-électrique convertissant l'énergie électrique en énergie mécanique (vibrations ultrasonores), et vice versa. La sonde est protégée par une plaque d'usure ou une lentille acoustique et supportée par un matériau amortisseur qui baisse le niveau sonore de la sonde après la génération de l'impulsion sonore. Ce sous-ensemble ultrasonore est monté dans un boîtier avec les connexions électriques nécessaires. Les sondes répandues de contact, à ligne à retard et d'angle utilisent cette conception de base. Les sondes à émission-réception séparées, généralement utilisées dans les applications de contrôle de la corrosion, ont toutefois une conception différente. En effet, elles ont des éléments différents pour l'émission et la réception séparés par un isolant acoustique, elles n'ont pas de support et elles ont une ligne à retard intégrale à la place de la plaque d'usure ou la lentille.

Les sondes les plus couramment utilisées pour le contrôle non destructif par ultrasons ont les propriétés fonctionnelles fondamentales suivantes :

- **Type** -- Les sondes sont classées selon leur type d'utilisation, comme les sondes de contact, à ligne à retard, d'angle ou d'immersion. Les caractéristiques des matériaux inspectés (comme la rugosité de surface, la température, l'accessibilité et la position des défauts dans le matériau, ainsi que la vitesse d'inspection) influent sur le choix du type de sonde.
- **Diamètre** -- Le diamètre de l'élément actif de la sonde, normalement logé dans un boîtier un peu plus grand.
- **Fréquence** -- Nombre de cycles d'onde complétés en une seconde, normalement exprimés en kilohertz (kHz) ou en mégahertz (MHz). La plupart des inspections par ultrasons industrielles sont effectuées à une gamme de fréquences de 500 kHz à 20 MHz. La plupart des sondes se situent donc dans cette gamme, mais il existe aussi sur le marché des sondes de moins de 5 kHz et de plus de 200 MHz. La pénétration augmente sous les basses fréquences, tandis que la résolution et la netteté de focalisation augmentent sous les fréquences élevées.

- **Bande passante** -- Spectre de fréquences entre les limites d'amplitude définies pour la réponse d'un appareil. Dans ce contexte, il convient de noter que les sondes CND typiques ne produisent pas d'ondes acoustiques à une fréquence unique pure, mais plutôt dans une gamme de fréquences situées au centre des fréquences nominales définies.
- **Durée de la forme d'onde** -- Nombre de cycles d'onde générés par la sonde chaque fois qu'elle est excitée. Une sonde à bande passante étroite a plus de cycles qu'une sonde à large bande passante. Le diamètre de l'élément, le matériau, le réglage électrique et la méthode d'excitation de la sonde influencent tous la durée de la forme d'onde
- **Sensibilité** -- Relation entre l'amplitude de l'impulsion d'excitation et celle de l'écho reçu d'une cible donnée.

Quand les ondes provenant de deux ou de plusieurs sources interagissent les unes avec les autres, leur déphasage se traduit par une augmentation ou une diminution de l'énergie des ondes au point de combinaison. Lorsque des ondes élastiques de la même fréquence se rencontrent de telle sorte que leur déplacement est parfaitement synchronisé (en phase ou en angle de déphasage de zéro degré), les énergies des ondes s'additionnent pour créer une onde de grande amplitude. Si elles se rencontrent de telle sorte que leur déplacement est exactement opposé (déphasage à 180°), les énergies des ondes s'annulent. Aux angles de déphasage de 0° à 180° , on constate une série de stades intermédiaires entre l'addition complète et l'annulation complète. La variation du moment d'émission des ondes à partir d'un grand nombre de sources permet à la fois de s'orienter et de focaliser le front d'ondes regroupées. C'est le principe essentiel qui sous-entend l'inspection par ultrasons multiéléments.

Dans les sondes à ultrasons conventionnels, les interférences constructives et destructives créent le champ proche et le champ lointain, ainsi que les divers gradients de pressions. En outre, une sonde d'angle à ultrasons conventionnels est munie d'un seul élément qui génère une onde dans un sabot. Les points sur ce front d'ondes ont différents intervalles de retard selon la forme du sabot. Ces derniers sont des intervalles mécaniques, par opposition aux retards électroniques utilisés dans l'inspection par ultrasons multiéléments. Lorsque le front d'ondes atteint le fond de la pièce inspectée, il peut être visualisé comme une série de sources ponctuelles et expliqué leur propagation de proche en proche (principe de Huygens et par réflexion la propagation est définie par les lois de Snell-Descartes).

En inspection par ultrasons multiéléments, les effets prévisibles de la combinaison et de l'annulation sont utilisés pour former et orienter le faisceau ultrasonore. L'excitation séparée d'éléments ou de groupes d'éléments à des moments différents crée une série d'ondes à partir de sources ponctuelles qui se combinent en un seul front d'ondes se déplaçant sous un angle choisi. Cet effet électronique est semblable

au retard mécanique produit avec un sabot classique, mais il peut être programmé en modifiant le modèle des retards.

Les éléments sont généralement excités en groupe de 4 à 32 afin d'améliorer la sensibilité effective en agrandissant l'ouverture, ce qui réduit les faisceaux indésirables et permet une focalisation nette.

Les échos sont captés par les différents éléments ou groupes d'éléments, et ils sont décalés dans le temps de façon à compenser les divers retards du sabot, et puis ils sont additionnés. Contrairement aux sondes monoéléments à ultrasons conventionnels, qui fusionnent les effets de toutes les composantes du faisceau qui frappent leur surface, les sondes multiéléments peuvent trier les fronts d'ondes réfléchis selon leur moment d'arrivée et leur amplitude à chaque élément. Lors du traitement par le logiciel de l'appareil, chaque loi focale retournée représente la réflexion à partir d'une composante particulière de l'angle du faisceau, un point particulier sur le parcours linéaire ou une réflexion à partir d'une profondeur focale particulière. L'information sur les échos peut être affichée en plusieurs formats standards.

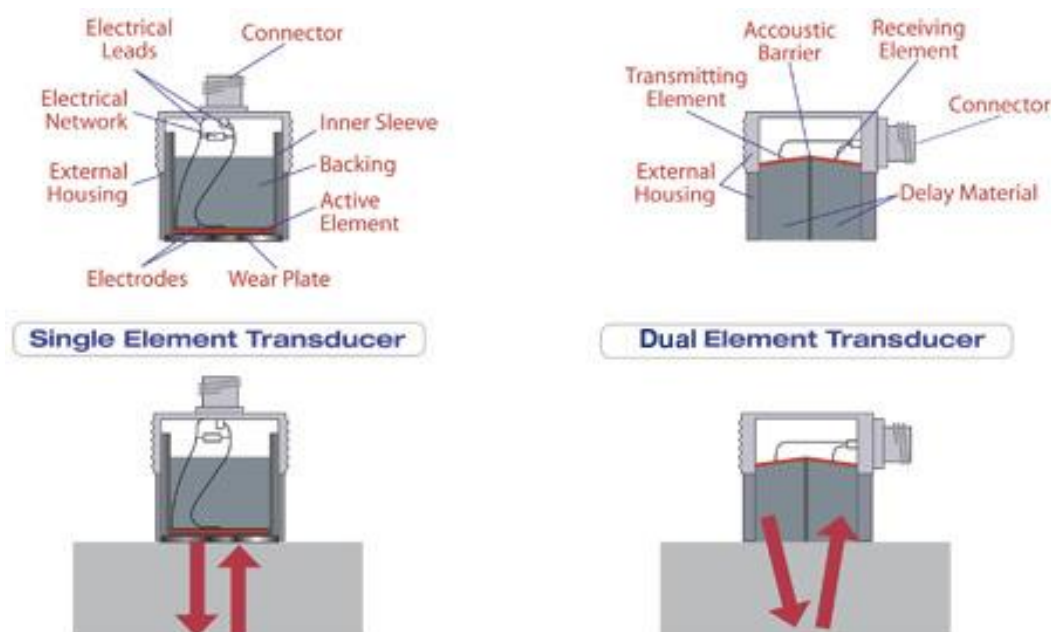


Figure 4 - Schémas illustrant la composition d'une sonde ultrasonore mono élément et à deux éléments

1.2.4. A-scan

Les appareils à ultrasons enregistrent normalement deux paramètres fondamentaux d'un écho : sa taille (amplitude) et l'endroit dans le temps où il apparaît par rapport au point zéro (temps de propagation de l'émission de l'impulsion). Le temps de propagation, à son tour, est généralement corrélé avec la distance ou la profondeur du réflecteur, en se basant sur la vitesse de propagation de l'onde ultrasonore du matériau contrôlé et sur la relation simple : $\text{Distance} = \text{vitesse} \times \text{temps}$.

La présentation de base des données d'une onde ultrasonore est la vue A-scan qui montre l'amplitude de l'écho et le temps de transit sur une grille simple dont l'axe vertical représente l'amplitude et l'axe horizontal représente le temps. Ainsi, on détectera la présence d'un défaut dans le matériau inspecté via l'apparition d'un écho après l'impulsion on pourra localiser ce défaut par rapport à la profondeur du matériau (Cf. Figure 5).

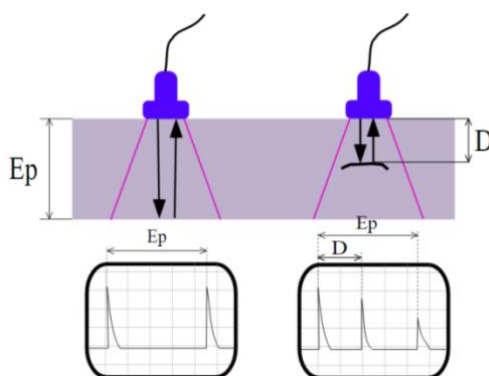


Figure 5 - Exemple d'A-scan (avec la détection du défaut à la distance D)

1.2.5. B-scan

Cette information peut aussi être présentée comme un B-scan de valeur unique. Ce format est communément utilisé avec les appareils de recherche de défauts classiques et les mesureurs d'épaisseur de la corrosion pour tracer la profondeur des réflecteurs en fonction de leur position. L'épaisseur est tracée en fonction du temps ou de la position, tandis que le balayage de la sonde le long de la pièce permet de tracer le profil de profondeur de la pièce. La corrélation des données ultrasonores avec la position réelle de la sonde permet de tracer une image proportionnelle, et permet de corréler les données aux zones spécifiques de la pièce contrôlée et de les tracer. Ce traçage de position se fait généralement à l'aide de dispositifs électromécaniques appelés codeurs. Ces codeurs sont utilisés soit dans des supports, dans le cas des inspections manuelles, ou dans des systèmes automatisés qui permettent de déplacer la sonde à l'aide d'un scanner programmable avec commande à moteur. Dans les deux cas, le codeur enregistre l'emplacement de chaque acquisition de données par rapport à un modèle de balayage donné défini par l'utilisateur et à la résolution en index.

1.2.6. C-scan

Le C-scan est une autre possibilité de présentation. Il s'agit d'une représentation des données en deux dimensions affichée comme une vue de dessus ou une vue planaire de la pièce à inspecter. Elle ressemble à une image radiologique quant à la perspective graphique, où la couleur représente l'amplitude ou la profondeur du signal. Les images planaires peuvent être générées sur les pièces à plat en collectant les données des emplacements sur les axes x et y ou sur les pièces cylindriques en collectant les données aux positions axiales et angulaires.

1.2.7. Avantages et défauts des ultrasons

Les avantages des systèmes à ultrasons multiéléments comparés à ceux conventionnels est la possibilité d'utiliser plusieurs éléments assemblés et formant une sonde pour orienter, focaliser et balayer les faisceaux. Ainsi, on peut balayer une surface ou un élément sous plusieurs angles sans avoir à déplacer la sonde. On peut aussi optimiser la forme et la taille des faisceaux via une focalisation électronique et non plus simplement mécanique. Ceci permet d'améliorer nettement le ratio signal sur bruit et d'obtenir plus simplement des C-scans.

Un phénomène associé aux ultrasons et la propagation via les sondes est la génération de lobes secondaires parasites. Il s'agit d'un phénomène de propagation de l'énergie acoustique sur d'autres angles différents du trajet principal. Ces trajectoires indésirables peuvent être réfléchi dans la surface étudiée et donc provoquer une mauvaise représentation de la qualité de la surface. La taille du pas, le nombre d'éléments, la fréquence et la bande passante peuvent être des facteurs affectant l'amplitude des lobes secondaires.

1.3. Valeur ajoutée

Dans des milieux industriels complexes, comme le nucléaire, l'aéronautique, les vibrations et les nombreuses contraintes mécaniques provoquent à la longue, des fissures dans les pièces présentant des possibilités de concentrations de contraintes.

Afin d'anticiper toutes prises de risques, la sûreté des infrastructures constituent des enjeux majeurs, pour garantir la sécurité lors des transports aériens de biens et de personnes.

Le contrôle non destructif (CND) est l'une des composantes du « manufacturing avancé », technologie générique clé destinée à mettre en œuvre l'industrie du futur. Il permet de contrôler, sans les altérer, des pièces en sortie d'usinage ou lors de maintenance, pour détecter des défauts dans des matériaux, effectuer des mesures de corrosion, vérifier la conformité des soudures. Tels sont les enjeux du CND par

ultrasons qui innove depuis une dizaine d'années avec le développement, des techniques de détection « multiéléments ».

Nous nous intéressons donc dans ce projet au CND par ultrasons, modalité qui consiste à émettre des ondes acoustiques dans le matériau à inspecter. Notre but est de réaliser un imageur ultrasonore. Pour cela, nous avons besoin d'une interface pour permettre à l'utilisateur d'observer les mesures et modifier les paramètres qu'ils souhaitent. A la demande du client, ce projet s'est effectué en langage Python. En effet, le client possède déjà un prototype fonctionnel sous Matlab. L'avantage de Python est qu'il s'agit d'un langage polyvalent. Il est possible de l'utiliser sur un logiciel libre et gratuit contrairement à Matlab. L'usage d'un logiciel gratuit permettrait à quiconque d'utiliser l'imageur ultrasonore sans avoir à souscrire Matlab. Nous voulions proposer une solution facilement intégrable dans le foyer grand public.

Cet aspect économique nous a beaucoup motivés de par le défi que cela relève : pouvons-nous proposer un produit abordable avec les moyens à disposition et le temps imparti ?

L'objectif est que l'utilisateur puisse visualiser les échos en temps réelle et en déduire une information spatiale sur l'objet inspecté, tel que la profondeur d'un défaut dans un matériau.

A travers ce projet, nous avons aussi pu apporter des conseils au client sur le langage le plus adapté à l'utilisation final. En effet, nous vous présenterons par la suite les critères de performances qui nous ont permis d'émettre une hypothèse sur le langage le mieux adapté. Nous avons réalisé deux prototypes :

- Un premier prototype fixe avec des données aléatoires
- Un deuxième prototype pour l'acquisition en temps réel à l'aide du système d'acquisition OEMPA d'AOS NDT

Le premier prototype est notre premier livrable suite auquel grâce à notre avancement, le client n'a pas hésité à nous confier son matériel pour l'acquisition temps réel.

Ces deux prototypes vous seront présentés dans le chapitre suivant. Ainsi, nous avons voulu à travers ce projet, tester notre capacité à faire communiquer différentes technologies entre elles dans un langage qui nous était très peu familier.

Pour que le programme soit plus facilement accessible, nous avons créé des exécutables afin que l'utilisateur n'ait pas nécessairement à télécharger un interpréteur Python.

Ce projet a nécessité un travail en groupe, une grande communication entre les membres. Nous avons eu l'opportunité de réaliser un projet de A à Z qui réunit de grands enjeux, tout en exploitant les compétences de chacun.

2. Méthodologie

2.1. Démarrage

Il a été primordial de se documenter au démarrage de ce projet. En effet, le langage utilisé tout au long de ce projet est Python, il a donc été important de très vite le maîtriser.

Python est un langage informatique interprété, c'est-à-dire que le code sera directement exécuté sans passer par une phase de compilation. Il est de typage fort ce qui signifie que chaque variable, à chaque instant, ne peut être que d'un type. Par contre, ce typage est dynamique, ce qui signifie qu'à la déclaration d'une variable, il n'est pas nécessaire de déclarer son type : c'est la valeur qu'on affectera à la variable qui déterminera son type.

Il permet un langage simplifié proche de celui de l'utilisateur qui sera, via l'interpréteur, traduit en code machine. Adapté à de nombreuses situations grâce à des bibliothèques spécialisées à chaque traitement, il est surtout utilisé comme langage de script. C'est un langage très prisé dans le monde scientifique. Il existe plusieurs versions de Python dont la plus récente est la version 3.6.

Pour notre projet, nous avons choisi d'utiliser la version récente afin d'avoir un choix plus large en ce qui concerne les librairies.

Nous utilisons l'IDLE «Integrated DeveLopment Environment » au départ pour coder, il s'agit d'un environnement de développement intégré pour le langage Python. Cependant, nous nous sommes très vite rendu compte que nous étions limités étant donné qu'il n'est pas possible de mettre des points d'arrêts pour déboguer et étudier les variables.

L'interpréteur choisi par la suite a été Pycharm. PyCharm est un environnement de développement intégré utilisé pour programmer en Python. Il offre la coloration syntaxique, l'auto-complétion du code, la vérification de code en direct, un débogueur graphique, l'intégration avec les principaux gestionnaires de versions, la gestion des environnements virtuels, la gestion des tests. Il permet aussi de pouvoir installer des librairies Python sans aucun téléchargement au préalable.

Les librairies les plus importantes utilisées sont : numpy, matplotlib et tkinter. Ces bibliothèques sont très utiles pour travailler avec des images sous Python.

NumPy est l'une des bibliothèques Python les plus importantes pour la manipulation de matrice. Elle permet de gérer des tableaux, de les importer, de les exporter et de travailler dessus.

Matplotlib est puissante et complète pour la visualisation des données avec de nombreuses possibilités de personnalisation.

Tkinter est un module intégré à la bibliothèque standard de Python. Il offre un moyen de créer des interfaces graphiques. Il est disponible sur Windows et la plupart des systèmes Unix et aura donc toutes les chances d'être portables d'un système à l'autre. Il existe d'autres bibliothèques pour créer des interfaces graphiques cependant tkinter à l'avantage d'être disponible par défaut, sans nécessiter une installation supplémentaire.

2.2. Modélisation

Notre stratégie au court de ce projet a été de commencer par des éléments « simples » pour ensuite monter en complexité. Pour rappel, le principe des ultrasons est un examen échographique qui se fait avec une sonde. De nos jours, lorsque nous pensons à une image échographique nous partons du principe qu'elle est forcément au format sectoriel. Cependant, il existe des sondes linéaires. Ce sont des sondes constituées de plusieurs cristaux alignés en rangée linéaire, qui nous donne une image reconstituée sous forme rectangulaire. Ce type d'affichage est plus abordable dans un premier temps, nous avons ainsi débuté par une modélisation de ce type.

La modélisation d'image synthétique de type « ultrasons » sous un format rectangulaire peut-être représentée comme sur la figure 6 ci-dessous.

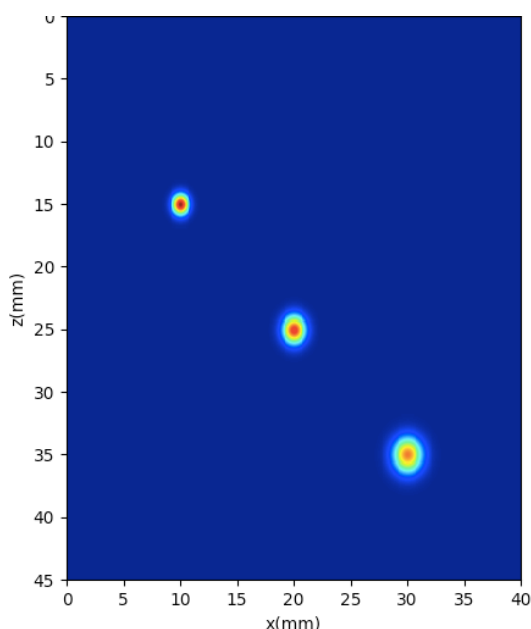


Figure 6 - Image matricielle

Nous avons ici une image matricielle représentée dans le plan x,z . Dans un cadre réelle, il faut imaginer que la sonde est positionnée en $x=0$, plus on se déplace en x (vers la droite de la figure) plus on descend en profondeur en z . Les 3 points

lumineux sur la figure peuvent être considérés comme des trous c'est-à-dire un défaut dans le matériau présumé. Nous utilisons ici N réflecteurs où chaque écho correspond à un modèle gaussien. Chaque écho a 5 paramètres :

- Position x
- Position z
- Ecart type x (équivalent à la largeur de l'écho en x)
- Ecart type z (équivalent à la largeur de l'écho en z)
- Amplitude

Il est important par la suite de normaliser l'image pour améliorer la qualité de l'image obtenue. Néanmoins, pour nous mettre dans des conditions réelles malgré la simulation, nous ajoutons du bruit gaussien de moyenne nulle. (Cf. Figure 7)

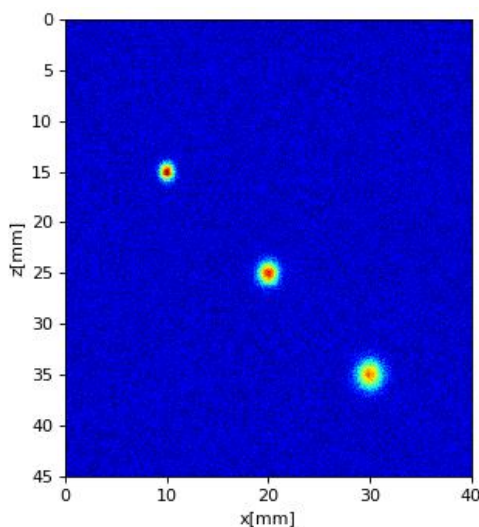


Figure 7 - Image matricielle bruitée

Comme énoncé dans la partie démarrage, au début nous avons effectué notre modélisation sur l'IDLE Python assez limité. Ainsi pour pouvoir modifier la variance, il était obligatoire de reprendre le code pour y effectuer une modification. Il est donc essentiel d'une part de créer une interface graphique pour pouvoir modifier les paramètres sans passer par le code. D'autre part, elle est nécessaire pour l'affichage des données

L'interface graphique vise à créer une interaction utilisateur-machine. L'interface dans ce prototypage rectangulaire propose à l'utilisateur les fonctionnalités suivantes (Cf. Figure 8) :

- Lancer le programme « RUN »
- Le stopper « stop »
- Modifier la variance pour le bruit
- L'affichage du FPS « Frame Per Second » qui nous expliquerons par la suite
- La taille de l'image $N = N_x \times N_z$
- Fermer l'interface graphique

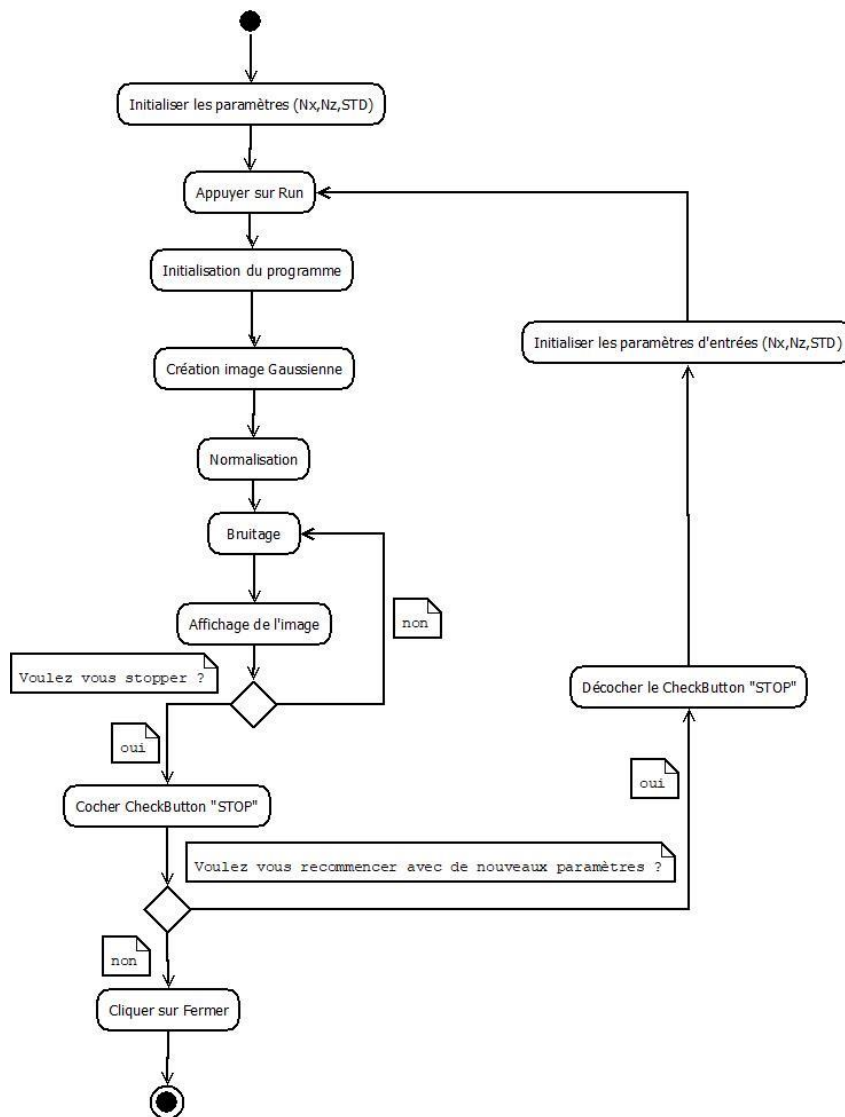


Figure 8 - Diagramme UML pour l'image matricielle

Pour une meilleure structuration de notre code, nous avons choisi le pattern MVC. (Cf. Figure 9) Ce pattern permet de découper notre code en 3 parties :

- **Modèle** : c'est là que se trouve les données nécessaires aux calculs. Dans notre cas, il s'agit des variables initialisées et des diverses fonctions créées.
- **Vue** : Ce que l'on nomme « la vue » est en fait une IHM. Elle représente ce que l'utilisateur a sous les yeux en l'occurrence l'image ultrasonore et les labels. Nous avons malgré tout de même créé une 4eme classe, **SidePanel**, uniquement en charge de la création et l'initialisation des labels. Cependant, c'est bien la classe vue qui permet leurs affichages.
- **Contrôleur** : permet de faire le lien entre la vue et le modèle lorsqu'une action utilisateur est intervenue sur la vue. C'est cet objet qui aura pour rôle de contrôler les données.

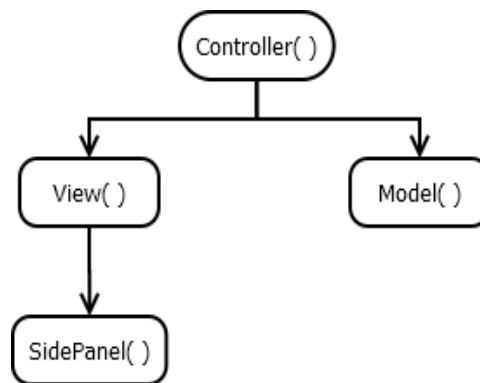


Figure 9 - Diagramme UML de classe

Avec notre application structurée en MVC, voici ce qu'il se passe :

- L'utilisateur a modifié les différents paramètres (variance, taille) et clique sur Run
- L'action est captée par le contrôleur, qui va vérifier la cohérence des données (taille, variance) et éventuellement les transformer afin que le modèle les comprenne
- Le modèle reçoit les données et change d'état
- Le modèle notifie la vue qu'il faut se mettre à jour
- L'affichage dans la vue est modifié en conséquence en allant chercher l'état du modèle

L'imagerie ultrasons est une succession d'images, il est ainsi possible de calculer le FPS « Frame Per Second » qui correspond au nombre d'image affichée par seconde.

Afin d'obtenir cette succession d'images à l'affichage et obtenir notre FPS, nous avons créé une boucle dans la classe Contrôleur. Cette boucle contient notre fonction de bruitage d'images jusqu'à ce que l'utilisateur décide d'arrêter le programme.

Pour calculer le FPS avant la boucle, nous initialisons le nombre d'itération à 1 et lançons un premier chronomètre (timer) de début. Après la fonction de bruit gaussien, nous pouvons arrêter ce chronomètre. Ainsi, nous connaissons le temps écoulé pour une image. Le FPS est obtenu en faisant la division du nombre d'itération actuel par le temps écoulé sachant que le nombre d'itération s'incrémente de 1 à chaque boucle faite.

Le passage du format rectangulaire au format sectoriel est plus évident une fois les étapes précédentes faites. En effet, le plus « dur » est fait au préalable : interface graphique, mise en place du pattern MVC, affichage d'image en boucle avec le calcul du FPS. La différence entre format rectangulaire et le format sectoriel est que nous

n'affichons plus les données dans le plan x,z. Nous nous retrouvons en coordonnées polaires. (Cf. Figure 10)

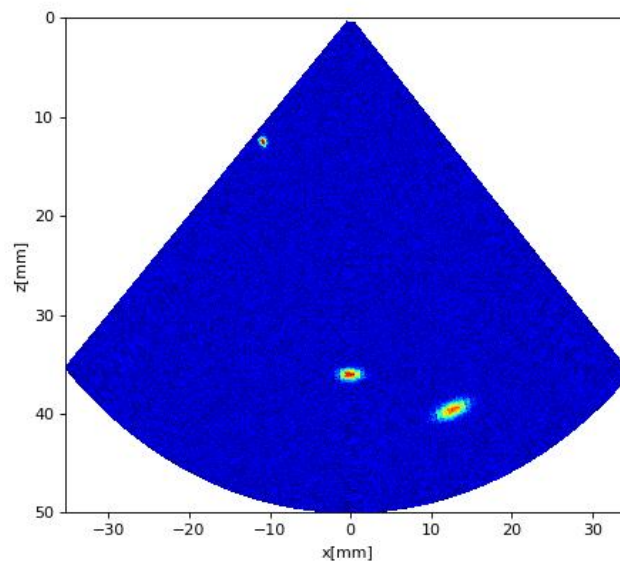


Figure 10 - Image sectorielle

L'interface graphique de ce prototypage sectoriel propose à l'utilisateur les fonctionnalités suivantes :

- Lancer le programme « Run »
- Le stopper « Stop »
- Angle minimum et maximal et le pas d'évolution
- Rayon minimum et maximal et le pas d'évolution

Le diagramme d'activité sera identique à la figure 8 à l'exception des paramètres d'entrées qui sont différents entre ces deux interfaces. Quant au diagramme de classe, il est identique à celui de la figure matricielle. (Cf. Figure 9)

La simulation de données fixes a été une grande étape du projet qui nous a permis de nous familiariser avec la programmation sous Python et ses librairies. Grâce à notre avance sur le planning, nous avons pu bénéficier des ressources matérielles du client pour la suite du projet.

2.3. Instrumentation

Dans cette partie, nous commençons les mesures grâce au système d'acquisition OEMPA développé par AOS NDT. Il permet l'acquisition rapide de signaux pour réaliser l'imagerie par focalisation en tout point en temps réel avec une grande précision.

La pièce à inspecter dans notre cas est un bloc d'aluminium contenant des petits trous percés. L'instrument électronique à 64 voies et le bloc à inspecter sont représentés sur la figure 11 ci-dessous.



Figure 11 - Matériel OEMPA

Comme nous l'avons déjà évoqué dans le premier chapitre, les sondes à ultrasons multiéléments sont classées en fonction des paramètres de base suivants :

- **Type** : généralement des sondes d'angle
- **Fréquence**: fréquences de 2 MHz à 10 MHz
- **Nombre d'éléments** : Les sondes à ultrasons multiéléments ont le plus souvent de 16 à 128 éléments, mais certaines en ont jusqu'à 256
- **Taille des éléments**

Nous utilisons une sonde de 128 éléments et travaillons autour d'une fréquence centrale de 5MHz. Afin de réaliser une adaptation d'impédance entre la sonde et le bloc d'aluminium, il faut ajouter de l'eau. En effet, l'eau est un très bon adaptateur d'impédance si bien que la majorité des contrôles est réalisée en immersion dans une cuve. La sonde est branchée au boîtier OEMPA qui va permettre d'acquérir les données émises par la sonde en temps réel.

Pour récupérer les données, la communication entre le boîtier OEMPA et notre ordinateur se fait à l'aide d'un câble Ethernet. Il existe plusieurs protocoles de communication en réseau. Ici, nous utilisons un TCP (Transmission Control Protocol), soit « protocole de contrôle de transmission ».

Concrètement, il permet de connecter deux applications et de leur faire échanger des informations. Ce protocole est dit « orienté connexion », c'est-à-dire que les applications sont connectées pour communiquer. Quand on envoie une information au travers du réseau, on peut être sûr qu'elle a bien été réceptionnée par l'autre application. Cela peut paraître évident mais le protocole UDP (User Datagram Protocol), par exemple, envoie des informations au travers du réseau sans se soucier de savoir si elles seront bien réceptionnées par la cible. Ce protocole n'est pas connecté, une application envoie quelque chose au travers du réseau en spécifiant une cible, mais la réception n'est pas garantie.

C'est donc le protocole TCP qui nous intéresse. Il est un peu plus lent que le protocole UDP mais plus sûr et, pour la quantité d'informations que nous allons transmettre. Il est préférable d'être sûr des informations transmises plutôt que de la vitesse de transmission.

Pour que le PC se connecte au boîtier il nous faut deux informations principales :

- **L'adresse IP** qui identifie le boîtier sur le réseau local. Dans notre cas, le boîtier possède déjà une adresse IP configurée 192.168.1.11
- **Un numéro de port**, qui est souvent propre au type d'information que l'on va échanger. Si par exemple, nous demandons une connexion web, le navigateur va en général interroger le port 80 si c'est en http ou le port 443 si c'est en connexion sécurisée (https). Ici, nous utiliserons le port 5001 qui est un port dédié aux protocoles TCP ou UDP.

Pour pouvoir communiquer avec l'adresse IP du boîtier, il est indispensable de s'assurer que l'ordinateur ait une adresse IP identique sur les 3 premiers bits du premier octet. En effet, cela permet à notre ordinateur d'être sur la même plage réseau que l'appareil. Nous avons, par exemple, défini notre adresse IP d'ordinateur 192.168.1.70.

Pour s'assurer de l'établissement d'une réelle connexion et communication entre les deux applications, nous avons utilisé le logiciel Wireshark. Wireshark est un analyseur de trafic réseau utilisé pour capturer et examiner les paquets transitant sur un réseau Open source et multiprotocole. (Cf. Figure 12) Destiné aux administrateurs réseaux et aux développeurs, Wireshark est une référence en matière d'analyse des transactions réseaux.

Time	Source	Destination	Protocol	Length	Info
5 2.460592	192.168.1.70	192.168.1.11	ICMP	1066	Echo (ping) request id=0x0001, seq=162/41472, ttl=255 (reply in 6)
6 2.460834	192.168.1.11	192.168.1.70	ICMP	1066	Echo (ping) reply id=0x0001, seq=162/41472, ttl=255 (request in 5)
7 2.461154	192.168.1.70	192.168.1.11	TCP	66	54607 → 5001 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
8 2.461570	192.168.1.11	192.168.1.70	TCP	66	5001 → 54607 [SYN, ACK] Seq=0 Ack=1 Win=256 Len=0 MSS=4460 WS=16 SACK
9 2.461597	192.168.1.70	192.168.1.11	TCP	54	54607 → 5001 [ACK] Seq=1 Ack=1 Win=65700 Len=0
10 2.533704	192.168.1.70	192.168.1.11	TCP	134	54607 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=65700 Len=80
11 2.533985	192.168.1.11	192.168.1.70	TCP	60	5001 → 54607 [ACK] Seq=1 Ack=81 Win=4096 Len=0
12 2.536147	192.168.1.11	192.168.1.70	TCP	106	5001 → 54607 [PSH, ACK] Seq=1 Ack=81 Win=4096 Len=52
13 2.561897	192.168.1.70	192.168.1.11	TCP	302	54607 → 5001 [PSH, ACK] Seq=81 Ack=53 Win=65648 Len=248
14 2.565769	192.168.1.11	192.168.1.70	TCP	1514	5001 → 54607 [ACK] Seq=53 Ack=329 Win=4096 Len=1460
15 2.566017	192.168.1.11	192.168.1.70	TCP	1514	5001 → 54607 [ACK] Seq=1513 Ack=329 Win=4096 Len=1460
16 2.566037	192.168.1.70	192.168.1.11	TCP	54	54607 → 5001 [ACK] Seq=329 Ack=2973 Win=65700 Len=0
17 2.566467	192.168.1.11	192.168.1.70	TCP	1414	5001 → 54607 [PSH, ACK] Seq=2973 Ack=329 Win=4096 Len=1360
18 2.591796	192.168.1.70	192.168.1.11	TCP	162	54607 → 5001 [PSH, ACK] Seq=329 Ack=4333 Win=64340 Len=108
19 2.594242	192.168.1.11	192.168.1.70	TCP	642	5001 → 54607 [PSH, ACK] Seq=4333 Ack=437 Win=4096 Len=588
20 2.619698	192.168.1.70	192.168.1.11	TCP	106	54607 → 5001 [PSH, ACK] Seq=437 Ack=4921 Win=65700 Len=52
21 2.622080	192.168.1.11	192.168.1.70	TCP	202	5001 → 54607 [PSH, ACK] Seq=4921 Ack=489 Win=4096 Len=148
22 2.647720	192.168.1.70	192.168.1.11	TCP	134	54607 → 5001 [PSH, ACK] Seq=489 Ack=5069 Win=65552 Len=80
23 2.650108	192.168.1.11	192.168.1.70	TCP	106	5001 → 54607 [PSH, ACK] Seq=5069 Ack=569 Win=4096 Len=52
24 2.675662	192.168.1.70	192.168.1.11	TCP	266	54607 → 5001 [PSH, ACK] Seq=569 Ack=5121 Win=65500 Len=212

Figure 12 - Capture d'un transfert de paquet (logiciel : Wireshark)

Grâce à Wireshark (Cf. Figure 12), nous pouvons confirmer que nous utilisons bien un protocole TCP et non UDP. Nous observons également un protocole ICMP. Le protocole ICMP (Internet Control Message Protocol) est le protocole de signalisation

des problèmes utilisé par le protocole IP. Son but est de tester la connectivité réseau mais aussi d'apporter une aide au diagnostic en cas de problèmes ou de défaillances. Il existe plusieurs types de messages ICMP ici 2 apparaissent :

- **Le type "Echo"**

Ce message ICMP est envoyé par l'ordinateur à destination du boîtier pour tester la connectivité réseau entre ces deux points. C'est ce message qui est envoyé par l'utilitaire "ping"

- **Le type "Echo Reply"**

Ce message ICMP est la réponse du boîtier au message "Echo". C'est ce message qui est analysé par l'utilitaire "ping" pour connaître le délai d'acheminement.

Le premier type de scan TCP que nous voyons ensuite est le TCP Syn scan ou Half Open scan (Half Open voulant dire "à moitié ouvert"), la traduction aide bien à comprendre le fonctionnement de ce scan. En effet, un TCP SYN scan va envoyer sur le port ciblé (5001) un paquet TCP SYN qui est donc le premier paquet qu'envoie l'ordinateur (qui demande à établir une connexion avec le boîtier). En temps normal si le port est ouvert sur le serveur avec un service tournant derrière, le serveur va renvoyer un SYNACK. Cela permet de valider la SYN du client et d'initialiser la connexion TCP, c'est-à-dire un paquet TCP avec les flags SYN et ACK à 1. On peut alors dire que le port est ouvert et détecte un service. Par la suite, nous voyons les échanges et la longueur des messages envoyés avec accusé de réception Ack à chaque élément.

Au début de cette partie instrumentation, le client nous a fourni une bibliothèque AOS afin de s'en inspirer. Il nous a notamment fourni un programme d'un de ses clients faisant appel à des DLL. Nous nous sommes inspirés des programmes qui nous ont été mis à disposition pour mettre en place une fois de plus notre architecture MVC.

Dans l'interface graphique l'utilisateur peut saisir l'adresse IP du boîtier auquel il souhaite établir la connexion et lancer la connexion en cliquant sur « connect ». Dans l'interface Pycharm (Cf. Figure 13), l'utilisateur reçoit différentes notifications :

- Un entier NewDev assigne à l'appareil un identifiant
- Booléen de connexion pour confirmer une connexion établie ou non
- Booléen de lecture d'un fichier contenant des paramètres d'acquisition tel que la profondeur des trous recherchés le nombre de cycle, la fréquence d'acquisition etc.
- Booléen pour l'activation du pulser d'ondes pendant une certaine durée définie

- Booléen de désactivation
- Booléen de suppression de la détection appareil
- Booléen de déconnexion quand l'utilisateur clique sur « disconnect »

```
NewDev:0
Connect:True
ok connexion
LoadSetup:True
ok loading file
EnablePulser:True
DisEnablePulser:True
DisConnect:True
ok deconnexion
```

Figure 13 - Fenêtre dialogue utilisateur

Une fois l'aspect connexion/déconnexion réglé nous avons porté notre attention sur les données d'acquisition. L'objectif premier de l'acquisition était de pouvoir déjà afficher un a-scan puis par la suite pouvoir en créer successivement à chaque mouvement nouveau de notre sonde. A chaque mouvement, il devrait être possible d'observer un nouvel écho. Pour cela, nous nous sommes inspirés une fois de plus des ressources Matlab fournies, notamment les DLL. Nous avons donc fait appel aux DLL afin de réaliser des fonctions spécifiques :

- New device() -> dll.mxNewDevice
- DeleteDevice() -> dll.mxNewDevice
- Connect() -> dll.mxConnect
- DisConnect() -> dll.mxConnect
- ReadFileWriteHW() -> dll.mxReadFileWriteHW
- GetSWCycleCount() -> dll.mxGetSWCycleCount
- GetAcquisitionAscanSize() -> dll.mxGetAcquisitionAscanSize
- EnableShot() -> dll.mxEnableShot
- DisEnableShot() ->
- GetAcquisitionAscanFifoIndex() -> dll.mxGetAcquisitionAscanFifoIndex
- GetAcquisitionFifoData() -> dll.mxGetAcquisitionAscanFifoData
- ReturnFifoData() -> fait appel aux 2 fonctions précédentes pour retourner les données

La différence majeure pour passer du A-scan au B-scan se fait lors du chargement du fichier grâce à la fonction de lecture du fichier de paramètres « ReadFileWriteHW() ». En effet dans le cas de l'A-scan, le fichier récupéré ne contient qu'un seul cycle et ne contient pas des paramètres précisant la profondeur d'exploration et bien d'autres. Or celui utilisé pour récupérer les paramètres B-scan nécessite plus d'un cycle et de préciser plus de paramètres. Le nombre total de cycle par défaut que nous utilisons vaut 51 dans le cadre de nos simulations. Cependant, il est toujours possible d'accéder au fichier pour modifier des paramètres.

Pour l'acquisition en continue aussi bien pour un A-scan qu'un B-scan, nous avons ajouté un bouton stop dans une boucle while. Tant que l'utilisateur ne clique pas sur « stop », l'acquisition continuera. Cependant, dans le cas du B-scan nous utilisons en plus une boucle for dans laquelle nous appelons notre fonction de retour de données « ReturnFifoData() » du cycle 0 au nombre de cycle total. Pour rappel, ce nombre de cycle total est récupéré dans le fichier paramètre, nous pouvons en vérifier la valeur grâce à la fonction GetSWCycleCount(). Lors de l'acquisition, il est important de noter que les premières données entrantes seront aussi les premières sortantes, c'est le principe du FIFO (First In First Out).

La figure 14, ci-dessous, permet d'illustrer à la fois le fonctionnement de notre interface et le parcours utilisé pour la création de l'A-Scan et B-Scan. En rouge dans l'A-Scan, nous avons souhaité mettre en avant la partie nécessaire à l'acquisition en continu également présente dans le B-scan. Dans les deux cas, nous remarquons bien que le bouton stop joue un rôle déterminant dans notre code pour l'acquisition

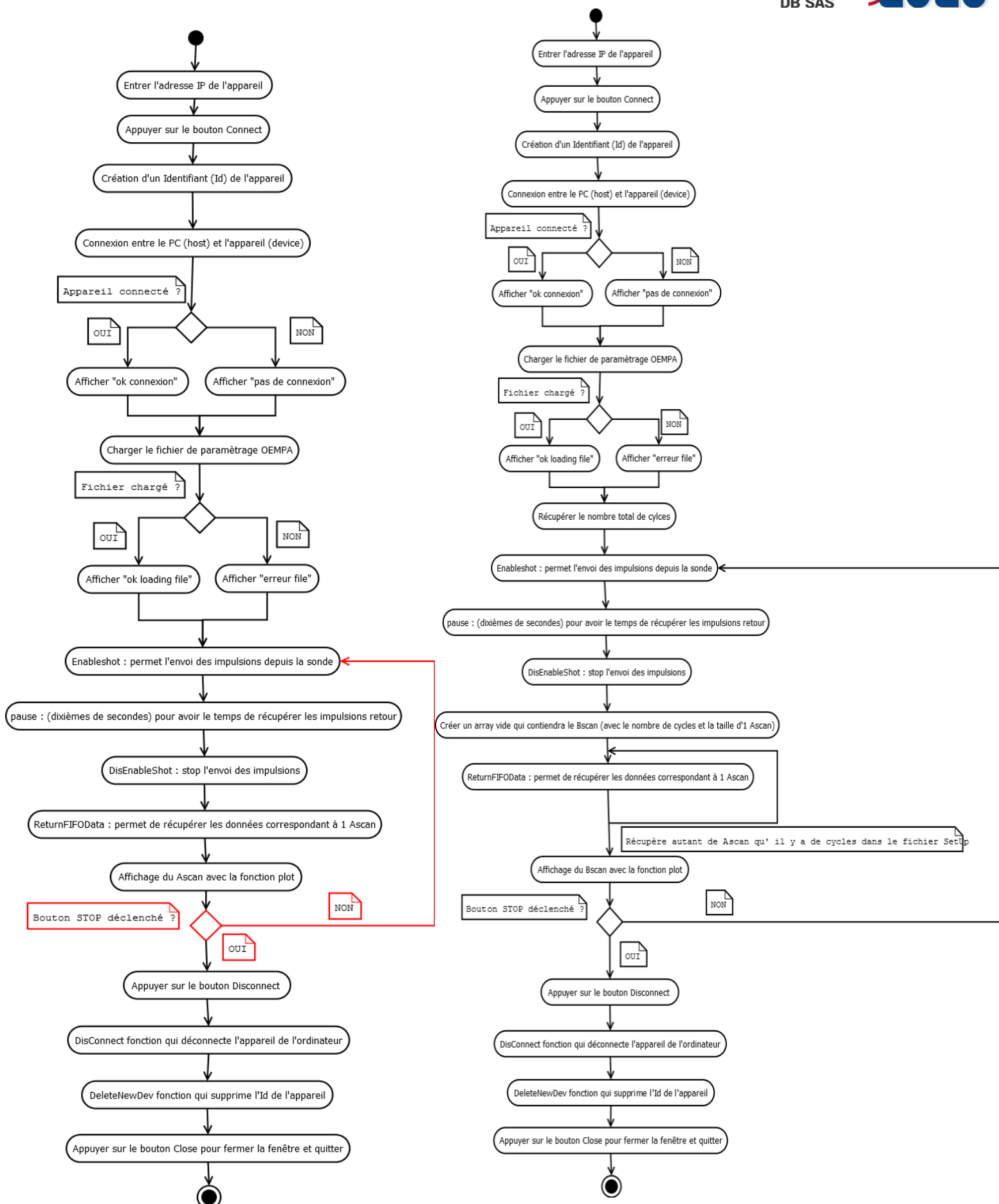


Figure 14 – Diagrammes UML A-Scan (à gauche) et B-Scan (à droite)

3. Résultat et analyse critique

3.1. Résultats

3.1.1. Modélisation

Pour rappel dans la partie simulation, les paramètres pouvant être modifié par l'utilisateur sont : la variance et la taille de l'image

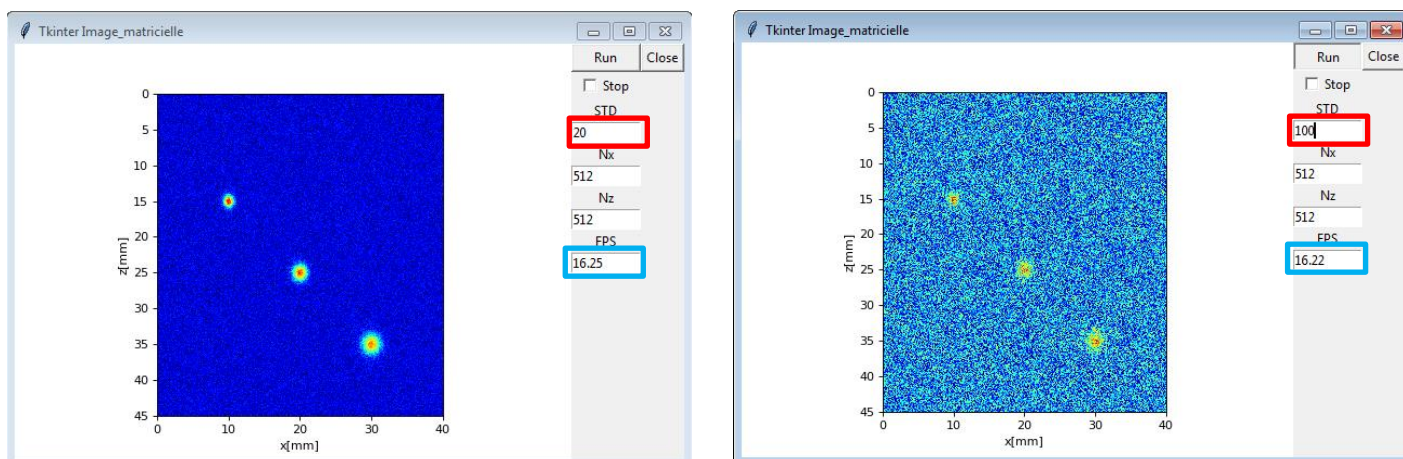


Figure 15 – Interfaces graphiques d'image matricielle pour une variance = 20 et une variance = 100

Comme vous pouvez le constater à travers la figure 15, plus l'utilisateur entre une variance croissante, plus il obtiendra une image bruitée de plus en plus claire en accord avec l'échelle des couleurs choisie. Ainsi, nous obtenons bien l'effet recherché. Il est également important de remarquer que la variance contrairement à la taille de l'image (Cf. Figure 16) n'a pas d'impact sur le FPS. Bien que sur les figures nous n'obtenons pas exactement le même FPS, il reste de l'ordre de 16 Hz. Ces écarts sont dus au fait que les captures ont été prises avant la stabilisation de la valeur.

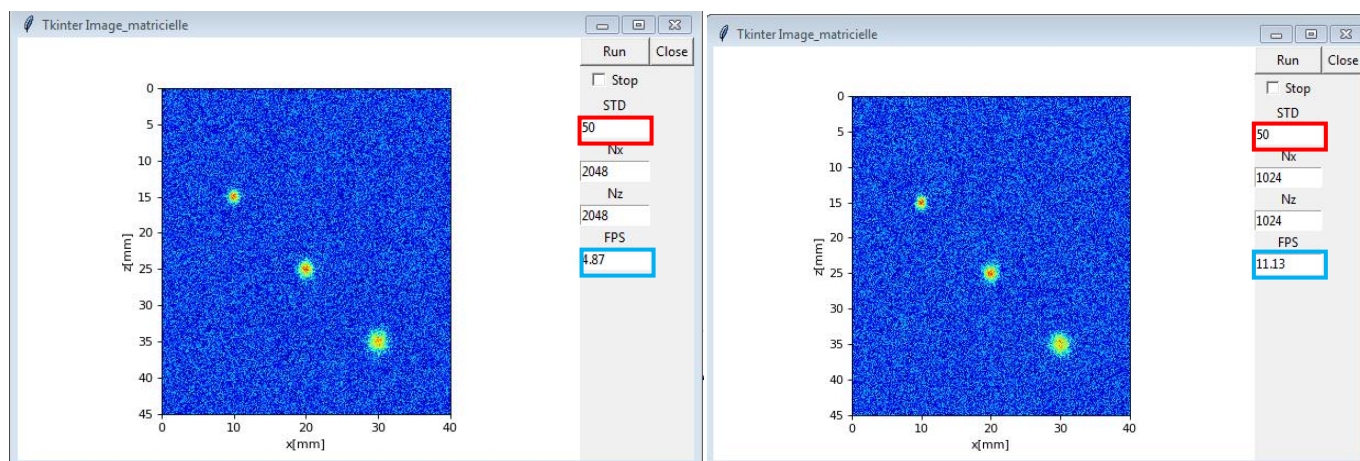


Figure 16 - Interfaces graphiques d'image matricielle pour taille 2048*2048 puis 1024*1024

Ces figures (Cf. Figure 16) permettent de confirmer l'hypothèse selon laquelle plus l'image va être grande, plus le temps d'affichage sera long et donc fera baisser le FPS. Dans ce cas précis pour une image 1024*1024, nous passons d'un FPS de 11.13 Hz à 4.87 Hz pour une image 2048*2048. Réciproquement, plus l'image sera petite plus le FPS sera élevée.

Dans le cas de l'image sectorielle, nous avons dans un premier temps effectué notre prototype sans modifier la taille de l'image puis nous avons ajouté le réglage de la taille afin de pouvoir comparer le FPS entre une image matricielle et une sectorielle. (Cf. Figure 17)

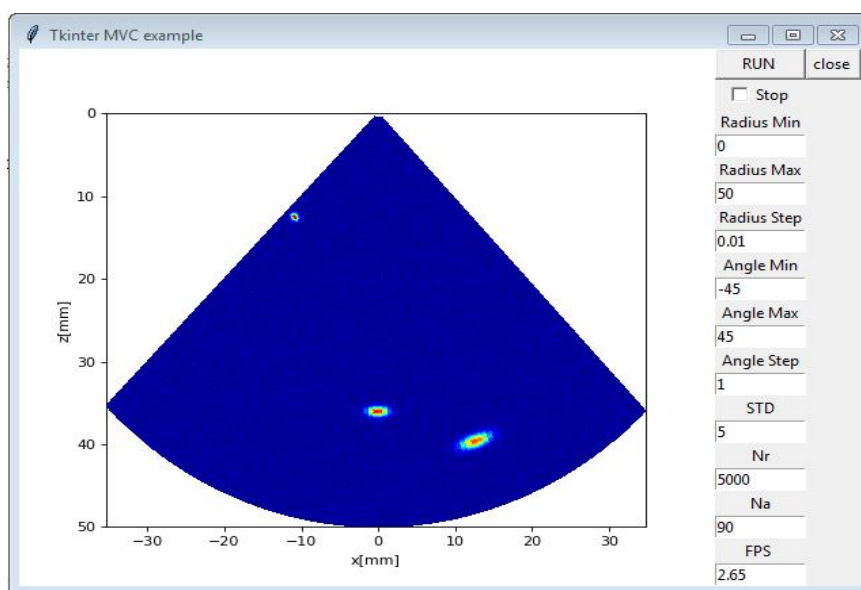


Figure 17 - Interface graphique d'une image sectorielle

3.1.2. Instrumentation

L'ajout d'un adaptateur d'impédance joue un rôle important, pour le prouver nous avons réalisé une acquisition sans ajout eau (Cf. Figure 18) et avec de l'eau par la suite. (Cf. Figure 19)

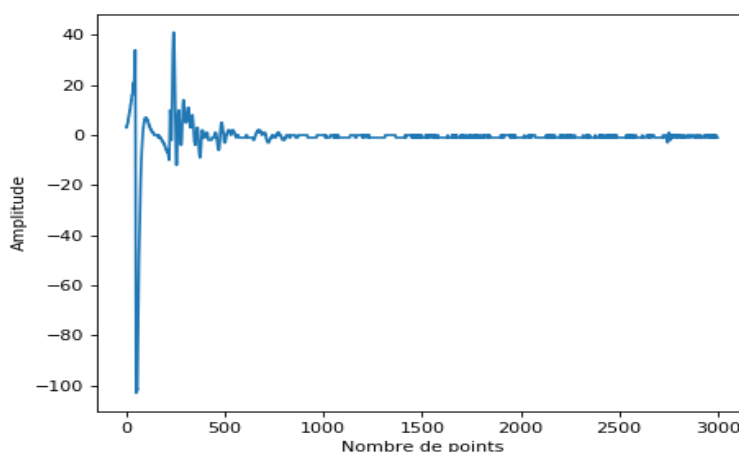


Figure 18 - A-scan émis/reçu sur le bloc aluminium sans ajout d'eau

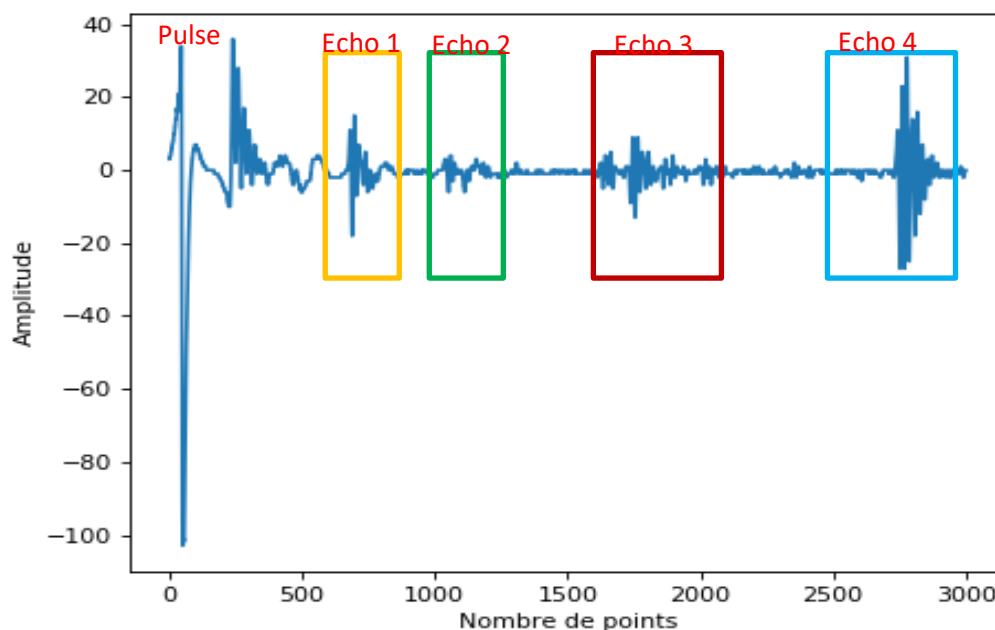


Figure 19 - A-scan émis/reçu sur le bloc d'aluminium avec ajout d'eau

Sur chacun des A-scans précédents, les deux premiers pics ont une amplitude importante traduisant l'impulsion envoyée. Néanmoins, sans l'ajout d'eau nous constatons qu'il nous ait impossible d'observer des échos. Grâce à l'adaptateur d'impédance, nous parvenons à obtenir l'écho 1 entrant suivi de 3 autres échos (Cf. Figure 19). La durée d'un A-scan est d'environ $30\mu s$ car nous avons une fréquence d'échantillonnage de 100 MHz.

Nous arrivons également aux mêmes résultats pour une acquisition d'A-scan en continu. Sur notre interface lors de l'acquisition en temps réel, les A-scan se superposent les uns sur les autres en changeant automatiquement de couleur comme vous l'observez sur la figure 20.

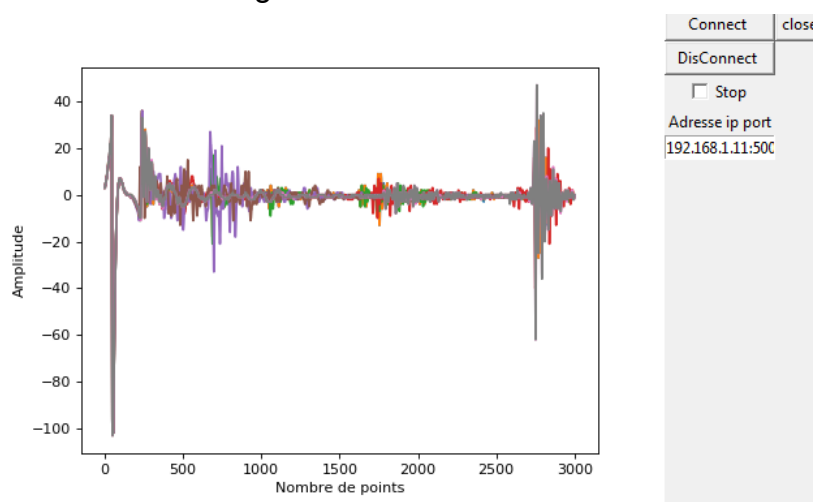


Figure 20 - A-scan en continu avec eau

Chaque nouveau A-scan obtenu est dû au déplacement de la sonde, autrement les différents A-scan se superposeraient à la perfection.

Il est aussi important de notifier que comme énoncé dans le chapitre précédent, nous avons un bouton Stop en plus dans l'interface utilisateur pour l'acquisition en temps réel de l'A-scan tout comme le B-scan (Cf. Figure 20).

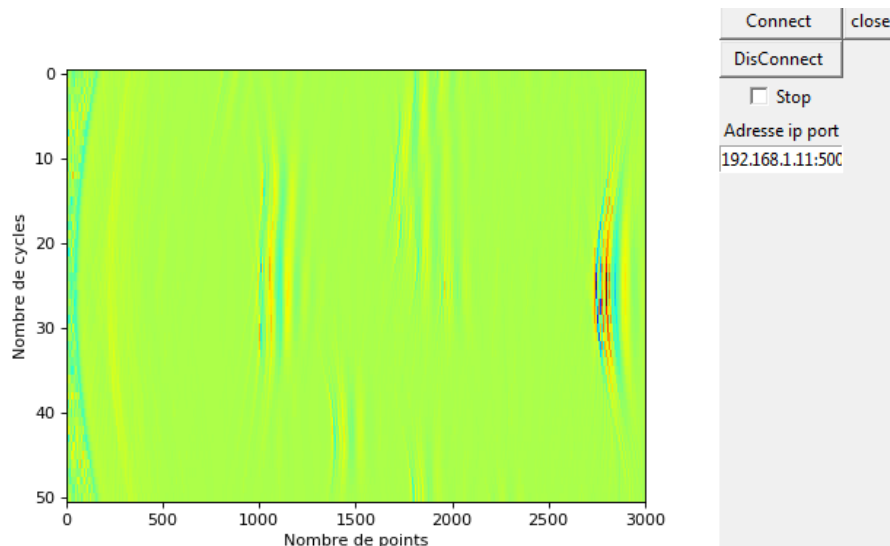


Figure 21 - B-scan

Le B-scan permet de détecter ainsi les défauts qui apparaissent en rouge. En se déplaçant sur le bloc, il est possible également de remarquer des courbures formées par l'alignement de certains trous. Il est donc aisément facile de détecter si les données sont cohérentes, ce qui est notre cas.

3.2. Critères d'évaluation

3.2.1. FPS

Pour valider notre projet, nous avons besoin de différents critères capables d'appuyer en faveur de nos premières impressions et espérances.

Le premier est le FPS (Frame Per Sample) c'est-à-dire la vitesse de rafraichissement de nos images (par secondes ou unité de temps) donnée en Hertz. Ainsi dans notre étude, nous avons opté pour une première analyse qui correspond à l'évolution du FPS en fonction de la taille des images. La seconde analyse porte sur la différence entre le FPS de Matlab et celui de Python.

Pour cela, nous avons imposé le calcul du FPS dans la boucle d'affichage des images. Bien entendu comme pour Python nous avons choisi une hiérarchie MVC

(Modèle-Vue-Contrôleur) et que sur Matlab tout est réalisé directement de manière séquentielle ; nous aurons une seconde partie sur l'utilisation d'un Profiler afin de déterminer si ce choix sur Python a une réelle incidence sur notre FPS.

Pour ces tests, nous avons testé les deux modèles c'est-à-dire la vue matricielle et la vue sectorielle.

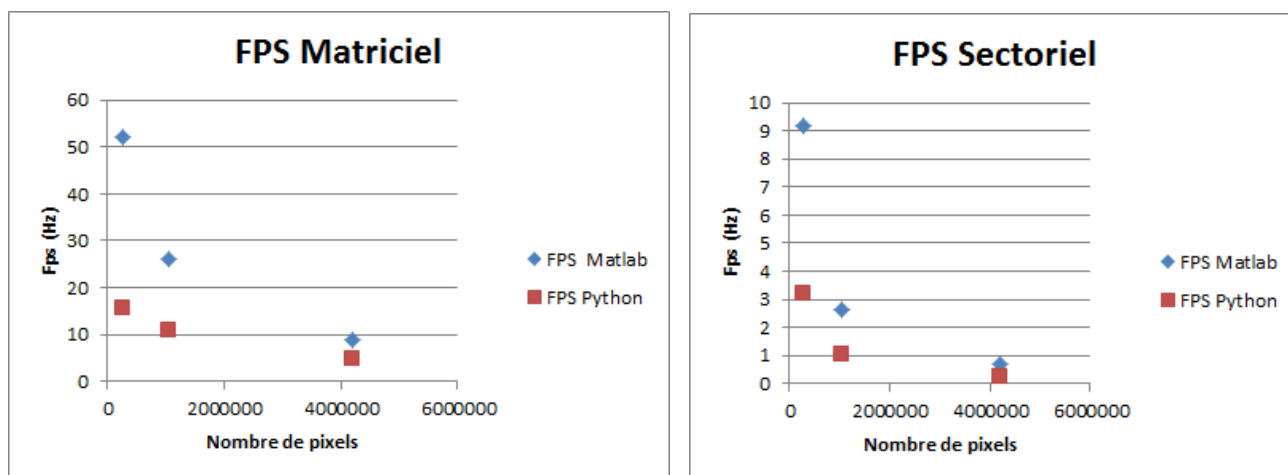


Figure 22 – Tests des FPS Python/Matlab en fonction de la taille des images

On remarque que dans les deux cas (matriciel et sectoriel), plus la taille de l'image est importante plus il y a convergence vers le même FPS par contre pour des images assez petites Matlab surpasse Python (Cf. Figure 22)

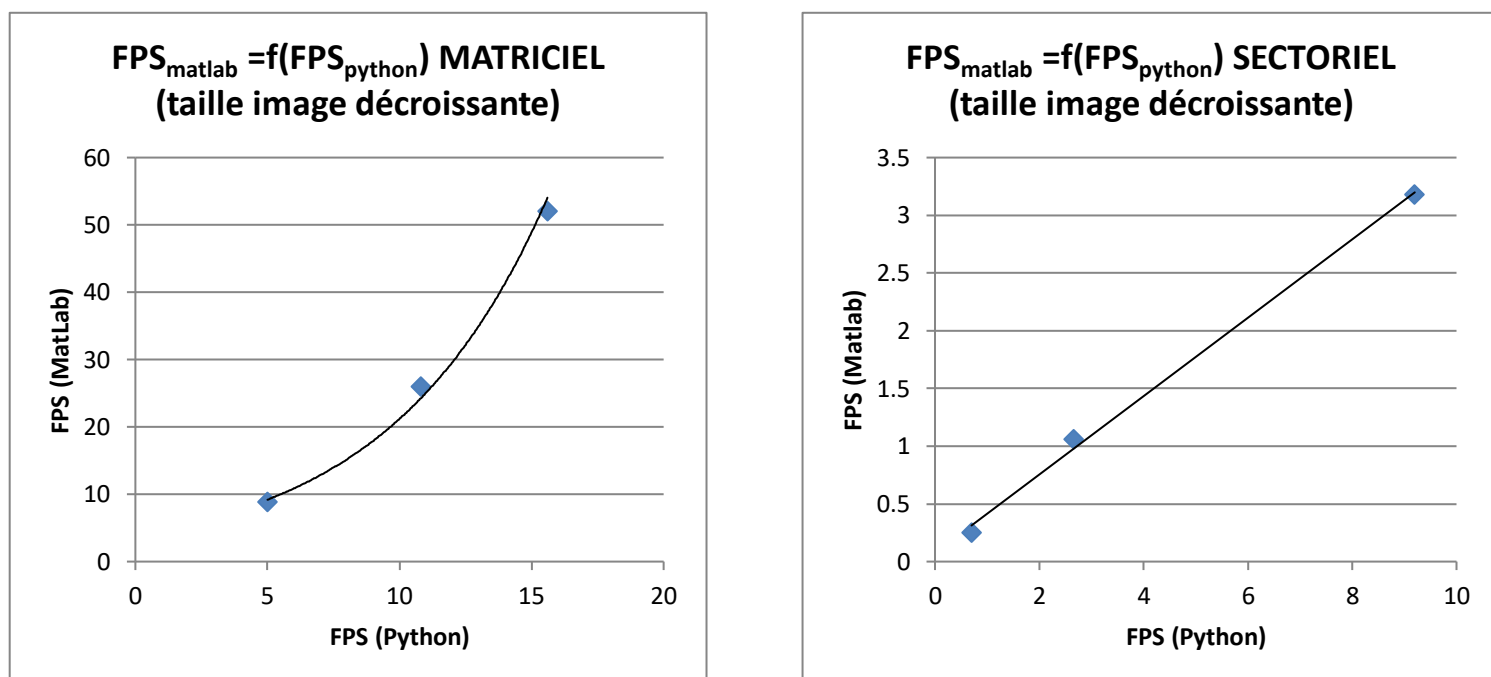


Figure 23 - Tests entre les FPS Matlab/Python pour l'image matricielle et sectorielle

Ici nous remarquons que le FPS de Matlab semble resté plus rapide que celui de Python dans les deux cas. (Cf. Figure 23) La différence entre le sectoriel et le matriciel réside dans l'écart de la vitesse de rafraîchissement. En effet, pour le sectoriel l'évolution est plus linéaire alors qu'en matriciel l'évolution est logarithmique.

On remarque aussi que le FPS, quel que soit le langage, reste plus important en matriciel qu'en sectoriel. Ceci paraît assez logique car la matriciel est un simple tableau de valeur de pixel alors que pour notre image sectorielle nous devons passer en coordonnées polaires.

Pour faire suite aux graphes réalisés sur l'évolution des FPS de Matlab et Python, nous avons cherché à mieux interpréter cette différence en utilisant un profiler. Un profiler ou profilage de code consiste à analyser l'exécution d'un logiciel afin de connaître son comportement lors de l'exécution.

Pour réaliser un profiler avec Python, il faut aller dans l'invite de commande puis se placer dans le fichier où se trouve le .py et écrire la commande suivante (penser à télécharger la librairie au préalable) :

python -m cProfile nomFichier.py

Ainsi, un grand nombre de lignes apparaissent maintenant il faut les interpréter afin d'utiliser ce qui a été réalisé.

La première ligne donne tout d'abord, le nombre d'appel de fonction ensuite entre parenthèses, nous obtenons le nombre d'appel à des fonctions non récursives, et pour finir le temps d'exécution total. (Cf. Figure 24)

Ensuite, les termes suivants apparaissent créant ainsi un tableau de données (Cf. Figure 24) : `ncalls tottime percall cumtime percall filename:lineno(function)`.

Nous allons donc plus détailler la signification de ces différents termes :

- ***ncalls*** : le nombre d'appel
- ***tottime*** : le temps total utilisé par la fonction correspondante (sans les sous-fonctions)
- ***percall*** : `tottime / ncalls`, le temps d'exécution pour un appel
- ***cumtime*** : le temps cumulé d'exécution des fonctions (y compris les sous-fonctions)
- ***percall*** : `cumtime / (nombre d'appel à des fonctions non récursives)`
- ***filename:lineno(function)*** : permet de connaître la fonction associée

Profile obtenu :

```
6142895 function calls (6032267 primitive calls) in 27.805 seconds
```

Ordered by: standard name

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
515	0.002	0.000	0.002	0.000	<frozen importlib._bootstrap>:103(release)
327	0.000	0.000	0.000	0.000	<frozen importlib._bootstrap>:143(__init__)
327	0.001	0.000	0.008	0.000	<frozen importlib._bootstrap>:147(__enter__)
327	0.000	0.000	0.002	0.000	<frozen importlib._bootstrap>:151(__exit__)
515	0.003	0.000	0.006	0.000	<frozen importlib._bootstrap>:157(get_module_lock)
325	0.001	0.000	0.001	0.000	<frozen importlib._bootstrap>:176(cb)
188	0.000	0.000	0.002	0.000	<frozen importlib._bootstrap>:194(lock_unlock_module)
424/5	0.000	0.000	0.841	0.168	<frozen importlib._bootstrap>:211(call_with_frames_removed)
2828	0.002	0.000	0.002	0.000	<frozen importlib._bootstrap>:222(verbose_message)
18	0.000	0.000	0.000	0.000	<frozen importlib._bootstrap>:232(requires_builtin_wrapper)
314	0.000	0.000	0.000	0.000	<frozen importlib._bootstrap>:307(__init__)
314	0.001	0.000	0.001	0.000	<frozen importlib._bootstrap>:311(__enter__)
314	0.002	0.000	0.003	0.000	<frozen importlib._bootstrap>:318(__exit__)
1256	0.000	0.000	0.000	0.000	<frozen importlib._bootstrap>:321(<genexpr>)
270	0.001	0.000	0.001	0.000	<frozen importlib._bootstrap>:35(new_module)
327	0.002	0.000	0.002	0.000	<frozen importlib._bootstrap>:369(__init__)
566	0.001	0.000	0.010	0.000	<frozen importlib._bootstrap>:403(cached)

Figure 24 - Extrait du profiler obtenu sous Python

Pour réaliser un profiler avec Matlab, il faut aller dans command window, puis taper dans command window **profile on**, ensuite lancer le programme puis l'arrêter et pour finir écrire à nouveau dans command window **profile viewer**. Une fenêtre s'affiche permettant d'analyser les résultats. (Cf. Figure 25)

Profile Summary
Generated 07-Dec-2017 09:13:02 using cpu time.














Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
rectangularView	9	12.784 s	0.015 s	
gui_mainfcn	9	12.769 s	0.031 s	
...hObject eventdata guidata(hObject)	2	12.393 s	0.000 s	
rectangularView>run_checkbox_Callback	2	12.378 s	12.191 s	
openfigLegacy	2	0.297 s	0.000 s	
gui_mainfcn>local_openfig	2	0.297 s	0.000 s	
hload	1	0.188 s	0.000 s	
FigFile.read	1	0.172 s	0.156 s	
FigFile.FigFile>FigFile.FigFile	1	0.172 s	0.000 s	
movegui	2	0.094 s	0.094 s	
num2str	101	0.078 s	0.032 s	
axis	2	0.047 s	0.016 s	
ylabel	2	0.032 s	0.032 s	

Figure 25 - Extrait du profiler Matlab

On peut voir sur le profiler différentes colonnes, expliquons leurs significations :

- **Function Name** : le nom des fonctions analysées
- **Calls** : le nombre de fois que la fonction est appelée tant que le profiler est en mode on
- **Total Time** : temps total utilisé par une fonction (incluant les sous-fonctions)
- **Self Time*** : temps total utilisé par une fonction seule
- **Total Time Plot** : représentation graphique du Self Time en fonction du Total Time

En cliquant sur la zone bleue des fonctions des informations plus détaillées sont données. (Cf. Figure 26)

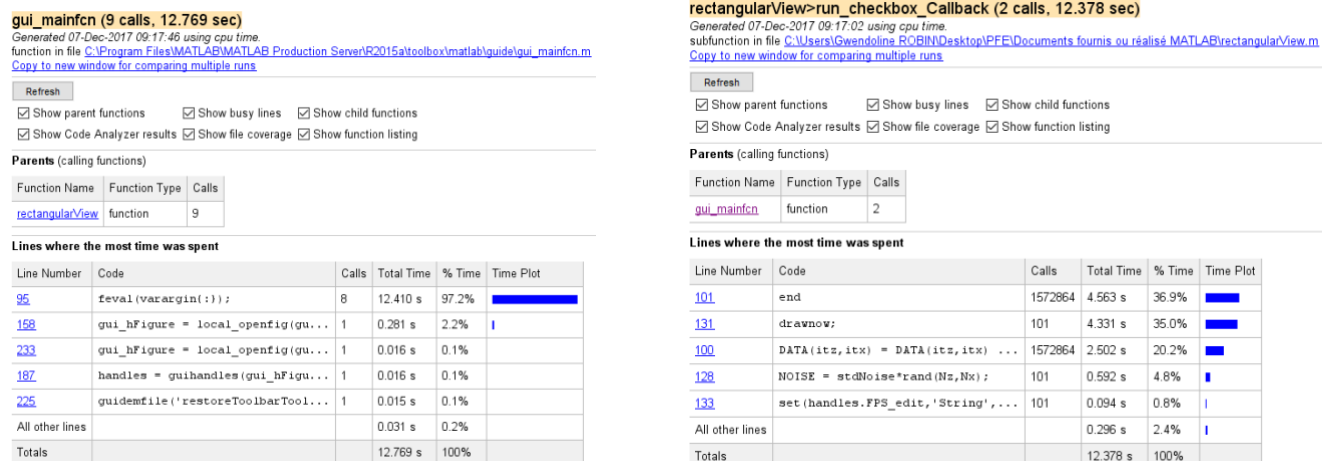


Figure 26 - Profiler de la fonction gui_mainfcn et run_checkbox_callback

Tout d'abord, les deux outils présentent des profilers bien différents. Le profiler de Matlab est plus « *user friendly* » que celui de Python.

Le plus important pour nous est de regarder quelle partie du code sous Python utilise le plus de temps. En effet, ce travail est réalisé afin de mieux comprendre la différence de FPS obtenue entre l'exécution du code sous Matlab et sous Python.

Premièrement si l'on regarde les fonctions dans l'ensemble (Cf. Figure 27) :

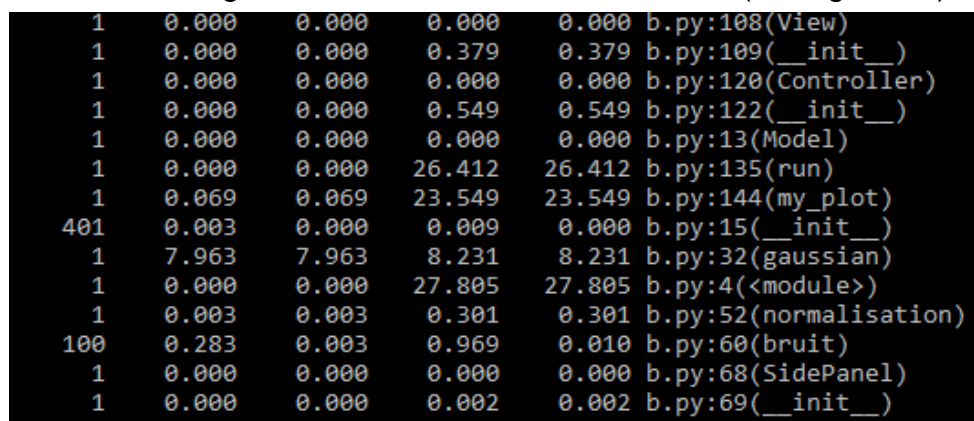


Figure 27 - Profiler de la vue matricielle

Nous pouvons constater que quatre fonctions se détachent des autres : run, my_plot, gaussian et <module>. La fonction réalisant une chose similaire à **my_plot** sous Matlab est aussi une des plus longues sous Matlab, seulement elle est de 12.378 secondes contrairement à 23.549 secondes sous Python. On observe donc à ce niveau que Matlab est deux fois plus performant. La méthode **gaussian** réalisée séparément sous Python est en plus réalisée dans le run sous Matlab. On peut alors ajouter les 8.231 secondes à la valeur du temps d'exécution du my_plot, ce qui creuse encore plus l'écart entre le run de Python et celui de Matlab à action équivalente respectivement : **31.78 secondes (33,05 secondes si l'on rajoute le**

temps de la normalisation et du bruit) et **12.378**. Ensuite, le **run** est la fonction permettant de lancer le programme.

La fonction <module>, qui est à 27.805 secondes, correspond à l'appel des librairies et est indispensable en Python afin d'avoir à notre disposition les librairies.

Plus précisément, lors de la **génération des valeurs (gaussian)** il y a environ **1 secondes d'écart**. (Cf. Figure 27 (gaussian) et Cf. Figure 26 (line 100 and 101)) En effet, pour Matlab on obtient 7.065 et sur Python nous avons 8.231. Contrairement à Matlab, où le temps de la ligne générant la data est d'environ 2 secondes en Python les calculs se font tellement vite que l'on obtient 0 secondes par l'utilisation de timer. La différence de temps semble se creuser lors de l'utilisation de la triple boucle for.

Il s'agit donc bien du my_plot (équivalent de run sous Matlab) qui gère l'affichage des données qui est plus lent sur Python.

Pour conclure sur ces tests effectués afin de valider ce critère du FPS, on peut affirmer que Matlab semble plus rapide tant dans l'exécution que dans l'affichage des données du scan. Cependant, il ne faut pas oublier que le code Python que nous avons fourni est en MVC et qu'il est peut être sous optimisé pour cet affichage. Tandis que Matlab effectue tout de manière séquentielle et utilise des interfaces préprogrammées. Sur Python, nous devons créer à la fois l'interface, faire l'acquisition et l'affichage des données. De plus, la plupart des fonctions utilisées sous Matlab sont optimisées alors que sur Python, étant open source, c'est la communauté qui améliore le langage et optimise ces fonctions.

3.2.2. Critère visuel

Le **critère visuel** peut être évalué visuellement par la représentation de la gamme de couleur, la présence et l'emplacement des boutons, l'orientation adéquate de l'axe et la représentation des valeurs pour **la phase simulation**.

Premièrement, au début du projet, nos données étaient représentées par des couleurs saturées, loin du résultat obtenu sur Matlab. Nous avons pu y remédier en y indiquant le type de colormap désirée et en modifiant notre manière de générer des valeurs aléatoirement. En effet, nous générions aléatoirement des valeurs avec la fonction randn. Or, cela permet d'obtenir des valeurs pseudo-aléatoires au sens de la loi normale réduite alors que nous désirions des valeurs comprises entre 0 et 1. Pour cela, nous avons donc utilisé la fonction rand de la librairie numpy de Python.

Ensuite, l'orientation de l'axe est importante. Il faut que les z soient orientés vers le bas avec le zéro en haut à gauche comme on peut le voir sur la figure ci-dessous. (Cf. Figure 28)

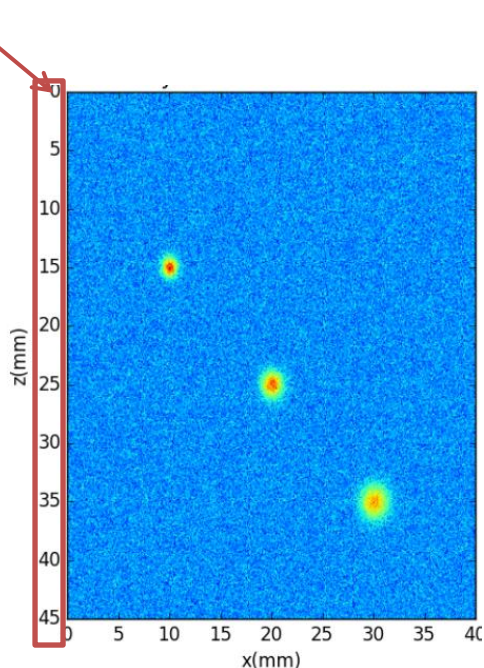


Figure 28 - Visualisation de l'orientation de l'axe

Pour finir, ce critère peut aussi être évalué graphiquement par l'emplacement des boutons, ou bien des zones de texte.

Le **critère visuel** peut être également interprété pour notre interface d'instrumentation. En effet, afin de regarder la cohérence de nos mesures, nous avons réalisé un A-scan sans adaptation d'impédance, sans eau (Cf. Figure 29) et un avec de l'eau ainsi un ou plusieurs échos devaient apparaître. Et, nous avons pu voir l'apparition de ces échos sur notre A-scan. (Cf. Figure 30)

Figure 29 - A-scan sans eau

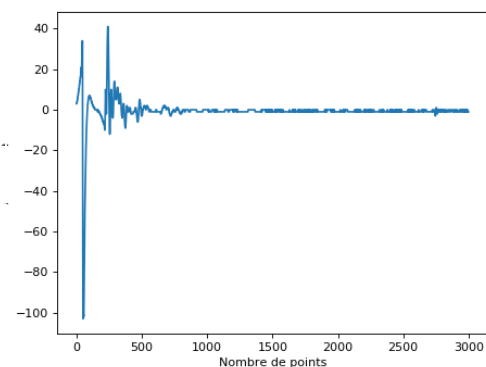
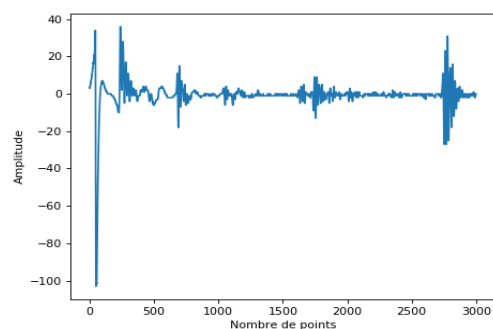


Figure 30 - A-scan avec eau



3.3. Analyse critique du projet

La date de la fin de notre projet ayant été atteinte, nous sommes maintenant en mesure de réaliser une analyse critique sur le déroulement de notre projet.

Nous pouvons diviser cette analyse en différentes parties. Tout d'abord, nous commencerons par étudier notre gestion de projet, ensuite la méthodologie employée et pour finir l'avancement de notre projet.

Premièrement, le projet de fin d'études est l'un de nos premiers projets d'une telle envergure que l'on réalise au sein de notre scolarisation à l'ESEO et surtout le premier projet que l'on fait pour un client provenant d'une entreprise. Après avoir suivi un cours en **gestion de projet** au début du semestre 9, nous avons dû mettre en pratique les méthodes de conduite de projet comme par exemple le planning de GANTT ou bien la matrice RACI. Le premier point réalisé avec le client et notre tuteur pédagogique fut par Skype. Pendant celui-ci, étant des personnes réservées, nous avons eu quelques difficultés à parler avec le client via Skype. De plus, nous aurions dû réfléchir au préalable à la répartition des tâches puisque à cette première réunion une question nous a été posée sur ce sujet et nous n'y avons pas réfléchi avant. Après cette rencontre, nous avons pris l'initiative de réaliser un planning en réfléchissant à la répartition des tâches. La bonne entente et la connaissance de chacun au sein de l'équipe projet a facilité la conduite de ce projet.

En ce qui concerne la **méthodologie employée** au sein du projet, au lancement de celui-ci, nous voulions faire trop de chose en même temps. Notre client, réactif, nous a rapidement aiguillé sur la méthodologie à employer. En effet, lors d'un projet comme celui-ci, nous avons appris qu'il faut toujours commencer par une phase de prototypage en commençant par le plus évident afin de valider des étapes au fur et à mesure. Puisque si l'on commence directement à tenter de se connecter à l'appareil et que cela échoue rien ne peut être mis en avant et validé ainsi il peut s'installer une forme de découragement vis-à-vis du projet à réaliser.

Pour finir, nous devions comme il est cité dans les parties précédentes du rapport, réaliser un imageur ultrasonore sur Python pour le contrôle non destructif. A la fin du projet, différentes choses ont été réalisées :

- **La phase de prototypage, modélisation** : Nous avons réalisé 3 codes permettant de générer des données aléatoires. Premièrement, un code permettant de réaliser une image sous forme rectangulaire, puis deux autres codes pour la représentation sectorielle dont un avec des réflecteurs. Malgré que nous fussions débutants en Python, nous avons réussi à créer ces trois scripts permettant l'implémentation d'une interface graphique pour chacun d'entre eux. Néanmoins, n'étant pas des experts en Python certaines parties du code peuvent, on pense, être optimisées afin de réduire le temps d'exécution des différents programmes.

- **La phase d'interfaçage avec l'appareil à notre disposition** : Nous avons réussi dans un premier temps à établir la connexion entre le module et notre code, la déconnexion et la lecture dans un document. Ensuite, nous avons pu récupérer les données issues de la sonde. Cette étape fût plus difficile, nous y avons consacré du temps et des moyens comme il est explicité dans la partie Méthodologie avec Wireshark. Ainsi, nous avons pu modifier notre interface graphique produite dans l'étape précédente afin d'afficher un A-scan fixe puis une A-scan mobile grâce à l'utilisation d'une boucle et pour finir afficher un B-scan.

Néanmoins, des améliorations peuvent être apportées sur l'implémentation et la conception de notre interface. En effet, premièrement, il aurait été bien de donner via l'interface la possibilité de modifier les paramètres du fichier de configuration. Ensuite, nous avons pu remarquer que lorsque la connexion renvoi le booléen FALSE notre code continue en donnant tous les autres booléens à FALSE. Il aurait été préférable de s'arrêter lorsque la connexion ne s'établit pas.

Et pour finir, étant donné que nous connaissions la fréquence d'échantillonnage qui est de 100 MHz, il aurait été bien d'afficher les différents graphiques non pas en nombre de point mais plutôt en fonction du temps.

Conclusion

Pour conclure le projet, nous pouvons affirmer que ce projet nous a permis de développer de nombreuses compétences aussi bien techniques, scientifiques que managériales.

En effet au niveau technique, nous avons revu et approfondi nos connaissances en Python. Par exemple, nous avons utilisé des bibliothèques numpy, scipy, matplotlib pour la partie mathématique et affichage de graphique scientifique. Nous avons aussi utilisé la bibliothèque socket pour la partie réseau.

Au niveau scientifique, nous avons pu revoir les notions d'imagerie ultrasonore ainsi que leur acquisition en image scan. Nous avons aussi revu les notions de réseaux et de protocole de communication (TCP/UDP/IPv4).

Dans un contexte plus managérial, nous avons appris les contraintes qu'imposent un projet d'une telle durée, l'envoi des rapports hebdomadaires, planifier et préparer des réunions, motiver l'équipe face à une difficulté ou après avoir validé un process.

Ce projet a été suivi par le client qui a émis certaines mises en garde ou conseils sur l'avancement du projet et a validé les étapes.

En effet, différents objectifs ont été atteints à travers ce projet d'industrialisation :

- Modélisation d'un prototype
- Comparaison FPS
- Connexion avec la machine
- Acquisition A-scan
- Acquisition B-scan
- Réalisation .exe

Quant aux perspectives sur ce projet, il sera certainement poursuivi au sein de l'entreprise ou dans le cas d'un futur projet de fin d'étude pour améliorer l'interface graphique. Bien que nous ayons fait des recherches sur l'amélioration de nos fonctions des différents codes, on peut aussi ajouter le fait de rechercher des fonctions Python plus optimisées afin d'améliorer le FPS.

Annexes

Annexe 1 : Gestion de projet

Pour accompagner le développement du projet dès le stade d'ébauche en septembre, un cours de gestion de projet a rapidement été mis en place. Ce cours introduit la méthodologie à suivre et les outils nécessaires au bon déroulement d'un projet. Étant dans une dimension ingénieur, cette gestion est d'autant plus importante que le respect des délais, des coûts et de la performance est essentiel dans la conception d'un système complexe. La gestion de projet permet d'autre part de distribuer les travaux à réaliser entre les membres de l'équipe mais également de créer une base de référence permettant de surveiller les écarts et l'évolution du projet afin d'assurer sa continuité.

1.1. Gestion générale de la répartition des tâches au sein du groupe



Afin de réaliser notre projet de fin d'études, nous avons mis en place une stratégie de travail en groupe précise. En effet, la plupart du temps nous avons réparti les tâches entre nous. Cependant, lors de gros soucis par exemple lors du débogage, nous nous sommes tous concentrés sur le problème afin de le résoudre le plus rapidement possible.

1.2. La matrice RACI

Afin de mieux percevoir la répartition des tâches au sein du groupe et des différents acteurs, nous vous présentons une matrice RACI (Cf. Figure 31)

En effet cet outil, aussi appelé matrice des responsabilités, permet d'indiquer les rôles de chaque intervenant pour une tâche donnée. En ce qui concerne les lettres (R, A, C et I) renseignés dans le tableau voici les significations :

- **R** : personne responsable de la réalisation, **Responsible**
- **A** : personne qui doit donner son approbation, **Accountable**
- **C** : personne que l'on consulte, **Consulted**
- **I** : personne que l'on doit tenir informé sur l'avancement de la tâche, **Informed**

Grille RACI	Sponsor Client	Tuteur pédagogique	Equipe projet : Danielle	Equipe projet : Nicolas	Chef de projet : Gwendoline	Externes/Experts
Compréhension du sujet	C	C	R	R	R	
Documentation	I		R	R	R	
Réunions						
1. Tuteur école : M.Longo		R	R	R	R	
2. Client : M.Carcreff	R		R	R	R	
Matlab à Python - Modélisation de données						
1. Interface graphique (GUI)	I/A	C		R	I/R	C
2. Générer des images aléatoires	I/A	C	R		I/R	C
3. Ajout des conditions réelles : bruit gaussien	I/A	C	R		I/R	C
4. Design MVC	I/A		R	R	I/R	C
5. Amélioration du FPS et résolution des soucis de l'interface	I/A				R	C
Connexion avec le module-instrument						
1. Se renseigner sur les librairies dll	C		R	R	I	
2. Fonctionnalités de base (connexion, newDevice, ...)	I/A	C	R	R	I	C
3. Récupération des données et affichage d'un Ascan fixe	I/A	C	R	R	I/R	C
4. Récupération des données et affichage d'un Ascan mobile	I/A			R	I/R	
5. Réalisation et affichage d'un B-scan	I/A	C	R	R	I/R	
6. Mettre à jour le GUI Python	I/A		R		I/R	
Tests						
1. Test visuel	I/A		R	R	I/R	
2. Test par suivi de variable	I/A		R	R	I/R	
Préparation de la soutenance et du fin de projet	C	C	R	R	I/R	
Soutenance de projet	I/A	C	R	R	R	

Figure 31 - Matrice RACI

Annexe 2 : Planning

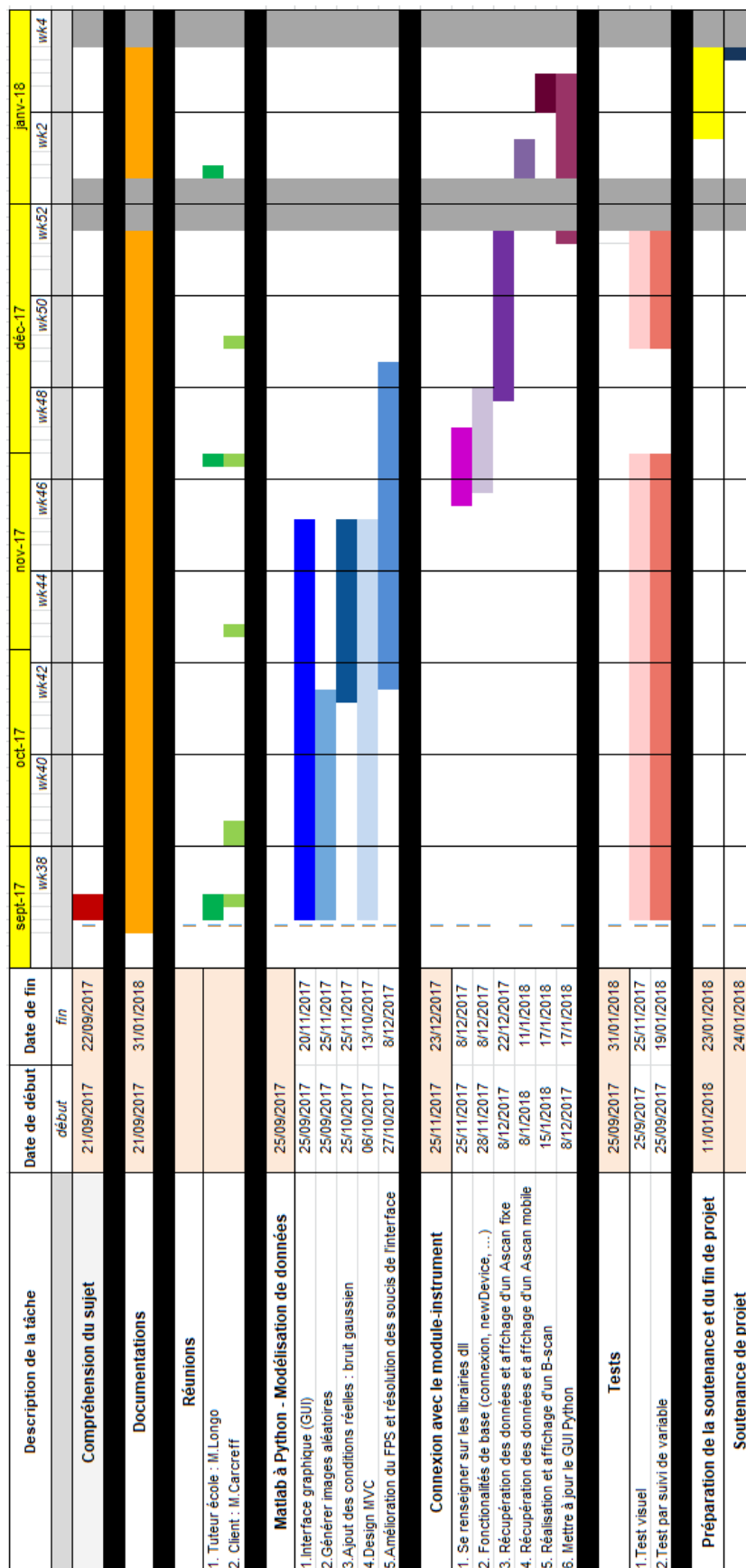


Figure 32 - Planning de GANTT

Annexe 3 : Difficultés rencontrées

Tout au long de ce projet, nous avons rencontrés diverses difficultés. Celles-ci ont pu être surmontées et nous ont permises d'apprendre de nouvelles choses.

3.1. Apprentissage d'un langage : Python

Durant la classe préparatoire, nous avons eu l'occasion de découvrir les bases du langage Python de par certain exercice de mathématiques. Nous n'avions pas eu l'opportunité de travailler avec des librairies scientifiques par exemple. Il nous a fallu réapprendre les bases et plus encore afin d'appréhender ce projet. En effet, celui-ci a nécessité l'utilisation de plusieurs librairies de traitement d'images chose que nous n'avions jamais faite en Python. D'autant plus que les librairies utilisées sont très prisées dans le domaine de la recherche et du développement. Ainsi, nous avons acquis de nombreuses compétences que nous pourrons valoriser à l'avenir pour notre parcours professionnel.

3.2. Installation des librairies

Python est un langage nécessitant des librairies afin de fonctionner de manière complète. Nous avons dû installer plusieurs librairies : numpy (pour la gestion des matrices), matplotlib (pour la gestion et l'affichage graphique). Pour cela, la commande pip-install devait être utilisée. Seulement, la commande pip n'était pas reconnue. Il a donc été nécessaire de réaliser des recherches afin de la rendre fonctionnelle et d'installer les librairies Python nécessaires au développement de notre interface. Afin que toute personne puisse installer ces librairies (y compris notre client), nous avons réalisé un tutoriel. (Cf. texte ci-dessous en italique)

Guide installation (Utilisation Pip)

- *Télécharger Python- 3.6.3.exe et Pycharm community 2017 version 2.3*

Pour que les librairies soient fonctionnelles :

1. *Aller dans l'invite de commande Windows*
2. *Aller dans le dossier qui contient les librairies et Python (téléchargements)*
3. *Exécuter la commande suivante " pip install Nom_librairie"*

Il y a deux possibilités soit votre pc reconnait pip (ce qui n'était pas notre cas) sinon:

1. *Retourner dans le dossier téléchargement dans l'invite commande « cd downloads » puis « python-3.6.3.exe » cela ouvre python Modify setup*
2. *Dans Optional Features sélectionner toutes les cases. **Attention** : surtout n'oublier pas de cocher la case pip qui nous intéresse pour l'installation) puis cliquer sur next*
3. *Dans Advanced options sélectionner à nouveau les cases. **Attention** : Ne pas oublier la case « add python to environment variables » puis cliquer sur install*

A nouveau soit le pc accepte immédiatement de reconnaître pip soit comme nous:

- *Aller dans panneau de configuration Système et sécurité -> Systèmes -> Paramètres systèmes avancés -> Fenêtre Propriétés systèmes aller dans variables environnement*
- *Si tout va bien la case variable utilisateur path indique bien l'environnement variable ajouté précédemment. Il faut copier ce chemin (path).*
- *Dans variables systèmes cliquer sur «nouvelle » et renseigner le nom de variable (ex: PYTHON3_HOME).*
- *Dans valeur de la variable coller le path copié au-dessus*
- *A nouveau dans variables systèmes, cliquer « nouvelle » pour renseigner le nom de variable (ex: PYTHON3_SCRIPT) et la valeur de la variable (reprendre le nom de la 1ere variable avec % ex: %PYTHON3_HOME% Scripts\)*
- *Enfin dans variable système sélectionner path et créer nouvelle variable (ex :%PYTHON3_SCRIPT%)*

Après toutes ces étapes pip devrait fonctionner. Pour vérifier dans commande taper "pip install Nom_librairie"

Cela nous a permis d'apprendre à expliquer le plus clairement possible les démarches à suivre afin d'installer les librairies utilisées indispensables au projet par la réalisation d'un tutoriel synthétique et précis.

Cependant, suite à l'installation de Pycharm pour développer le code Python toute cette démarche est obsolète.

Dorénavant, via le logiciel Pycharm, on peut directement installer les librairies par de simples clics dans settings présent dans l'onglet file du logiciel.

3.3. Choix de l'architecture

Afin de réaliser notre interface graphique, nous avons rencontré quelques difficultés pour choisir le modèle adéquate pour notre application. Nous avons été rencontré un professeur d'informatique de l'ESEO afin d'obtenir le plus de conseil possible.

Nous avons finalement opté pour un modèle MVC (Modèle Vue Contrôleur). Il s'agit d'un modèle où la partie réalisant des calculs sera dans une classe Modèle, la partie réalisant l'esthétisme de l'interface sera dans la classe Vue et la partie permettant d'interagir sera dans une classe nommée Contrôleur.

Cette étape du projet peut parfois paraître « inutile » seulement c'est entièrement l'inverse puisqu'un code structuré permet la meilleure compréhension de celui-ci par un acteur externe par exemple.

3.4. Débogage

Au début de notre projet, nous n'avions pas d'IDE. Nous utilisions l'IDLE Python par défaut qui ne permettait pas de mettre des points d'arrêt pour le suivi de variables lors du débogage. Nous connaissions certains logiciels de développement tels que Eclipse mais ne savions pas s'il était adapté pour le langage Python. Nous avons donc demandé conseil à nos enseignants qui nous ont aiguillés vers Pycharm.

Ensuite, lorsque notre IDE fut fonctionnel et surtout le debugger, nous avons pu faire des tests unitaires afin de déboguer notre programme avec l'aide d'internet et de nos connaissances en programmation dans d'autres langages (C, C++, Matlab,...).

Nous retenons de cette difficulté, qu'il est primordial d'avoir l'IDE adéquate pour développer correctement.

3.5. Récupération des données issues de l'appareil

Cette étape fut sans doute la plus compliquée du projet. Malgré que nous ayons de la documentation et des ressources software (dll, fonctions mex Matlab) données par le client, pendant presque deux semaines nous sommes restés bloqués sur l'acquisition des données. En effet, lors de cette acquisition nous arrivions à récupérer une taille de données identique à celle de Matlab. Néanmoins, nous n'arrivions pas à les afficher car les mesures obtenues étaient non pas un vecteur de valeurs mais un scalaire nul.

Suite à ce problème, nous avons commencé par utiliser un logiciel nommé Wireshark afin de suivre le transfert de paquets et ainsi être sûr que des données transitaient bien et que la connexion était établie avec le boîtier.

Notre erreur venait, finalement, d'une mauvaise manipulation des fichiers dll et d'un mauvais dimensionnement des données pour l'affichage des mesures dans notre interface.

Cette tâche a pu être accomplie suite à la persévérance des membres du groupe et à la recherche du problème par l'intégration d'un autre outil : Wireshark.

Annexe 4 : Remerciements

Au terme de ce projet, nous tenons à exprimer notre profonde gratitude et nos sincères remerciements à notre client Ewen CARCREFF pour le temps qu'il nous a consacré, les directives précieuses et pour la qualité de son suivi pour toute la période de notre projet.

Nous tenons aussi à remercier vivement le responsable de l'option, qui a également été notre tuteur pédagogique, M.LONGO Roberto, qui a su nous inculquer la méthodologie à tenir lors de la réalisation d'un projet.

Nos vifs remerciements s'adressent aussi à M.AUBIN, M.CHAVIN, M.ROBINET, et M.CLAVREUIL pour leur gentillesse, disponibilité et le temps qu'ils nous ont consacrés.

Nos remerciements vont enfin à toutes les personnes qui ont contribué de près ou de loin à l'élaboration de ce projet.

Annexe 5 : Glossaire

ACK : ACKnowledgement pour l'accusé de réception d'information

CND : Contrôle Non Destructif

FPS : Frame Per Seconde

GUI : Graphic User Interface

ICMP : Internet Control Message Protocol

IDE, IDLE : Integrated DeveLopement Environment

MVC : Modèle Vue Contrôleur

PSH : PuSH (drapeau flag TCP)

SYN : Demande l'établissement de la connexion

TCP : Transmission control Protocol

UDP : User Datagram Protocol

Annexe 6 : Table des illustrations

Figure 1 - Présentation rapide de l'entreprise cliente	4
Figure 2 - Exemple de contrôle non destructif sur une aile d'avion	5
Figure 3 - Représentation d'un faisceau US (en rouge les zones de haute énergie et en bleu et vert les zones plus basse d'énergie).....	7
Figure 4 - Schémas illustrant la composition d'une sonde ultrasonore mono élément et à deux éléments.....	9
Figure 5 - Exemple d'A-scan (avec la détection du défaut à la distance D)	10
Figure 6 - Image matricielle	14
Figure 7 - Image matricielle bruitée.....	15
Figure 8 - Diagramme UML pour l'image matricielle	16
Figure 9 - Diagramme UML de classe	17
Figure 10 - Image sectorielle	18
Figure 11 - Matériel OEMPA	19
Figure 12 - Capture d'un transfert de paquet (logiciel : Wireshark)	20
Figure 13 - Fenêtre dialogue utilisateur	22
Figure 14 – Diagrammes UML A-Scan (à gauche) et B-Scan (à droite)	24
Figure 15 – Interfaces graphiques d'image matricielle pour une variance = 20 et une variance = 100	25
Figure 16 - Interfaces graphiques d'image matricielle pour taille 2048*2048 puis 1024*1024.....	25
Figure 17 - Interface graphique d'une image sectorielle	26
Figure 18 - A-scan émis/reçu sur le bloc aluminium sans ajout d'eau.....	26
Figure 19 - A-scan émis/reçu sur le bloc d'aluminium avec ajout d'eau	27
Figure 20 - A-scan en continu avec eau	27
Figure 21 - B-scan	28
Figure 22 – Tests des FPS Python/Matlab en fonction de la taille des images.....	29
Figure 23 - Tests entre les FPS Matlab/Python pour l'image matricielle et sectorielle	29
Figure 24 - Extrait du profiler obtenu sous Python	31
Figure 25 - Extrait du profiler Matlab.....	31
Figure 26 - Profiler de la fonction gui_mainfcn et run_checkbox_callback.....	32
Figure 27 - Profiler de la vue matricielle.....	32
Figure 28 - Visualisation de l'orientation de l'axe.....	34
Figure 29 - A-scan sans eau	34
Figure 30 - A-scan avec eau	34
Figure 31 - Matrice RACI.....	39
Figure 32 - Planning de GANTT.....	40