RAPPORT FINAL DU PROJET N° 51

Titre du projet :

Conception et prototypage d'un dispositif pouvant mesurer avec précision la capacité d'un vase antique

Client partenaire extérieur
M. Grégory CHAMBON

Encadrants techniques
M. Kevin HEGGARTY
M. Ioannis KANELLOS

Encadrants de groupe de pilotage
M. Daniel STOENESCU
M. André THEPAUT

Équipe projet
M. Hichem BINOUS
M. Alban GOUGOUA
M. Idris HAMOUD
M. Juan Pablo IRUIT
M. Oussama SEGHAIER

Auteurs	Relecteurs	Destinataires	Version	Date
Équipe projet	*Équipe projet*	❖*Client partenaire extérieur* ❖*Encadrants techniques* ❖*Encadrants de groupe de pilotage*	2.0	21/06/2018

Mots clés : vase ; antiquité ; mesure de volume ; archéologie ; métrologie ; modélisation 3D ; capacité ; prototype ; télémètre ; précision ; Arduino ; appareil de mesure ; moteur pas à pas ; LIDAR.

Table des matières

LISTE DES ANNEXES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ DU PROJET

Ce projet s'inscrit dans le cadre d'un projet européen plus étendu dénommé « ARCHEO-METROLOGIA » [1]. L'objectif de notre projet est de déterminer le volume d'un vase antique à l'aide d'un appareil de mesure pouvant scanner l'intérieur du vase sans qu'il ait contact. À l'aide d'un logiciel, faire la représentation 3D de l'intérieur du vase et calculer avec précision la capacité du vase. Ce dispositif qui est un prototype, doit être léger et facilement transportable pour être utilisé sur des sites archéologiques.

Notre partenaire effectue des recherches concernant les pratiques de mesures du III^{e} au I^{er} millénaire avant J.-C. au Proche Orient [1]. L'intérêt de ses recherches est de comprendre le système d'échanges et plus avant du commerce en vigueur dans des civilisations en contact pendant cette période archéologique. Mais il n'existe pas actuellement un appareil de mesure de l'intérieur d'un vase. Par ailleurs, il existe des méthodes de représentation graphique chronophage très coûteuses en temps et approximatives. De plus, les vases de cette époque ne sont pas tous en bon état, la plupart d'entre eux sont reconstitués avec des parties manquantes (trous) et sont fragiles d'utilisation.

Notre mission est donc de réaliser un prototype d'appareil de mesure de la capacité d'un vase antique (avec tous ses défauts) pratique d'utilisation via un logiciel. Pour mener à bien cette mission, nous nous intéressons à l'optique, l'électronique, la mécanique et l'informatique. Les principales tâches à faire pour la réalisation du dispositif sont les suivantes : en premier lieu, *le pilotage de la caméra* (capteurs optiques à utiliser à l'intérieur du vase), *le montage et le pilotage mécaniques* (réalisation du support mécanique pour permettre le mouvement de la caméra à l'aide d'un moteur) ; en second lieu, *l'acquisition des données* (écriture des données recueillies par la caméra sur une carte mémoire), *l'interpolation des données manquantes* (faire la correction des éventuels trous présents dans le vase par des valeurs approximées et précises), *la modélisation 3D* (faire un modèle en trois dimensions du vase en utilisant les données recueillies et en les traitant) ; en dernier lieu, *le calcul du volume* (affichage du volume du vase par le logiciel).

À la fin de ce projet, ce présent rapport technique qui explicite les choix effectués et le fonctionnement du système dans son ensemble est à fournir au client ainsi que le prototype de l'appareil de mesure. L'utilisation d'un tel appareil est une solution à la mesure de volume des objets. En effet, il permettra d'apporter un grand plus aux domaines de la métrologie, de l'archéologie, des sciences sociales et voire des expositions de vases dans les musées dans le monde. Ce dispositif entend, avec l'avancée certaine des capteurs optiques et la miniaturisation des appareils électroniques dans les années à venir, améliorer la précision qu'elle offre déjà et s'intégrer à des robots de terrain pour réduire en temps et en coûts, mais aussi en précision, le calcul de volume de divers vases et objets.

INTRODUCTION

Le projet *Archeo-Metrologia* est un projet à l'échelle européenne qui vise à étudier les méthodes et unités de mesure employées durant l'Antiquité, permettant ainsi aux historiens de récolter des informations concernant les échanges commerciaux de l'époque. Cette initiative réunit différents acteurs de différentes disciplines : archéologues, anthropologues, historiens, etc.

Notre client et partenaire Grégory CHAMBON est directeur d'études à l'École des Hautes Études en Sciences Sociales (EHESS) et dirige ce projet. Il s'est tourné vers notre école pour concevoir et prototyper un dispositif capable de mesurer précisément la capacité d'un vase antique.

Cet appareil se devra d'être facilement transportable pour être utilisé sur des zones de fouilles. Il permettra notamment d'obtenir des informations quantitatives sur les capacités et donc sur les unités de mesures volumétriques utilisées à l'époque. On voit ainsi en quoi notre projet s'inscrit dans le projet *Archeo-Metrologia*.

Ce document présentera de façon détaillée notre solution à ce problème en expliquant les difficultés rencontrées ainsi que les différentes étapes de notre réflexion. Nous nous attarderons sur les raisons qui nous ont poussé à choisir cette solution expliquée dans le chapitre 2, en nous attardant sur les composants qui la constitue. Ces choix seront basés sur des aspects financiers, des préférences en termes de langage de programmation et surtout sur la performance et les dimensions des composants. Nous présenterons également des résultats évaluant la précision des composants et celle du prototype complet. Ces résultats permettront de valider ou non notre solution, et de permettre à notre client d'avoir des critères d'évaluation.

CHAPITRE 1 : PRÉSENTATION GLOBALE DU PROJET ET ÉTUDE DE L'EXISTANT

Dans cette première partie, nous ferons en premier lieu une présentation du projet, ses enjeux, objectifs et contraintes ; en second lieu, nous montrerons les méthodes existantes en faisant une étude de celles-ci.

1. PRÉSENTATION GLOBALE DU PROJET

Dans le cadre de notre projet ingénieur nous avons pour but de réaliser le prototype d'un dispositif capable non seulement de mesurer la contenance d'un vase antique, malgré les éventuels trous (voir *Figure 1*), mais aussi de faire la représentation 3D du vase. Ce produit est conçu pour être utilisé sur les lieux de fouilles et faciliter considérablement le travail des archéologues de terrain.

Figure 1 : Exemple de vases en mauvais état [2]

1.1. Enjeux du projet

Afin de dépasser les moyens de mesures existants, notre dispositif doit se proposer comme solution afin de pallier trois défis principaux. D'une part et essentiellement, l'appareil de mesure doit permettre une précision de moins de cinq pourcents (5%) sur le volume. D'autre part, il doit aussi être beaucoup plus rapide que les manœuvres utilisées actuellement, on vise

donc un temps moyen minimum de quinze minutes. Finalement, en considérant l'utilisation d'un tel outil par les archéologues sur le terrain, il faut que le dispositif soit portatif.

1.2. Contraintes liées au projet

Pour le bon déroulement du projet, on a défini plusieurs contraintes qui doivent être respectées et tirées au clair dès le début. Les plus importantes sont :

- Le budget : l'école a mis à notre disposition une somme de cinq cents euros qui doit couvrir les dépenses nécessaires à l'achat des composantes dont on aura besoin ;
- Le calendrier : le projet doit être réalisé dans un temps limité, soit du 13 février 2018 au 28 juillet 2018, tout en respectant les dates des différents livrables ;
- La qualité : c'est la contrainte principale traduisant la précision et la rapidité de fonctionnement du dispositif telle que spécifiée dans le cahier des charges (voir *annexe 1*) ;
- La technologie : en considérant les technologies existantes, nous sommes contraints à utiliser des composantes en cours d'expansion, et donc moins documentées.

2. ÉTUDE DE L'EXISTANT

Actuellement les méthodes utilisées [3] par les archéologues afin de calculer la capacité d'un vase dépendent de son état de conservation.

Le plus simple serait de le remplir avec un liquide mais cette méthode insinue que le vase est entier et étanche. Ceci rend cette méthode peu utilisée par les archéologues.

Une deuxième méthode, dans le cas d'un vase fragile ou fissuré, consiste à le remplir par un matériau suffisamment fragmenté pour être considéré comme un liquide.

Finalement, si le vase peut être reconstitué ou est assez endommagé, il est possible de le dessiner (voir *Figure 2*) et calculer géométriquement sa contenance par le biais d'un programme informatique permettant de numériser les dessins, et calculer ensuite la contenance.

Néanmoins, ces procédés présentent d'importantes contraintes pour les archéologues qui manipulent des vases antiques et donc souvent fissurés ou fragiles, et sont donc obligés d'utiliser des méthodes dont la précision varie entre sept (7%) et dix pourcents (10%), mais qui prennent au minimum trois heures à réaliser.

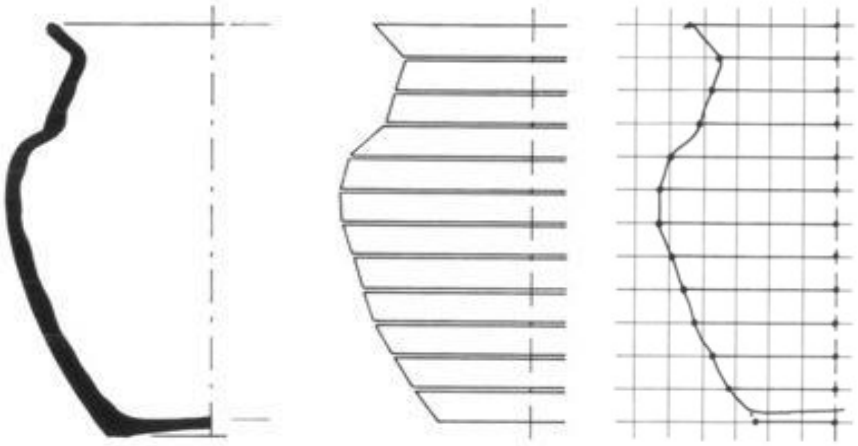


Figure 2 : Quelques étapes de la modélisation du vase par méthode graphique (dessin)

CHAPITRE 2 : ANALYSE DU PROJET

Dans cette seconde partie, nous analyserons le besoin du client pour fournir une solution appropriée au problème en vue de réaliser les objectifs du projet. Ainsi, nous comptons présenter le concept du projet, rechercher les composants utiles à sa réalisation et effectuer un choix final pour la réalisation de l'appareil de mesure.

1. PRÉSENTATION DU CONCEPT

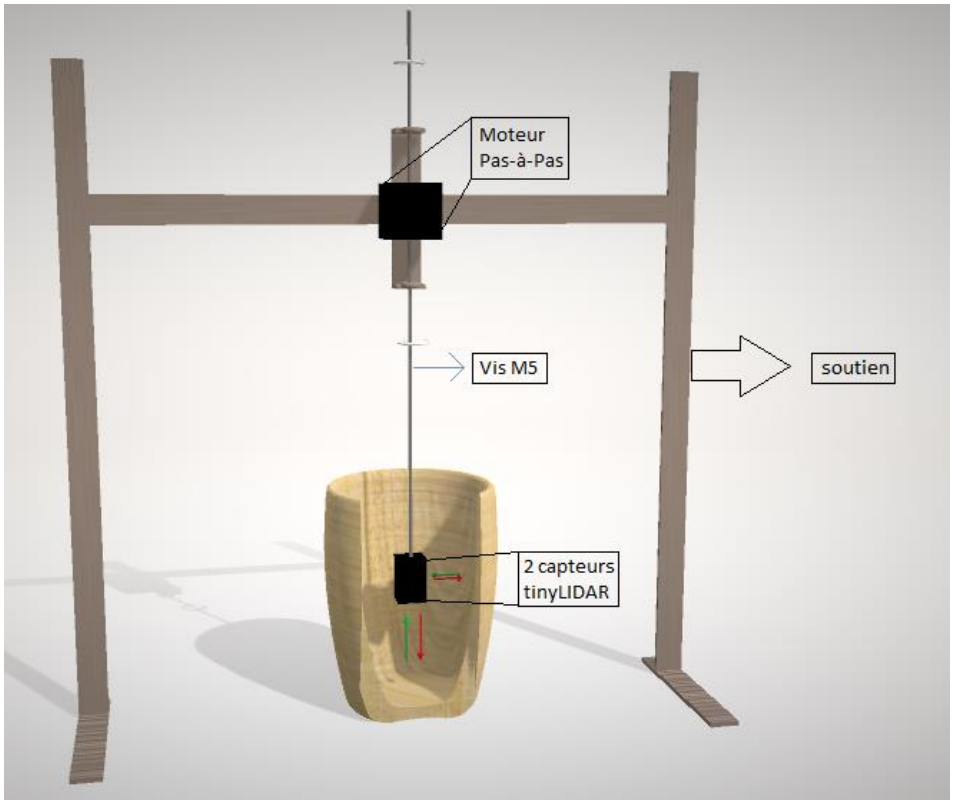


Figure 3 : Représentation d'un prototype de l'appareil de mesure

Notre projet consiste principalement à concevoir le prototype d'un appareil de mesure du volume d'un vase. L'idée est donc de fabriquer à l'aide de composants électroniques et optiques existants un boîtier de mesure de petite taille pouvant entrer à l'intérieur d'un vase d'ouverture minimum égale à six centimètres (6 cm).

Le principe est d'insérer un boitier composé de capteurs optiques, de cartes électroniques, fixé à une vis lui permettant de monter et descendre à l'intérieur d'un vase (voir *Figure 3*). Une fois à l'intérieur du vase, le boitier est capable de scanner l'intérieur du vase en recueillant une grande quantité de points qui seront plus tard traités par un logiciel qui calcule le volume de ce vase. Pour faire bouger la vis, un moteur, relié à un support mécanique au-dessus du vase, est utilisé et une carte électronique est aussi utilisée pour commander le moteur.

La vis peut tourner à l'intérieur du moteur, monter et descendre tout en entraînant le boîtier de mesure dans son mouvement pour effectuer les mesures.

Pour vérifier ce principe et le réaliser, nous faisons une recherche de composants optiques, électroniques, mécaniques pour l'appareil de mesure, et des outils informatiques pour le logiciel.

2. RECHERCHE DE COMPOSANTS EXISTANTS

Cette partie explique de façon détaillée les raisons qui nous ont poussé à choisir ces différents composants pour pouvoir réaliser les objectifs du projet. Elle contiendra des comparatifs techniques des différents moteurs, capteurs et microcontrôleurs que nous avons considérés.

2.1. Recherches de capteurs optiques

L'énoncé du projet ne nous imposait pas de méthode à utiliser pour mesurer le volume, il fallait toutefois qu'elle soit non invasive et sans contact pour préserver l'intégrité du vase.

Notre première réunion avec nos encadrants techniques nous a aiguillé dans nos recherches. En effet, il fallait également prendre en compte le fait que le vase pouvait ne pas être complet (voir *Figure 1*) après reconstitution rendant ainsi les méthodes utilisant des liquides ou des gaz impossibles. Nous avons ensuite cherché des appareils déjà existants capables de scanner de l'extérieur le vase, cependant la requête du client précisait que la représentation devait se faire de l'intérieur. Une fois ces détails pris en considération nos options ont été réduites à deux solutions (voir *Figure 4*) :

- La première consistait à faire entrer un capteur de type télémètre capable de mesurer la distance au vase ;
- La seconde prévoyait d'utiliser un télémètre à l'extérieur du vase et d'utiliser un miroir capable d'orienter le rayon pour prendre plusieurs mesures à chaque hauteur.

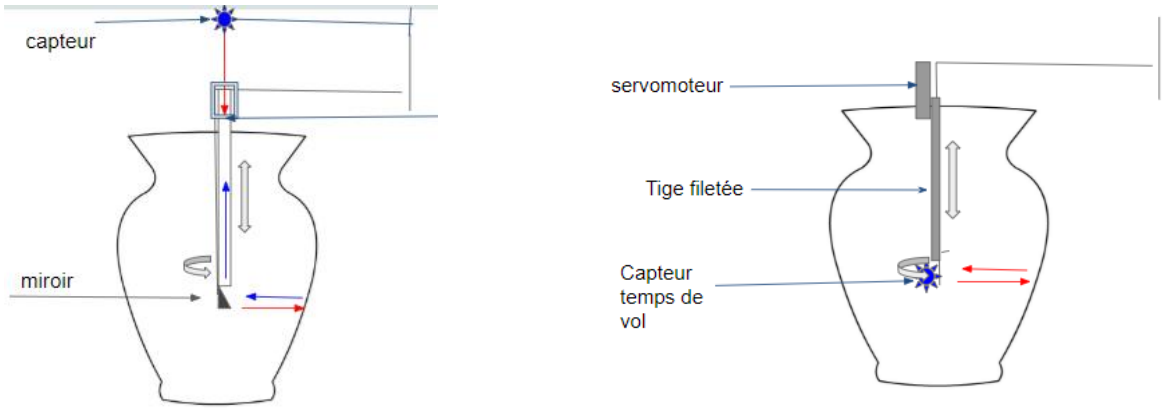


Figure 4 : Illustrations des solutions 1 et 2 respectivement à droite et à gauche

La seconde option nous paraissait trop compliquée en termes de calibration et de réglages nous avons donc décidé d'opter pour la première. Ainsi, il nous fallait donc disposer de capteurs d'assez petites tailles pour s'insérer dans un vase dont le diamètre pouvait être de six centimètres (6 cm).

2.1.1. Télémètres lasers

Les télémètres ont des prix abordables et ont une précision intéressante pour une grande portée. Mais nous n'avons pas d'information concernant la précision pour les petites portées. Sachant que nous voulons mesurer des distances allant de six centimètres (6 cm) à un mètre (1 m), cette option ne semble donc pas appropriée. Les télémètres ne communiquent qu'avec une application consacrée et il sera donc difficile de récupérer les valeurs dont nous aurons besoin.

Modèles	Précision (en mm)	Prix	Portée (en m)	Temps de mesure (en s)	Dimensions	Fonctionnalités
Bosch PLR 50 C	2	115,47 €	50	1 (en moyenne)	11,5 × 5 × 2,3 cm^3	Communication Bluetooth Smartphone
Bosch Zamo	3	47,40 €	20	1-4	12,8 × 11,6 × 2,7 cm^3 80 g	
Bosch GLM 50 C	1,5	119,90 €	50	0,5-4	10,6 × 4,5 × 2,4 cm^3	Communication Bluetooth Smartphone

Tableau 1 : Présentation de quelques télémètres lasers [4]

2.1.2. Caméras temps de vol

Les caméras temps de vol *TerraBee* ont une portée de plusieurs kilomètres et permettent donc d'obtenir une représentation 3D de l'espace environnant. Cependant leur précision pour une courte portée est peu intéressante.

2.1.3. Scanneurs tridimensionnels (LIDAR)

Le *Scanse Sweep* (voir *Figure 5*) est un appareil permettant d'obtenir un modèle 3D à trois cent soixante degrés (360°). L'ouverture du dispositif est très réduite (0,5°), il faut donc faire monter l'appareil à l'intérieur du vase pour obtenir un modèle 3D de l'ensemble de celui-ci. Il possède une application dédiée et est compatible avec Raspberry Pi cependant sa précision en dessous d'un mètre est très peu intéressante comme le montre le graphique ci-dessous.

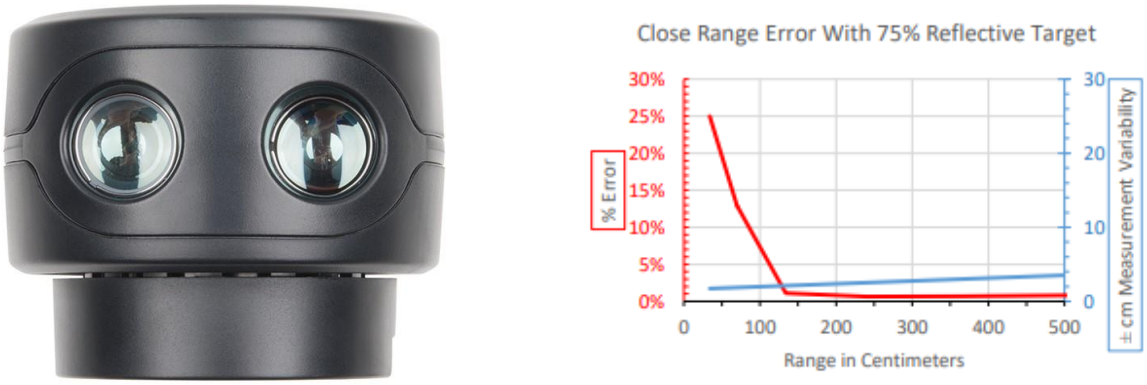


Figure 5 : Illustration du Scanse Sweep et graphique de précision [5]

2.1.4. Capteurs ToF (Time of Flight) LIDAR Arduino

Modèles	Précision (en %)	Prix	Portée (en cm)	Temps de mesure	Dimensions	Fonctionnalités
Adafruit VL6180X	5	$13,95	0,5-10 Peut aller jusqu'à 20 cm.	3,69 s	20,55 × 18,0 × 3,0 mm^3	1. Smartphones / portables touchscreen devices 2. Easy integration
Adafruit VL53L0X	5	$14,95	3-100	100 ms	21,0 × 18,0 × 2,8 mm^3	1. Smartphones / portables touchscreen devices 2. Easy integration
tinyLidar VL53L0X	3	$24,95	3-200	16,67 ms (60 Hz) Sample rates	21 × 25 × 8,3 mm^3	Pre-calibrated Very easy integration
Pololu VL53L0X	3 in best conditions	$9,95	3-200		13 × 18 × 2 mm^3	Moins de documentations que les autres.

Tableau 2 : Présentation de quelques capteurs ToF LIDAR Arduino

Les valeurs du tableau ci-dessus sont obtenues dans les conditions de réflectivité de 17% (faible).

Les capteurs temps de vol présentés dans le *tableau 2* offrent une précision d'au mieux 4,5 mm sur chaque mesure (pour une distance de 15 cm). C'est plus intéressant que les caméras temps de vol *TerraBee* plus adaptées à l'extérieur et aux grandes distances. La plupart des capteurs ci-dessus utilisent le capteur VL53L0X de *ST Microelectronics*.

Pour des raisons de performance, c'est le *tinyLidar VL53L0X* que nous choisirons, il offre la meilleure précision et son API (*Application Programming Interface*) sur Arduino est très bien documentée et est facile d'utilisation.

2.2. Recherche de composants électroniques

Pour que le moteur et les dispositifs de mesure fonctionnent correctement, des microcontrôleurs sont nécessaires pour contrôler la direction et la vitesse du moteur et pour contrôler la mesure des capots de distance et stocker ces mesures. Nous avons étudié deux microcontrôleurs qui répondent aux spécifications nécessaires pour le travail : l'*Arduino UNO* et le *Raspberry Pi*.

Figure 6 : Illustrations de cartes Arduino UNO et Raspberry Pi respectivement à gauche et à droite

D'une part, l'*Arduino UNO* est un microcontrôleur avec de nombreuses bibliothèques et beaucoup de documentation, il est spécial pour les projets électroniques et électromécaniques. D'autre part le *Raspberry Pi* est un mini-ordinateur qui le rend beaucoup plus complet que l'Arduino mais en même temps beaucoup plus complexe et plus de dépenses d'énergie inutiles pour ce projet. Nous avons donc opté pour l'Arduino. Deux cartes Arduino sont nécessaires, l'un pour contrôler les appareils de mesure et stocker leurs données, et l'autre pour contrôler le moteur.

Ensuite, nous devons stocker les mesures des capteurs pour pouvoir les utiliser plus tard. Pour eux, un *Shield SD* sera utilisé pour l'Arduino qui permet de stocker toutes les données dans une carte mémoire SD.

Enfin, l'Arduino qui entre dans le vase doit communiquer avec l'Arduino qui contrôle le moteur pour l'avertir quand il finit de descendre et commence à monter. Deux solutions ont été trouvées à cela. La première est la communication filaire, vu que le mécanisme doit tourner, le câble s'emmêlerait avec la structure. La seconde est la communication sans fil. Pour cela, il y a la communication Wi-Fi et la communication Bluetooth. La communication Wi-Fi consomme

plus d'énergie pour avoir plus de portée et plus de vitesse ce qui n'est pas nécessaire dans ce projet, donc nous avons opté pour la communication Bluetooth.

2.3. Recherche de supports et appareils mécaniques

Compte tenu des premiers croquis de solutions (voir *Figure 3*), on a pu avoir une idée des composantes mécaniques nécessaires à la réalisation du dispositif. En premier lieu, on devait trouver un système capable de commander avec précision le mouvement des capteurs à l'intérieur du vase, soit un moteur pas à pas commandant la montée et la descente d'une tige filetée. Ensuite, un support mécanique pour tenir le moteur au-dessus du vase.

2.3.1. Moteurs

Modèle	Couple	Longueur vis	Vis filetée	Précision
Moteur pas-à-pas 2,8V 1,68A 4,4kg-cm RepRap SOYO	4,4 kg-cm		0,8 mm	± 5%
Moteur pas-à-pas Bipolaire NEMA-17	3,7 kg-cm	Tige filetée de 28 cm	Tr 8 × 8 mm^2	
Moteur pas-à-pas 29 oz/pouce à arbre fileté	21 N.cm	100 mm	M5 Distance des pas : 0,1 mm	± 5%

Tableau 3 : Présentation de quelques moteurs pas-à-pas

On n'a pas trouvé de moteur correspondant à nos besoins exacts, néanmoins on choisira le *moteur pas-à-pas 29 oz/pouce à arbre fileté* et on remplacera sa vis par une autre plus longue. Ce sera donc une tige filetée M5 de longueur un mètre (1 m). Pour contrôler le moteur, on prendra un *Contrôleur de Moteur Pas-à-Pas Bipolaire EasyDriver SFE*.

2.3.2. Supports mécaniques

Pour pallier le problème de fixation du moteur-vis au-dessus du vase, nous avons trouvé deux possibilités :

- Construction d'un support à partir de *Profilé aluminium 20x20 I-Type rainure 5 ;*

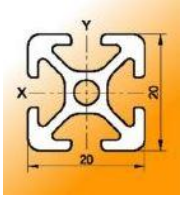


Figure 7 : Illustration d'un profilé en X

- Utilisation d'un trépied d'appareil photo, ayant une ouverture assez grande.

Nous sommes amenés à choisir le montage par profilés pour deux raisons principales. La première étant l'ouverture nécessaire (vases de diamètre 1 m), la seconde est le budget. Cependant, l'utilisation d'un trépied est concevable pour les archéologues de terrain qui peuvent avoir parmi leurs outils de grands trépieds.

2.4. Recherche d'outils informatiques

Le choix du langage de programmation des composants *Arduino* et du logiciel n'est pas imposé par le client ni par les encadrants techniques. Ainsi, pour choisir le bon langage pour notre projet, nous avons pris en considération deux critères principaux qui sont la maîtrise du langage par l'équipe de développement et la richesse des bibliothèques nécessaires pour répondre aux besoins du projet (représentations 3D et 2D ; interface graphique).

Les deux langages bien maîtrisés par l'équipe de développement sont JAVA et Python, avec plus d'expérience en Python. Nous avons commencé par faire une comparaison de ces deux derniers et étudier leurs meilleures bibliothèques de graphisme (voir *Tableau 4*).

	JAVA	Python
Bibliothèques	Java3D, Xith3D, jMonkey Engine, Aviatrix3D, JOGL, LWJGL	Visbyl, PyQtGraph, Matplot3d, Panda3d, VPython, Pyqwt

Tableau 4 : Présentation des bibliothèques graphiques des langages JAVA et Python

Les deux langages possèdent les outils nécessaires, donc nous avons choisi de travailler avec Python. Et nous avons choisi la bibliothèque PyQtGraph comme elle nous permet de faire des représentation 3D, 2D et des interfaces graphiques en même temps. Elle utilise OpenGL [6] pour le rendu 3D et PyQt (4 ou 5) [7] pour les interfaces graphiques.

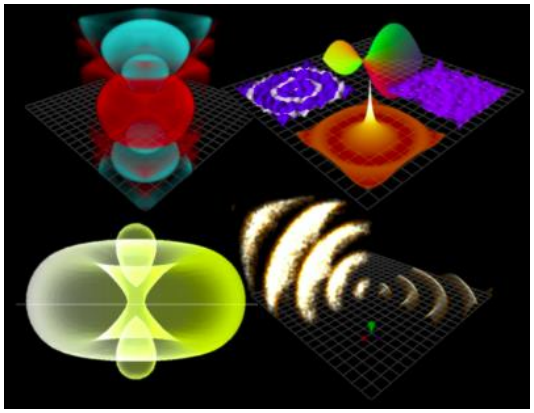


Figure 8 : Exemple de représentations 3D par PyQtGraph

3. CHOIX DE LA SOLUTION

Nous arrivions sans peine au choix d'une solution fiable et viable entre les deux solutions présentées plutôt dans la partie *2.1* de ce chapitre. Notre choix s'était donc porté sur la solution 1, celle d'utiliser un capteur de type *ToF LIDAR* pour l'insérer dans le vase et effectuer les mesures.

Au final, le dispositif de mesure sera constitué, en faisant une synthèse de toutes les synthèses de recherches de composants, :

- Deux capteurs *tinyLidar VL53L0X*, petits, précis et de prix raisonnables ;

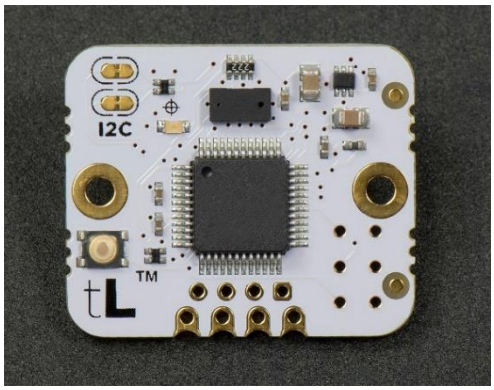


Figure 9 : Capteur tinyLidar VL53L0X [8]

- Deux cartes Arduino UNO (voir *Figure 6*) très simples d'utilisation pour commander les capteurs et le moteur ;
- D'un *moteur pas-à-pas 29 oz/pouce à arbre fileté* petit et précis ;

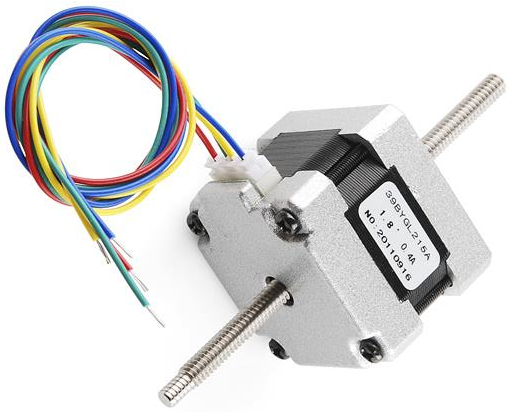


Figure 10 : Moteur pas-à-pas 29 oz/pouce à arbre fileté [9]

- De profilés en aluminium pour supporter le moteur.

Le logiciel, quant à lui, sera développé dans le langage Python et utilisera les bibliothèques PyQtGraph et OpenGL pour les représentations 2D, 3D et l'interface graphique.

Après, avoir conclu du choix de la solution avec les encadrants techniques et le client, nous avons entamé la phase d'ingénierie de ce projet.

CHAPITRE 3 : RÉALISATION ET RÉSULTATS DU PROJET

Dans cette dernière partie, d'une part nous ferons la conception et la réalisation de l'appareil de mesure, du logiciel, d'autre part nous testerons la performance de nos différents outils.

1. RÉALISATION DU PROJET

1.1. Cahier des charges fonctionnelles

Le prototype doit pouvoir mesurer le volume d'un vase antique. Répondre à ce besoin nécessite d'abord de définir le corpus de vases que l'appareil sera capable de mesurer avec précision. Nous considérerons des vases dont l'ouverture ne sera pas inférieure à 6 cm et dont la largeur ne dépasse pas 1 m. Pour formuler le besoin du client, on utilise des fonctions détaillant les services rendus par le produit et les contraintes auxquelles il est soumis. La fonction principale de notre appareil est d'obtenir la capacité d'un vase antique et un modèle 3D du vase. Cette fonction principale nécessite la mise en place de nombreuses fonctions secondaires que nous allons lister ci-dessous :

- Permettre à l'appareil de mesure de tourner et de descendre/monter ;
- Mesurer avec précision la distance séparant l'appareil du bord du vase visé ;
- Envoyer chacune des mesures réalisées au logiciel ;
- Modéliser en 3D l'intérieur du vase à partir des mesures ;
- Extraire l'information concernant le volume ;
- Afficher le modèle 3D du volume.

Ces fonctions secondaires seront plus détaillées dans le cahier des charges que vous pourrez trouver en annexe (voir *Annexe 1*). Nous devrons respecter des contraintes d'intégrité concernant le vase, des contraintes de précision concernant les mesures et des contraintes concernant le temps d'acquisition des résultats. D'après notre client les méthodes graphiques actuelles peuvent prendre des heures et n'offre une précision que de 10% environ. Notre appareil lui se devra d'avoir une précision de moins de 5%.

1.2. Conception et réalisation de l'appareil de mesure

Tout d'abord, suite aux recherches et documentations et une fois la solution choisie, nos premiers croquis indiquait que le travail repose sur deux axes principaux, soient l'électronique et la mécanique. Nous devions alors choisir les différentes composantes nécessaires à la réalisation du dispositif selon les règles de notre solution.

Ensuite, concernant la partie électronique nous devions concevoir une solution pour commander le moteur et les capteurs en synchronisation afin d'avoir les mesures nécessaires et éviter de heurter le fond du vase. Nous avons alors choisi de mettre un premier capteur dirigé vers le fond du vase et un second vers la paroi perpendiculairement au premier (voir *Figure 11*). Cette mise en place permettra de déclencher le mouvement du moteur et capteurs en parallèle, l'arrêt du moteur quand la tige atteint le fond du vase et d'actionner la phase de montée et d'acquisition des données. Les deux capteurs seront connectés à un multiplexeur qu'on insère sur la carte électronique. On connecte aussi à la carte un *Shield SD* et un module Bluetooth. Une pile de cinq volts (5 V) pour alimenter les composants est introduite dans le boîtier.

Après, la partie mécanique consistait à réaliser un système (support-moteur-tige-capteurs) qui permet un mouvement bien précis des capteurs à l'intérieur du vase afin d'acquérir un nuage de points de mesures interprétables par le logiciel. Le boîtier de mesure s'illustre bien par la figure ci-dessous.

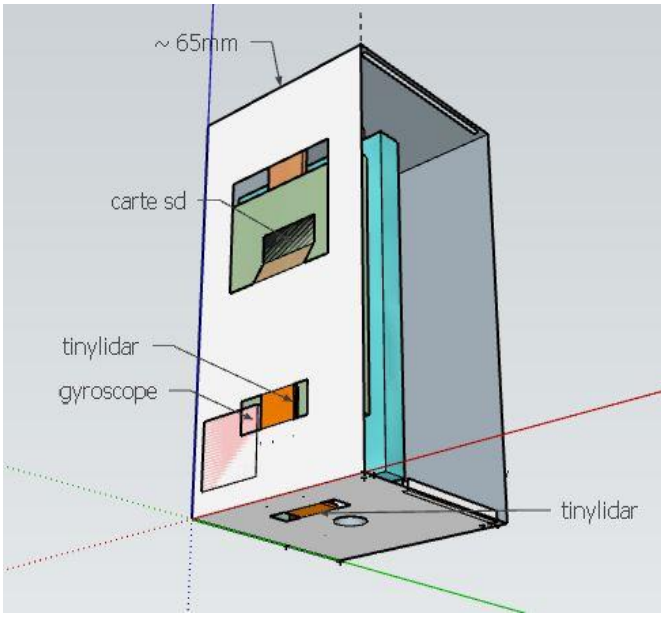


Figure 11 : Illustration du boîtier de l'appareil de mesure

Puis, pour respecter le corpus des vases étudiés, le boitier doit avoir une largeur et une longueur inférieures à 6 cm. Afin de le satisfaire, on a choisi de faire traverser la tige filetée dans le boîtier. Ensuite on fixera le boîtier à l'aide d'écrous à la vis. Cette phase de conception composée de plusieurs phases a fait que les commandes des différentes composantes nécessaires, ont été faites sur deux reprises.

Enfin, suivant l'ordre de livraison des achats nous avons procédé en premier à la manipulation des capteurs et nous avons pu alors tester les capteurs et voir les différents modes de fonctionnement. D'une part, on pouvait donc procéder à la programmation du déroulement de la manœuvre, soient la communication entre les deux cartes, la synchronisation des capteurs et l'enregistrement des mesures. D'autre part, à l'arrivée des moteur et tige, on a remarqué que les dimensions fournies par le vendeur du moteur ne correspondaient pas à celui fourni. En effet la tige fournie avec le moteur avait un pas de filetage différent, et on n'a pas pu la remplacer par la tige M5. Compte tenu des circonstances, on a dû prélever les dimensions exactes de la vis, et commander une nouvelle qui correspond à ces mesures : « TIGE FILETÉE BSW ACIER

DROIT BRUT 4.8 1 MÈTRE 3/16 » (filetage britannique ; 24 Filets par pouce). Ensuite, une fois qu'on a reçu les profilés on a pu procéder à leurs montages, et construire un support de hauteur réglable par vissage d'une barre horizontale à deux autres verticales chacune de taille 1m50 et chacune tenant debout sur un profilé de 30 cm. Parallèlement, on avait entamé la conception des deux boîtiers vérifiant les dimensions du matériel utilisé et on a procédé à leur impression au Téléfab.

1.3. Développement informatique

1.3.1. Algorithmes d'utilisation des moteur, télémètre, Bluetooth et carte mémoire sur Arduino

Pour contrôler tous les appareils électroniques et électromécaniques, deux codes différents sont nécessaires, un pour chaque Arduino.

- **Carte Arduino avec les capteurs** (voir *annexe 2*)
  D'abord pour pouvoir utiliser les deux tinyLIDAR en même temps, vous devez préconfigurer les adresses *I2C* du tinyLIDAR pour pouvoir communiquer en même temps avec l'Arduino (ils viennent par défaut avec la même adresse *I2C*), puis utiliser chaque adresse pour envoyer les commandes et recevoir les valeurs de distance (2 octets par mesure) à travers le même port *I2C*. Ils sont aussi configurés pour effectuer des mesures de haute précision, avoir plus de temps entre les mesures mais augmenter notablement la précision desdites mesures. Ensuite, toutes les deux cents millisecondes (200 ms) la carte Arduino demandera à chaque tinyLIDAR simultanément une mesure, puis lira la valeur des mesures et les sauvegardera dans un fichier d'extension *.csv* dans la carte mémoire SD. Enfin, cet Arduino devra communiquer à l'autre Arduino lorsque la distance jusqu'à la base du vase pour qu'il commence à tourner vers l'autre côté. Pour cela, il a fallu préconfigurer deux modules Bluetooth *HC-05* afin qu'ils communiquent automatiquement entre eux lorsqu'ils sont allumés de cette façon, les Arduino peuvent communiquer sans fil via le port série. Lorsque la distance au fond du vase est inférieure à un certain seuil, l'Arduino envoie l'autre signal à l'autre Arduino via le port série.

- **Carte Arduino pour le moteur**
  Une fois configuré, le moteur n'a pas un programme très complexe. L'Arduino ne fait que déplacer le moteur à la vitesse désirée, compter le nombre de pas effectués et quand il reçoit le signal de l'autre Arduino, cela va changer sa vitesse et sa direction faire les mêmes pas que quand il est descendu. Alors l'opération termine dans la même position de début.

1.3.2. Algorithmes de traitement des données

Tout d'abord, avant de pouvoir traiter les données de la carte mémoire, un algorithme de lecture de fichier *.csv* appelé *readFile.py* permet de lire un fichier *.csv* et mettre toutes les

données contenues dans une liste de distances et de hauteurs pouvant être utilisée plus facilement par les autres algorithmes de traitement. En effet, cette fonction ouvre et lit le fichier *.csv*, puis récupère chaque information dans une liste préliminaire pour effectuer des corrections – ajout d'une hauteur et d'une distance à chaque valeur du tableau pour se mettre à un point origine défini par l'intersection perpendiculaire des plans horizontal et vertical des capteurs. À la suite des corrections, cet algorithme fait une interpolation du volume de la partie inférieure du vase (en dessous de 2 mm) et corrige les valeurs prises par le capteur qui est en dessous du vase – il renvoie des valeurs d'erreur comprises entre 0 et 9 lorsque les distances sont en dessous de 10 mm et au-dessus de 1 m. Cette fonction, donne finalement en sortie une liste de distances et de hauteurs de chaque endroit du vase.

Ensuite, nous utilisons deux catégories d'algorithmes de traitement de données : la première se charge de faire la correction des valeurs erronées enregistrées par le capteur placé sur le côté, et fait une approximation de rayons à partir d'une liste de points ; la seconde se charge de transformer des listes en d'autres listes : liste de distances et hauteurs en liste de coordonnées *(x, y, z)* et d'index des valeurs erronées, liste de coordonnées *(x, y, z)* et index des valeurs erronées en liste de rayons et hauteurs (R_i, h_i) et d'index des valeurs erronées. Cette dernière transformation de liste est utilisée pour calculer le volume du vase.

Enfin, tous les algorithmes de traitement de données seront présentés en *annexe 2* avec plus d'explication.

1.3.3. Développement de l'interface graphique

Le développement de l'interface graphique avec les représentations 3D et 2D du vase est faite selon 2 étapes principales :

- **Étape 1 : conception**

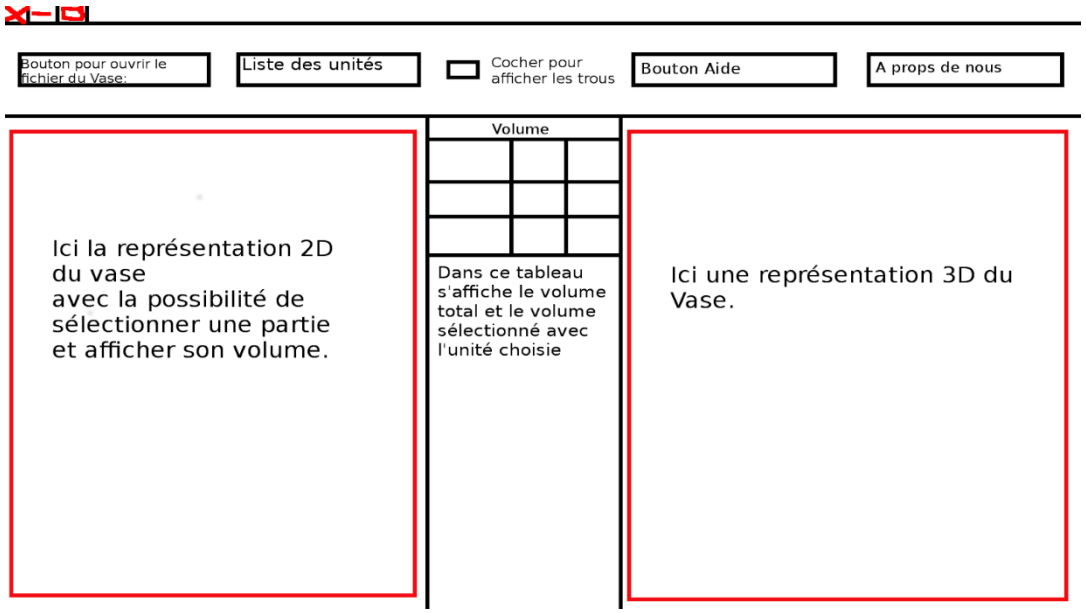


Figure 12 : Maquette du logiciel

Nous avons commencé par concevoir la maquette du logiciel (figure ci-dessus) que nous avons choisie de le faire en une seule fenêtre principale avec 4 interfaces :

- Interface 1 (voir *Figure 13*) : contient les différents boutons et réglages pour
  - Bouton « *Open file* » pour ouvrir et lire un fichier *.csv* qui contient les données du vase ;
  - Une liste pour choisir l'unité du volume souhaité parmi une liste extensible (mm^3, cm^3, dm^3, m^3, mL, cL, dL, L) ;
  - Une case à cocher si on veut afficher la position des trous dans le vase (s'il y en a) dans la représentation 3D ;
  - Bouton « *Help* » qui affiche une fenêtre d'aide à l'utilisation du logiciel et l'appareil de mesure ;
  - Bouton « *About us* » qui affiche une autre fenêtre avec des informations sur le projet.

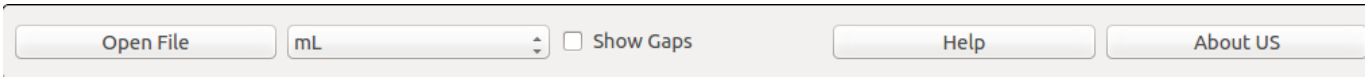


Figure 13 : Capture d'écran de l'interface 1

- Interface 2 (voir *Figure 14*) : Cet espace contient une représentation 2D du vase avec une ligne horizontale pour sélectionner une partie du vase et afficher son volume.

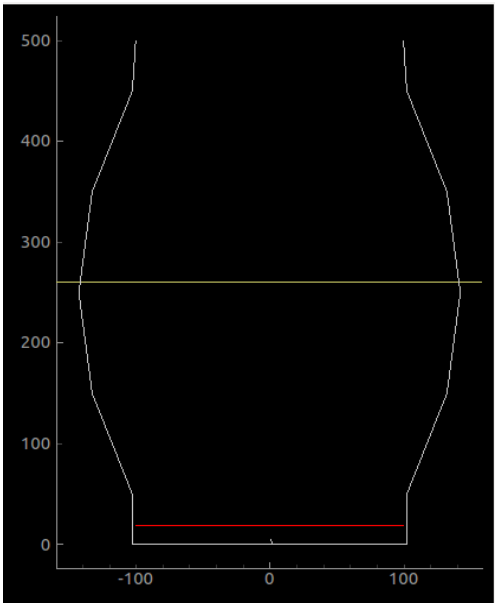


Figure 14 : Capture d'écran de l'interface 2

- Interface 3 (voir *Figure 15*) : cette interface contient les informations sur le volume du vase. Il contient un tableau de deux lignes, une pour afficher le volume sélectionné dans l'interface 2 et l'autre pour afficher le volume total.

Volume		
	Value	Unit
Selected	5519.46	mL
Total	7597.08	mL

Figure 15 : Capture d'écran de l'interface 3

- Interface 4 : Il s'agit d'une fenêtre avec la représentation 3D du Vase.

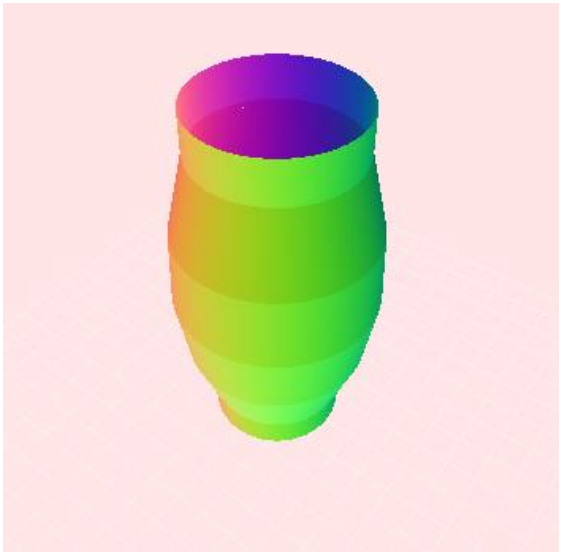


Figure 16 : Capture d'écran de l'interface 4

❖ Étape 2 : Développement

En faisant l'assemblage de toutes les interfaces et en utilisant les algorithmes de traitement des données derrière le bouton d'ouverture du fichier « *Open File* », on obtient la fenêtre ci-dessous (*Figure 17*) qui est l'interface graphique de notre logiciel. Les détails de cette partie de développement se trouve en *annexe 2*.

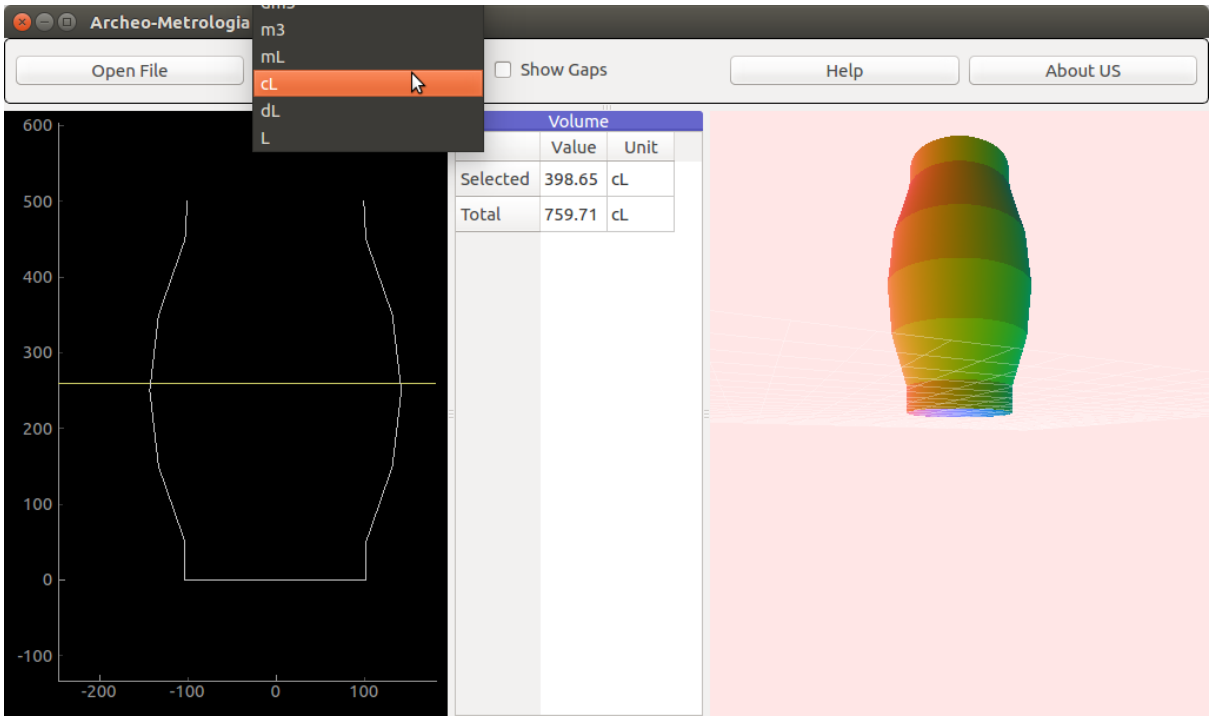


Figure 17 : Capture d'écran de l'interface graphique du logiciel

2. RÉSULTATS DU PROJET

2.1. Performance de l'appareil de mesure

Des premières mesures sur le capteur tinyLidar nous ont permis de réévaluer sa précision effective. En effet les 3% ne sont pas très optimistes, il existe en effet un mode d'acquisition haute précision sur le capteur qui permet d'obtenir une précision de trois millimètres pour une distance de trente centimètres. Cela équivaudrait à une précision de 1%, pour une fréquence de mesures de cinq hertz (5 Hz).

Cependant il faudrait effectuer des mesures de précision pour différentes distances pour obtenir une évaluation précise de la précision du composant. D'ailleurs, il nous est impossible pour l'instant de donner des résultats réels parce que nous n'avons pas jusque-là pu réaliser des tests. En effet, le retard des composants et les fausses informations données sur les sites de vente du moteur par exemple, ne nous ont pas permis de respecter notre diagramme de Gantt et faire des tests comme cela était prévu.

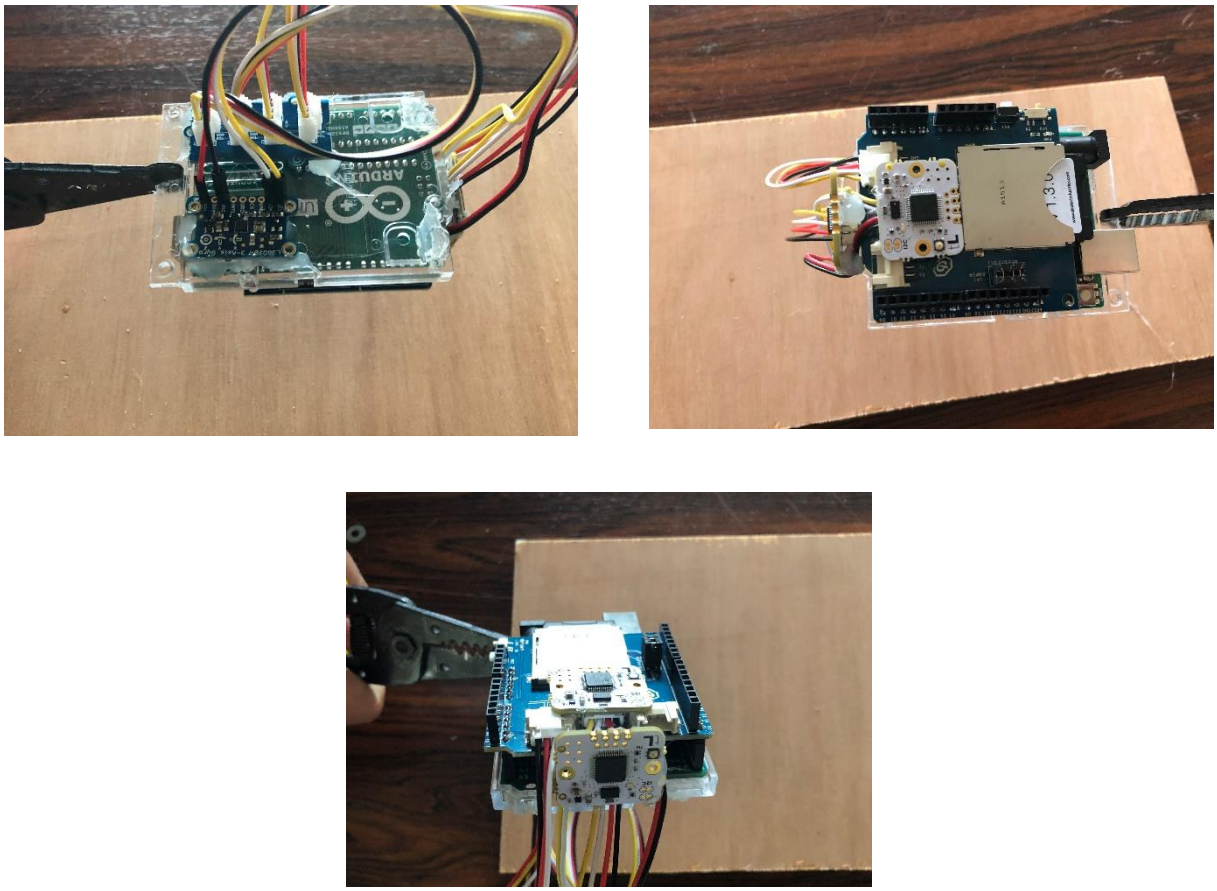


Figure 18 : Quelques photos de l'appareil de mesure composée de l'Arduino, d'un gyroscope, des deux capteurs ToF tinyLidar et de la carte mémoire SD

2.2. Performance et précision du calcul de volume

Pareillement à la précédente partie, nous ne pourrons fournir des résultats réels de calcul de volume par le logiciel. Toutefois, des tests ont été effectués sur des valeurs aléatoires contenues dans un fichier .csv et garantisse la robustesse et la précision du calcul de volume.

3. SOLUTION DE REPLI

Le prototype n'est toujours pas terminé pour l'instant et le risque concernant le montage mécanique fait que nous avons dû imaginer une solution de secours dans laquelle notre dispositif de mesure devra être piloté manuellement par une manivelle. Les acquisitions des mesures ne seront plus régulières, il nous faudra donc un gyroscope trois axes capable de mesurer l'angle à chaque étape pour pouvoir reconstituer le modèle 3D avec exactitude. Cette solution n'est bien évidemment pas optimale et ne serait pas convenable pour des archéologues faisant des fouilles sur terrain mais elle permettrait de faire une preuve de notre concept. Cette solution qui paraît plus rudimentaire a toutefois des avantages indéniables :

- Moins cher que la première solution ;
- Pas de risques concernant les communications Bluetooth ;
- Plus qu'une seule pile neuf volts (9 V) pour alimenter l'Arduino donc plus écologique ;
- Un dispositif plus facile à transporter pour le client.

Son inconvénient principal vient de la durée nécessaire pour faire une mesure précise sur un vase de taille maximale (1 mètre), il faut en effet au moins dix minutes pour obtenir des résultats satisfaisants (meilleurs que l'existant) d'après nos calculs.

Nous nous sommes rendus compte d'une faiblesse importante de notre solution initiale qui nécessitait une régularité parfaite sur les mesures et une précision importante sur chaque pas du moteur. D'où l'idée d'utiliser un gyroscope, nous avons opté pour le gyroscope illustré par la figure ci-dessous.

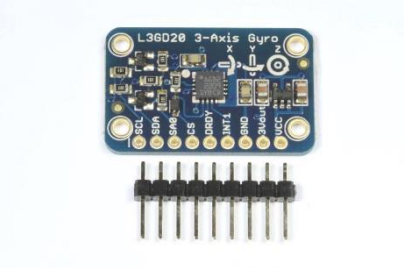


Figure 19 : Illustration du gyroscope : Adafruit L3gd20h (3 axes)

Ce gyroscope est fourni avec une documentation riche et une bibliothèque qui nous permet de gérer les acquisitions très facilement. Il a également l'avantage de retourner les résultats dans des unités du Système International, et d'utiliser le protocole *I2C* de communication que l'on utilise également pour les acquisitions des deux capteurs de distance. Il offre différents modes de précision, le mode le plus précis permet de mesurer jusqu'à +-250°/s. On peut utiliser du 3 V pour l'alimenter et donc utiliser la même broche que pour les deux capteurs *tinyLidar*.

CONCLUSION ET PERSPECTIVES

Nous avions sous-estimé les délais de livraison des composants de notre prototype notamment le moteur pas à pas, et notre manque d'expertise en mécanique ont ralenti le déroulement du projet. Il nous a fallu donc nous rabattre sur une solution de repli qui est expliquée plus en détail dans la partie 3 du chapitre 3. Avant la soutenance nous devons :

- Utiliser un accéléromètre pour pouvoir mesurer l'angle et ne pas dépendre de la régularité du moteur ;
- Réaliser le montage avec manivelle pour pouvoir contrôler manuellement les capteurs.

Nous ne perdons toutefois pas de vue les objectifs et critères fixés par le client, et nous proposons donc des nouvelles améliorations qui pourraient être apportées à notre solution. Ainsi dans l'avenir nous comptons :

- Piloter le boîtier à l'aide de deux moteurs pas à pas, un permettant au boîtier de monter et descendre et un autre permettant au boîtier de tourner ;
- Utiliser un Arduino Nano pour réduire la taille du prototype ;
- Penser à un prototype plus facile à porter comme un trépied pour pouvoir tout mettre dans une mallette.

BIBLIOGRAPHIE

[1] EHESS. *Le projet archeo-metrologia : Des mesures et des hommes | EHESS*. **Disponible sur** : https://www.ehess.fr/fr/espace-num%C3%A9rique/projet-archeo-metrologia-mesures-et-hommes. (14/02/2018)

[2] INRAP. *Découvertes remarquables à Toulouse : Serrures métalliques médiévales – Inrap*. **Disponible sur :** https://multimedia.inrap.fr/atlas/Grand-Toulouse/decouvertes-toulouse/p-20719-Serrures-metalliques-medievales.htm#.Ww1Gx0iFPIU. (29/05/2018)

[3] PERSÉE. *Méthode géométrique simple de calcul du volume des contenants en céramiques – Persée*. **Disponible sur** : https://www.persee.fr/doc/dam_0184-1068_1981_num_4_1_912. (14/06/2018)

[4] LEROY MERLIN. *Télémètre connecté Laser BOSCH Plr 50 c 50 m | Leroy Merlin*. **Disponible sur** : https://www.leroymerlin.fr/v3/p/produits/telemetre-connecte-laser-bosch-plr-50-c-50-m-e1401128917. (20/06/2018)

[5] SCANSE. *User's Manual, Sweep V0.991*. **Disponible sur** : https://s3.amazonaws.com/scanse/Sweep_user_manual.pdf. (13/06/2018)

[6] OPENGL. *OpenGL – The Industry Standard for High Performance Graphics*. **Disponible sur** : https://www.opengl.org/. (13/06/2018)

[7] PYQTGRAPH. *PyQtGraph – Scientific Graphics and GUI Library for Python*. **Disponible sur** : http://pyqtgraph.org/. (13/06/2018)

[8] MICROELECTRONICDESIGN. *tinyLIDAR – MicroElectronicDesign*. **Disponible sur** : https://microed.co/product/tinylidar/. (25/03/2018)

[9] ROBOTSHOP. *Moteur Pas-à-Pas 29 oz/pouce à Arbre Fileté – RobotShop*. **Disponible sur** : https://www.robotshop.com/eu/fr/moteur-pas-pas-29-oz-pouce-arbre-filete.html#Sp%C3%A9cifications. (25/03/2018)

ANNEXES

1. ANNEXE 1

CAHIER DES CHARGES
PROJET 51 :
Conception et prototypage d'un dispositif pouvant mesurer avec précision la capacité d'un vase antique

Version : 2

Date : 23/03/2018

Client	***Prestataire***	
Grégory Chambon *Directeur d'études à l'EHESS*	Nom1 : BINOUS Hichem Nom2 : GOUGOUA Alban Nom3 : HAMOUD Idris Nom4 : IRUIT Juan-Puablo Nom5 : SEGHAIER Oussama	
Cahier des charges approuvé dans sa version le 23 / 03 / 2018 par M. Gregory CHAMBON		

1. Introduction

1.1. Objet du document

Ce document décrit tous les services que doivent rendre le produit et ses livrables et toutes les exigences qu'ils doivent satisfaire.

1.2. Portée du document

Ce document est destiné à formaliser le besoin du client dans le cadre du projet S4 INGÉ.

1.3. Abréviations

Abréviations	Signification	Libellé
MTBF	Mean time before failure	Temps moyen entre chaque panne pour un composant
VIT	Vitale	Exigences fonctionnelles ou non fonctionnelles indispensables
IMP	Importante	Exigences souhaitées mais non exigées
MIN	Mineure	Exigences non exigées immédiatement, mais qui devront être prises en compte ultérieurement par le produit (impact sur l'évolutivité)

2. Les objectifs du produit

2.1. Définition du produit

Le client nous demande de lui fournir le prototype d'un appareil pouvant mesurer avec une bonne précision le volume d'un vase antique.
*Il sera donc constitué d'une partie **physique** capable d'effectuer les mesures et d'une partie **logicielle** qui traitera les données afin d'obtenir un modèle 3D et le volume du vase.*
Sachant que le vase peut être incomplet et que l'appareil doit être capable de modéliser l'intérieur du vase tout en préservant l'intégrité de l'artefact qui peut être unique et dont la valeur est inestimable.

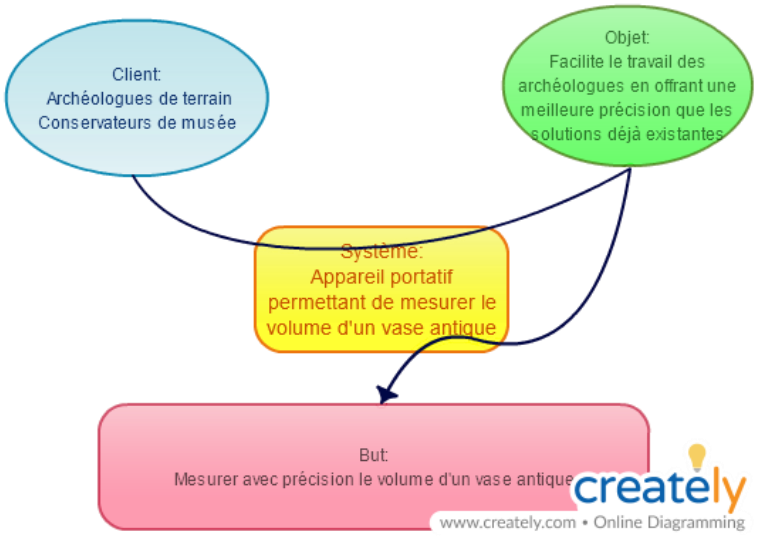
2.2. Contexte économique du produit

D'après la fiche descriptive du projet procurée par notre partenaire extérieur, il n'existe pas aujourd'hui d'appareil permettant de mesurer avec précision le volume d'un vase pour les archéologues de terrain.
Les mesures réalisées actuellement sont basées sur des représentations graphiques n'offrant que des résultats approximatifs, et le temps d'acquisition est de quelques heures environ.

3. Exigences sur le produit

3.1. Capacités Fonctionnelles

Description des fonctionnalités



Bête à corne

Nom	*Mesurer avec précision la distance séparant l'appareil du bord du vase visé*
Description	*Mesure physique et enregistrement des données*
Événement déclencheur	*L'utilisateur enclenche la mesure*
Entrées	*Aucune*
Sorties	*Données concernant la distance*
Contraintes	*Précision < 5 %* *Largeur du vase comprise entre 10cm et 1m*
Importance	*VIT*

Nom	*Envoyer chacune des mesures réalisées au logiciel*
Description	*Communication entre la partie logicielle et la partie physique réalisant les mesures*
Événement déclencheur	*Le système enclenche la communication automatiquement*
Entrées	*Informations*
Sorties	*Acquittement de l'envoi*

Contraintes	*Il faut que la communication soit suffisamment rapide pour avoir lieu entre deux mesures consécutives* *Nous souhaitons que le temps d'acquisition soit inférieur à 10 minutes.*
Importance	*VIT*

Nom	*Modéliser en 3D l'intérieur du vase*
Description	*Créer un modèle 3D à partir des informations recueillies par l'appareil de mesure*
Événement déclencheur	*Le système enclenche la modélisation automatiquement*
Entrées	*Informations*
Sorties	*Modèle 3D*
Contraintes	*Aucune*
Importance	*VIT*

Nom	*Extraire l'information concernant le volume*
Description	*Calculer le volume à partir du modèle 3D de l'intérieur du vase*
Événement déclencheur	*Le système enclenche le calcul automatiquement*
Entrées	*Le modèle 3D*
Sorties	*Valeur du volume du vase*
Contraintes	*Aucune*
Importance	*VIT*

Nom	*Contrôler la montée et la descente du capteur*
Description	*Connaître avec précision la hauteur à laquelle est situé le capteur par rapport à son support et à la base du vase.*
Événement déclencheur	*Automatique*
Entrées	*Aucune*
Sorties	*Aucune*
Contraintes	*Assurer l'intégrité du vase* *Contrôler avec précision la façon dont tourne et descend le capteur*
Importance	*VIT*

Nom	*Permettre à l'appareil de mesure de tourner et de descendre/monter*
Description	*Pour représenter en 3D l'intérieur du vase il faut pouvoir acquérir des mesures sur l'ensemble du vase*
Événement déclencheur	*Enclenchement de la mesure par l'utilisateur*

Entrées	*Aucune*
Sorties	*Aucune*
Contraintes	*Assurer l'intégrité du vase* *Contrôler avec précision la façon dont tourne et descend le capteur*
Importance	*VIT*

Nom	*Afficher le modèle 3D/volume*
Description	*Afficher le modèle 3D et le volume à l'utilisateur*
Événement déclencheur	*Automatiquement une fois la mesure lancée et la modélisation 3D effectuée*
Entrées	*Modèle 3D et Volume*
Sorties	*Aucune*
Contraintes	*Aucune*
Importance	*IMP*

3.2.Exigences non fonctionnelles

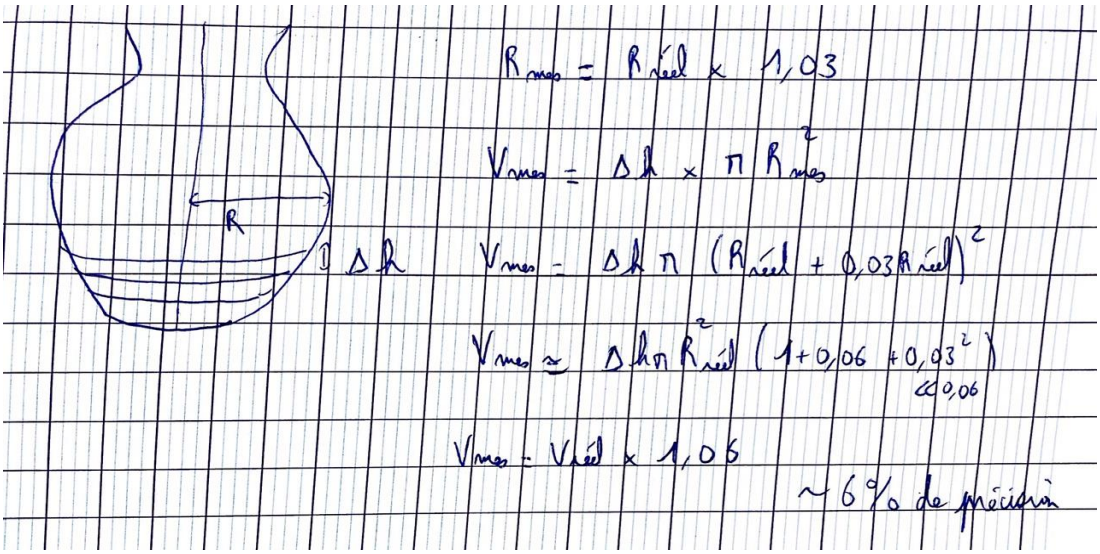
3.2.1. Rapidité et Précision

Notre client souhaite que le temps d'acquisition soit inférieur à celui des méthodes existantes, nous chercherons donc à avoir un temps d'acquisition inférieur à 10 minutes.

Nous allons essayer d'approximer la précision sur la mesure du volume en sachant que notre meilleur capteur a une précision de 3 %.

On suppose que le volume est évalué comme étant la somme de petits cylindres d'épaisseur Δh. (voir schéma ci-dessous).

Nous ne considérerons que l'imprécision apportée par le capteur ainsi on se contentera d'évaluer l'imprécision pour un petit cylindre :



$R_{mes} = R_{réel} \times 1{,}03$
$V_{mes} = \Delta h \times \pi \times R_{mes}^2$
$V_{mes} = \Delta h \times \pi \times R_{réel}^2 \times (1 + 0{,}06 + 0{,}03^2)$ *; on a 0,03*0,03<<0,03 on peut donc le négliger*
$V_{mes} = V_{réel} \times 1{,}06$ *; on a donc* ***6 % de précision dans le pire des cas****.*

Nous allons effectuer un grand nombre de mesures ce qui nous permettra d'améliorer la précision après lissage de nos résultats.

1.1. 3.2.2. Facilité d'utilisation

Le prototype final que nous proposerons à notre client viendra accompagné de la spécification du capteur choisi. Nous produirons également une notice d'utilisation détaillée du logiciel comportant des exemples d'utilisation de l'appareil.
Le logiciel comportera toutefois une interface intuitive et ne nécessitera pas de formation préalable de la part de l'utilisateur.

1.2. 3.2.3. Maintenabilité

La partie physique de notre prototype se devra d'avoir une durée de vie d'au moins5 ans. Nous devrons donc choisir nos composants en faisant attention à leur MTBF.

Le code de la partie logicielle se devra lui d'être facilement maintenable, il devra être factorisé et commenté clairement.

2. ANNEXE 2

2.1. Algorithmes de programmation des capteurs sur Arduino

```
#define bottom_tinyLidar 0x10   //adress bottom TinyLidar
#define front_tinyLidar 0x12    //adress front TinyLidar
#define chipSelect 4            //dependes on the card and the arduino
#include <I2C.h>
#include <SPI.h>
#include <SD.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_L3GD20_U.h>
int compteur;
float angle=0;
unsigned long looptime=0;
// No need to specify pins for I2C
Adafruit_L3GD20_Unified gyro = Adafruit_L3GD20_Unified(20); //initialize the
gyroscope
void setup() {
  //setup
 //int i;
 gyro.enableAutoRange(true);

 I2c.begin();
 pinMode(10, OUTPUT);           // SD pin output if not inicializated, it is an
input.
     if (!SD.begin(chipSelect))
              return;           // doesnt continue
 //set both tinyLidars to high accuracy
 I2c.write(bottom_tinyLidar, "PH");//set tinylidar to high accuracy
 I2c.read(bottom_tinyLidar, 1);
 delay (50);
 I2c.write(front_tinyLidar, "PH");//set tinylidar to high accuracy
 I2c.read(front_tinyLidar, 1);
 delay (1000); //give tinyLidars time to reset
 SD.remove("data.csv");
 //gyros part
 Serial.begin(9600);

 if(!gyro.begin())
 {
   /* There was a problem detecting the L3GD20 ... check your connections */
   Serial.println("Ooops, no L3GD20 detected ... Check your wiring!");
   while(1);
 }

} //end setup
void loop() {
 unsigned long seconds2=0;
 unsigned long seconds1=millis();
 uint16_t high;
 uint16_t distance;

 File dataFile = SD.open("data.csv", FILE_WRITE);
 I2c.write(bottom_tinyLidar,'D'); //take single measurement
```

```
  I2c.read(bottom_tinyLidar,2); //request 2 bytes from the bottom tinyLidar
  high = I2c.receive();  //receive MSB byte
  high = high<<8 ;
  high = I2c.receive(); // receive LSB byte and put them together

  compteur++;
  I2c.write(front_tinyLidar,'D'); //take single measurement
  I2c.read(front_tinyLidar,2); //request 2 bytes from the front tinyLidar
  distance = I2c.receive();  //receive MSB byte
  distance = distance<<8;
  distance = I2c.receive(); // receive LSB byte and put them together

  sensors_event_t event;
  gyro.getEvent(&event);
// Serial.print("Y: "); Serial.print(event.gyro.y); Serial.print("  ");
  //Serial.println("rad/s ");
  float velocity=event.gyro.x;
  angle=angle+velocity*looptime/1000;//calculate the angle with the speed mesured
by the gyroscope


  if (dataFile)   //if archive found store every value
   {    dataFile.print(distance);
        dataFile.print(";");
        dataFile.print(angle);
        dataFile.print(";");
        dataFile.print(high);
        dataFile.print(";");
        dataFile.println(compteur);
        } //end if
   dataFile.close();

 delay(200);

 seconds2=millis();
 looptime=seconds2-seconds1;//calculate the time of every loop
 Serial.println(angle);

 }//end loop
```

2.2. Algorithmes de traitement de données

Nous utilisons deux fichiers pythons *readFileGyroscope.py* et *dataTreatmentGyroscope.py* pour traiter les données que nous recueillons sur la carte mémoire SD.

2.2.1. Lecture de fichiers .csv

Notre premier fichier *readFileGyroscope.py* nous permet de lire le fichier *.csv* qui est stocké sur la carte mémoire. Afin, de l'utiliser, il est nécessaire d'importer le module *csv* de Python. Puis, nous importer la fonction *floor()* du module *math* pour faire les arrondis. On importe de même les fonctions du fichier *dataTreatmentGyroscope.py* car elles nous servent à faire un test ensuite.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Archeo-metrologia project
@author = Alban GOUGOUA & Oussama SEGHAIER

New treatment with the gyroscope (addition of angle)
"""

import csv
from math import floor
from dataTreatmentsGyroscope import NUMBER_POINTS_IN_CIRCLE, INFINITY,
correction, distancesToPoints, pointsToCircles, calculVolum

INITIAL_HEIGHT = 2 #: The device of measurement can go down in one vase until a
height of 2 millimeter maximum to avoid a contact.
DISTANCE = (1 + 2.5 + 30) #: The distance between the sensor and the screw.
HEIGHT = 10 #: The height between the sensor which is under the box and the
sensor which is the side.

def readFile(file) :
    """
    @author = Alban GOUGOUA

    This function allows of reading a file .csv including the values of
distances, angles and associated heights
        measured by the device.
    It adds a distance d and a height h for correcting and allowing to each value
to locate at
        the intersection of the perpendicular of the two sensors as center of a
landmark.
    It returns a list of lists of distances, angles and associated heights.

    :param file : the file .csv including data.
    """

    listDistAngHeights = [] #: The list of distances, angles and associated
heights.
    table = [] #: A table which will include values of file .csv.
    correction = [] #: A list of values of heights to correcting.

    #: Opening of the file .csv and adding of values from file in table. Then we
close this file .csv.
    with open(file, newline='') as csvfile :
        csvreader = csv.reader(csvfile)
        for row in csvreader :
            table.append(row)
    csvfile.close()

    #: We take values from table in listDistAngHeights in the form of floating
numbers. We respectively add d and h to each distance and each height.
    for i in range(1, len(table)) :
        listDistAngHeights.append([float(table[i][0]) + DISTANCE,
float(table[i][1]), float(table[i][2]) + HEIGHT, float(table[i][3])])

    #: Sort of listDistAngHeights in order descending of the height.
    listDistAngHeights = sorted(listDistAngHeights, key=lambda d : d[2],
```

```
reverse=True)

   #: Correction of values less than INFINITY
   for l in listDistAngHeights :
       if(l[2] < INFINITY) :
           correction.append((l[2], l[3])) #: We add each tuple of (height,
number) in this list for height less than 10 mm.
   correction = sorted(correction, key=lambda d : d[1]) #: We sort this list in
ascending order of number.
   counter = 0 #: We initialize a counter.
   initial_height = INITIAL_HEIGHT - 1 #: We initialize a variable to 1. This
variable is incrementing of 1.
   while(counter <= int(floor(NUMBER_POINTS_IN_CIRCLE / 2))) : #: We compare
counter to the whole part of the half of NUMBER_POINTS_IN_CIRCLE.
       if(len(correction) == 0) :
           counter = NUMBER_POINTS_IN_CIRCLE #: If the list correction is empty,
we leave the loop while.
       else :
           value = correction[0] #: We use the first tuple of correction.
           for l in listDistAngHeights :
               if(l[3] == value[1]) :
                   l[2] = initial_height + 1 #: We change the value of false
height of listDistAngHeights by initial_height + 1. We begin to 2 mm.
                   correction.remove(value) #: We remove the first tuple of
correction.
                   counter += 1 #: We increment counter.
           if(counter == int(floor(NUMBER_POINTS_IN_CIRCLE / 2))) :
               const = NUMBER_POINTS_IN_CIRCLE %
int(floor(NUMBER_POINTS_IN_CIRCLE / 2)) #: We figure out the rest of euclidian
division of NUMBER_POINTS_IN_CIRCLE by the whole part of its half.
               if(const == 0) : #: If the rest is 0, we reinitialize counter to
0 and increment initial_height to 1.
                   counter = 0
                   initial_height += 1
               elif(const == 1) : #: If the rest is 1, we reinitialize counter
to 0, increment initial_height to 1 and figure out the number of remaining
points to make one round of measurement.
                   counter = 0
                   initial_height += 1
                   for i in range(NUMBER_POINTS_IN_CIRCLE -
int(floor(NUMBER_POINTS_IN_CIRCLE / 2))) :
                       if(len(correction) == 0) :
                           break #: If the correction is empty, we leave the
loop for.
                       else :
                           value = correction[0]
                           for l in listDistAngHeights :
                               if(l[3] == value[1]) :
                                   l[2] = initial_height + 1
                                   correction.remove(value)
                   initial_height += 1

   #: We remove the column of number.
   listDistAngHeights1 = listDistAngHeights[:]
   listDistAngHeights = []
   for l in listDistAngHeights1 :
       listDistAngHeights.append([l[0], l[1], l[2]])
```

```
    return listDistAngHeights

file = 'fileTestWithAngle.csv'
listDH = readFile(file)
print("\nListe de distances, angles et de hauteurs associées = ", sorted(listDH,
key=lambda d : d[2]))
a =  correction(listDH)
print("\n---------------------- Test de la fonction de correction --------------
----------------")
print("\nIndex des distances infinies =", a[1])
print("\nCorrection =", a[0])

counter = 0
for d in a[0] :
    if d[0] < INFINITY :
        counter += 1
        print(":-( :-(")

if counter == 0 :
    print("\nCorrection Successful")

print("\n---------------------- Test de la fonction de distancesToPoints -------
------------------------")
a = distancesToPoints(a[0], a[1])
print("\nListe des coordonnées = ", a[0])

print("\n---------------------- Test de la fonction de pointsToCircles ---------
----------------------")
print("\nNUMBER_POINTS_IN_CIRCLE = ", NUMBER_POINTS_IN_CIRCLE)
a = pointsToCircles(a[0], a[1])
print("\nListe des rayons et hauteurs = ", a[0])

print("\n---------------------- Test de la fonction de calculVolum -------------
-----------------")
a = calculVolum(a[0])
print("\nVolume =", a, "mm3")
```

Ce code permet de lire un fichier .csv, mettre ses données dans un tableau qui sera converties en liste de distances, angles, hauteurs associées et numéros. Cette liste est ensuite corrigée car le capteur donne des valeurs erronées en dessous de 10 mm. En utilisant le nombre de cercles que nous désirons avoir pour le calcul de volume, nous faisons une approximation des valeurs d'hauteurs erronées par approximation d'un millimètre pour la moitié d'un cercle. En effet, on considère qu'un cercle fait 2 mm car un tour du capteur autour de la vis fait 2 mm. Puis, nous copions cette liste corrigée dans une nouvelle liste pour supprimer la dernière colonne des numéros qui nous ont servis à savoir l'ordre d'acquisition de chaque point, un moyen de rendre l'acquisition pertinente et précise pour le traitement des données.

2.2.2. Traitements de données

Les commentaires de l'ensemble des fonctions contenues dans cette partie permettent une bonne compréhension du traitement des données.

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Archeo-metrologia project
@author = Alban GOUGOUA & Oussama SEGHAIER

New treatment with the gyroscope (addition of angle)
"""

from math import pi, fabs, cos, sin, sqrt #: Import a few of constant and
functions of module math.
import numpy as np #: Import all functions of module numpy.
import random #: Import module random.


INFINITY = 10 #: All value less than 10 millemeters and greater than 1000
millimeters
                  #: aren't detected by the sensor.
                  #: We must complete with other values by approximation.
SPEED_MOTOR = 2.09 #: The angular speed of motor in radians/second.
FREQUENCY_SENSOR = 4 #: The frequency of sensor in hertz (1/second).
NUMBER_POINTS_IN_CIRCLE = int(FREQUENCY_SENSOR * ((2*pi) / SPEED_MOTOR)) #: The
number of measured points when the screw doing one round on itself.

################################################## BEGINNING
##################################################

def correction(listDistAngHeights) :
   """
   @author = Alban GOUGOUA

   This function allows of averaging the values of the listDistAngHeights which
are infinites to values
       near of their neighbours not infinite.
   It returns a couple composed of one list of lists (corlistDistAngHeights) and
another list (indexInfDistList1).

   :param listDistAngHeights : The list of distances, angles and associated
heights from a file .csv.
       This list is a list of lists [distance, angle, height].
       Distance and height are in millemeter.
       Angle is in degree.
   """

   #: Sort of listDistAngHeights in order descending of the height.
   listDistAngHeights = sorted(listDistAngHeights, key=lambda d : d[2],
reverse=True)

   #: Share the listDistAngHeights in three lists : first for the distances,
second for the angles and third for the associated heights.
   listDist = []
   listAng = []
   listHeight = []
   for l in listDistAngHeights :
       listDist.append(l[0])
       listAng.append(l[1])
       listHeight.append(l[2])
```

```
    #: Put the index of infinite distances in a list.
    indexInfDistList = []
    for i in range(len(listDist)) :
        if listDist[i] < INFINITY :
            indexInfDistList.append(i)

    #: Copy of the list of indexes of infinite distances in another list.
    indexInfDistList1 = indexInfDistList[:]

    #: First correction in putting all infinte distances near INFINITY.
    for i in indexInfDistList :
        listDist[i] = random.randint(10, 15) #: This value is abritrary.

    #: Give an averaged value to infinite values in counting the number of
successive indexes.
    sucIndex = [] #: The list of successive indexes.
    nonsucIndex = [] #: The list of nonsuccessive indexes.
    i = 0 #: First index of the list. His value doesn't never change during the
processing.
    while i < len(indexInfDistList) :
        #: Case 1 : the lenght of the list of indexes is greater than 1.
        if len(indexInfDistList) > 1 :
            #: We check if the list of successive indexes is empty.
            if len(sucIndex) == 0 :
                if indexInfDistList[i] == (indexInfDistList[i+1] - 1) : #: We
check if the two first elements are successives.
                    #: We add the two first elements to the list of successive
indexes.
                    sucIndex.append(indexInfDistList[i])
                    sucIndex.append(indexInfDistList[i+1])
                    #: We delete the two first elements to the list of indexes.
When we remove the first element of the list, the second element locates at the
index 0 now.
                    indexInfDistList.remove(indexInfDistList[i])
                    indexInfDistList.remove(indexInfDistList[i])
                else : #: If the two first elements aren't successives, do below.
                    nonsucIndex.append(indexInfDistList[i]) #: We add the first
element in the list of nonsuccessive indexes.
                    indexInfDistList.remove(indexInfDistList[i]) #: And we remove
it of the list of indexes.
            #: We check if the list of successive indexes isn't empty, that is to
say, is greater than or equal to 2.
            elif len(sucIndex) >= 2 :
                sucIndex.sort() #: We sort this list in ascending order.
                #: We check if the first element of the list of indexes is equal
to last element of the list of successive indexes.
                if indexInfDistList[i] == (sucIndex[-1] + 1) :
                    sucIndex.append(indexInfDistList[i]) #: We add it in the list
of successive indexes.
                    indexInfDistList.remove(indexInfDistList[i]) #: And we remove
it of the list of indexes.
                #: In the reverse case, do below.
                else :
                    minDist = listDist[sucIndex[0] - 1] #: We take the value of
element locating before the first element of the list of successive indexes in
the list of distances.
                    maxDist = listDist[sucIndex[-1] + 1] #: We take the value of
```

```
element locating after the last element of the list of successive indexes in the
list of distances.
                    newMin = min(minDist, maxDist) #: We take the smallest value.
                    #: We correct the value of infinite distances in finite
values in doing a mean.
                    for s in sucIndex :
                        listDist[s] = newMin + (fabs(maxDist -
minDist)/(len(sucIndex) + 1))
                        newMin = listDist[s]
                    sucIndex = [] #: We empty this list.
        #: Case 2 : the lenght of the list of indexes is equal to 1.
        else :
            #: We check if the list of successive indexes isn't empty, that is to
say, is greater than or equal to 2.
            if len(sucIndex) >= 2 :
                sucIndex.sort() #: We sort this list in ascending order.
                #: We check if the first element of the list of indexes is equal
to last element of the list of successive indexes.
                if indexInfDistList[i] == (sucIndex[-1] + 1) :
                    sucIndex.append(indexInfDistList[i]) #: We add it in the list
of successive indexes.
                    indexInfDistList.remove(indexInfDistList[i]) #: And we remove
it of the list of indexes.
                #: In the reverse case, do below.
                else :
                    minDist = listDist[sucIndex[0] - 1] #: We take the value of
element locating before the first element of the list of successive indexes in
the list of distances.
                    maxDist = listDist[sucIndex[-1] + 1] #: We take the value of
element locating after the last element of the list of successive indexes in the
list of distances.
                    newMin = min(minDist, maxDist) #: We take the smallest value.
                    #: We correct the value of infinite distances in finite
values in doing a mean.
                    for s in sucIndex :
                        listDist[s] = newMin + (fabs(maxDist -
minDist)/(len(sucIndex) + 1))
                        newMin = listDist[s]
                    sucIndex = [] #: We empty this list.
            #: We check if the list of successive indexes is empty.
            else :
                nonsucIndex.append(indexInfDistList[i]) #: We add the one element
in the list of nonsuccessive indexes.
                indexInfDistList.remove(indexInfDistList[i]) #: And we remove it
of the list of indexes.

    #: We correct the value of last infinite distances which are in the list of
successive indexes.
    if len(sucIndex) > 0 :
        minDist = listDist[sucIndex[0] - 1]
        if (sucIndex[-1] + 1) > 0 :
            maxDist = listDist[0]
        else :
            maxDist = listDist[sucIndex[-1] + 1]
        newMin = min(minDist, maxDist)
        for s in sucIndex :
            listDist[s] = newMin + (fabs(maxDist - minDist)/(len(sucIndex) + 1))
            newMin = listDist[s]
```

```
    #: We correct the value of infinite distances which are in the list of
nonsuccessive indexes.
    for i in nonsucIndex :
        if i+1 > len(listDist) :
            newMin = min(listDist[i-1], listDist[0])
            listDist[i] = newMin + (fabs(listDist[i-1] - listDist[0]) / 2)
        else :
            newMin = min(listDist[i-1], listDist[i+1])
            listDist[i] = newMin + (fabs(listDist[i-1] - listDist[i+1]) / 2)

    #: Put the value of angles in radians.
        #: Source : https://fr.wikihow.com/convertir-des-degr%C3%A9s-en-radians
    for a in range(len(listAng)) :
        listAng[a] *= pi/180

    #: Put the list of distances, the list of angles and the list of associated
heights in a list.
    corlistDistAngHeights = [] #: The corrected list of distances, angles and
heights.
    for d in range(len(listDist)) :
        corlistDistAngHeights.append([listDist[d], listAng[d], listHeight[d]])


    return (corlistDistAngHeights, indexInfDistList1)

    ##############################################################################
##############################################################################
###

def distancesToPoints(corlistDistAngHeights, indexInfDistList) :
    """
    @author = Alban GOUGOUA

    This function allows to figure out coordinates of points from a distance.
    It does it for a list of distances and returns a list of coordinates (x,y,h)
where x and y are calculated with distances
        and angles ; h corresponds to the height associated to each distance.
    It returns also the list of indexes of infinite distances (indexInfDistList).

    :param corlistDistAngHeights : The list of distances, angles and associated
heights from a file .csv which was corrected.
        This list is a list of lists [distance, angle, height].
        Distance and height are in millemeter.
        Angle is in radian.

    :param indexInfDistList : The list of indexes of infinite distances.
        It is a list of integers.

    :param speed : The rotation speed of the motor which is in radians/second.
        It is a positif real number.

    :param frequency : The frequency of the sensor which is hertz (1/second).
        It is a positif real number.
    """

    #: Initialization of differents variables to 0.
    listCoordAndHeights = []
```

```
    x = 0
    y = 0
    h = 0

    #: We figure out the coordinates (x, y, h) of each distance of the list of
distances, angles and heights.
    for d in corlistDistAngHeights :
        #: We figure out coordinates (x, y) from distance and associated angle. h
receives the value of associated height.
        x = d[0] * cos(d[1])
        y = d[0] * sin(d[1])
        h = d[2]

        listCoordAndHeights.append((x, y, h)) #: We add each calculated data in
the list.

    #: We return the list of coordinates and associated heights and the list of
indexes of infinite distances.
    return (listCoordAndHeights, indexInfDistList)

    ##########################################################################
##########################################################################
###

def pointsToCircles(listCoord, indexInfDistList) :
    """
    @author = Alban GOUGOUA & Oussama SEGHAIER

    This methods finds the least squares circle fitting a set of 2D points (x,y).
    It returns the list of the best circles using the least square method and the
associated height ;
        it returns also, the list of gaps.

    INPUTS :
      @ListCoord = [(x,y,h) .... ] : (x,y) the cartesien coordinates of a point ,
h : height of the point.

    source of methods : http://scipy-
cookbook.readthedocs.io/items/Least_Squares_Circle.html
        We choose the method 2b : "leastsq with jacobian".
    """

        ################################################## Beginning of this
method ##################################################
    # == METHOD 2b ==
    # Advanced usage, with jacobian

    def pointsToRadius(X, Y) :
        """
        @author = Alban GOUGOUA

        This function figures out from a numpy array list of coordinates the
least squares circle.
        It returns the radius of this circle.

        :param X : a numpy array list including NUMBER_POINTS_IN_CIRCLE
coordinates in x.
```

```
        :param Y : a numpy array list including NUMBER_POINTS_IN_CIRCLE
coordinates in y.
        """

        from scipy import optimize as op


        def calc_R(xc, yc):
            """ calculate the distance of each 2D points from the center c=(xc,
yc) """
            return np.sqrt((X-xc)**2 + (Y-yc)**2)

        def f_2b(c):
            """ calculate the algebraic distance between the 2D points and the
mean circle centered at c=(xc, yc) """
            Ri = calc_R(*c)
            return Ri - Ri.mean()

        def Df_2b(c):
            """ Jacobian of f_2b
            The axis corresponding to derivatives must be coherent with the
col_deriv option of leastsq"""
            xc, yc     = c
            df2b_dc    = np.empty((len(c), X.size))

            Ri = calc_R(xc, yc)
            df2b_dc[ 0] = (xc - X)/Ri                   # dR/dxc
            df2b_dc[ 1] = (yc - Y)/Ri                   # dR/dyc
            df2b_dc       = df2b_dc - df2b_dc.mean(axis=1)[:, np.newaxis]

            return df2b_dc


        # coordinates of the barycenter
        x_m = np.mean(X)
        y_m = np.mean(Y)

        #: We figure out the center of the circle.
        center_estimate = x_m, y_m
        center_2b = op.leastsq(f_2b, center_estimate, Dfun=Df_2b, col_deriv=True)

        #: We figure out the radius of the circle.
        xc_2b, yc_2b = center_2b[0]
        Ri_2b        = calc_R(xc_2b, yc_2b)
        R_2b         = Ri_2b.mean()

        return R_2b

        ################################################## End of this method
################################################

        ##########################################################################
################
        ##                           CoordsAndHeight(ListCoord)-->>
X,Y,H                       ##
        ##########################################################################
################
```

```
    def coordsAndHeight(listCoord) :
        """
        @author = Oussama SEGHAIER

        This function transforms a listCoord as explained above to two numpy
arrays X and Y
            constituted respectively of the x coordinates and the y coordinates.

        the funcion returns
        @X : a numpy array of the x coordinates
        @Y : -------------------- y -----------
        @H : a list of the h associated coordinates
        """

        #: We create the list of each coordinate x, y.
        X_r = []
        Y_r = []
        H = []

        #:We add each value of the list of coordinates and associated heights in
lists above.
        for coord in listCoord :
            X_r.append(coord[0])
            Y_r.append(coord[1])
            H.append(coord[2])

        #: We transform these lists in numpy array lists.
        X = np.array(X_r)
        Y = np.array(Y_r)

        return X,Y,H


    #: We figure out the radius of each circle to each NUMBER_POINTS_IN_CIRCLE
points.
    counter = 0 #: A counter forar counting the number of points reaching in the
listCoord.
    numOfCircle = 0 #: The number of circle.
    pseudoListCoord = [] #: A list for storing the NUMBER_POINTS_IN_CIRCLE values
of listCoord.
    listRadiAndHeights = [] #: A list which will return at the end of this
function.
    for coord in listCoord :
        counter += 1
        pseudoListCoord.append(coord) #: We add each value of listCoord.
        #: We figure out the radius when the counter is equal to
NUMBER_POINTS_IN_CIRCLE.
        if(counter == NUMBER_POINTS_IN_CIRCLE) :
            numOfCircle += 1 #: We increase the number of circle to 1.
            X = coordsAndHeight(pseudoListCoord)[0] #: We recover the numpy array
list of coordinates x.
            Y = coordsAndHeight(pseudoListCoord)[1] #: We recover the numpy array
list of coordinates y.
            H = coordsAndHeight(pseudoListCoord)[2][0] #: We recover the last
value of the list of heights.
            R = pointsToRadius(X, Y) #: We figure out the radius.
            listRadiAndHeights.append((R, H)) #: We add the value of radius and
the associated height in the list.
```

```
            counter = 0 #: We reinitialize the counter to 0.
            pseudoListCoord = [] #: We empty this list.

    if(len(pseudoListCoord) != 0) :
        numOfCircle += 1 #: We increase the number of circle to 1.
        X = coordsAndHeight(pseudoListCoord)[0] #: We recover the numpy array
list of coordinates x.
        Y = coordsAndHeight(pseudoListCoord)[1] #: We recover the numpy array
list of coordinates y.
        H = coordsAndHeight(pseudoListCoord)[2][0] #: We recover the last value
of the list of heights.
        R = pointsToRadius(X, Y) #: We figure out the radius.
        listRadiAndHeights.append((R, H)) #: We add the value of radius and the
associated height in the list.
        counter = 0 #: We reinitialize the counter to 0.
        pseudoListCoord = [] #: We empty this list.


    return (listRadiAndHeights, indexInfDistList)

    ##############################################################################
##############################################################################
###

def calculVolum(rayList) :
    """
    @author = Alban GOUGOUA & Oussama SEGHAIER

    This function figures out the volume in millimeter cube from a list of rays
and heights.

    rayList = [(R,h).....] list of radius and associated heights that calculates
the vase volume
    """

    def volume(r1, r2, h) :
        """calculates the volume of a cone truncated with parameters R = radius
and H = height.
        source : https://calculis.net/volume/cone-tronque
        """
        return (h*pi/3)*((r1**2)+(r2**2)+r1*r2)

    volum = 0
    #: We figure out the volume by sum of each cone's volume.
    for i in range(len(rayList)) :
        if(i != len(rayList) - 1) :
            h = rayList[i][1] - rayList[i+1][1]
            r1 = rayList[i][0]
            r2 = rayList[i+1][0]
            volum += volume(r1, r2, h)

    return volum

################################################## END
##################################################
```

2.3. Étapes de développement de l'interface graphique

Vous trouverez dans cette annexe les étapes du développement de l'interface graphique. Tout d'abord, on importe les modules à utiliser :

```
# -*- coding: utf-8 -*-
"""
@author: Seghaier Oussama
@Project: 51
"""
from pyqtgraph.Qt import *
import pyqtgraph as pg
import pyqtgraph.opengl as gl
from pyqtgraph.dockarea import *
import numpy as np
import sys
import time
from readFile import *  # module to read the file written by us
from dataTreatements import * # module with all the date treatement functions
```

2.3.1. Développement du module de représentation 3D

Avec les modules de traitement de données expliquées dans la partie 1.3.2 du rapport, les données finales du vase sont représentées sous forme d'une liste de cercles avec leurs hauteurs par rapport au sol (ou base du vase) : $List = [(R_1, H_1), \dots, (R_n, H_n)]$ avec H_1 la hauteur du vase et $H_n = 0$. Pour représenter cette liste sous forme d'un vase, on crée d'abord un module qui représente un cône tronqué à partir de deux couples (R_1, H_1) et (R_2, H_2).

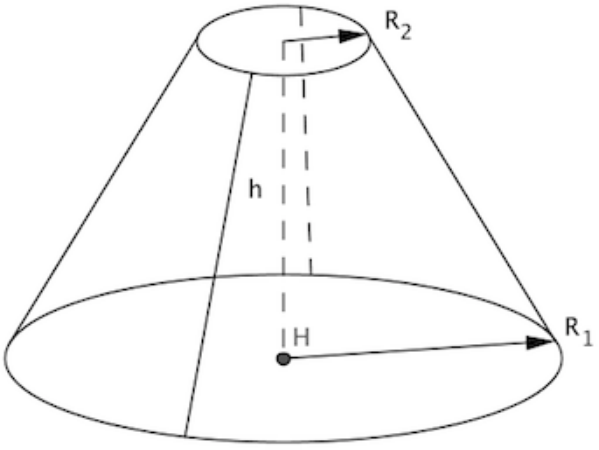


Figure 1 : Exemple d'un cône tronqué

On modélise le vase comme la superposition des plusieurs cônes tronqués avec une cercle commun. Par exemple, on représente le premier cône tronqué à partir de (R_1, H_1) et (R_2, H_2) avec hauteur H_2 par rapport à l'origine puis au dessous duquel on représente le deuxième cône tronqué à partir de (R_2, H_2) et (R_3, H_3) à une hauteur H_3 par rapport à l'origine. Et ainsi de suite jusqu'à la fin de la liste. Exemple : pour la liste [(5,20), (8,10), (3,5)] qui contient 2 cônes tronqué on obtient la représentation suivante :

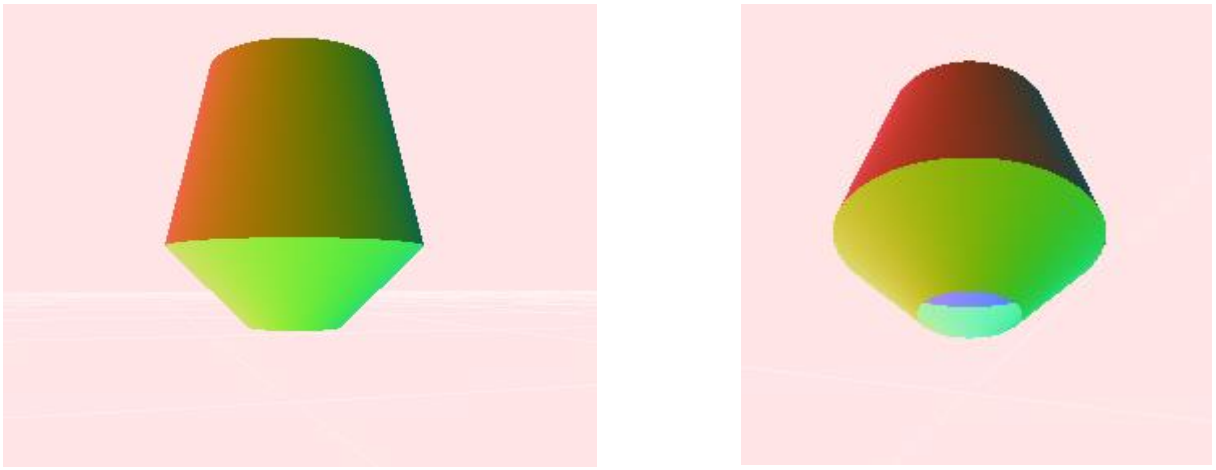


Figure 2 : Exemple de représentation 3D de deux cônes tronqués

Vous trouverez ci-dessous le code de la fonction qui dessine le vase en 3 dimensions :

```
def drawWithoutGaps(ConesList,w5):

	"""Draw the 3D modelisation of the vase
	ConesList : List = [(R1,H1),...,(Rn,Hn)] model of the vase by circles qnd
their heights
	w5     : the window where we will draw the vase
	"""
	w5.items = [] # the items list set to empty in case we change another vase
	for i in range(len(ConesList)-1) :
	# we draw the Cone number i
	md = gl.MeshData.cylinder(rows=10, cols=100,
radius=[ConesList[i+1][0],ConesList[i][0]],length=(ConesList[i][1]-
ConesList[i+1][1]))
	colors = np.ones((md.faceCount(), 4), dtype=float)
	colors[::2,0] = 0.2
	colors[:,1] = np.linspace(0, 1, colors.shape[0])
	md.setFaceColors(colors)
	m5 = gl.GLMeshItem(meshdata=md, smooth=True, shader='viewNormalColor',
glOptions='opaque')
	# ze translate the cone number i to the right position
	m5.translate(0,0,ConesList[i+1][1])
	# wz then draw the cone
	w5.addItem(m5)
```

2.3.2. Développement du module de représentation 2D

Comme mentionné précédemment, nous avons comme résultat du traitement des données du vase la liste $List = [(R_1, H_1), \dots, (R_n, H_n)]$. Notre objectif est de représenter en 2D le vase à partir de cette liste. On se place dans un repère (x, y), si on représente tous les points de coordonnées (R_i, H_i), on obtient la figure suivante :

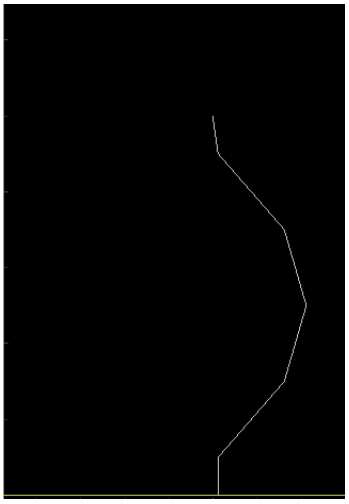


Figure 3 : Représentation 2D de la liste des rayons et hauteurs

On représente maintenant dans le même graphe la liste $[(-R_1, H_1), \dots, (-R_n, H_n)]$ et une ligne qui relie (R_n, H_n) et $(-R_n, H_n)$, on obtient la représentation du vase en 2D :

Figure 4 : Représentation 2D du vase

On ajoute maintenant une ligne horizontale qui bouge avec le mouvement de la souris et on écrit une fonction qui nous donne l'ordonnée de la position de la souris dans le graphe qui nous serait utile pour calculer le volume de la partie sélectionnée du vase.

Figure 5 : Représentation 2D du vase avec une ligne horizontale jaune qui bouge avec le curseur de la souris

Voici la partie du code qui permet de représenter cela :

```
        # We create the horizontal line
        hLine = pg.InfiniteLine(angle=0, movable=True)

        def update2D(ConesList):
        """Function to call every time we import a new file  """
        w.clear() # we clear the last data to draw a new vase
        w.addItem(hLine, ignoreBounds=True) # we add the horizontal line and we
draw the vase as explained
        Rays1 = [ConesList[i][0] for i in range(len(ConesList))]
        Rays2 = [-Rays1[i] for i in range(len(Rays1))] #
        heig1 = [ConesList[i][1] for i in range(len(ConesList))] # heights list

        w.plot(Rays1,heig1)
        w.plot(Rays2,heig1)
        w.plot([Rays1[-1],Rays2[-1]],[heig1[-1],heig1[-1]])
        w.enableAutoScale()

        vb = w.vb

        def mouseMoved(evt): # every time we move the mose , the line will move
with it and recover
        pos = evt[0]  ## using signal proxy turns original arguments into a tuple
        if w.sceneBoundingRect().contains(pos):
                mousePoint = vb.mapSceneToView(pos)
                index = int(mousePoint.x())
                hLine.setPos(mousePoint.y())
                Selected = VolVasePortion(ConesList,vb.mapSceneToView(evt[0]).y())
                Total = CalcVolume(ConesList)
                l =
[[str(Convert(Selected,Selected[1],Combo.currentText())[0]),str(Combo.currentTex
```

```
t())],[str(Convert(Total,Total[1],Combo.currentText())[0]),str(Combo.currentText
())]]
            w4.setData(l)
            w4.setHorizontalHeaderLabels("Value;Unit;".split(";"))
            w4.setVerticalHeaderLabels("Selected;Total;".split(";"))

      proxy = pg.SignalProxy(w.scene().sigMouseMoved, rateLimit=60,
slot=mouseMoved)
```

2.3.3. Développement de l'interface graphique principale

C'est la dernière partie où on va assembler tous les modules et fenêtres et les mettre dans une seule fenêtre principale. On définit la fonction *gui* (*graphical user interface*) et on définit la fenêtre principale de taille *1024x576* :

```
def gui():
      """ This is the principal function that will be enable the user to
interact with the software functionnalities """

      app = QtGui.QApplication([]) # the application
      win = QtGui.QMainWindow() # the main window
      HelpWindow = QtGui.QWidget()
      area = DockArea()
      win.setCentralWidget(area)
      win.resize(1024,576)
      win.setWindowTitle('Archeo-Metrologia') # Window Title
```

Puis on crée les 4 espaces pour placer les 4 interfaces expliquées dans la partie conception selon les tailles dans le tableau ci-dessous :

Interfaces	Taille
Interface 1 : les boutons	1024x30
Interface 2 : la représentation 2D	385x273
Interface 3 : tableau du volume	244x273
Interface 4 : la représentation 3D	405x273

Tableau 1 : Tailles des interfaces

Voici la partie du code pour créer ces différentes interfaces :

```
      ## Create docks, place them into the window one at a time.
      ## The Docks Are the Windows That will contain the necessary information
      """
      I will create 6 Docks:
      d1 : this dock contains the necessary buttons like open file,.. and the
ability to change the global variables
      d4 : this dock will show the volume details
```

```
        d5 : _____________ show the 3D Contents
        d6 : hier we will have a 2d representation of the vase and be able to
select a part of the vase
        """
        d1 = Dock("Parameters", size=(1024,30))
        d4 = Dock("Volume", size=(244,273))
        d5 = Dock("3D Model",size = (405,273))
        d6 = Dock("2D Model",size = (385,273))
        d1.hideTitleBar()
        d5.hideTitleBar()
        d6.hideTitleBar()

 #### positionning docks ######
        area.addDock(d1, 'top')
        area.addDock(d5, 'bottom',d1)
        area.addDock(d4, 'left',d5)
        area.addDock(d6, 'left',d4)
```

On a fait ces choix pour respecter la maquette faite dans la conception. Maintenant on ajoute à chaque espace ces fonctionnalités.

```
        w1 = pg.LayoutWidget()

        w4 = pg.TableWidget()
        w4.setWindowTitle("Volume Details")
        w4.setRowCount(2)
        w4.setColumnCount(2)
        w4.setHorizontalHeaderLabels("Value;Unit;".split(";"))
        w4.setVerticalHeaderLabels("Selected;Total;".split(";"))

        w5 = gl.GLViewWidget()

        w6 = pg.GraphicsWindow(title="2D Vase")
        w = w6.addPlot()
## Adding widget to docks areas
        d1.addWidget(w1)

        ouvrir = QtGui.QPushButton('Open File')

        afficherTrous = QtGui.QCheckBox('Show Gaps')

        apropos = QtGui.QPushButton('About US')

        Help = QtGui.QPushButton('Help')

        Combo = QtGui.QComboBox()
        Combo.setWindowTitle('Choose Volume Unit')
        Combo.addItems(Units)

        textbox = QtGui.QTableWidgetItem()

        filedialog = pg.FileDialog(None,"Load Vase file..",'C:/', "Vase File
(*.csv)")

######################## Design #################
        w1.addWidget(ouvrir, row=0, col=0)
```

```
        w1.addWidget(afficherTrous, row=0, col=5)
        w1.addWidget(Help, row=0, col=10)
        w1.addWidget(apropos, row=0, col=15)
        w1.addWidget(Combo,row = 0,col = 2)


        ConesList = []
        def openfile():
             ConesList = []
             FileName = unicode(filedialog.getOpenFileName(None,"Load Vase
file..",'C:/', "Vase File (*.csv)"))
             ConesList1 = TreatData(FileName)[:-1]

             for i in range(len(ConesList1)):

                ConesList.append((round(ConesList1[i][0]/10,2),ConesList1[i][1]))
             print(ConesList)
             drawWithoutGaps(ConesList,w5)
             w.clear()
             update2D(ConesList)

        ouvrir.clicked.connect(openfile)

        w5.setBackgroundColor(255,230,230,0)
        w5.show()
        w5.setWindowTitle('Vase 3D Representation')
        w5.setCameraPosition(distance=200)

        g = gl.GLGridItem()
        g.scale(50,50,50)
        w5.addItem(g)
        d5.addWidget(w5)

        d4.addWidget(w4)
        d6.addWidget(w6)
        #apropos.clicked.connect(afficher3D)
        layout = QtGui.QGridLayout()
        HelpWindow.setLayout(layout)
```

À la fin, on affiche la fenêtre et on exécute la fonction de l'interface principal :

```
        win.show()
        import sys
        if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()
################################################################################
############
if __name__ == '__main__':
        gui()
```

On obtient la fenêtre suivante (avec l'exemple d'un vase) :

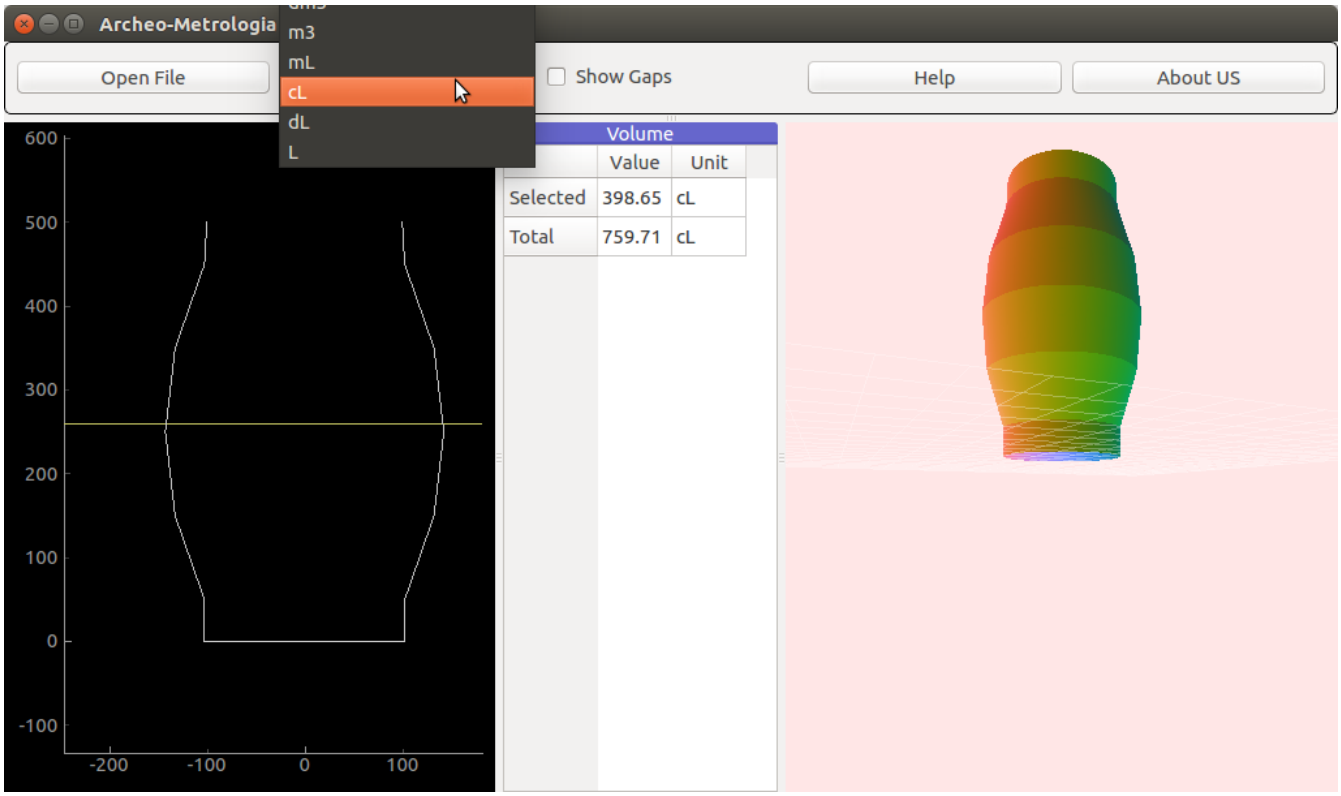


Figure 6 : Capture d'écran du logiciel