Analyseur de lobe de détection de capteurs radar et infrarouge sur MATLAB®

Ing. N. MAILLARD
Ing. R. P. DELMOT
Ir O. GILLIEAUX
GRAMME – Liège

L'analyseur de lobe est un outil développé sur MATLAB, permettant de tracer et mesurer des lobes de détection de capteurs radar et infrarouge. Les mesures sont réalisées par l'intermédiaire de deux lasers scanners. L'application est constituée d'une plateforme software et d'une interface hardware. De plus toute la conception s'est basée sur l'utilisation de l'USB de manière à optimiser la rapidité et la flexibilité de l'outil.

Mots-clefs : MATLAB, laser, mesure, USB, interface graphique, lobe radar, lobe infrarouge, détecteur, analyseur

The lobe analyzer is a tool developed on MATLAB, which enables to trace and measure detection lobes of radar and infrared sensors. Measurements are taken by two lasers scanners. The application consists of a platform software and an interface hardware. Besides, the whole conception is based on the use of USB to maximize the speed and the flexibility of the tool.

Keywords : MATLAB, laser, measure, USB, graphical interface, radar lobe, infrared lobe, sensor, analyzer

1. Introduction

« Bureau d'Electronique Appliquée » est une société active à l'échelle internationale dans le secteur de la détection de personnes et de véhicules. Le marché de la détection pour portes automatiques piétonnes et industrielles représente une partie importante de ses activités.

Le travail de fin d'études se porte sur un analyseur de lobe qui consiste en un logiciel développé sur MATLAB, capable de tracer et mesurer des lobes de détection de capteurs radar et infrarouge. Cet outil exploite les mesures de distance fournies par deux lasers scanners qui sont un nouveau produit conçu par « Bureau d'Electronique Appliquée ».

Cette société développe entre autres des détecteurs radar à antenne planaire qui consistent à émettre des micro-ondes se propageant suivant un lobe, qui ressemble fort à un ballon de baudruche. Un lobe radar représente la zone de détection de mouvement du détecteur, un objet en mouvement hors de celui-ci ne sera donc pas détecté.

Le lobe dépend du type d'antenne utilisée. Il existe en générale deux types d'antenne planaire, une à lobe étroit et une à lobe large.

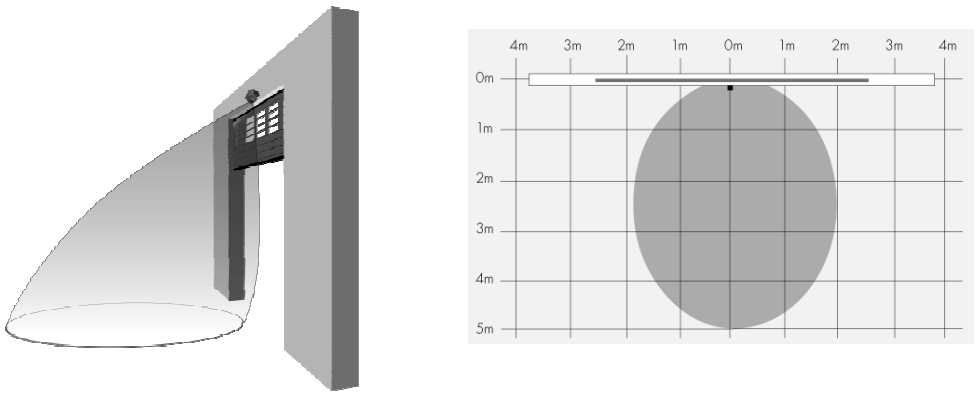

Figure 1 : Représentation du lobe d'un détecteur radar

Il existe aussi des capteurs infrarouges qui consistent à émettre de la lumière infrarouge suivant plusieurs spots. Le lobe infrarouge représente la zone de détection de présence du détecteur. Un objet en dehors de celui-ci n'est pas détecté.

Pour un capteur radar, il est difficile de déterminer de manière précise l'endroit où l'objet va être détecté. En effet la détection va dépendre de la taille de l'objet, de la matière dont il est constitué et de sa direction de déplacement.

Il est donc important d'analyser les lobes de détection produits par les capteurs, afin de définir les zones de détection, de vérifier la dispersion de ceux-ci, mais aussi de pouvoir définir un produit étalon pour la production.

La méthode utilisée pour mesurer des lobes consiste à déplacer une cible dans la zone de détection, puis à enregistrer sa position absolue lorsque celle-ci est détectée par le capteur. Sa position est calculée grâce aux mesures reçues par les deux lasers scanners.

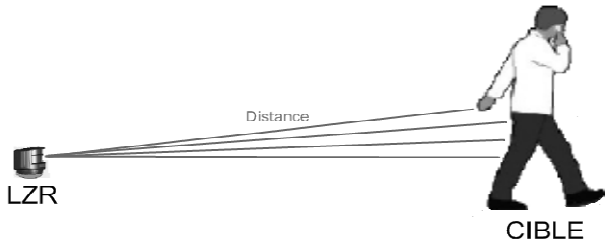

Figure 2 : Méthode de mesure

2. Intérêt de l'utilisation de MATLAB

MATLAB est un langage de calcul scientifique de haut niveau et un environnement interactif pour le développement d'algorithmes, la visualisation et l'analyse de données, ou encore le calcul numérique. En utilisant MATLAB, il est possible de résoudre des problèmes de calcul scientifique plus rapidement qu'avec les langages de programmation traditionnels, tels que C, C++ et Fortran.

MATLAB est utilisable dans une grande variété d'applications, incluant le traitement du signal et d'images, les communications, la conception de systèmes de contrôle, les tests et les mesures, la modélisation et l'analyse financière, ainsi que la biologie informatique. Il est possible d'intégrer son code MATLAB avec d'autres langages et applications, et de distribuer ses algorithmes et applications.

3. Cahier des charges

L'objectif du travail est le développement d'une interface graphique sous MATLAB, permettant de communiquer via USB avec les deux lasers scanners et avec une carte d'acquisition USB.

L'accès aux périphériques externes se fait donc uniquement via USB, ce qui optimise l'utilisation de l'outil sur laptops. En effet, le port série RS232 n'est plus accessible sur les portables de dernière génération.

Le travail à réaliser peut se diviser en trois parties.

3.1 Acquisition des signaux des lasers scanners au sein de MATLAB

La connexion de l'outil avec le laser scanner doit se faire de manière automatique. Cet outil doit intégrer également la possibilité de configurer les paramètres utiles de l'appareil. Le nombre de laser scanner à connecter à l'application doit être analysé.

Il faut une solution qui permette d'atteindre le port USB à partir de MATLAB. Pour cela, nous allons nous baser sur une application écrite en C++, qui utilise un pilote ainsi qu'une librairie USB. De plus nous utiliserons un protocole défini spécialement pour le laser scanner, qui permet de lui envoyer des requêtes afin de recevoir les données souhaitées.

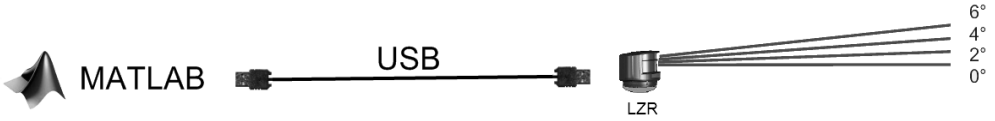

Figure 3 : Méthode d'acquisition des LZR

3.2 Conception d'un périphérique USB

Lorsque le capteur sous test détecte un objet en mouvement ou en présence, il commute une sortie relais afin de provoquer l'ouverture de la porte automatique. Cette commutation est signalée en synchronisation au logiciel par l'intermédiaire d'une carte d'acquisition USB.

La conception de cette carte d'acquisition fait partie du travail à réaliser. Elle se fait sur base d'un PIC 18F2550 et utilise une librairie ainsi qu'un pilote USB « mpusbapi » fournis par Microchip. De même que pour l'acquisition des lasers scanners, il doit être possible d'acquérir ce périphérique USB depuis MATLAB.

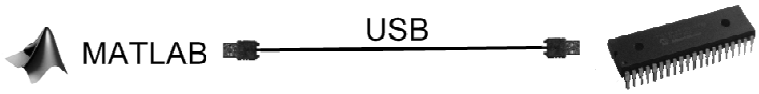

Figure 4 : Méthode d'acquisition de la carte USB

3.3 Conception d'une interface graphique sous MATLAB

Le travail consiste en l'élaboration d'une interface graphique sous forme de fenêtre regroupant un certain nombre de fonctionnalités. De plus, la plate-forme software doit être transportable, et utilisable aussi bien sur desktop que sur laptop.

Cette interface doit notamment permettre à l'utilisateur d'afficher, de modifier, de comparer et d'exporter les mesures réalisées. Il s'agit de tracer les limites de la zone de détection pour le capteur sous test. Pour cela l'outil doit exploiter les informations de distance fournies par le laser scanner. En fonction du capteur sous test, les limites de détection peuvent être conditionnées par la direction et le sens du déplacement. L'outil doit ainsi prendre en compte et afficher ces différentes informations.

Le lobe obtenu par les mesures doit pouvoir être édité en vue de supprimer certains points de mesure jugés incohérents ou en ajouter avec une reconstruction de la limite du lobe. Il intègre un moyen d'afficher une échelle et/ou une grille de distance sur l'image.
Les tracés peuvent être évidemment sauvegardés et rechargés dans l'interface. De plus, l'affichage doit permettre la superposition de plusieurs tracés en différentes couleurs en vue de comparaison.

L'ajout de commentaires doit également être possible. Une des procédures de tracé de lobe consiste à se déplacer dans la zone de manière aléatoire. Le système enregistre les distances mesurées par les deux lasers scanners lors

des détections pour en calculer le lobe. Reste au système à tracer un nuage de points correspondant aux différentes positions de détection. Le centre de gravité, la direction ainsi que la vitesse doivent être définis. L'utilisateur peut exporter un tracé ou des tracés superposés sous la forme d'une image, mais aussi les valeurs de mesure dans une feuille Excel.

4. Acquisition des mesures

4.1 Concept

L'utilisation de plusieurs lasers scanners permet d'avoir une meilleure précision au niveau de la recherche du centre de l'objet, de la direction du déplacement et de la vitesse de celui-ci. Dans notre cas, il a été décidé d'utiliser deux lasers scanners afin de ne pas trop alourdir la gestion de l'USB. Il s'agit d'un compromis entre la rapidité et la précision du système.

Le laser scanner réalise une série de mesures de distance suivant plusieurs plans inclinés à différents angles. Chaque plan est constitué de 274 mesures, et ce sur une ouverture de 95°. Il constitue l'élément majeur de l'application car lui seul fournit les données brutes. Son concept repose sur le temps de vol, qui consiste à mesurer le temps que prend un pulse laser pour se réfléchir contre une cible puis revenir vers l'appareil.

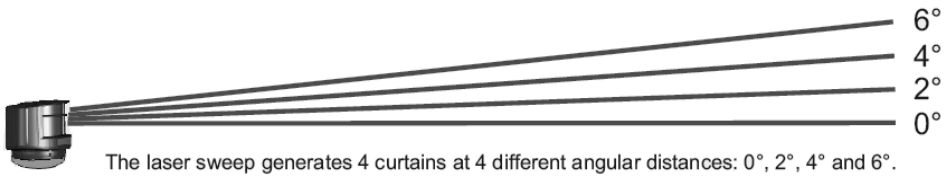

Figure 5 : Aperçu des plans de mesure du laser scanner

4.2 Communication USB sous MATLAB

Introduction

Le projet consiste à réaliser une acquisition de données provenant des lasers scanners, par le port USB, au sein même de MATLAB. Il est évident que MATLAB ne fournit pas d'accès au port USB, ainsi il est possible de le faire uniquement en utilisant une interface externe à MATLAB*.*

Par interface externe, il faut comprendre une application non codée sous MATLAB. En effet, il est possible de compiler du code C/C++ ou du JAVA dans MATLAB*.*

« Sourceforge.net[1] » propose l'application LibUSB qui regroupe une librairie et un driver. LibUSB-Win32 est une version de la librairie USB « libusb » pour les systèmes d'exploitation Windows (Win98SE, WinME, Win2k, WinXP). La librairie permet aux applications d'accéder en mode utilisateurs à n'importe quel « device USB » sur Windows. Et cela, de manière générique sans devoir écrire une seule ligne de code de pilote au niveau du noyau Windows.

Fonctions LibUSB :

- Ce pilote peut-être utilisé comme un filtre pour des périphériques USB déjà installés. Cette fonction permet à l'application LibUSB-Win32 de communiquer avec n'importe quel périphérique déjà installé.
- Peut être utilisé en tant que pilote pour des périphériques n'en possédant pas.
- Supporte le transfert « Bulk » et « Interrupt transfers ».
- Supporte toutes les requêtes standards.

Pour pouvoir utiliser cette application, il faut nécessairement un système d'exploitation Windows XP, un compilateur C++ et la DDK[2] pour Windows XP.

[1] http://libusb-win32.sourceforge.net/

[2] Driver Development Kit

Il existe deux moyens pour utiliser une librairie (C/C++) au sein de MATLAB :

- Télécharger la librairie dans MATLAB avec la commande « loadlibrary »
- Créer un MEX-file qui utilise la librairie

Seule la deuxième solution a été exploitée, car la première réalise une conversion automatique de type[3] qui n'est pas optimisée.

Définition d'un MEX-File

Un MEX-file contient une gateway routine (passerelle) et une computational routine (routine de calcul). Cette dernière contient le code C/C++ que l'on désire exécuter. En réalité la gateway routine permet de faire le transfert entre le code MATLAB et le code C/C++. C'est donc par celle-ci que passent les variables que l'on désire échanger entre ces deux langages.

Lors de l'appel du MEX-file, MATLAB utilise un compilateur C/C++ par défaut, qui s'appelle « lcc ». Cependant, il est possible de choisir celui-ci, par la commande « mex –setup ».

A la base un MEX-file, n'est rien d'autre qu'un fichier source (.c/.cpp), qui une fois créé, ne nécessite plus que l'utilisation de la commande « mex filename.cpp ». Cette commande permet de compiler le fichier source et de créer le MEX-file dans MATLAB. En réalité, un fichier binaire « .mexw32 » est créé lors de la compilation. Ce qui va permettre à MATLAB de traiter le MEX-file tout comme un M-file[4].

Ainsi, lors de l'appel du MEX-file, MATLAB utilise le fichier binaire « .mexw32 ». Il faut évidemment associer à ce fichier la librairie USB utilisée.

Comme déjà énoncé, la gateway routine est le point d'entrée vers le MEX-file, c'est donc à travers celle-ci que MATLAB accède à la computational routine.

---

[3] Convertir une valeur d'un type (source) dans un autre (cible). On parle aussi de coercition ou de cast.

[4] Fichier contenant du code MATLAB

4.3 Pilotes

Les pilotes permettent une interface logicielle avec le matériel. Ces derniers sont conçus pour assurer un fonctionnement transparent au niveau de l'utilisateur.

Un point fort de l'USB est sa configuration automatique, on l'appelle aussi le « plug & play ». Cela signifie que si l'utilisateur connecte un périphérique USB, Windows détecte automatiquement ce périphérique et charge le pilote approprié s'il est disponible dans les fichiers de Windows.

Un pilote a le plus souvent l'extension (SYS). Un fichier SYS est un fichier compilé et il est impossible de le modifier pour l'adapter à un autre type de périphérique. Sans celui-ci, le composant ne sera pas installé et ne pourra pas communiquer avec le PC.

Les pilotes utilisés dans ce cas ci, proviennent de Sourceforce.net et de Microchip. Ils nous permettent d'accéder facilement au port USB. Le principe de base consiste à récupérer un pointeur sur le flux en entrée et en sortie du PC. Ainsi, nous pouvons écrire sur le bus USB en utilisant les fonctions données par les librairies « libusb » et « mpusbapi ».

5. Dispositif de détection

L'application nécessite une communication entre le détecteur sous test et le logiciel. En effet, le détecteur sort un signal de détection qui doit être observé par une interface de communication qui comme son nom l'indique communique avec le logiciel.

Le protocole utilisé est évidemment l'USB, il s'agit donc d'une interface USB qui a le même rôle qu'une carte d'acquisition. Le microcontrôleur utilisé est un PIC 18F2550 de Microchip. Il permet notamment de réaliser des communications USB full-speed.

Ce périphérique, consiste donc à communiquer au logiciel l'état des relais des détecteurs. Dans notre cas, il est possible de relier quatre relais sur le périphérique.

Voici comment est implémenté le module USB :

Figure 6 : Visualisation de l'interface USB

Le choix du microcontrôleur s'est porté sur un PIC de la famille des 18F pour la simple raison que Microchip propose pour celle-ci une librairie et un pilote permettant de réaliser des communications USB.

5.1 Hardware

L'interface a été conçue de manière à être autoalimentée par l'USB. Celui-ci en effet fournit une alimentation 5V avec un courant de 500 mA maximum. L'absence de circuit d'alimentation simplifie évidemment la conception de l'interface. Celle-ci a été réalisée sur PROTEL 2004.

La carte est constituée des éléments suivants :

- Quatre entrées digitales permettant de raccorder les relais des détecteurs
- Un connecteur USB de type B
- Un connecteur permettant de réaliser une programmation « In circuit » par l'intermédiaire d'un PICKIT 2, de Microchip
- Quatre LEDs pour la signalisation.

Les entrées relais, ont été pensées de manière à éviter des courts-circuits accidentels. En effet, l'utilisateur pourrait par mégarde relier un potentiel positif ou négatif au lieu des deux broches du relais qui sont toujours libres de potentiel. Dans ce cas, le courant pourrait devenir trop grand et provoquer des dommages à la carte.

Certains ports USB possèdent une sécurité contre les courts-circuits, mais ce n'est pas le cas de tous. Et enfin, un filtre passe bas est présent à chaque entrée pour rejeter les hautes fréquences.

Les quatre LEDs permettent premièrement de signaler à l'utilisateur la présence d'un signal de détection pour chaque entrée digitale et deuxièmement de l'alerter en cas de problème de fonctionnement de l'application, tel que la perte des périphériques USB par l'ordinateur.

Voici un aperçu en trois dimensions de la carte électronique.

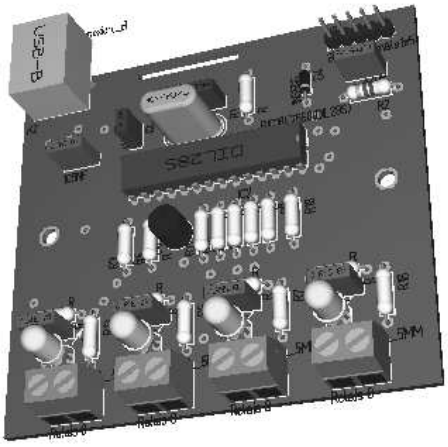

Figure 7 : Représentation en trois dimensions de la carte électronique de l'interface USB

5.2 Software

Le microcontrôleur contient un programme qui a été codé en C et compilé avec le compilateur MCC18 qui est dédié à la famille des PIC 18F.

La base du programme est en réalité fourni par Microchip, mais il doit être modifié de façon à ce qu'il corresponde à l'application. En effet, la programmation d'un périphérique USB est fort complexe, il y a beaucoup de paramètres en jeu et cela nécessite une très bonne connaissance de la norme USB dans tous ses détails. C'est aussi pour cela que Microchip

fournit un pilote USB, évitant ainsi au programmeur de devoir en concevoir un par lui même.

6. Interface Graphique

L'interface graphique développée pour l'application se base sur l'utilisation de l'outil GUIDE fourni par MATLAB. Cet outil permet de développer des Interfaces Graphiques (GUI[5]), se basant sur du code MATLAB. Il permet de gérer assez facilement des objets graphiques, ainsi que les événements qui y sont associés.

Implémentation de l'interface graphique

L'interface graphique est conçue autour d'une fenêtre principale, à partir de laquelle l'utilisateur peut accéder à l'ensemble des fonctions disponibles. Il s'agit réellement d'une application orientée vers l'utilisateur, en effet, aucune ligne de code n'est nécessaire pour exploiter l'outil dans sa globalité. Tout est accessible à partir des fenêtres qui sont toutes liées entre elles.

Toutes les figures de l'application ont un M-file (.m) associé, dans lequel, se trouve le code MATLAB permettant d'interagir avec l'utilisateur.

L'ensemble du programme se base sur une seule fenêtre, donc sur un M-file principal « *LOBE_ANALYZER.m* ». C'est d'ailleurs dans celui-ci que le processus d'acquisition a lieu.

[5] GUI : Graphical User Interface

L'interface possède trois niveaux hiérarchiques que voici :

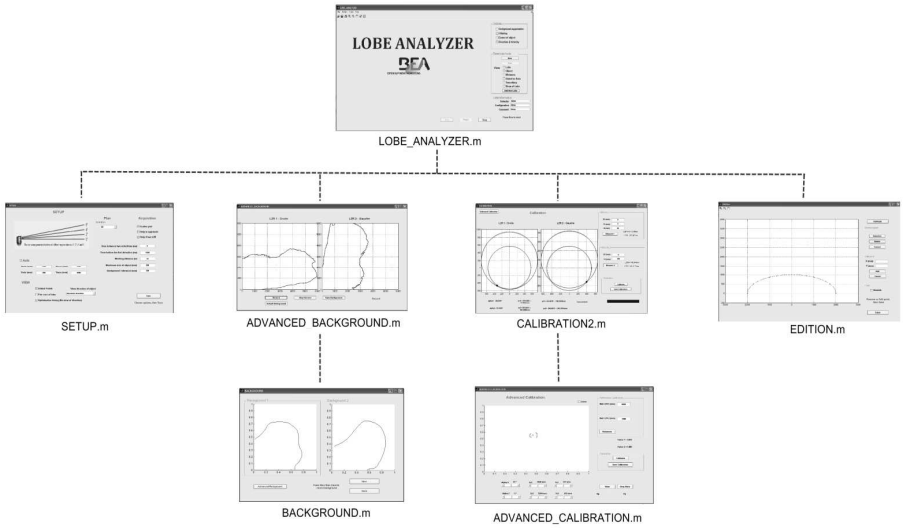

Figure 8 : Représentation de l'interface graphique

La fenêtre LOBE_ANALYZER, permet d'afficher les mesures réalisées par les lasers scanners, mais aussi de gérer l'outil, en sélectionnant des modes de fonctionnement ou en accédant à des fenêtres spécifiques. Il y a quatre fenêtres secondaires, la première qui s'appelle SETUP, permet de configurer l'outil, en cochant des options d'affichage ou d'acquisition, ou en entrant des valeurs de timings pour le procédé d'acquisition.

ADVANCED_BACKGROUND, qui est la deuxième fenêtre, permet de définir la zone de détection de l'outil, en effet, celui-ci doit réaliser une sorte de photo de la zone où l'on va effectuer les tests de détection. La troisième fenêtre CALIBRATION, réalise comme son nom l'indique une calibration de l'outil, de manière à ce qu'il puisse se repérer sur la zone de détection.

Vient ensuite la fenêtre EDITION, qui laisse la possibilité d'éditer les résultats, en supprimant ou en ajoutant des données.

Le dernier niveau est composé de deux fenêtres qui sont assez peu utilisées puisqu'elles ne sont pas directement utiles au fonctionnement de l'application. Il y a la fenêtre BACKGROUND, qui a le même rôle que la

fenêtre ADVANCED_BACKGROUND mais cependant avec une liberté moindre au niveau du type de zone de détection. La fenêtre ADVANCED_CALIBRATION, permet juste d'affiner la calibration de l'outil, au cas où il serait nécessaire de le faire.

7. Analyse globale du procédé d'acquisition

L'implémentation de l'application sous MATLAB, nécessite la définition d'un algorithme d'acquisition permettant de diviser le processus en plusieurs opérations logiques et implémentables individuellement.

En effet, tout logiciel de calculs et de mesures répond à un processus algorithmique, consistant à traiter un flux de données entrant pour sortir un flux de variables utiles. Dans ce processus, se trouvent plusieurs sous processus, qui réalisent une partie du traitement global. Il s'agit en réalité de fonctions implémentées dans différents M-file.

Voici un aperçu des opérations présentes dans ce cycle.

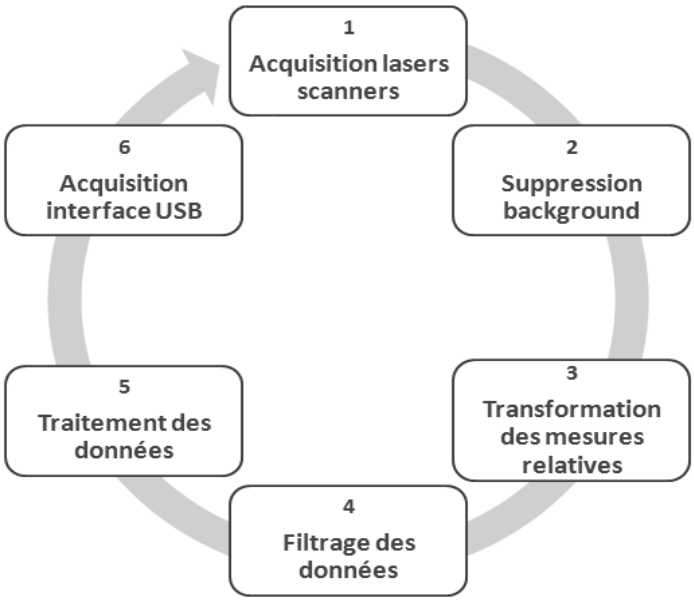

Figure 9 : Processus d'acquisition

7.1 Suppression de l'arrière-plan

La suppression de l'arrière ou le *background suppression*, est une technique fort utilisée dans le domaine de l'imagerie et de l'infrarouge. Le principe est très simple et efficace, puisqu'il consiste à enregistrer « l'image » de l'arrière-plan puis à supprimer tout ce qui lui ressemble. Cette méthode permet de mettre en évidence uniquement la partie intéressante de l'image, c'est-à-dire les objets n'appartenant pas à cet arrière-plan.

Dans le cas de l'analyseur de lobe, cela permet de recueillir uniquement les mesures de la cible présente dans la zone de détection. Voici un schéma permettant de mieux comprendre le principe.

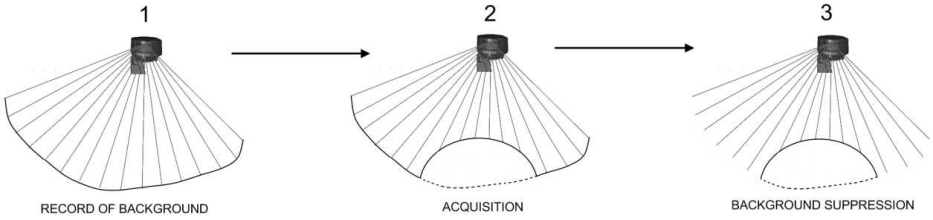

Figure 10 : Méthode de la suppression de l'arrière-plan

Cette suppression de l'arrière-plan représente le tout premier traitement à réaliser sur les mesures envoyées par les lasers scanners. C'est donc une étape très importante dans le processus d'acquisition global.

7.2 Recherche des coordonnées absolues

Après la suppression de l'arrière-plan, le logiciel obtient une série de mesures de distance de la cible par rapport aux lasers scanners. La deuxième étape, consiste à calculer les coordonnées absolues des points constituant la cible.

Un des objectifs de l'analyseur de lobe est d'avoir un dispositif flexible, pouvant s'adapter à n'importe quelles applications. Pour cela, la position des lasers scanners ne doit pas être figée, en effet l'utilisateur doit pouvoir les placer où il le souhaite.

Une calibration de l'outil permet de déterminer la géométrie du dispositif. En effet, pour pouvoir exploiter les deux appareils de mesure, leurs positions ainsi que leurs orientations doivent être connues de manière précise.

Méthode de calibration

Le dispositif est composé par un système d'axes absolus fixé par rapport au détecteur sous test, et par deux systèmes d'axes relatifs rattachés aux lasers scanners.

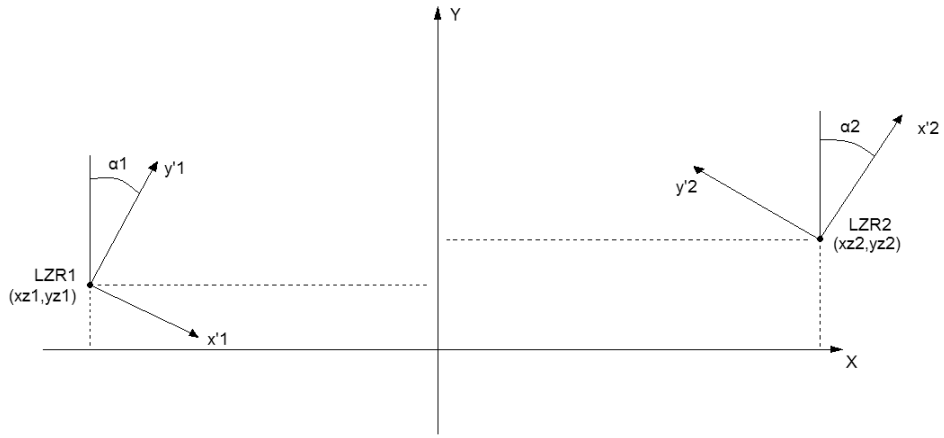

Figure 11 : Dispositif vu suivant les axes X et Y

Par convention, le système d'axes absolus (X,Y) est fixé de manière à ce que l'axe X soit parallèle à la porte automatique et que la position du détecteur corresponde à l'origine des axes. Cela permet d'avoir un lobe de détection symétrique par rapport à l'axe Y.

Le nombre d'inconnues du système est égal à six. En effet, il y en a quatre pour les coordonnées des lasers scanners et deux pour leurs orientations. La solution pour déterminer ces inconnues consiste à les positionner par rapport à deux points situés dans le système d'axes absolus.

La calibration consiste à mesurer la distance entre un objet de référence (un cylindre en l'occurrence) et le laser en lui-même. Cette mesure est réalisée par le logiciel suite à une série d'approximations au sens des moindres carrés.

Une fois la calibration réalisée, l'outil est capable de positionner une cible à l'aide des mesures de celle-ci par rapport à chaque appareil. Dès que la position de la cible est connue, il est possible de déterminer sa direction de déplacement ainsi que sa vitesse.

Utilisation des données

Après avoir réalisé l'acquisition d'un lobe, l'utilisateur se retrouve avec une série de données qui caractérisent la cible lors de sa détection. On connaît en effet, le centre de la cible lorsque le détecteur l'a détecté. Mais aussi, ses directions, sa vitesse et surtout son périmètre. A partir de toutes ses données, il est possible de demander à l'outil de les afficher séparément.

Ainsi, lorsque l'utilisateur demande d'afficher le lobe, ce sont tous les points représentant le centre de la cible qui s'affichent.

Ci-dessous, un aperçu de la fenêtre principale de l'analyseur de lobe avec l'affichage d'un lobe de détection. Il existe un ensemble de points qui représentent les mesures du laser scanner de gauche et d'autres qui représentent les mesures de celui de droite. Ainsi les points plus foncés, représentent le centre de la cible qui est calculé par le logiciel.

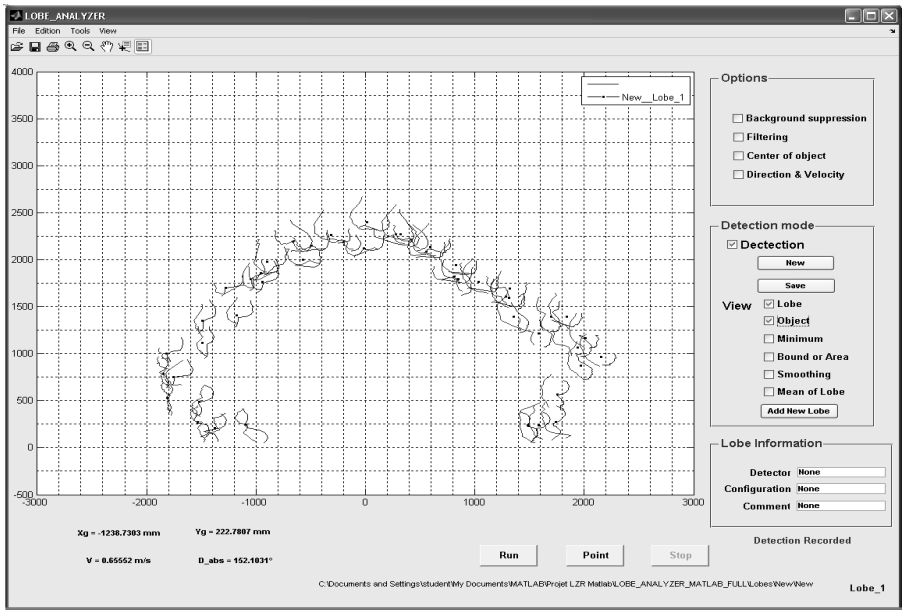

Figure 12 : Affichage d'un lobe, vu suivant X et Y

Un des avantages de l'analyseur de lobe est qu'il fait apparaître très distinctement les perturbations qu'il peut exister sur un lobe.

Le premier critère pour juger un lobe est sa symétrie. En effet, il est plus important d'avoir une symétrie du lobe que d'avoir une variation de celui-ci dans une zone localisée. C'est le cas du lobe ci-dessous, puisque le rétrécissement du lobe est symétrique par rapport à l'axe Y.

L'utilisateur peut s'il le souhaite, afficher l'extrémité du lobe. Cette option n'est utile que lorsqu'il y a un grand nombre de mesures réalisées. Voici donc l'extrémité du lobe mesuré.

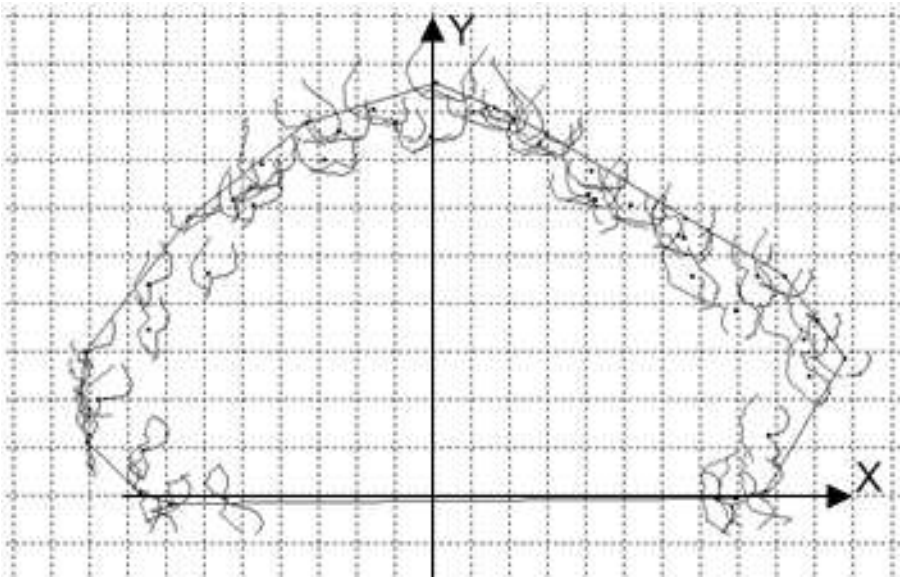

Figure 13 : Affichage de l'extrémité d'un lobe de détection

Il est possible aussi de comparer ce lobe avec des modèles, tels que des ellipses ou des cercles, mais aussi avec d'autres lobes déjà enregistrés. Cette possibilité est très importante pour l'utilisateur afin de pouvoir réaliser une analyse approfondie.

Voici une comparaison de ce lobe avec une ellipse.

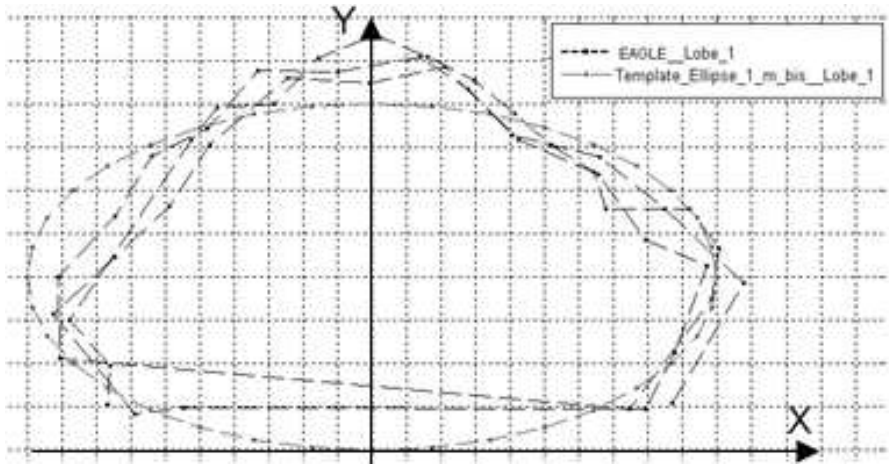

Figure 14 : Comparaison d'un lobe avec une ellipse

Cette comparaison permet ici de mieux voir les écarts du lobe par rapport à un modèle idéal. Pour finir, il est aussi possible d'afficher la moyenne du lobe qui, comme on le voit ci-dessous, est assez cohérente.

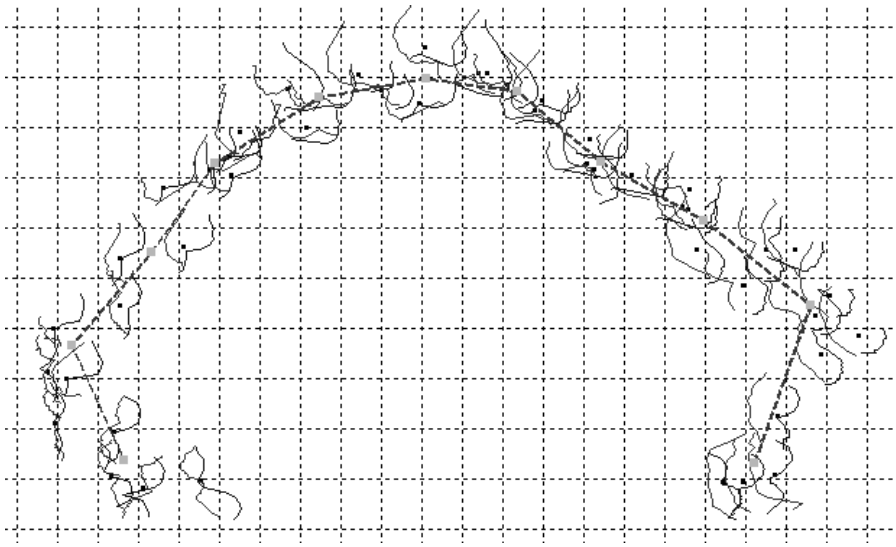

Figure 15 : Affichage de la moyenne d'un lobe

8. Transportabilité de l'application

Il n'est pas nécessaire de posséder MATLAB pour pouvoir transporter l'application sur une autre machine. En effet, *MATLAB Compiler* permet de partager des applications MATLAB telles qu'un exécutable ou une librairie partagée.

Les exécutables et les librairies créés avec l'outil *MATLAB Compiler* utilisent un moteur de calcul appelé le *MATLAB Compiler Runtime* (MCR). Le MCR est fourni par MATLAB pour la distribution d'applications et cela, libre de droits.

Cet outil permet d'exécuter l'analyseur de lobe en dehors de l'environnement MATLAB. Ainsi on ne perd pas de temps à recoder manuellement le code dans un autre langage. De plus, il permet d'intégrer des applications orientées utilisateur, c'est à dire les GUI, mais il permet également d'incorporer du code C/C++ en tant que librairie partagée.

Il est possible d'intégrer des applications utilisant d'autres langages de développement. En réalité, *MATLAB Compiler* sert à packager des applications pour créer des composants appartenant au logiciel.

L'application a été compilée de manière à créer deux exécutables. Un exécutable *package* « *LOBE_ANALZR_pkg.exe* » qui comme son nom l'indique est un package qui contient le MCR. Lors de l'installation de ce package sur une machine, il suffit d'y ajouter les librairies utilisées dans l'application. Ensuite, l'application peut être exécutée à l'aide du deuxième exécutable « LOBE_ANALZR.exe » qui contient l'ensemble du code de l'application.

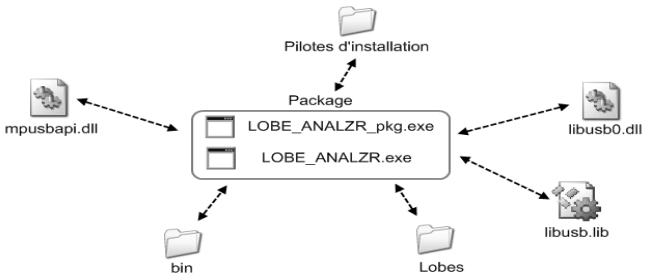

Figure 16 : Package d'installation de l'application

9. Conclusion

MATLAB a permis de développer un outil complet, comportant une interface graphique et des algorithmes de traitement de données. Ces algorithmes ont pu bénéficier de la puissance de calcul fournie par MATLAB.

L'interface graphique remplit bien son objectif puisqu'elle permet d'analyser des lobes de façon approfondie. Ainsi, il est possible d'enregistrer, de traiter, d'éditer, de comparer et enfin d'afficher les mesures.

L'application fonctionne de manière automatique, en gérant la communication USB avec les différents périphériques et en traitant directement les mesures réalisées par les deux lasers scanners. En plus d'être automatique, l'outil est flexible car il peut être exécuté sur des machines ne possédant pas MATLAB, mais aussi et surtout, il peut s'adapter à tout type d'environnements grâce à une calibration du système.

Au niveau de ses performances, l'outil a été testé et évalué sous ses différents aspects. Ce qui a permis de voir qu'il se prêtait mieux aux applications de types radar courtes distances et infrarouge. Les erreurs obtenues grâce aux mesures expérimentales rejoignent bien les erreurs théoriques développées mathématiquement. Ainsi la précision de l'outil peut aller jusqu'au centimètre.

Ensuite, un autre aspect qui a été pris en compte, est le choix d'algorithmes permettant d'avoir un compromis entre la précision et la rapidité des mesures. Ainsi, l'acquisition a été optimisée de manière à réduire au maximum son temps d'exécution. Pour information l'outil est capable de calculer le centre, la direction du déplacement et la vitesse d'une cible seize fois par seconde.

Pour terminer, l'analyseur de lobe est conçu pour tout utilisateur. En effet, l'application possède une page de démarrage et des dialogues permettant de guider l'utilisateur. En outre, elle possède également un *User Guide* qui décrit les différentes étapes nécessaires au bon fonctionnement de l'outil.

10. Références bibliographiques

10.1 Ouvrage

AXELSON, J., *USB Complete*, 2nd Edition, Madison, Lakeview Research, 2001.

ALFIO, Q., RICCARDO, S., FAUSTO, S., *Approximation au sens des moindres carrés discrets, Méthodes Numériques*, Springer.

DON, A., *USB System Architecture (USB 2.0). Universal Serial Bus System,* Architecture, 2nd Edition, MindShare Inc.

KUHN, M., *L'USB et sa Norme*, 2002.

ROEGEL, D., *Intersection de deux cercles dans le plan*.

THE MATHWORKS MATLAB, *External Interfaces,* Version 7, The MathWorks Inc., 2004.

10.2 Sources Internet

MICROCHIP, *PIC 18 MCU*, http://www.microchip.com

MICROSOFT, *Aide et Support,* http://support.microsoft.com

SOURCEFORGE NET, *LibUSB-Win32*, http://libusb-win32.sourceforge.net/

THE MATHWORKS, *Documentation MATLAB*, http://www.mathworks.com/support/tech-notes