LIÈGE université Library

http://lib.uliege.be https://matheo.uliege.be

Élaboration d'un système de mesure de surface non-réfléchissante dans le visible par déflectométrie infrarouge

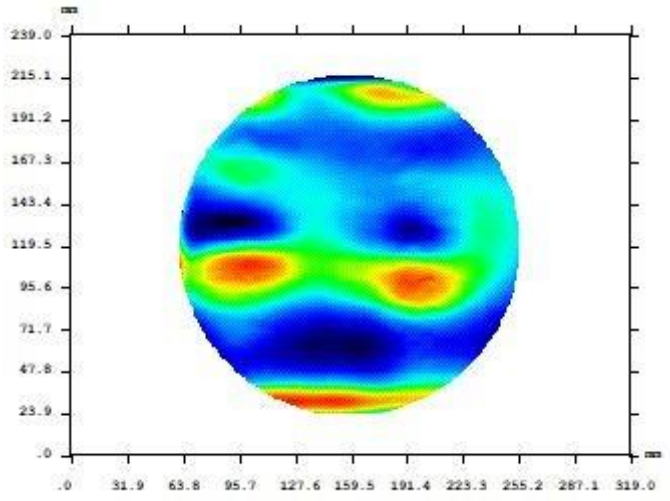
Auteur : Bolen, Thomas
Promoteur(s) : Habraken, Serge; 8565
Faculté : Faculté des Sciences
Diplôme : Master en sciences spatiales, à finalité spécialisée
Année académique : 2018-2019
URI/URL : http://hdl.handle.net/2268.2/8438

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

Élaboration d'un système de mesure de surface non-réfléchissante dans le visible par déflectométrie infrarouge



Mémoire présenté par **Thomas BOLEN**

pour l'obtention du **Master en Sciences Spatiales à finalité spécialisée**

à l'**Université de Liège**

Année académique 2018-2019

Promoteur pour l'Université :

Serge HABRAKEN, Université de Liège

Promoteur pour la société :

Fabrice WOLFS, AMOS

Membres du jury :

Marc GEORGES, Centre Spatial de Liège

Vincent MOREAU, AMOS

Gregor RAUW, Université de Liège

Parce qu'un travail de cette ampleur ne s'effectue jamais seul, je tiens à remercier

Messieurs Fabrice Wolfs et Serge Habraken pour leur confiance dans le cadre de la réalisation de ce mémoire,

Messieurs Philippe Gilson et Pierre Gloesener pour leur accueil chaleureux au sein de la société AMOS,

Monsieur Moïse Baron pour nos partages, nos discussions et l'entraide dans la réalisation de notre projet commun,

Monsieur Marc Georges pour son initiation au monde de la déflectométrie,

Monsieur Alexis Bechet pour nos nombreuses discussions sur la découverte de la déflectométrie,

Ma maman, Madame Françoise Gauthier, pour la relecture et la correction de cet ultime travail,

Ma chérie, ma famille, mes collègues et mes amis pour leur indéfectible soutien et tous leurs encouragements tout au long de la réalisation de ce projet.

Table des matières

Table des figures ..... E
Introduction 1
Travaux et documentation sur la déflectométrie infrarouge 5
1. Les principes de la déflectométrie 5
2. La déflectométrie infrarouge 9
3. L'intégration d'un gradient 11
4. Les travaux réalisés par AMOS 13
5. Le projet SLOTS 17
6. La structure du programme 19
L'instrument de déflectométrie infrarouge 21
1. Vision globale de l'instrument 21
2. Le scan du fil chauffé 22
3. La caméra infrarouge 23
4. Le chariot élévateur 24
5. La sonde de distance 25
6. Le programme de l'instrument 25
7. Les différentes installations 27
8. Mon apport au développement du dispositif 28
Calcul des pentes 29
1. Présentation de mon code 30
2. Analyse des performances 33
3. Les résultats obtenus 34
Intégration du gradient 35
1. Le générateur de surface 35
2. L'intégrateur de gradient 37
3. Les résultats 39
Programme complet de reconstruction par déflectométrie infrarouge 45

1. Le programme 45
2. Résultats obtenus 48
3. Analyse des performances 52
Octopus 53
1. Présentation du programme 53
2. Résultats obtenus 61
3. Analyse des performances 66
Améliorations, points d'attention et autres options envisageables 69
1. Les points abordés précédemment 69
2. Les nouvelles idées 71
Conclusion 73
Bibliographie 75
Annexe 1 77
1. Deflect.m 77
2. JPEG.m 79
3. Intensite.m 81
4. Coordonnees.m 83
5. Pentes_X.m 85
6. Pentes_Y.m 86
Annexe 2 87
1. Fct_pour_int2D.m 87
2. Cartes des pentes de chaque surface avec et sans bruit 91
Annexe 3 93
Annexe 4 101
1. Deflect.m 101
2. JPEG.m 104
3. Info_doss_im.m.m 106
4. Coordonnees.m 107
5. Intensite.m 108
6. Pentes_X.m 109
7. Pentes_Y.m 110
8. Integrad_2D.m 111
Annexe 5 119
1. Structure de base de la fonction GUIDE 119
2. Octopus.m 121
3. Ouvrir.m 124

4. Parametres.m 132
5. Centre_de_masse.m 141
6. Centre_de_masse_boutons.m 149
7. Pentes.m 151
8. Ouvrir2.m 157
9. Pentes2.m 165
10. Surface_Finale.m 177

Table des figures

Figure 1 - Un des ATS présents sur le site du VLT à Cerro Paranal au Chili (Crédits : Iztok Boncina/ESO) 1
Figure 2 - Photographie du système de mesure SLOTS [1] 3
Figure 3 - Photographie du système de mesure par déflectométrie infrarouge développé par AMOS en 2014 [4] 3
Figure 4 - Schéma du principe de base de la déflectométrie [6] 6
Figure 5 - Schéma du calcul de la phase φ [6] 6
Figure 6 - Représentations du test d'Hartmann (a) et du "test inversé d'Hartmann" (b) [7] 7
Figure 7 - Représentation de la configuration du problème et des termes sous-jacents [2] 8
Figure 8 - Photographie d'un dispositif de mesure par déflectométrie infrarouge à l'aide d'un écran codé [9] 10
Figure 9 - Photographie d'un dispositif de déflectométrie infrarouge à l'aide d'un fil chauffé [9] 10
Figure 10 - Photographie de l'écran couvert d'un quadrillage de résistances dans le visible (A) et dans l'infrarouge avec plusieurs résistances "allumées" (B) [9] 11
Figure 11 - Photographie montrant la reconstitution du motif d'une déflectométrie infrarouge à l'aide d'un écran chauffé par laser [9] 11
Figure 12 - Représentation des voisins du pixel surface qui permettent sa reconstruction [11] 12
Figure 13 - Photographie du premier dispositif créé par l'équipe d'AMOS [4] 14
Figure 14 - Photographies du second dispositif créé par l'équipe d'AMOS [4] 14
Figure 15 - Premiers résultats obtenus par le dispositif créé par AMOS. À gauche : comparatif des mesures de la surface par mesure 3D et déflectométrie. À droite : comparaison des mêmes mesures après avoir retiré les premiers polynômes de Zernike. [4] 15
Figure 16 - Résultats de mesure de la surface du miroir obtenus par mesure 3D et par le second dispositif de mesure par déflectométrie infrarouge [4] 16
Figure 17 - Carte des surfaces obtenues par SLOTS (a), par un interféromètre (b) et la différence entre les deux cartes (c) [2] 18
Figure 18 - Photographie globale du dispositif [5] 22
Figure 19 - Photographie de la partie supérieure du dispositif [5] 22
Figure 20 - Représentation du mouvement de scan le long des axes x et y [5] 23
Figure 21 - Photographie de la première caméra utilisée pour le projet [5] 24
Figure 22 - Visuel du menu principal du programme de Monsieur Baron [5] 25
Figure 23 - Visuel de la caméra lors de l'alignement du fil en x (à gauche) et en y (à droite) [5] 26
Figure 24 - Photographie de la seconde installation de l'instrument 27
Figure 25 - Cartes des intensités maximales (Figure Matlab 1) et des références (Figure Matlab 2) lors du scan en x 32
Figure 26 - Cartes des intensités maximales (Figure Matlab 3) et des références (Figure Matlab 4) lors du scan en y 32
Figure 27 - Pentes calculées selon x (Figure Matlab 5) et selon y (Figure Matlab 6) 33

Figure 28 - Carte du quatrième terme de Zernike générée 36
Figure 29 - Carte générée à partir de polynômes de Legendre 37
Figure 30 - Fenêtre montrant la surface générée sans masque, avec un masque circulaire et avec un masque rectangulaire 38
Figure 31 - Intégration depuis le pixel rouge le long des lignes vertes, puis le long des lignes bleues en ne passant pas dans les zones jaunes 38
Figure 32 - Intégration trapézoïdale de la surface du quatrième terme de Zernike 40
Figure 33 - Intégration trapézoïdale de la surface générée par des polynômes de Legendre 41
Figure 34 - Carte d'erreurs de la carte de focus reconstruite sans bruit 42
Figure 35 - Carte d'erreurs de la carte de focus reconstruite avec bruit 42
Figure 36 - Carte d'erreurs de la carte de polynômes de Legendre reconstruite sans bruit 43
Figure 37 - Carte d'erreurs de la carte de polynômes de Legendre reconstruite avec bruit 43
Figure 38 - Représentation de la différence d'axes entre Matlab et notre projet 47
Figure 39 - Surface du miroir plat reconstruite 48
Figure 40 - Surface du miroir plat mesuré à l'interféromètre 49
Figure 41 - Surface du miroir concave reconstruit 49
Figure 42 - Surface du miroir concave mesuré à la 3D 50
Figure 43 - Surface du miroir concave reconstruit en ayant retiré la reconstruction du miroir plat comme référence 50
Figure 44 - Surface du miroir concave reconstruite sans le facteur -1 51
Figure 45 - Fenêtre de l'application GUIDE lors du démarrage de la réalisation d'une nouvelle interface 54
Figure 46 - Première fenêtre du programme liée au script "Octopus.m" 54
Figure 47 - Message de confirmation de la manipulation à effectuer 55
Figure 48 - Fenêtre du programme liée au script "Ouvrir.m" 56
Figure 49 - Fenêtre du programme liée au script "Parametres.m" 56
Figure 50 - Fenêtre du programme liée au script "Centre_de_masse.m" 57
Figure 51 - Fenêtre du programme liée au script "Pentes.m" 58
Figure 52 - Image des pentes en x affichée dans NDA, ensuite masquée et finalement avec les 10 premiers termes de Zernike retirés 59
Figure 53 - Fenêtre du programme liée au script "Ouvrir2.m" 60
Figure 54 - Fenêtre du programme liée au script "Pentes2.m" 61
Figure 55 - Fenêtre du programme lié au script "Surface_Finale.m" 62
Figure 56 - Évolution de la carte de la surface avant et après le masquage et retrait des 10 premiers polynômes de Zernike sur NDA 62
Figure 57 - Cartes des hautes fréquences du miroir plat par interférométrie (gauche) et déflectométrie infrarouge (droite) 63
Figure 58 - Cartes des hautes fréquences du miroir concave par 3D (gauche) et déflectométrie infrarouge (droite) 64
Figure 59 - Cartes des basses fréquences de la TWF sur le miroir en SiC par 3D (gauche) et déflectométrie infrarouge (droite) 64
Figure 60 - Cartes des hautes fréquences de la TWF du miroir en SiC par 3D (gauche) et déflectométrie infrarouge (droite) 65
Figure 61 - Erreurs de répétabilité du déplacement du fil en x [5] 65
Figure 62 - Résultats obtenus par Monsieur Bechet concernant les chemins des rayons incident la caméra [12] 72
Figure 63 – Zoom sur la carte des pentes x de la surface du quatrième terme de Zernike sans bruit 91
Figure 64 – Zoom sur la carte des pentes y de la surface du quatrième terme de Zernike avec bruit 91
Figure 65 – Zoom sur la carte des pentes x de la surface en polynôme de Legendre sans bruit 92

Figure 66 – Zoom sur la carte des pentes y de la surface en polynôme de Legendre avec bruit 92

Introduction

Dans le cadre du Master en Sciences Spatiales à l'Université de Liège, il est demandé à chaque étudiant de réaliser un mémoire afin de finaliser leur cursus académique. Je me suis tourné vers la société AMOS, située dans le Parc Scientifique de Liège, pour réaliser mon travail de fin d'études. Le sujet de ce travail est :

Élaboration d'un système de mesure de surface non réfléchissante dans le visible par déflectométrie infrarouge

AMOS (Advanced Mechanical and Optical Systems) est une société liégeoise qui produit des miroirs pour satellites et télescopes ainsi que des pièces mécaniques et des structures pour télescopes. La société est née des suites d'un besoin des Ateliers de la Meuse. La société était à la recherche de locaux propres pour le développement d'une cuve à vide. De plus, profitant du don du département de polissage optique de la part de l'Institut d'Astrophysique de Liège, Monsieur Bill Colin fonda la société AMOS (à l'époque : Ateliers Meuse Optique Spatiales) en 1983. Des premiers jours à aujourd'hui, AMOS s'est distinguée comme une des références mondiales en ce qui concerne la conception de miroirs pour l'optique spatiale notamment grâce à son expertise et son désir de vouloir pousser ses résultats au plus proche, voire au-delà, des attentes de ses clients. L'une des plus belles vitrines de la société est le développement complet d'un des télescopes ATS du Very Large Telescope situé au Chili.

Figure 1 - Un des ATS présents sur le site du VLT à Cerro Paranal au Chili (Crédits : Iztok Boncina/ESO)

Le sujet de mon mémoire se place directement dans les intérêts et besoins de la société AMOS en ce qui concerne la fabrication de miroirs. En effet, lors de sa fabrication, un miroir passe par plusieurs étapes qui consistent à le convertir de la matière brute achetée à un sous-traitant vers un miroir fabriqué

avec une précision de l'ordre du nanomètre afin d'obtenir les meilleures spécifications possibles. Au cours de ce processus, le miroir doit être mesuré à de multiples reprises afin de contrôler l'évolution de sa forme. À cet effet, AMOS possède plusieurs instruments de mesure. Le premier est la machine 3D, qui à l'aide d'un palpeur, permet la mesure de la surface du miroir alors qu'il n'est pas encore réfléchissant. Ensuite, afin d'accélérer la procédure lorsque le miroir est devenu réfléchissant, celui-ci est mesuré à l'aide d'un interféromètre dans le visible. Alors que la machine 3D peut prendre plus de dix heures afin de mesurer un miroir de taille moyenne à l'aide de 20.000 points de palpage, un interféromètre ne prendra que quelques minutes pour mesurer ce même miroir.

Cependant, cette procédure est appliquée uniquement pour des miroirs allant jusqu'à une taille de l'ordre du mètre, car la machine 3D ne permet pas d'en accueillir de plus grands. AMOS étant capable de produire des miroirs d'une taille allant jusqu'à 4 mètres de diamètre, cela pose donc un problème de mesure pour les plus grands au début de leur conception. À l'heure actuelle, les opérateurs de chez AMOS sont contraints de travailler avec un profilomètre afin de faire évoluer ce type de miroirs de la pièce brute vers une pièce réfléchissante. Ce n'est qu'à partir de ce moment qu'ils ont l'occasion de réaliser les premières mesures concrètes du miroir et de lancer les calculs de temps de travail des robots sur la surface.

Afin de pallier à cette déconvenue technique, la société AMOS m'a proposé de participer à l'élaboration d'un système de mesure permettant l'obtention de cartes de la surface de ce type de miroirs dans les premières heures de leur conception. Pour réaliser cela, Monsieur Fabrice Wolfs m'a expliqué qu'il serait intéressant pour la société AMOS de réaliser un dispositif de mesure par déflectométrie infrarouge. Cette technique de mesure possède plusieurs intérêts. Le premier est que la déflectométrie, comparée à l'interférométrie, ne nécessite pas l'achat de références pour mesurer un miroir ce qui diminue drastiquement le coût financier de cette technique. D'autre part, la mesure dans l'infrarouge permet la mesure d'un miroir non réfléchissant dans le visible ce qui est très intéressant dans le cadre de la problématique que rencontre la société AMOS.

Plusieurs projets de ce type ont déjà été réalisés à travers le monde par différentes équipes de scientifiques et ingénieurs. Le projet qui fait le plus parler de lui est celui de la machine SLOTS (Scanning Longwave Optical Test System) développée par une équipe de scientifiques de l'Université d'Arizona aux États-Unis. Petit frère du projet SCOTS (Software Configurable Optical Test System) qui mesure des surfaces par déflectométrie dans le visible, SLOTS les mesure dans l'infrarouge en utilisant comme source un fil chauffé qui scanne la surface à mesurer dans deux directions orthogonales [1]. En termes de résultats, l'équipe de l'Université d'Arizona a obtenu une précision de 31 nm RMS sur un miroir sphérique de 4 pouces [2] et de 0,69 µm RMS sur un miroir primaire hors axe de 4,2 m de diamètre [3]. De son côté, la société AMOS a aussi développé en 2013-2014 un prototype de mesure par déflectométrie infrarouge afin d'évaluer l'intérêt de cette technique. Pour ce projet, les équipes d'AMOS ont utilisé une plaque chauffante partiellement masquée comme source afin de réaliser un *phase shifting*. Les résultats obtenus par ce dispositif se sont révélés assez convaincants. Cependant, suite à des problèmes liés au contraste de l'image reçue par la caméra, l'instrument fut abandonné au profit de celui développé et présenté au travers de ce mémoire [4].

Le but de mon mémoire est donc, à la demande de la société AMOS, de réaliser un instrument de mesure comparable à celui fabriqué par l'équipe de chercheurs en Arizona, mais pour les grandes optiques uniquement. Pour m'aider dans cette tâche, je fus accompagné de Moïse Baron, employé chez AMOS comme métrologue et contrôleur qualité. Dans le cadre de son Bachelier en cours du soir sur l'automation à l'Université de Louvain, le travail de fin d'études de Monsieur Baron s'est tourné vers le développement technique et le fonctionnement de l'instrument qui va permettre l'acquisition d'images

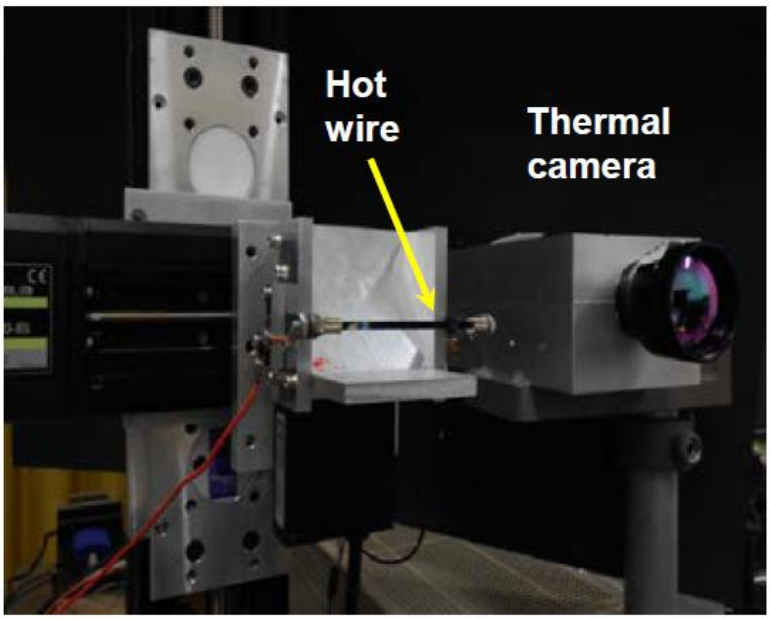


Figure 2 - Photographie du système de mesure SLOTS [1]

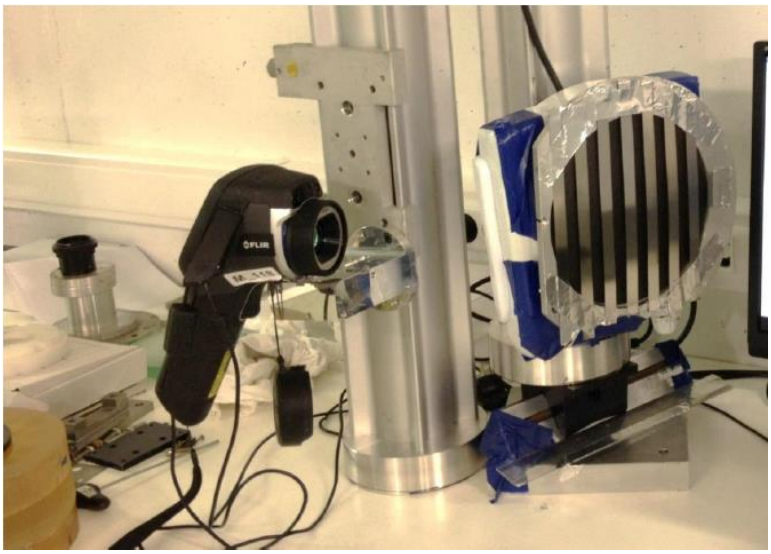


Figure 3 - Photographie du système de mesure par déflectométrie infrarouge développé par AMOS en 2014 [4]

du miroir pour une mesure en déflectométrie infrarouge. En ce qui me concerne, mon mémoire se tourne vers l'analyse de ces images au travers des points suivants :

- Réaliser une recherche d'articles concernant un ou plusieurs systèmes de mesures comparables et existants afin de définir les méthodes d'algorithmes à implémenter ;
- Participer au développement du robot cartésien implémenté par Monsieur Baron dans le cadre de son mémoire ;
- Réaliser un algorithme de reconstruction du miroir sur base des images obtenues ainsi qu'une interface permettant la manipulation de la reconstruction ;
- Monter l'instrument sur le banc de test afin de réaliser des mesures sur des miroirs de grandes tailles.

Au travers de ce mémoire, je tente de retransmettre l'entièreté du parcours que fût la réalisation de cette ample tâche durant près de 6 mois. Dans un premier temps, je présente toute la partie recherche

réalisée sur la déflectométrie, la déflectométrie infrarouge, les méthodes d'intégration de gradient ainsi que les travaux entrepris par l'équipe de scientifiques de l'Université d'Arizona afin de déterminer la meilleure manière de définir les algorithmes qui me furent utiles au cours de ce travail. Ensuite, j'introduis brièvement le dispositif conçu par Monsieur Baron pour réaliser l'acquisition des images en déflectométrie infrarouge. Le but de cette section est de lister le matériel à ma disposition pour effectuer mes calculs de mesure, les améliorations qui ont été nécessaires à mon travail ainsi que les contraintes qui m'ont été imposées. La partie la plus importante de mon mémoire se concentre sur la présentation de l'évolution de mon algorithme qui permet la reconstruction de miroirs mesurés avec Monsieur Baron. Cette partie est divisée en plusieurs sections, chacune présentant les modifications apportées, les améliorations qui en ont découlé, mais aussi les désavantages. Au fur et à mesure de mes explications, toutes ces sections sont accompagnées des lignes de code que j'ai implémentées. Mes codes se trouvent entièrement et commentés en annexe afin de favoriser une lecture globale pour le lecteur qui le désire. Enfin, je présente les améliorations, possibilités de modification ainsi que les points qui pourraient être intéressants d'investiguer dans une dernière section préalablement à la conclusion de ce mémoire.

Travaux et documentation sur la déflectométrie infrarouge

Dans cette section, je me consacre à présenter l'ensemble de mes recherches bibliographiques au sujet de la déflectométrie infrarouge et ses débouchés. Tout d'abord, je présente le principe de la déflectométrie dans le visible et dans l'infrarouge au travers de documents théoriques et de recherches. Ensuite, je développe plus en détail les projets et résultats réalisés par la société AMOS et l'équipe de scientifiques de l'Université d'Arizona. Enfin, en fin de section, j'introduis la structure du programme que j'ai conçu pour la société AMOS à partir de toutes ces informations.

Au fil de cette section, je m'attarde en profondeur uniquement sur l'analyse des images récoltées ainsi que la reconstruction de la surface d'un miroir mesuré par cette technique. L'aspect mécanique de l'instrument étant développé par Monsieur Baron, je n'introduis pas de recherches approfondies sur cette partie du projet dans ce mémoire. Cependant, je présente les choix de Monsieur Baron de manière synthétique dans la section suivante. Pour les plus intéressés, tous ces résultats sont évidemment référencés dans la section « Bibliographie » [5].

1. Les principes de la déflectométrie

Dans un premier temps, il est important de comprendre en quoi consiste le principe de mesure par déflectométrie avant de s'attarder sur le cas particulier de la déflectométrie dans le domaine de l'infrarouge. Tout au long de cette sous-section, il est aisé de constater que la déflectométrie est une méthode de mesure relativement simple à comprendre et que son utilisation peut être réalisée de multiples manières.

La déflectométrie consiste en l'analyse d'une source lumineuse réfléchie par une surface spéculaire. Ce principe de mesure permet la détermination des pentes de la surface analysée. Une technique de reconstruction de la surface, comme l'intégration d'un gradient, est donc nécessaire pour analyser la surface mesurée.

Pour réaliser une mesure déflectométrique, un projecteur (un écran par exemple) est généralement utilisé comme source lumineuse. Un motif qui correspond à une figure facilement analysable est affiché sur ce projecteur. Le plus utilisé est le motif de franges parallèles sinusoïdales noires et blanches. Cependant, des franges sphériques peuvent aussi être utilisées ainsi que tout autre motif, bien que les franges parallèles dominent dans le cadre de la déflectométrie. Comme un objet spéculaire réfléchit la lumière dans une seule direction, l'utilisation d'une source lumineuse de grande

taille est de rigueur. Les rayons lumineux réfléchis sont ensuite captés par une caméra afin d'être analysés [6]. Afin de permettre la reconstruction de la surface, la mesure des pentes dans plusieurs directions est donc nécessaire. Deux mesures orthogonales sont réalisées ; elles permettent une reconstruction par intégration du gradient obtenu. Cette analyse des pentes peut être réalisée de plusieurs manières qui sont présentées dans les paragraphes suivants.

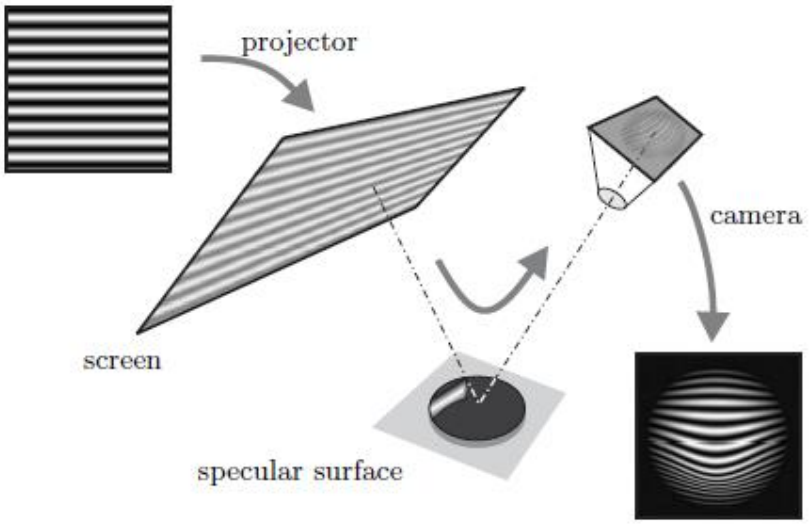


Figure 4 - Schéma du principe de base de la déflectométrie [6]

La première méthode consiste en l'évaluation de la phase du modèle sinusoïdal utilisé [6]. Chaque point de la surface réfléchit une partie de l'écran vers la pupille de la caméra. Le calcul de la phase réfléchie par la surface peut donc être effectué en chaque pixel de la caméra. Ensuite, grâce à la formule suivante :

$$\varphi = d \,.\, tan2\alpha \tag{1}$$

où φ représente la phase mesurée, d la distance entre l'écran et la surface et α la pente locale de la surface (voir Figure 5), la pente locale de la surface peut être mesurée.

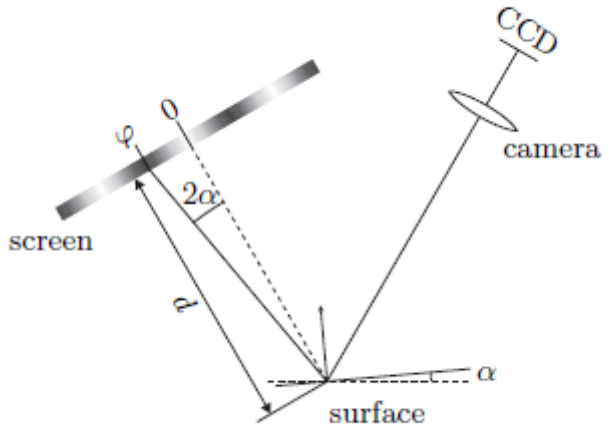


Figure 5 - Schéma du calcul de la phase φ [6]

Cependant, le contraste de l'image reçue décroit comparé à celui de l'image de départ. Il est donc utile d'évaluer le modèle sinusoïdal reçu. Pour résoudre ce problème, un décalage de phase dans l'image d'origine peut être effectué. En projetant N images décalées chacune d'une phase de $2\pi/N$, le modèle de la sinusoïdale reçu est caractérisé. Cette solution est appliquée à chaque pixel de la caméra séparément afin de déterminer comment chacun reçoit l'image réfléchie par la surface spéculaire. Sur base de cette référence et en prenant en compte que la connaissance de la frange dans laquelle le pixel se situe doit être déterminée, la phase φ de l'image est calculée ainsi que la pente angulaire résultante.

Une seconde méthode permet le calcul de la pente locale grâce à la connaissance des coordonnées de l'écran, de la surface spéculaire et de la caméra. Elle pourrait être nommée le « test inversé d'Hartmann » [7].

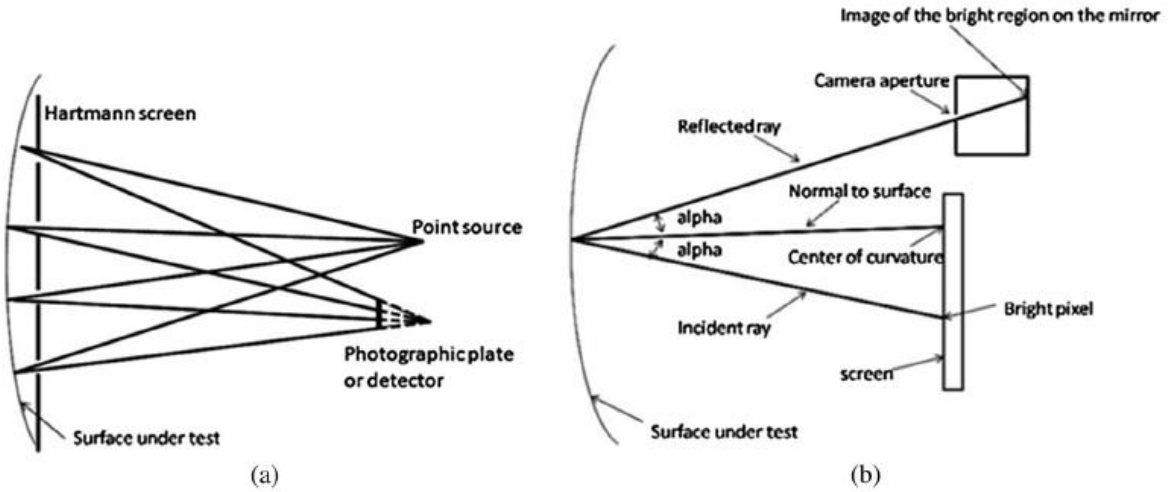


Figure 6 - Représentations du test d'Hartmann (a) et du "test inversé d'Hartmann" (b) [7]

Un test d'Hartmann est un test au cours duquel les pentes d'une surface sont mesurées. Pour cela, une source lumineuse ponctuelle est placée à une certaine distance de la surface et son image réfléchie est reçue par un détecteur ou une plaque photographique. La subtilité du test est qu'un écran d'Hartmann est placé juste devant la surface à mesurer. Il s'agit d'un masque percé de trous disposés sous la forme d'un quadrillage orthogonal. De cette manière, seuls les points non masqués apparaissent sur le détecteur. En analysant la disposition des points finaux et en la comparant à celle qui aurait correspondu à une surface parfaite, la pente en chaque point non masqué peut être calculée.

Dans le cas d'un « test inversé d'Hartmann », il faut considérer que le chemin des rayons lumineux est inverse. Ici, le détecteur est remplacé par l'écran lumineux de la déflectométrie et le point lumineux par l'ouverture de la caméra. En chaque point de la surface (appelé pixel de la surface durant la suite de ce mémoire), un seul pixel de l'écran est réfléchi vers l'ouverture de la caméra. La position de ce pixel de la surface correspond donc à la situation unique où l'angle d'incidence du pixel de l'écran et l'angle de réflexion vers la caméra sont égaux. Grâce à ces informations, la bissectrice de ces deux rayons, qui est normale à la surface, peut être déterminée ainsi que la pente qui en résulte.

Sur base de la connaissance de la position de chaque pixel de l'écran, de chaque pixel de la surface et de la position de la caméra, les pentes locales de chaque pixel de la surface peuvent être calculées à l'aide des équations suivantes :

$$w_x(x_m, y_m) = \frac{\frac{x_m - x_s}{d_{m2s}} + \frac{x_m - x_c}{d_{m2c}}}{\frac{z_{m2s} - W(x_m, y_m)}{d_{m2s}} + \frac{z_{m2c} - W(x_m, y_m)}{d_{m2c}}}$$
$$w_y(x_m, y_m) = \frac{\frac{y_m - y_s}{d_{m2s}} + \frac{y_m - y_c}{d_{m2c}}}{\frac{z_{m2s} - W(x_m, y_m)}{d_{m2s}} + \frac{z_{m2c} - W(x_m, y_m)}{d_{m2c}}} \quad (2)$$

avec w_x et w_y les pentes locales aux coordonnées (x_m, y_m), x_m et y_m les coordonnées du pixel de la surface observée, x_s et y_s les coordonnées du pixel de l'écran réfléchi, x_c et y_c les coordonnées de l'ouverture de la caméra, d_{m2s} la distance entre la surface et l'écran, d_{m2c} la distance entre la surface et la caméra, z_{m2s} la distance dans la direction Z entre la surface et l'écran, z_{m2c} la distance dans la direction Z entre la surface et la caméra et $W(x_m, y_m)$ la hauteur de la surface aux coordonnées (x_m, y_m). Tous ces termes sont représentés sur le schéma de la Figure 7 [2].

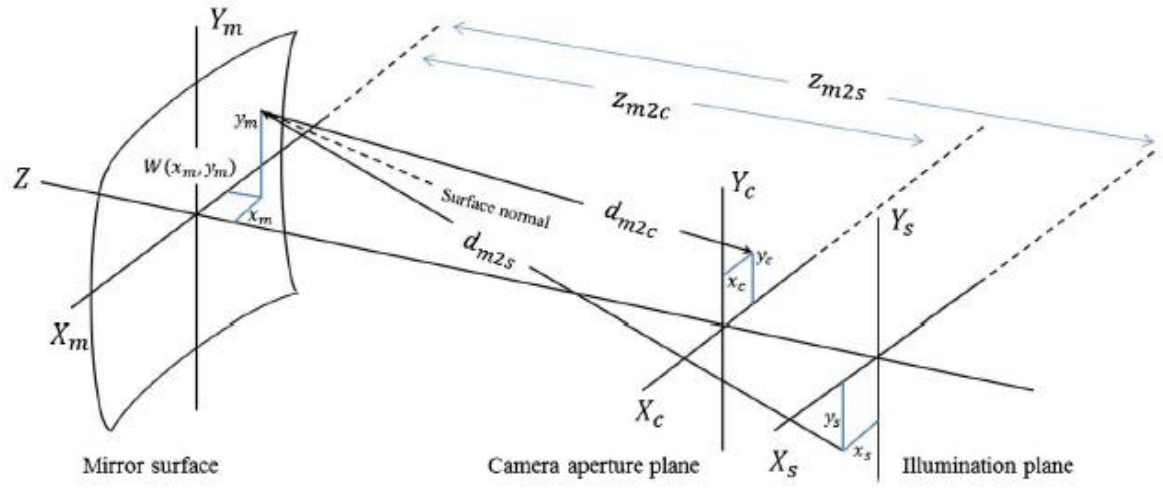


Figure 7 - Représentation de la configuration du problème et des termes sous-jacents [2]

Évidemment, comme expliqué dans un des paragraphes précédents, la mesure doit être effectuée dans deux directions orthogonales afin de permettre la reconstruction de la surface sur base du gradient de la surface observée. Cette affirmation est autant applicable au calcul de la phase et du modèle sinusoïdal reçu par la caméra dans la première méthode qu'au calcul de la pente sur base du « test inversé d'Hartmann ».

À l'heure actuelle, la déflectométrie représente un adversaire de plus en plus sérieux de l'interférométrie [8]. En effet, cette dernière devient de plus en plus inflexible et coûteuse notamment lorsqu'il s'agit de mesurer des surfaces *freeform* et qu'il est nécessaire d'acheter ou de fabriquer des CGHs de plus en plus complexes. D'autre part, cette technique requiert beaucoup de précautions face au bruit cohérent, mais aussi face aux vibrations et aux fluctuations de température pour ne citer qu'eux.

Le monde scientifique a donc mis au point une technique de mesure qui permet de contrer la plupart des soucis de l'interférométrie. Il a donc développé la déflectométrie, une technique qui permet la mesure directe de la déflexion de la lumière et qui est très peu coûteuse car elle ne requiert pas l'achat de référence pour ces mesures. De plus, comme la déflectométrie est soutenue entièrement par une approche géométrique du problème, cette technique est donc incohérente étant donné qu'elle ne dépend pas de la nature la lumière.

Cependant, la déflectométrie possède aussi ses inconvénients. Comme expliqué précédemment, la déflectométrie mesure la pente de la surface et non la surface elle-même. Ce détail implique les erreurs voire la perte d'informations sur les basses fréquences. Sur un champ de 100 mm, la précision de la déflectométrie est passée de plusieurs microns à quelques-uns au fil des innovations technologiques, mais reste très largement supérieure à la précision de quelques nanomètres obtenue par interférométrie. La déflectométrie risque donc, dans les années futures, de devenir un challenger de taille pour l'interférométrie.

2. La déflectométrie infrarouge

Au cours de la sous-section précédente, bien que cette technique soit plutôt simple à comprendre et de plus en plus précise, nous avons pu observer que la déflectométrie est entièrement dépendante de la réflectivité des surfaces spéculaires qu'elle mesure [9]. Cependant, la réflectivité dépend de la rugosité de la surface. Une surface trop rugueuse engendre une diffusion trop importante de la lumière et une perte importante de contraste sur l'image prise par la caméra.

Dans les entreprises de fabrication de miroirs pour satellites et télescopes comme la société AMOS, la conception de miroirs de grande taille est une des activités phares. Cependant, en début de fabrication, les pièces qui deviendront des miroirs sont d'une grande rugosité. Les sociétés sont donc confrontées au problème de ne pas pouvoir mesurer les grandes pièces ni à l'aide d'une machine de palpage 3D en raison de leur dimension, ni à l'aide d'un interféromètre en raison de leur faible réflectivité dans le visible. Pour pallier à ce problème, une solution a été trouvée. En effet, comme expliqué précédemment, la réflectivité dépend de la rugosité. Cependant, pour être plus précis, la rugosité dépend en réalité du ratio entre la rugosité et la longueur d'onde de la lumière incidente. Ainsi, en augmentant la taille de la longueur d'onde de cette lumière, une surface rugueuse diffuse moins la lumière et le contraste sur l'image finale est plus élevé. C'est de ce principe qu'est née la déflectométrie infrarouge.

Plusieurs méthodes ont été développées au fil des années pour adapter la source de lumière qu'est l'écran dans le visible vers une source de lumière infrarouge. Elles peuvent être divisées en deux grandes catégories : les modèles statiques et les modèles dynamiques. La différence entre eux vient de la possibilité de modifier et moduler le modèle au fil du temps dans les modèles dynamiques ce qui n'est pas le cas dans les modèles statiques. Ces derniers sont des modèles qui sont donc déplacés et pivotés mécaniquement afin de modifier l'image réfléchie par la surface spéculaire.

Un premier modèle statique est l'utilisation d'un écran codé avec différentes émissivités dans l'infrarouge. Une plaque est entièrement chauffée pour émettre de la lumière dans l'infrarouge. Cependant, cette plaque est partiellement masquée par des bandes métalliques. Le but de ce montage est d'utiliser l'émissivité quasi nulle du métal pour créer un écran avec des bandes émettant de la lumière et d'autres non ressemblant à un code-barre. Cet écran est utilisé comme un écran pour la déflectométrie dans le visible. Il est déplacé parallèlement à ces bandes pour évaluer les pentes dans une direction. Ensuite, l'écran est pivoté afin de permettre la mesure des pentes dans le sens orthogonal au premier. C'est ce modèle de source statique qui fut utilisé par les équipes de chez AMOS en 2013-2014 pour réaliser leur prototype de déflectométrie infrarouge [4].

Un second modèle statique est l'utilisation d'un fil chauffé par un courant le traversant comme source de lumière infrarouge. Dans ce cas, le fil chauffé est déplacé d'un bout à l'autre du miroir dans le sens de la pente que l'on mesure afin de couvrir tout le miroir. Cette méthode nécessite l'acquisition de

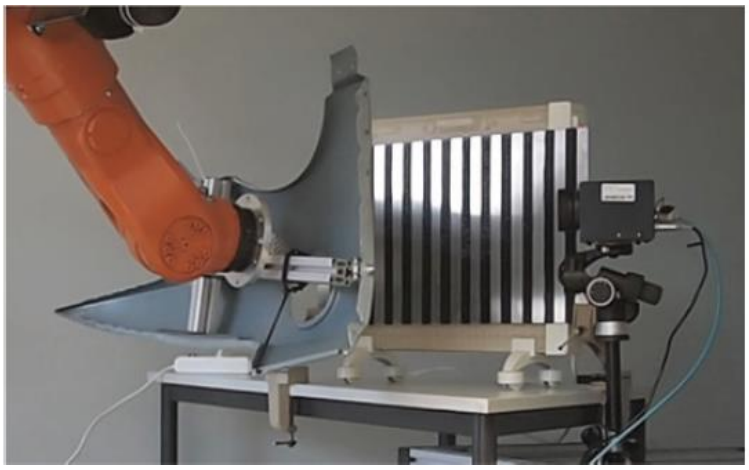


Figure 8 - Photographie d'un dispositif de mesure par déflectométrie infrarouge à l'aide d'un écran codé [9]

nombreuses images pour caractériser le déplacement de l'image du fil réfléchi par le détecteur de la caméra. Sur base de la connaissance de la position du fil sur l'image et de sa position réelle, il est possible de déterminer la pente sur le pixel de la surface impacté. Il est donc important de connaître avec précision la position du fil à chaque acquisition. Pour déterminer cela, le calcul du centre de masse par rapport à la position du fil et à son intensité à chaque acquisition est réalisé. Le projet SLOTS développé par l'Université d'Arizona utilise cette méthode pour mesurer des surfaces spéculaires en infrarouge [1, 2, 10].

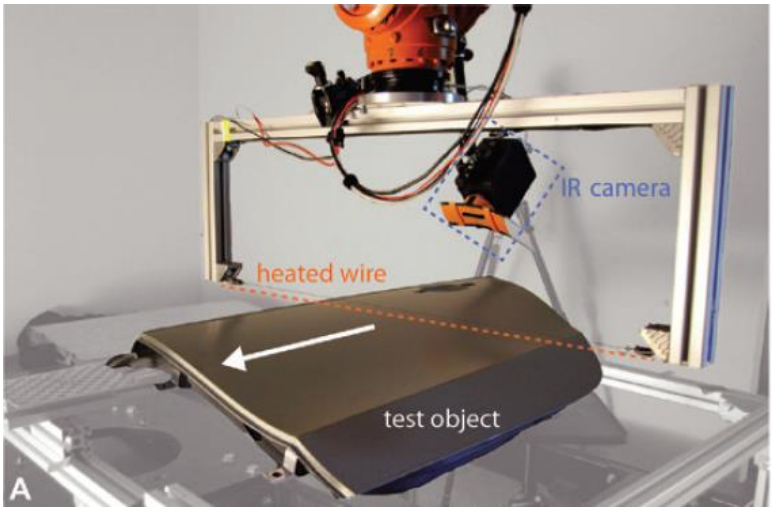


Figure 9 - Photographie d'un dispositif de déflectométrie infrarouge à l'aide d'un fil chauffé [9]

Comme détaillé précédemment, il existe une deuxième famille de modèles qui permettent l'émission de lumière infrarouge : les modèles dynamiques. Un de ceux-ci consiste en une plaque sur laquelle est disposé un quadrillage de résistances. Grâce au contrôle du courant passant par chaque résistance, il est possible de choisir le motif qui sera affiché sur la plaque : lignes, formes géométriques voire même des formes plus complexes comme des lettres de l'alphabet. Le principal défi de ce type de modèle est d'éviter la propagation de la chaleur du matériel qui soutient le quadrillage ainsi que le refroidissement des résistances lorsqu'elles viennent de s'éteindre (lorsque le courant ne les traverse plus et donc qu'elles n'émettent plus de lumière dans l'infrarouge).

Le dernier modèle qui est présenté dans cette section est le modèle qui utilise un écran chauffé par un laser. Un laser est pointé sur un écran situé à proximité de la surface à mesurer. Le laser chauffe

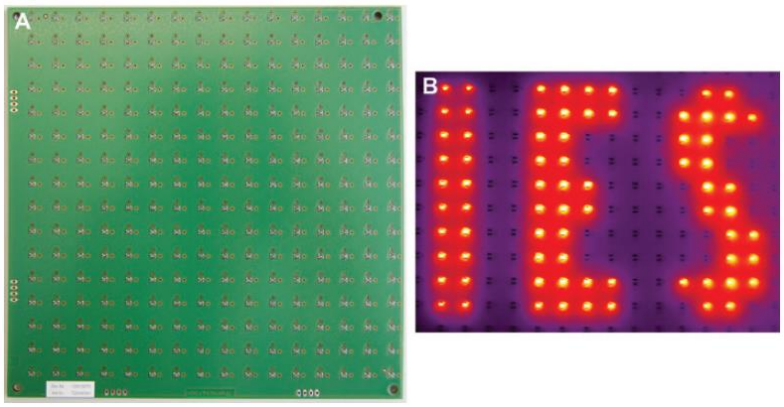


Figure 10 - Photographie de l'écran couvert d'un quadrillage de résistances dans le visible (A) et dans l'infrarouge avec plusieurs résistances "allumées" (B) [9]

la surface et la zone de surface chauffée émet une lumière dans l'infrarouge qui va pouvoir se réfléchir sur les surfaces dont la rugosité est élevée. Ainsi, en déplaçant le laser sur l'écran, il est possible de faire apparaître des motifs au choix comme dans le cas du quadrillage de résistances. Il est cependant nécessaire de faire attention lors de l'utilisation de cette méthode pour deux raisons. La première est que la création d'un motif complet sur un écran n'est pas instantanée ; le temps de finir le motif, ses premiers traits sont déjà plus faibles en intensité. Deuxièmement, la chaleur se diffuse autour de la zone chauffée par le laser ce qui modifie le motif créé. Cette méthode nécessite donc que la création du motif soit filmée et que la position du laser à chaque image soit caractérisée afin de reconstruire le motif complet comme illustré dans la Figure 11.

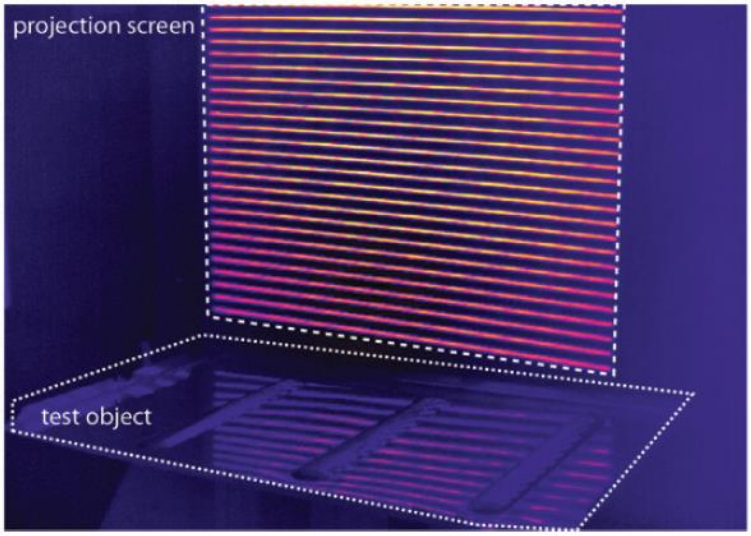


Figure 11 - Photographie montrant la reconstitution du motif d'une déflectométrie infrarouge à l'aide d'un écran chauffé par laser [9]

3. L'intégration d'un gradient

La déflectométrie est une mesure de surface qui donne les cartes de pente de la surface spéculaire mesurée. Il est donc nécessaire de réaliser une intégration du gradient obtenu afin de reconstruire la carte de la surface. Dans cette sous-section, j'introduis le modèle d'intégration de gradient que j'ai choisi pour la réalisation du projet de mon mémoire.

Pour effectuer ce choix, j'ai décidé de me baser sur les travaux de l'équipe de scientifiques de l'Université d'Arizona puisqu'il m'a été demandé de participer à un projet qui développe un instrument similaire au leur. Dans un des papiers publiés par cette équipe [2], il est expliqué que l'intégration de Southwell a été choisie pour réaliser les reconstructions de surface sur base de gradients. Mon choix s'est donc également porté sur cette option.

L'intégration de Southwell est une méthode itérative où la hauteur de chaque pixel de la surface est calculée en fonction des quatre points qui l'entourent comme représenté dans la Figure 12 [11].

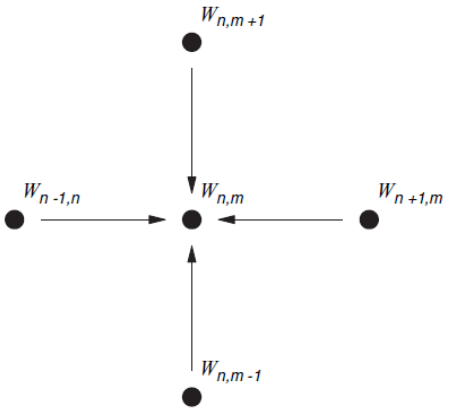


Figure 12 - Représentation des voisins du pixel surface qui permettent sa reconstruction [11]

Effectivement, chaque pixel de surface de coordonnées (n, m) peut être mesuré à l'aide des équations suivantes :

$$\begin{aligned} W_{n,m} &= W_{n-1,m} + \frac{d}{2}[w_x(n-1,m) + w_x(n,m)] \\ W_{n,m} &= W_{n+1,m} - \frac{d}{2}[w_x(n+1,m) + w_x(n,m)] \\ W_{n,m} &= W_{n,m-1} + \frac{d}{2}[w_y(n,m-1) + w_y(n,m)] \\ W_{n,m} &= W_{n,m+1} - \frac{d}{2}[w_y(n,m+1) + w_y(n,m)] \end{aligned} \tag{3}$$

où W représente la hauteur du pixel de la surface, d la distance entre le centre de deux pixels de la surface (pour des pixels représentant la même surface, il s'agit de la taille du pixel de la surface) et w la pente locale.

De ces quatre équations, il est possible de créer une équation qui en est la moyenne pondérée et qui permet d'évaluer $W_{n,m}$:

$$\begin{aligned} W_{n,m} = &\frac{(\sigma_{n-1,m}W_{n-1,m} + \sigma_{n+1,m}W_{n+1,m} + \sigma_{n,m-1}W_{n,m-1} + \sigma_{n,m+1}W_{n,m+1})}{(\sigma_{n-1,m} + \sigma_{n+1,m} + \sigma_{n,m-1} + \sigma_{n,m+1})} \\ &+ \frac{(\sigma_{n-1,m}S^x_{n-1,m} - \sigma_{n+1,m}S^x_{n+1,m} + \sigma_{n,m-1}S^y_{n,m-1} - \sigma_{n,m+1}S^y_{n,m-1})}{(\sigma_{n-1,m} + \sigma_{n+1,m} + \sigma_{n,m-1} + \sigma_{n,m+1})} \end{aligned} \tag{4}$$

avec $\sigma_{n,m}$ qui sert à la pondération et vaut 1 dans le cas où le pixel surface (n,m) possède une pente locale mesurée et $S^x_{n-1,m}$ défini comme

$$S^{x}_{n-1,m} = \frac{d}{2}[w_x(n-1,m) + w_x(n,m)] \tag{5}$$

Comme décrit précédemment, l'intégration de Southwell est une méthode itérative. Celle-ci doit être répétée un nombre de fois égal au nombre de pixels de la surface couverte par l'intégration. Le but de cette itération est de permettre la minimisation d'erreurs dues au calcul de la pente locale par exemple.

Il est rapide de constater que pour appliquer la méthode de Southwell, il est nécessaire d'avoir obtenu une première carte représentant la surface reconstruite sur base du gradient. C'est de cette première carte qu'est calculée la première itération avec l'intégration de Southwell. Dans le livre « Optical Shop Testing » de Daniel Malacara [11] où j'ai obtenu les informations concernant l'intégration de Southwell, j'ai découvert une méthode d'intégration basique qui permet une première reconstruction de la surface sur base du gradient uniquement. Il s'agit de l'intégration trapézoïdale.

En repartant de deux des quatre équations présentées dans les Équations (3), à savoir

$$\begin{aligned} W_{n,m} &= W_{n-1,m} + \frac{d}{2}[w_x(n-1,m) + w_x(n,m)] \\ W_{n,m} &= W_{n,m-1} + \frac{d}{2}[w_y(n,m-1) + w_y(n,m)] \end{aligned} \tag{6}$$

il est possible d'écrire l'équation suivante qui qualifie l'intégration non plus le long des axes x ou y, mais selon une ligne diagonale :

$$\begin{aligned} W_{n,m} = W_{n-1,m-1} + \frac{d}{2}[&\big(w_x(n-1,m-1) + w_x(n,m-1)\big) \\ &+ \big(w_y(n,m-1) + w_y(n,m)\big)] \end{aligned} \tag{7}$$

Grâce à l'Équation (7) et en prenant une hauteur arbitraire pour un des pixels de la surface, il est possible de dérouler cette intégration de point en point jusqu'à couvrir l'entièreté de la surface. Pour que la précision de cette intégration soit optimale, il est conseillé de l'appliquer uniquement sur des surfaces qui possèdent du tilt, du focus ou de l'astigmatisme. De plus, afin que l'intégration soit correcte, il est conseillé dans le livre de Monsieur Malacara d'intégrer la surface dans le chemin opposé également afin de minimiser les erreurs de propagation. Cependant, comme cela est présenté plus loin, ces deux conseils d'optimisation n'ont pas été pris en compte, car le but de l'utilisation de cette intégration dans ce projet est d'obtenir une première carte qui sera corrigée par l'intégration de Southwell et non une carte la plus correcte possible.

4. Les travaux réalisés par AMOS

Au cours des années 2013-2014, la société AMOS a fabriqué un prototype de mesure par déflectométrie infrarouge. Le but de ce test était de voir s'il était intéressant pour la compagnie de développer un instrument de mesure plus complexe pour aider à la manufacture de ses miroirs de grande taille.

Pour ce prototype, l'équipe de Monsieur Fabrice Wolfs a donc imaginé un instrument utilisant le modèle de l'écran codé pour mesurer des miroirs par déflectométrie infrarouge. Afin de capturer l'image réfléchie par la surface spéculaire, ils ont utilisé dans un premier temps une caméra infrarouge de 160 x 120 pixels, puis une caméra de 640 x 480 pixels.

Le montage initial était de petite envergure afin de tester si le modèle de franges émis par la plaque chauffante était bien réfléchi par une surface. Des premiers tests d'algorithme de mesure furent effectués et ceux-ci se révélèrent fructueux. On peut observer dans la Figure 15 que les résultats obtenus par déflectométrie sont proches de ceux réalisés par une mesure 3D (CMM). De plus, comparés aux résultats de la mesure 3D, les résultats du dispositif montrent une carte de la surface moins pixelisée et plus nette visuellement. Cela vient du fait que la déflectométrie couvre l'entièreté de la surface mesurée alors qu'une mesure 3D voit son palpeur se déplacer de proche en proche sur la surface. De plus, les résultats obtenus après avoir retiré les premiers polynômes de Zernike sont eux aussi très bons. Cela a encouragé la société AMOS à investiguer plus en profondeur la piste de la déflectométrie infrarouge.

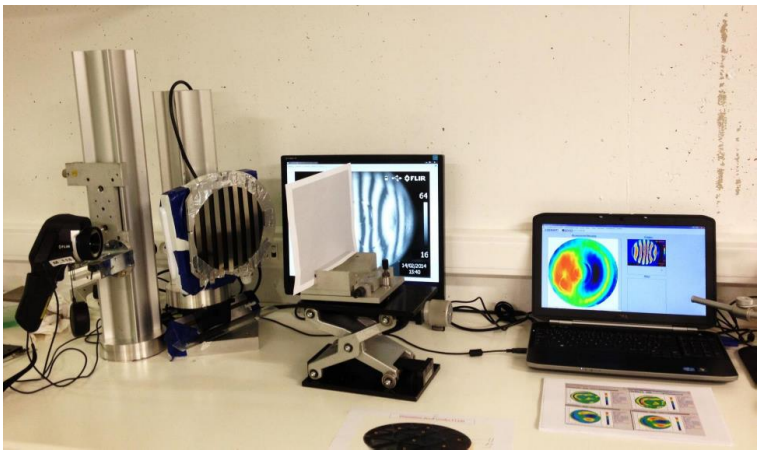


Figure 13 - Photographie du premier dispositif créé par l'équipe d'AMOS [4]

Suite à ces premiers résultats concluants, le dispositif fut automatisé. La plaque chauffante fut fixée sur un bras robotique contrôlée par un ordinateur. Pour les tests suivants, un miroir hexagonal fut mesuré. Celui-ci possédant un rayon de courbure de plusieurs mètres, la caméra dut être placée à cette distance de rayon de courbure pour pouvoir focaliser l'image de la grille chauffante sur la surface. Afin de régler le problème de la distance entre les différents instruments du dispositif, une lentille fut installée devant la caméra pour lui servir de zoom optique.

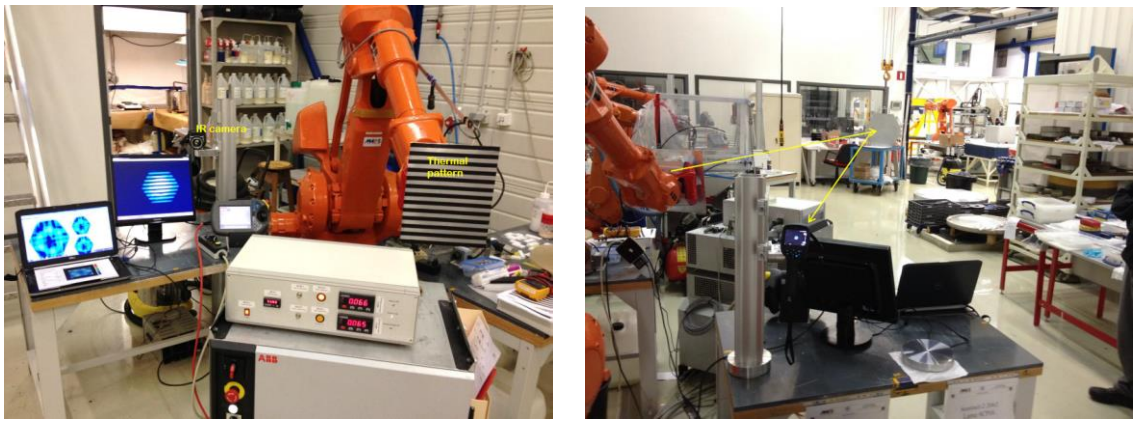


Figure 14 - Photographies du second dispositif créé par l'équipe d'AMOS [4]

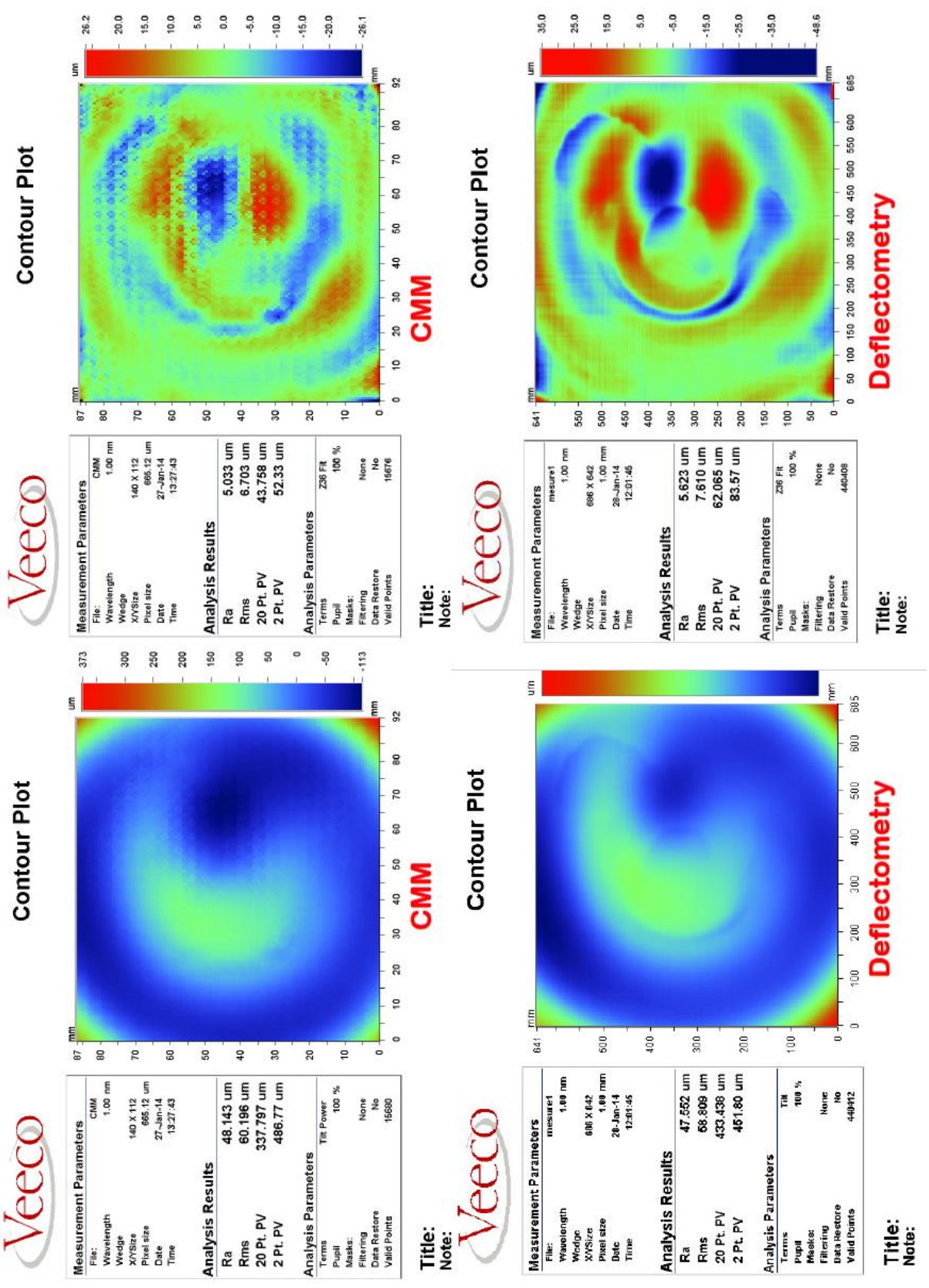


Figure 15 - Premiers résultats obtenus par le dispositif créé par AMOS. À gauche : comparatif des mesures de la surface par mesure 3D et déflectométrie. À droite : comparaison des mêmes mesures après avoir retiré les premiers polynômes de Zernike. [4]

Plusieurs mesures furent réalisées à l'aide de ce dispositif. Cependant, suite à un manque de contraste dans l'image résultant de la distance entre les différents éléments du dispositif, les résultats obtenus par l'équipe d'AMOS furent de moins bonne qualité que lors du premier test. En effet, comme on peut le constater sur la Figure 16, bien que les valeurs numériques obtenues soient fort semblables, la qualité de l'image est beaucoup moins nette en raison de ce manque de contraste. L'instrument fut donc abandonné au profit de la machine de déflectométrie développée avec Monsieur Baron plusieurs années après ces premiers essais.

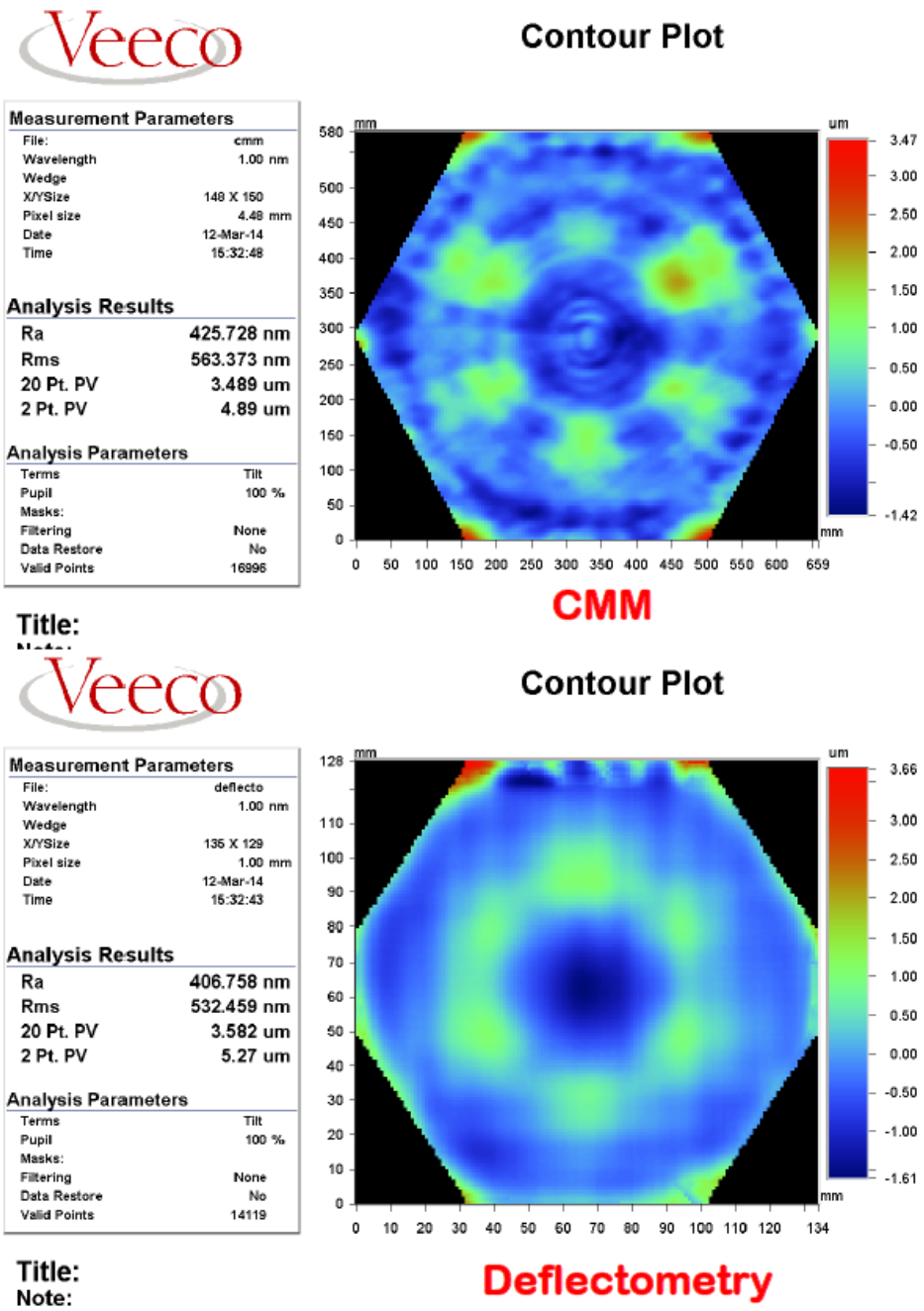


Figure 16 - Résultats de mesure de la surface du miroir obtenus par mesure 3D et par le second dispositif de mesure par déflectométrie infrarouge [4]

5. Le projet SLOTS

À l'Université d'Arizona, une équipe de scientifiques a mis au point un instrument de mesure par déflectométrie infrarouge dans la suite du développement de leur instrument SCOTS (Software Configurable Optical Test System). Cet appareil nommé SLOTS (Scanning Longwave Optical Test System) a pour but de mesurer des surfaces de plus haute rugosité et donc non réfléchissantes dans le visible [2].

Pour cet instrument, ils ont décidé de produire deux prototypes en utilisant le modèle du fil chauffé. Le premier dispose de deux niveaux linéaires alors que le second n'en possède qu'un seul ainsi qu'un axe de rotation pour le faire pivoter. Le fil choisi est en tungstène et est chauffé à 800°C lors du scan. Il va permettre une mesure de surface à moindre coût comparé à l'achat d'un interféromètre infrarouge. De plus, afin d'éviter des réflexions fantômes sur l'image réfléchie, une plaque métallique est fixée de manière inclinée à l'arrière du fil afin de négliger ses effets néfastes pour la mesure. Enfin, deux caméras de résolutions différentes sont utilisées dans chaque prototype : 320 x 240 pixels dans le premier cas et 640 x 480 pixels dans le second.

La procédure de mesure de la surface d'un miroir est identique pour les deux prototypes et se déroule en plusieurs étapes. La première consiste en l'alignement du système. Pour cela, l'opérateur qui réalise la mesure place la caméra au niveau du centre de courbure du miroir à mesurer. L'intérêt de cette démarche est de pouvoir résoudre l'image réfléchie par la surface. Ensuite, l'opérateur ajuste le miroir pour que celui-ci soit entièrement scanné par le fil. Enfin, les distances entre les différents éléments (caméra, fil et miroir) sont prises, car le projet SLOTS calcule les pentes locales du miroir grâce au « test inversé d'Hartmann ».

La seconde étape consiste en la collecte des données. Le fil chauffé scanne la surface du miroir dans deux directions orthogonales. La vitesse du fil est constante et la caméra recueille des images à des intervalles de temps fixés. Les images sont sauvegardées pour ensuite être analysées ensemble.

La troisième étape consiste à calculer la position exacte du fil lorsqu'il est réfléchi avec un maximum d'intensité pour chaque pixel de la surface. Pour cela, le calcul du centre de masse de la position est effectué à l'aide de l'équation suivante :

$$x_s = \frac{\sum_i x_i I_i}{\sum_i I_i} \tag{8}$$

où x_s correspond à la position pondérée du fil pour un pixel de la surface donné, x_i correspond à la position du fil chauffé lors de l'acquisition i et I_i l'intensité que mesure le pixel de la caméra qui représente le pixel de la surface donnée à la position i du fil.

La quatrième étape est le calcul de la pente sur base des mesures effectuées dans les étapes précédentes ainsi que le calcul de la position du fil pour chaque pixel miroir. Pour cela, le calcul se fait à l'aide des Équations (2). Cependant, comme on peut le constater dans ces formules, pour calculer la pente de chaque pixel de la surface, sa hauteur est nécessaire. Mais, le but du projet SLOTS étant justement de déterminer cette hauteur inconnue, il a été nécessaire de trouver une alternative à ces deux équations.

En fabrication optique, la hauteur W est généralement largement inférieure aux distances z_{m2s} et z_{m2c}. Dans le premier cas, on parle de l'ordre du micromètre voire millimètre et, dans le second cas, de plusieurs mètres. Cela étant noté, il est possible de faire une approximation des Équations (2) où les erreurs induites sont négligeables. Les équations simplifiées donnent les formules suivantes :

$$w_x(x_m, y_m) = \frac{1}{2}\left(\frac{x_m - x_s}{z_{m2s}} + \frac{x_m - x_c}{z_{m2c}}\right)$$
$$w_y(x_m, y_m) = \frac{1}{2}\left(\frac{y_m - y_s}{z_{m2s}} + \frac{y_m - y_c}{z_{m2c}}\right) \quad (9)$$

Comme les Équations (9) ne dépendent plus de la valeur W de chaque pixel de la surface, il est possible de déterminer les valeurs des pentes locales en chacun d'eux. L'équipe de scientifiques d'Arizona s'est basé sur ces Équations pour réaliser leur calcul de pentes.

La dernière étape de leur procédure consiste en la reconstruction de la surface par l'intégration du gradient obtenu à l'aide de l'intégration de Southwell.

Au fil du temps, le projet SLOTS a déjà abouti à plusieurs résultats. Un de ceux-ci concerne la mesure d'un miroir de 100 mm de diamètre, dont le rayon de courbure est de 1,9 m. L'équipe de scientifiques a décidé lors de la reconstruction de la surface de retirer les dix premiers polynômes de Zernike sur les cartes des pentes calculées avant d'intégrer le gradient. Ensuite, après la reconstruction, elle a décidé de retirer à nouveau les dix premiers polynômes de Zernike. Le but de cette manœuvre est de retirer toutes les basses fréquences de la carte de la surface reconstruite. Comme expliqué précédemment, la déflectométrie engendre des erreurs au niveau des basses fréquences. Grâce au retrait des dix premiers polynômes de Zernike, l'équipe en charge du projet SLOTS a pu comparer les hautes fréquences mesurées par déflectométrie infrarouge avec celles trouvées lors d'une mesure interférométrique. En soustrayant une carte à l'autre, les scientifiques ont obtenu la carte des différences. L'erreur sur la mesure de l'entièreté de la surface a été déterminée et vaut 31 nm RMS.

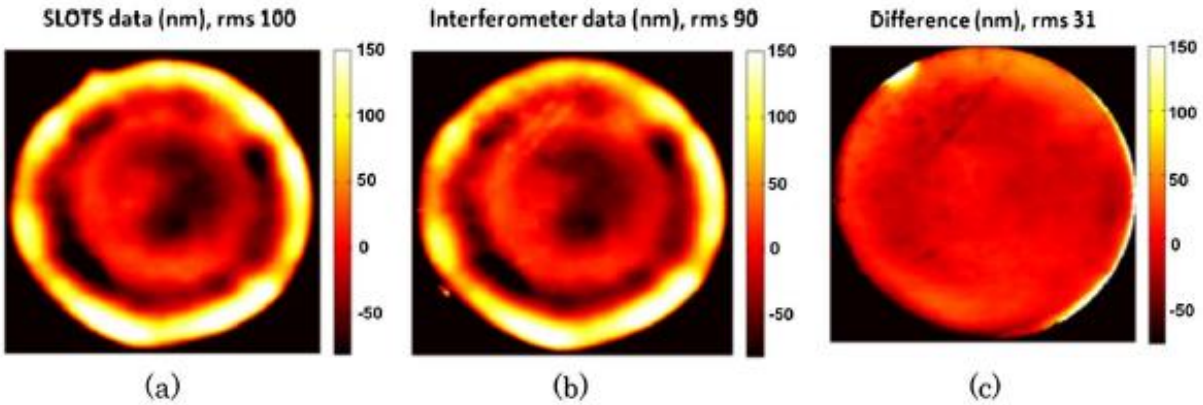


Figure 17 - Carte des surfaces obtenues par SLOTS (a), par un interféromètre (b) et la différence entre les deux cartes (c) [2]

6. La structure du programme

La théorie concernant la déflectométrie, l'intégration d'un gradient ainsi que les travaux de la société AMOS et de SLOTS étant présentés, il est temps d'aborder maintenant la structure du programme que j'ai pu réaliser au cours de mon projet.

L'instrument de déflectométrie étant calqué sur celui développé par l'équipe de l'Université d'Arizona, la procédure à suivre est déjà toute tracée. Cependant, le projet étant réparti entre Monsieur Baron et moi-même, il est important de commencer par délimiter le travail de chacun.

Dans le cadre de son travail de fin d'études, la manufacture ainsi que le fonctionnement de l'appareil ont été attribués à Monsieur Baron. Ainsi, si l'on compare à la procédure en 5 étapes définies par le projet SLOTS, Monsieur Baron devait prendre en charge les deux premières étapes : l'alignement et la collecte de données.

Dans le cadre de mon mémoire, il me restait les 3 derniers points à aborder : le calcul du centre de masse du fil pour chaque pixel de la surface, le calcul des pentes et l'intégration du gradient. Ces trois points sont ceux qui seront développés au travers de mes programmes et décrits dans ce mémoire.

Ces trois étapes étant nommées ainsi de manière globale, il semble néanmoins important de développer les différents aspects qui durent être abordés au cours de ma programmation. Voici la liste des étapes à programmer obligatoirement lors de mon travail :

- Lire les données sauvées par le programme de Monsieur Baron ;
- En extraire les informations concernant l'intensité de chaque pixel à chaque image ;
- Réaliser le calcul du centre de masse pour chaque pixel de la surface ;
- Définir les paramètres importants pour le calcul des pentes au travers de mon programme ou des données fournies par le programme de Monsieur Baron ;
- Calculer les pentes locales pour chaque pixel de la surface ;
- Réaliser un masque pour définir la zone à intégrer ;
- Intégrer le gradient obtenu avec l'intégration trapézoïdale ;
- Intégrer le gradient à l'aide de l'intégration de Southwell ;
- Afficher les résultats obtenus.

Ces différentes étapes forment la base du programme que j'ai défini. Au cours de l'avancée de mon travail, plusieurs autres aspects ont été programmés afin de rendre le programme plus ergonomique et agréable à utiliser. Parmi ces aspects on peut citer :

- La possibilité d'interagir avec certains paramètres importants du programme comme la position de la caméra ou la valeur du seuil de l'intensité à prendre en compte pour la mesure du centre de masse ;
- Le contrôle de l'avancée du programme au travers de l'affichage de certaines cartes ou messages d'alertes ;
- La possibilité du retrait des polynômes de Zernike pour analyser mes résultats comme l'équipe du projet SLOTS ;
- L'enregistrement des résultats obtenus en format lisible par des programmes de la société AMOS ;
- Une interface pour l'utilisation de mon programme.

L'analyse des lignes de code de mon programme est faite en deux parties : une première qui permet l'obtention des pentes depuis la lecture des données sauvées par Monsieur Baron, la seconde

qui correspond au masquage et à l'intégration complète de mon gradient ainsi qu'à sa précision. Ensuite, une analyse du programme complet est faite en deux temps : en premier, l'analyse du programme avec le calcul des pentes et de l'intégration combinés ainsi que les changements apportés et ensuite, la présentation de mon programme avec son interface et mes premiers résultats concrets.

L’instrument de déflectométrie infrarouge

Avant de rentrer dans le vif du sujet et la présentation du programme que j’ai développé, il est intéressant de réaliser une description de l’instrument de déflectométrie qui a été fabriqué dans le cadre de ce projet. Comme précisé à plusieurs reprises, cet instrument a été développé par Monsieur Moïse Baron dans le cadre de son travail de fin d’études pour l’obtention de son Bachelier en automation à l’Université de Louvain.

Le but de cette section est de montrer au lecteur toute la structure qui a été mise en place pour permettre la collecte de données par déflectométrie infrarouge. De plus, au travers de cette section, je présente les différents aspects de la conception de l’instrument ou du programme de Monsieur Baron pour lesquels je suis intervenu. Ces aspects peuvent être sous forme d’aide ou de requête pour le bon déroulement de nos deux travaux.

Je tiens tout de même à préciser que cette section est une approche globale de l’instrument et de son fonctionnement. Les détails techniques évoqués au travers des pages qui suivent sont ceux qui ont servi l’intérêt de mon travail. Pour plus de détails concernant tous les aspects techniques qui n’auront pas été évoqués, je vous invite chaudement à parcourir le travail écrit de Monsieur Baron [5].

1. Vision globale de l’instrument

L’instrument de déflectométrie développé au long de ce projet est un modèle où la source de lumière infrarouge est un fil chauffé. Dans ce cadre-là, plusieurs instruments sont nécessaires et obligatoires au bon fonctionnement de l’appareil : un fil chauffé, une caméra infrarouge et un ordinateur. Pour l’appareil de ce projet, voici la liste des éléments principaux qui le constituent :

- Un fil chauffé ;
- Une caméra infrarouge ;
- Un ordinateur ;
- Une plateforme CNC ;
- Un capteur de distance ;
- Un chariot élévateur.

La disposition des différents éléments est représentée dans les Figures 18 et 19.

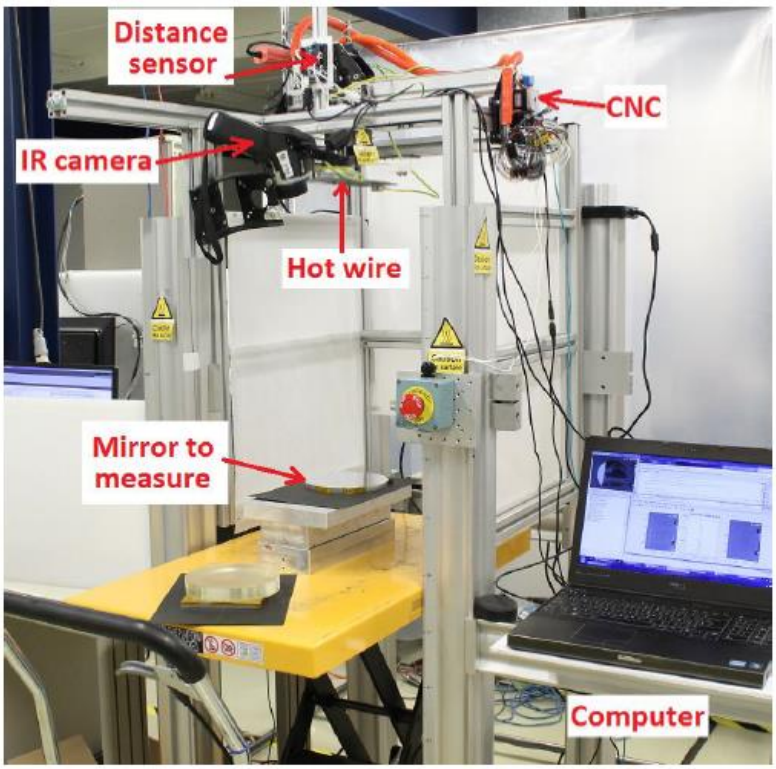


Figure 18 - Photographie globale du dispositif [5]

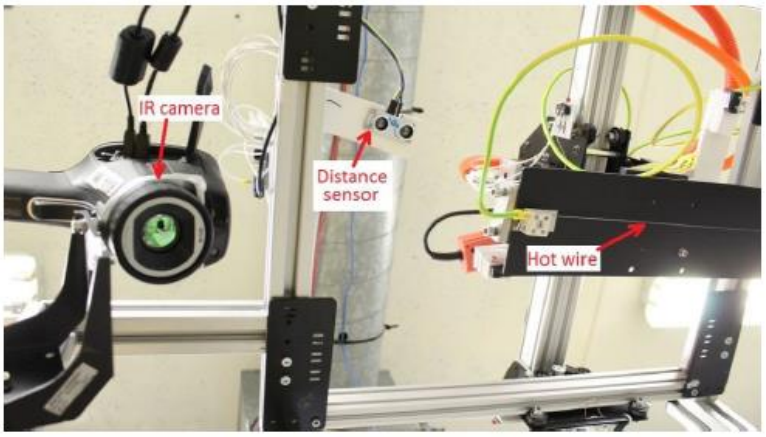


Figure 19 - Photographie de la partie supérieure du dispositif [5]

L'élément le plus important à retenir dans la disposition de l'instrument est que le fil chauffé et la caméra sont situés à la même hauteur sur la structure. Leur distance par rapport au miroir est donc égale. Cette information sera utile dans le cadre du calcul des pentes à l'aide des Équations (9).

2. Le scan du fil chauffé

Dans le cadre d'une mesure déflectométrique infrarouge avec le modèle choisi pour notre projet, il est nécessaire que le fil chauffé se déplace le long de deux axes orthogonaux afin de le scanner. Dans ce but, une plateforme CNC a été mise en place.

La plateforme est constituée de 3 moteurs pas-à-pas qui vont permettre la mise en mouvement du fil le long des axes x et y et sa rotation. Il est utile de noter que le mouvement du fil ne se fait pas à vitesse constante comme pour le projet SLOTS, mais bien avec un déplacement de pas-en-pas. Avec cette disposition, un capteur n'est pas nécessaire pour connaître la situation du fil à chaque acquisition puisque les images sont prises à chaque pas. Ainsi, en connaissant la taille du pas, son numéro et la position de départ du fil, il est aisé de situer le fil chauffé pour chaque acquisition effectuée.

La structure permet au fil de se déplacer le long des axes x et y sur une distance de 270 mm. La position initiale x du fil lors du scan en x est 0. Il en est de même pour la position y du fil lors du scan en y. Ainsi, à l'acquisition i, la position du fil sera donnée par le produit de i et de la taille du pas. Lors du développement de son programme, Monsieur Baron a ajouté une fonctionnalité qui permet le déplacement de la position de départ du fil. Ainsi, pour obtenir la position du fil à chaque image, il faut ajouter cette position de départ au produit précédemment calculé.

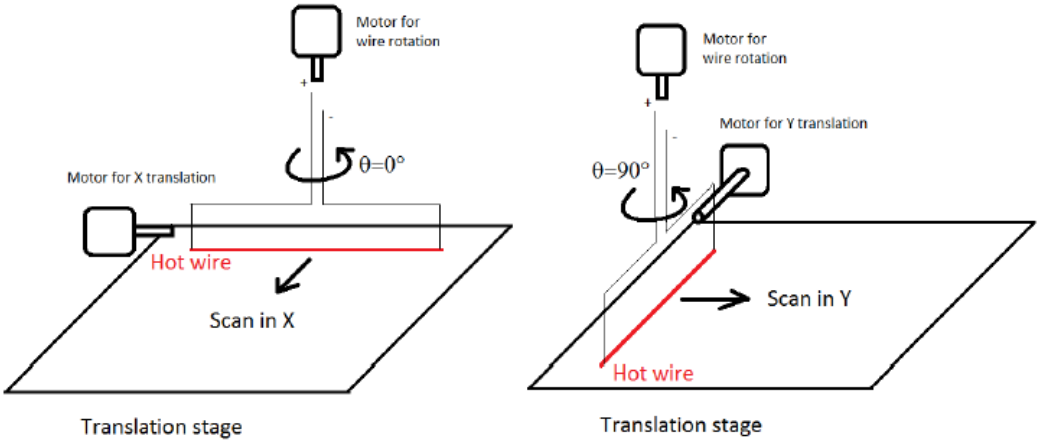


Figure 20 - Représentation du mouvement de scan le long des axes x et y [5]

La taille du pas était initialement d'une longueur de 1 mm. Suite à l'optimisation de l'instrument par Monsieur Baron, cette taille a varié. Cependant, cette information est référencée à chaque mesure dans un fichier fourni par l'instrument. Ce détail est utile à la compréhension de l'évolution de mon programme.

Le fil chauffé est fixé sur une plaque métallique comme on peut le constater dans la Figure 19. Cette plaque n'étant pas inclinée, il se peut qu'il y ait une présence de réflexions fantômes lors de la collecte des données et qu'elle constitue une source d'erreurs lors du calcul de centre de masse du fil.

3. La caméra infrarouge

Afin de permettre l'acquisition de données, l'utilisation d'une caméra infrarouge est nécessaire. Au cours du développement du projet, il a très vite été décidé que la caméra utilisée initialement serait remplacée par une caméra de plus haute résolution à long terme. Ne sachant pas la date à laquelle ce changement serait effectué, mon programme a été écrit de telle sorte que le changement de résolution n'impacte pas son déroulement.

La première caméra est une caméra infrarouge d'une résolution de 320 x 240 pixels. Elle réalise l'acquisition d'images en valeur de gris. De plus, l'échelle d'observation d'intensité de la caméra peut

être fixée manuellement et est employée à chaque acquisition afin que toutes les images utilisent la même échelle de valeurs de gris. Une variation de cette échelle aurait engendré des erreurs pour la mesure du centre de masse du fil qui se base sur l'évolution de l'intensité de chaque pixel.

Figure 21 - Photographie de la première caméra utilisée pour le projet [5]

La seconde caméra infrarouge possède une résolution de 640 x 480 pixels. Pour celle-ci, la fixation de l'échelle d'intensité se réalise numériquement à l'aide de différentes lignes de code qui ont été implémentées dans le programme de Monsieur Baron.

Au cours du développement du projet, le dispositif a subi deux configurations comme expliqué dans les sections suivantes. Le changement de caméra fut opéré lors de la modification de la configuration. Cependant, suite à des problèmes d'accès numériques aux paramètres de la seconde caméra pour fixer l'échelle d'intensité, aucune acquisition n'a pu être réalisée avant la rédaction de ce mémoire. Toutes les images récoltées, utilisées et analysées dans ce travail sont donc celles acquises lors de la configuration avec la première caméra.

4. Le chariot élévateur

Le chariot élévateur sert de table de pose pour le miroir mesuré par l'instrument. Il permet aisément son positionnement en x et y grâce à ses pieds à roulette. De plus, sa fonction élévatrice permet de placer le miroir à la position optimale pour la mesure.

Comme expliqué dans la section théorique, afin d'effectuer les meilleures prises de données de la surface du miroir, il est nécessaire que la caméra soit placée au niveau du centre de courbure dans le cas de miroirs sphériques. En ce qui concerne les miroirs plats, aucune condition de placement n'est recommandée.

Dans notre dispositif, la structure ne permet pas le déplacement de la caméra, car celle-ci est fixée. Il est donc nécessaire de pouvoir modifier la distance caméra-miroir d'une autre manière. Le choix d'un chariot élévateur s'est donc très vite imposé afin de résoudre ce problème.

5. La sonde de distance

Suite à l'utilisation des Équations (9) pour le calcul des pentes de la surface spéculaire mesurée, il est obligatoire de connaître les distances caméra-miroir et fil-miroir. Cependant, comme cela a déjà été précisé, ces deux distances sont égales grâce à la disposition particulière de notre instrument. Il reste néanmoins à déterminer cette valeur pour pouvoir mesurer la surface du miroir.

La première méthode utilisée a été de graduer l'un des pieds du premier dispositif (marques noires sur le pied avant droit sur la Figure 18). La hauteur zéro a été choisie comme étant celle du fil et de la caméra. Les graduations augmentaient donc en valeur en partant de cette hauteur vers le sol. L'objectif était de pouvoir déterminer aisément la valeur de la hauteur à chaque changement de miroir et donc de disposition du chariot élévateur.

Ensuite, dans le but d'obtenir une mesure plus précise de cette hauteur, une sonde a été placée sur notre dispositif. La valeur obtenue ne doit donc plus être encodée manuellement comme dans le cas de nos graduations, mais s'enregistre directement dans les données fournies par le programme de Monsieur Baron.

Lors du changement de dispositif, nous avons rencontré un problème avec la sonde. Dans son nouvel environnement, elle n'a plus produit aucune valeur cohérente. La raison de ce problème n'est pas réellement définie à l'heure actuelle. Les hypothèses sont une trop grande réflexion des ondes qu'elle émet dans son entourage proche ou une trop grande distance mesurée par cette sonde achetée à moindre coût en magasin. Quelle que soit la raison, la sonde ne permet plus la réalisation de mesures dans le deuxième dispositif, tout comme la caméra jusqu'à ce que cela soit résolu.

6. Le programme de l'instrument

Parallèlement au développement de son instrument, Monsieur Baron a également développé un programme en Matlab pour en assurer le fonctionnement. Le but de ce programme est de permettre l'alignement du dispositif (Tâches 1, 2, 3) et la collecte de données pour le calcul des pentes (Tâche 7). Monsieur Baron a aussi intégré à son programme une première méthode de calcul de pentes dans l'attente de la finition du mon programme, mais celle-ci n'est pas développée dans ce mémoire.

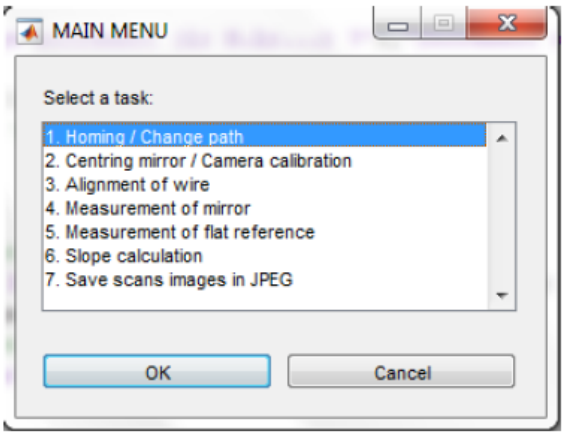


Figure 22 - Visuel du menu principal du programme de Monsieur Baron [5]

La tâche « Homing / Change path » sert à calibrer les zéros de l'appareil. Grâce à des capteurs de contact, l'appareil oriente son fil en direction de scan x et y pour palper ces points définis comme les 0 des valeurs x et y.

La tâche « Centring mirror / Camera calibration » permet de faire apparaître le visuel vidéo de la caméra à l'écran. Dans un premier temps, cela permet de positionner le miroir de manière centrée par rapport au champ de vue de la caméra. Ensuite, le fil chaud se positionne au centre de la structure et s'allume. En étant réfléchi par le miroir en position centrale, l'opérateur va pouvoir régler manuellement l'échelle d'intensité observée sur la première caméra. Lors de l'utilisation de la seconde caméra, le réglage se fera numériquement.

Ensuite, avec la tâche « Alignement wire », l'opérateur peut vérifier que le fil est correctement orienté pour un scan des axes x et y orthogonal. Si ce n'est pas le cas, l'opérateur peut ajuster cet alignement à l'aide de boutons affichés à l'écran.

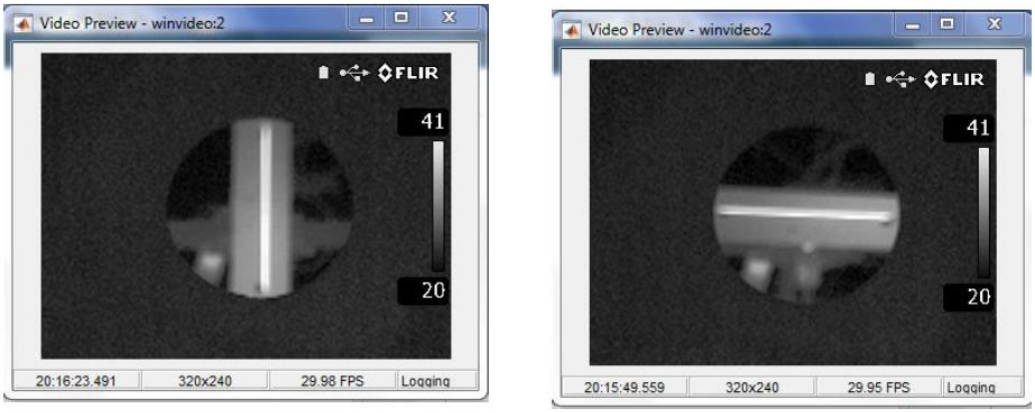


Figure 23 - Visuel de la caméra lors de l'alignement du fil en x (à gauche) et en y (à droite) [5]

Enfin, la tâche « Save scans images in JPEG » consiste au scan automatique du miroir le long des axes x et y par le fil. Lors du déplacement du fil pas-à-pas, la caméra acquiert une image à chaque pas et les sauvegarde dans deux fichiers : un pour le scan en x (fichier « x »), un pour le scan en y (fichier « y »). Dans un premier temps, les fichiers images portaient des noms particuliers en fonction de leur acquisition. Ensuite, suite à une modification de Monsieur Baron, leur nom correspond à l'itération de pas à laquelle le fil se situait au moment de la capture. Un changement fût opéré dans mes programmes suite à cette modification.

Avec les deux dossiers images, le programme de Monsieur Baron fournit aussi deux autres fichiers en format « .mat ». Ce type de fichier spécifique au logiciel Matlab permet de sauvegarder des variables et leur valeur ainsi que des tableaux complets de données. De ces deux fichiers appelés « mirror.mat » et « slope.mat », plusieurs informations utiles sont extraites à chaque lancement de mon programme afin de permettre le calcul des pentes et la reconstruction de la surface :

- La hauteur ;
- Les positions x et y de la caméra ;
- La taille du pas de déplacement du fil ;
- La position de départ du scan du fil en x et en y.

Ces deux dossiers et ces deux fichiers se trouvent toujours dans un dossier commun choisi et nommé par l'opérateur de la mesure.

7. Les différentes installations

Au cours de l'avancée du projet, le dispositif de mesure est passé par deux stades d'installation différents. Le premier stade permettait de développer l'instrument au sol. Pour cela, le dispositif était fixé sur quatre pieds de 1,60 m (voir Figure 18). Ainsi, le dispositif était facilement accessible pour Monsieur Baron et moi-même si des modifications devaient y être apportées. De plus, avoir l'instrument en main nous a permis aussi d'apprivoiser la machine, de connaître ses défauts et de comprendre comment les résoudre.

Comme expliqué précédemment, la première caméra fût utilisée sur cette installation. Tous les acquisitions et résultats présentés dans ce mémoire ont été obtenus dans cette configuration.

La seconde installation est l'installation considérée comme finale de l'appareil. Il fut placé sur un banc de test à près de 6 m du sol sans ses pieds de 1,60 m, car ceux-ci empêchaient sa bonne installation. Le champ de vue de la caméra donne sur une ouverture de la face inférieure du banc de test. Il a fallu ajuster au mieux la position de la machine afin que les bords de l'ouverture ne gênent pas la vue. Enfin, la caméra fut remplacée par une caméra de plus haute résolution.

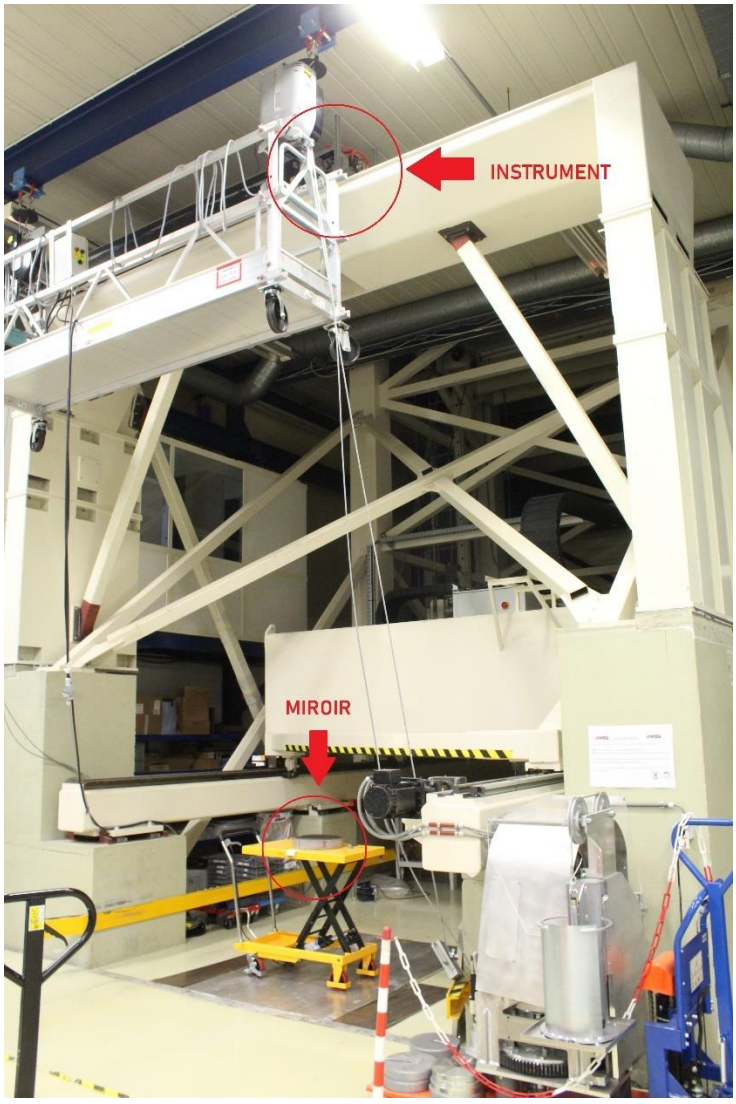


Figure 24 - Photographie de la seconde installation de l'instrument

Suite aux problèmes techniques liés aux paramètres de la caméra qu'il reste à régler, et parce que la sonde n'émet plus de valeur fixe et cohérente, aucune acquisition n'a encore été réalisée avec cette installation. Monsieur Baron et moi-même espérons pouvoir régler tout cela dans les plus brefs délais afin que l'instrument devienne parfaitement opérationnel.

8. Mon apport au développement du dispositif

Mon apport au dispositif et au travail de Monsieur Baron peut se découper en trois parties : le partage d'informations, les besoins pour mon travail et le développement technique de l'instrument.

Au niveau du partage d'informations entre Monsieur Baron et moi-même, nous avons passé plusieurs heures autour de la machine à nous demander comment améliorer le dispositif continuellement. Nous nous sommes demandés aussi quels étaient les futurs points à investiguer au cours des prochains mois afin d'améliorer nos résultats. De nombreuses idées ont germé en fonction des besoins de chacun et de nos connaissances respectives. Par exemple, l'idée de la sonde fût évoquée lors de nos entretiens et installée peu de temps après. Plusieurs des recherches que j'avais menées ont aussi permis à Monsieur Baron de développer un premier prototype de mesure de calcul de pentes qui a été intégré dans son programme (Tâches 4, 5, 6).

En ce qui concerne les besoins pour mon travail, il m'était nécessaire d'avoir accès à des données utilisées lors du scan de l'instrument. Cette affirmation s'est avérée d'autant plus vraie lorsque Monsieur Baron a amélioré et modifié des valeurs qui étaient précédemment fixées, comme la taille des pas ou les positions de départ x et y du scan. J'ai donc demandé à Monsieur Baron de modifier son code selon certaines instructions précises afin que je puisse récolter toutes ses données qui se trouvaient être vitales pour mon projet personnel. Évidemment, aucune de ces demandes ne remettait en cause le travail de mon collègue. La sauvegarde de certaines données ou la non-modification d'autres représentant l'essentiel de mes demandes, cela n'a pas impacté son travail à plus d'une ligne de code pour chaque demande.

Enfin, à plusieurs reprises, j'ai apporté mon aide à Monsieur Baron lors de modifications physiques de l'instrument. Que cela soit pour maintenir la structure, serrer des boulons, surveiller l'écran de contrôle lors de réglages ou tests manuels ou encore déplacer l'instrument de sa première installation vers la finale, j'ai répondu présent dès que mon collègue se trouvait dans une situation nécessitant une assistance technique supplémentaire.

Calcul des pentes

Dans les prochaines sections, je présente le programme que j'ai développé pour la reconstruction des surfaces des miroirs mesurés par déflectométrie infrarouge. J'y décris l'évolution de mon travail en partant de mes premières ébauches jusqu'à atteindre la version finale.

Mon programme fut développé sur le logiciel Matlab. Ce choix fut ainsi fait pour être dans la continuité du travail de Monsieur Baron et ainsi permettre un relais facile entre nous. Mon cursus universitaire ne m'ayant pas introduit à cette forme de programmation, j'ai commencé par me faire la main en développant quelques petits codes afin d'en apprendre les rudiments. Il est possible que mes codes ne soient pas complètement optimisés par des fonctions prédéfinies sur Matlab que je ne maîtrise pas : il n'en reste pas moins que mes programmes sont totalement opérationnels.

Chaque programme présenté au fil des prochaines sections se retrouve entièrement dans les Annexes. Chaque programme étant composé de plusieurs scripts, il sera présenté dans une annexe distincte et divisée en plusieurs sous-sections pour chacun des scripts afin d'en faciliter la lecture. Le choix de diviser mes programmes en plusieurs scripts permet une lecture allégée de mes lignes de codes tant pour leur utilisation que leur compréhension. De plus, en le divisant en plusieurs scripts, cela me permettait de faire directement appel à eux dans chaque nouvelle version de mon travail sans avoir à les modifier. Enfin, chacun de mes scripts est divisé en modules. Un module est introduit par les lignes de code suivantes :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Titre du module %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Ces lignes de commentaires dans mes codes ont pour but de décrire plus simplement mes programmes et d'en permettre au lecteur une meilleure compréhension.

Dans cette première section, je m'attarde sur la première version de mon programme qui permet le calcul des pentes d'un miroir mesuré. Lors de mon arrivée chez AMOS, la machine développée par Monsieur Baron n'était pas encore au point. Cependant, quelques jeux de données avaient pu être réalisés pour me permettre d'entamer mon travail. Comme ces données n'étaient pas totalement fiables, j'ai donc décidé de diviser mon travail en deux parties : la première consiste en la réalisation d'un premier algorithme de calcul de pentes à l'aide de ce jeu de données, la seconde porte sur la réalisation d'un algorithme d'intégration de gradient, non pas sur base de ces données mais sur base de cartes que j'ai moi-même créées. Le code de mon intégration est donc présenté dans une autre section afin de mettre en évidence les différences dans le travail effectué.

1. Présentation de mon code

Mon programme de calcul des pentes débute par l'exécution du script « Deflect.m » (voir Annexe 1.1). Ce script constitue la colonne vertébrale de mon programme et fait appel à d'autres scripts en fonction de l'avancée du calcul des pentes.

Le premier module de ce script est le module « Nettoyage ». Il permet de vider la fenêtre de commandes, l'espace de travail et de fermer les fenêtres déjà ouvertes par Matlab. L'objectif de ce script est de permettre un travail plus fluide et éviter l'arrêt du programme en raison d'un paramètre inattendu.

Le second module « Lien du dossier contenant les scripts » permet de définir une variable contenant l'adresse où se situent les scripts. Le logiciel Matlab doit nécessairement se trouver dans le même dossier que les scripts pour pouvoir les exécuter. Cependant, mon travail me mène à travailler dans d'autres dossiers, il m'était donc nécessaire de définir cette variable pour pouvoir voyager plus facilement dans ces dossiers.

Le troisième module « Sélection du dossier avec les données » demande à l'utilisateur de choisir le dossier de données qu'il souhaite utiliser. Il en définit le chemin d'accès, mais aussi les accès vers les dossiers « x » et « y » qu'il contient.

Ensuite, le module « Conversion des dossiers x et y en dossiers JPEG » permet d'enregistrer les images collectées par l'appareil au format JPEG dans deux nouveaux dossiers. Comme précisé précédemment, le programme de Monsieur Baron enregistre déjà les images au format JPEG. Cependant, dans les jeux de données reçus pour entamer mon travail, les images n'étaient pas toutes dans le même format (parfois JPEG, parfois BMP). J'ai donc trouvé intéressant de créer ce module qui permet la recherche de tous les formats d'images et leur conversion dans le format JPEG.

Ce module exécute le script « JPEG.m ». Ce dernier enregistre donc un dossier d'images de formats différents en un dossier d'images au format JPEG. Il est appliqué à deux reprises dans le module afin de convertir les dossiers en deux dossiers images. Ce script est présenté intégralement dans l'Annexe 1.2.

Le premier module de ce script est nommé « Noms des images ». Il permet de créer un tableau contenant la liste de tous les noms des images trouvées dans le dossier sélectionné. Les différents formats couverts par ce module sont BMP, PNG, GIF, HDF, BPM, PGM, PPM et JPEG.

Le module suivant nommé « Création du dossier JPEG » permet la création d'un dossier JPEG qui va accueillir les nouvelles images. Dans le cas de ce programme, le dossier JPEG est situé dans le dossier contenant les dossiers « x » et « y » d'origine. Le module permet la création des dossiers « x » et « y » dans le dossier « JPEG » s'ils n'existent pas.

Le dernier module de ce script est l'« Enregistrement des images en JPEG ». Celui-ci travaille dans le sous-dossier du dossier « JPEG ». Il appelle une à une les images du dossier de départ et les enregistre au format JPEG dans le nouveau dossier. Lorsque la conversion est terminée, le programme retourne au script « Deflect.m » et en continue la lecture où il s'était arrêté.

Le module suivant du script « Deflect.m » s'appelle « Références des meilleures intensités ». Ce module permet de définir une carte qui est de la taille des images du dossier sélectionné et qui contient pour chaque pixel le numéro de l'image possédant la plus grande intensité pour ce pixel. En effet, dans la première version de mon programme, j'ai décidé de ne pas réaliser le calcul du centre de masse pour chaque pixel, mais de me concentrer uniquement sur la position du fil qui donnait la plus grande intensité. J'ai fait ce choix uniquement dans un souci de simplification dans un premier temps afin de

maîtriser l'outil Matlab. Néanmoins, le calcul du centre de masse sera directement introduit dans la seconde version de mon programme lors du calcul des pentes.

Ce module fait appel au script « Intensite.m » qui permet la lecture des fichiers « x » et « y » et donne pour chacun d'eux une carte avec la meilleure intensité et une carte avec la référence de l'image qui correspond à cette intensité pour chaque pixel. Ce script est constitué d'une boucle de deux itérations (une pour chaque dossier). Il est retranscrit dans la section Annexe 1.3.

Le module commence sa boucle par un premier module : « Dossier itéré ». Il permet de se placer dans le dossier sélectionné et d'en ressortir deux caractéristiques : la liste de noms et le nombre de fichiers qu'il contient.

Ensuite, « Préparation à la cartographie » permet de définir les cartes qui sont remplies par les valeurs des maximums d'intensité et leur référence. Dans ce module, la première image de la liste est ouverte et est convertie du format RGB en niveau de gris. L'objectif de cette manipulation est de faciliter le traitement de l'image et surtout de faire passer le nombre de dimensions de la variable qui la contient de 3 à 2. Ainsi, la création des deux cartes peut se faire plus aisément.

Le module « Réalisation des 2 cartes » ouvre ensuite les images du dossier une à une et lit chacun de ses pixels. Si l'intensité dans un ou plusieurs des pixels d'une image est supérieure à celle dans le pixel correspondant, la valeur est remplacée et la nouvelle référence notée. Lorsque toutes les images ont été lues, deux cartes distinctes de références sont créées, une pour chaque dossier.

La dernière étape de la boucle de ce script consiste en l'« Affichage des deux cartes » pour chaque dossier. Les cartes créées pour chaque dossier sont converties en images et sont affichées dans des fenêtres. Le but de cette étape est d'évaluer le bon fonctionnement du programme à l'aide de visuels simples.

La dernière étape avant d'entamer le calcul des pentes consiste en la détermination de la position du fil pour chaque acquisition et des coordonnées de chaque pixel de la surface. Pour cela, le module « Calcul des coordonnées » est lu par le script « Deflect.m ». Ce module consiste entièrement en la lecture du script « Coordonnees.m » qui réalise ces différents calculs.

Pour ces calculs, il est néanmoins important de présenter quelques hypothèses et estimations réalisées. En effet, comme cela a déjà été cité, les jeux de données avec lesquels j'ai préparé ce premier programme sont le fruit d'un premier test effectué à l'aide de l'instrument de Monsieur Baron qui était à peine en développement à l'époque. J'ai donc dû effectuer plusieurs estimations grossières pour pouvoir progresser dans mon travail. Premièrement, ne connaissant pas la distance parcourue par le fil lors d'un scan, j'ai estimé une distance de 500 mm. J'ai aussi estimé que la taille de pas était fixée dans chaque direction et correspondait aux 500 mm divisés par le nombre d'acquisitions dans la direction donnée. Ma seconde estimation portait sur la surface couverte par la caméra. J'ai estimé que la longueur de l'image correspondait également à 500 mm et que la largeur était proportionnelle à cette distance par rapport aux nombres de pixels respectifs.

Le premier module du script « Coordonnees.m » réalise la « Définition des distances estimées ». De plus, ce module lance la boucle de deux itérations, chacune représentant un des dossiers « x » ou « y ».

Le module « Calcul de la position du fil » détermine ensuite la position du fil à chaque acquisition en se basant sur la taille estimée du pas.

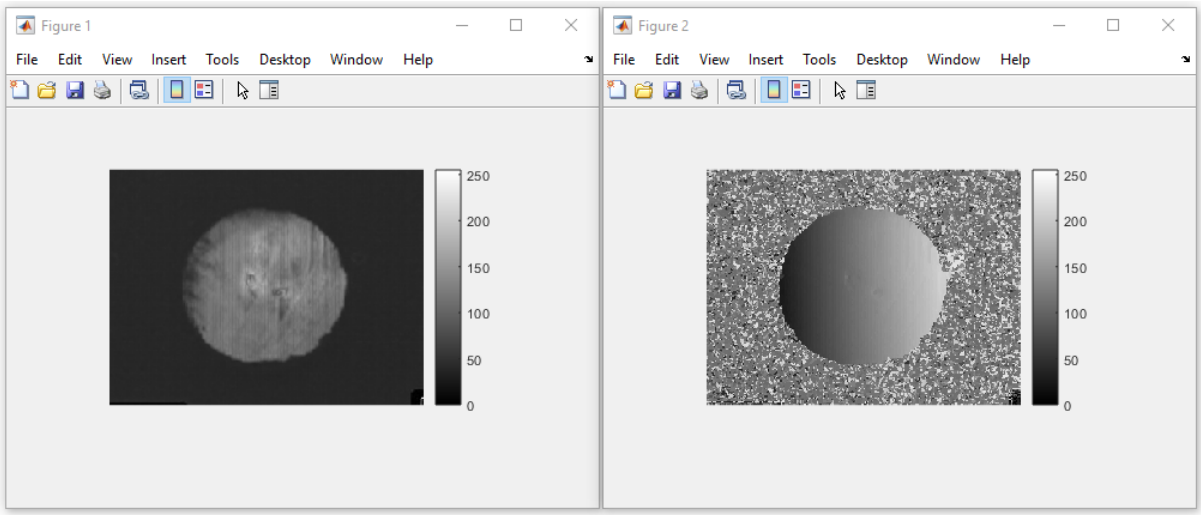


Figure 25 - Cartes des intensités maximales (Figure Matlab 1) et des références (Figure Matlab 2) lors du scan en x

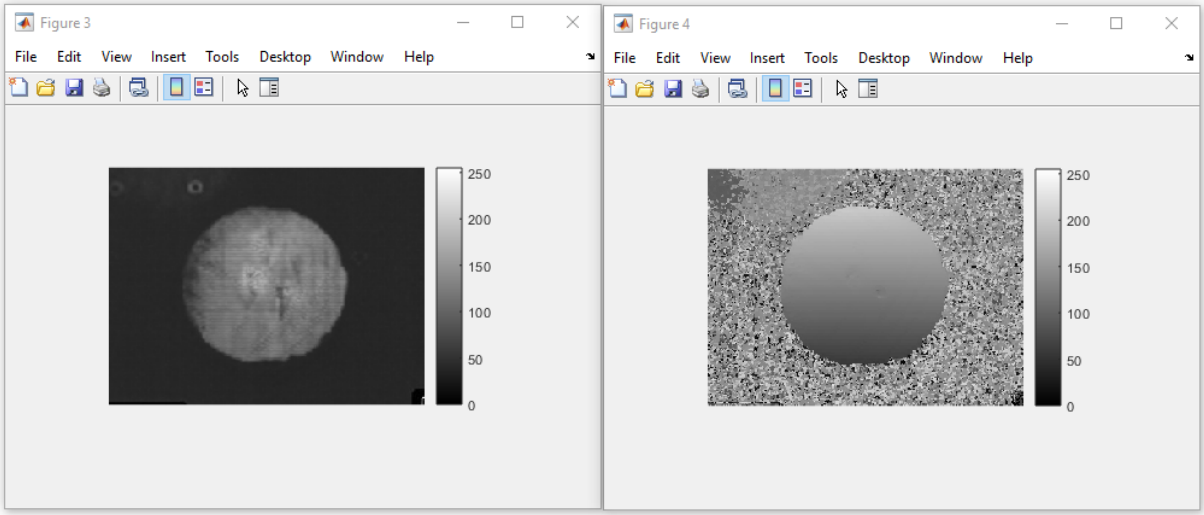


Figure 26 - Cartes des intensités maximales (Figure Matlab 3) et des références (Figure Matlab 4) lors du scan en y

Le module « Calcul de la position des pixels de la surface » quant à lui détermine les dimensions réelles de la surface, observées sur base des estimations réalisées. Ensuite, après avoir déterminé la taille d'un pixel de la surface dans la direction observée, l'algorithme détermine la position de chacun des pixels toujours dans la même direction. Pour réaliser ce calcul, je pars du principe que l'image n'est pas soumise à une distorsion. Cette hypothèse sera retravaillée dans les futures évolutions de mon programme.

Le dernier module de ce script nommé « Enregistrement des données respectives » sauvegarde les valeurs calculées sous un nom de variable propre à chaque direction de travail.

Finalement, les deux derniers modules du script « Deflect.m » sont similaires à quelques différences près. Nommés « Calcul des pentes x » et « Calcul des pentes y », ces deux modèles lancent respectivement les scripts « Pentes_X.m » et « Pentes_Y .m». Chacun de ces scripts est écrit de la même manière :

- Un premier module nommé « Paramètres » où sont définis les paramètres nécessaires de la caméra (la hauteur de la caméra n'est pas redéfinie dans le deuxième script, car elle existe toujours) ;

- Un second module nommé « Calcul des pentes » qui détermine la valeur de la pente en chaque pixel surface selon la direction sur base des Équations (9) ainsi que les pentes maximum et minimum observées ;
- Le dernier module « Affichage » est destiné à la présentation des résultats obtenus sous forme d'image.

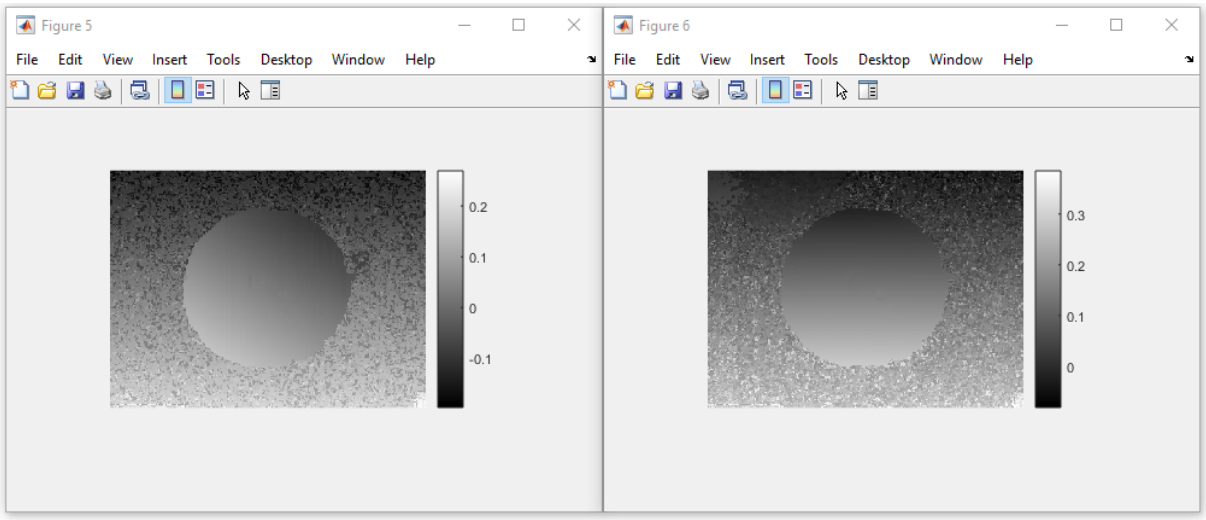


Figure 27 - Pentes calculées selon x (Figure Matlab 5) et selon y (Figure Matlab 6)

Ces dernières étapes clôturent la première version de mon calcul des pentes par déflectométrie infrarouge. Au terme de l'exécution du calcul, l'utilisateur se trouve face à 6 fenêtres Matlab ouvertes :

- Les deux cartes des intensités maximales ;
- Les deux cartes des références ;
- Les deux cartes contenant les valeurs des pentes.

2. Analyse des performances

Plusieurs aspects sont pris en considération dans l'analyse des performances de ce premier programme :

- Le temps de l'opération ;
- L'interface
- Les erreurs et les risques d'arrêt du programme.

Tout d'abord, les différents résultats de durée d'opérations obtenus pour les trois dossiers d'images qui m'avaient été fournis pour réaliser ce premier programme sont présentés dans la Tableau 1. Ces résultats sont accompagnés des caractéristiques des dossiers, à savoir les dimensions de leurs images et les tailles des dossiers « x » et « y ».

Au vu des résultats et en toute logique, le temps d'opération semble être plus long pour des dossiers contenant plus d'images. Le constat de l'augmentation de la durée suite à un plus grand nombre d'itérations permet également de conclure que le temps aurait été plus long pour des images de plus grandes tailles.

Dossier	Taille des images	Dossier « x »	Dossier « y »	Temps
1	320 x 240 pixels	230 images	210 images	9,560 s
2	320 x 240 pixels	74 images	100 images	/
3	320 x 240 pixels	137 images	154 images	7,0422 s

Tableau 1 - Temps obtenus pour le calcul des pentes

Le second dossier, ne donne aucun résultat. Le programme s'arrête au moment où il doit convertir les images RGB en niveau de gris. Cela vient du fait que les images dans ces dossiers sont déjà en niveau de gris. Ainsi, le programme s'arrête car il ne reçoit pas d'images RGB à convertir. Ce défaut a été réglé dans la version suivante du calcul des pentes.

L'interface, quant à lui, n'a pas encore été défini. L'opérateur ne peut aucunement interagir sur le déroulement des opérations. Seules les 6 fenêtres présentées précédemment donnent des informations à l'utilisateur quant à l'avancée de l'opération et aux résultats. De plus, les résultats ne sont pas sauvegardés au cours de l'opération. Ce manque d'interactivité sera amélioré au fur et à mesure des évolutions du programme comme présenté dans les prochaines sections.

Au sujet des erreurs et potentiels risques liés à l'arrêt du programme, deux sources d'erreurs ont été découvertes. La première concerne le format RGB des images comme ci-dessus. La seconde concerne l'utilisation du programme. En effet, si un utilisateur venait à sélectionner un dossier ne comprenant pas les dossiers « x » et « y » ou encore si ceux-ci sont vides, une erreur sur la fenêtre de commande Matlab va obligatoirement apparaître. Ce sont les deux seules erreurs qui ont été découvertes et rencontrées.

Enfin, on peut noter une erreur concernant les barres de couleurs des images présentées dans les figures Matlab 2 et 4. Effectivement, les barres sont graduées de 0 à 255 (l'échelle représente la taille des octets définissant chaque pixel de l'image). Il aurait donc fallu les modifier pour qu'elles soient à l'échelle de la taille des dossiers. La valeur de gris de chaque pixel reste néanmoins correcte tant que la taille des dossiers ne dépasse pas 256 images. L'erreur n'aurait été visible que dans le cadre de dossiers dépassant cette limite.

3. Les résultats obtenus

Aucun résultat concluant et à peu près correct ne peut être obtenu quant à la forme des miroirs en raison des nombreuses estimations et hypothèses développées pour ce premier essai, l'état d'avancement du développement de la machine de Monsieur Baron ne permettant pas plus de précisions.

Cependant, les images présentées prouvent tout de même que certaines parties de l'algorithme fonctionnent correctement. En effet, dans les figures Matlab 2 et 4, on observe le déplacement du fil sur les images. Ces déplacements possèdent le même sens que le déplacement réel du fil lors du scan. On peut donc s'attendre à ce que le reste de l'algorithme fonctionne correctement, au détail près des estimations et hypothèses réalisées.

Enfin, en termes de résultats, on observe sur les figures 5 et 6 que les pentes sont en valeurs tangentielles et sont donc sans unités.

Intégration du gradient

Dans cette section, je présente la première version de la deuxième partie du projet que j'ai développé : l'intégration d'un gradient. Cette section est divisée en plusieurs parties. En effet, comme précisé dans la section précédente, les résultats de pentes obtenus contiennent des erreurs. Il était donc inenvisageable de me baser sur ces résultats pour développer un programme d'intégration de qualité. Ainsi, pour pallier à ce manque de données, j'ai réalisé deux cartes de surfaces. Je les ai dérivées pour obtenir leur gradient et pouvoir les réintégrer par la suite. De cette manière, j'ai pu comparer les cartes d'origine et celles reconstruites afin d'évaluer la qualité de mon intégration.

Dans cette section, je présente donc le programme générateur de surface que j'ai développé dans un premier temps ainsi que les raisons du choix des cartes. Ensuite, je présente le programme d'intégration que j'ai codé. Enfin, je réalise une analyse des résultats obtenus par rapport aux cartes de départ.

1. Le générateur de surface

Les codes du générateur de surface que j'ai créés se trouvent en Annexe 2. Le programme qu'il représente permet la génération d'une carte de surface et de ses cartes de pentes. L'utilisateur a le choix entre 4 options de génération :

- Une carte de surface représentant le quatrième terme de Zernike (le focus) et ses cartes de pentes ;
- La même carte que la précédente, mais avec un bruit aléatoire généré sur les cartes de pentes ;
- Une carte de surface représentant un polynôme de Legendre du 17ème degré le long des x et du 14ème degré le long des y et son gradient ;
- La même carte que la précédente, mais avec un bruit aléatoire généré sur les cartes de pentes.

Cette possibilité de choix représente le premier module du script « Fct_pour_int2D.m » et est nommée « Choix de la carte ». Le choix parmi ces quatre options est justifié. Le but est de proposer deux cartes de base différentes : une carte de basse fréquence et l'autre avec des hautes fréquences. Ainsi, il est possible de voir comment le programme d'intégration réagit face à ces deux types de fréquences. Ensuite, la possibilité d'offrir du bruit sur les cartes de pentes permet de voir comment celles-ci peuvent influencer les résultats d'intégration du gradient. Cependant, le choix du quatrième terme de Zernike ou de ces degrés de polynômes de Legendre est totalement arbitraire et n'est aucunement justifié à part pour les basses et hautes fréquences qu'ils présentent. D'autres termes auraient pu être choisis et convenir tout autant.

Les cartes obtenues sont toutes d'une taille de 300 x 300 pixels. Avec elles sont fournies les valeurs maximale et minimale de la surface, les dimensions des cartes et la distance entre chaque pixel.

Dans les deuxième et troisième modules, la surface générée correspond donc au quatrième terme de Zernike centré sur le centre de l'image. L'équation de ce terme est donnée par

$$z(x, y) = 2 * (x^2 + y^2) - 1 \quad (10)$$

Comme cela peut être observé dans les lignes de code, un coefficient multiplicateur a été ajouté à cette équation. Le but était de permettre à la carte générée de donner un PV de 10 µm. De cette manière, comme il s'agit des ordres de grandeur qui intéressent AMOS, les erreurs de reconstruction sont aussi celles qui intéressent la société. La valeur 150 est retirée à i et j dans les équations pour permettre le centrage de la figure sur l'image. Dans le troisième module, un bruit aléatoire est ajouté aux cartes des pentes. Celui-ci est de l'ordre du microradian. Ce type de bruit sera typiquement rencontré lors de l'utilisation de l'instrument développé par Monsieur Baron. Il était donc intéressant de générer un bruit du même ordre de grandeur. Dans l'Annexe 2, les cartes de pentes avec et sans bruit sont présentées afin de permettre au lecteur d'effectuer une comparaison visuelle.

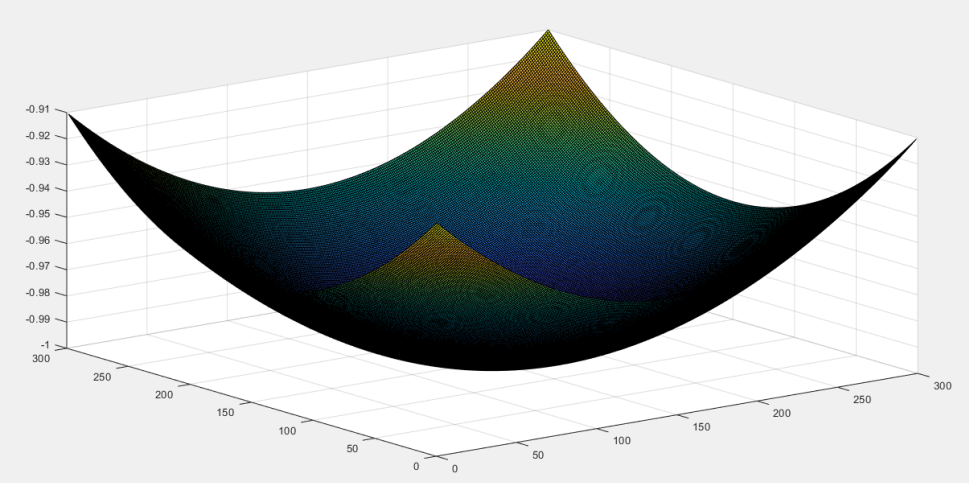


Figure 28 - Carte du quatrième terme de Zernike générée

Dans les quatrième et cinquième modules, le 17ème degré et le 14ème degré des polynômes de Legendre sont générés respectivement le long des axes x et y afin de construire une surface de hautes fréquences. La forme est générée par l'Équation (11).

À nouveau, dans les lignes de code, un facteur multiplicatif a été ajouté à cette équation afin d'obtenir une surface avec un PV de 10 µm. La valeur 150 est à nouveau retirée afin que la figure soit centrée sur l'image. Enfin, dans le 5ème module, un bruit similaire à celui du 3ème module a été ajouté aux cartes des pentes afin de voir comment les résultats évoluent. Les cartes avec et sans bruit sont présentées en Annexe 2.

Après un simple coup d'œil sur les cartes de pentes avec et sans bruit des deux surfaces, il est aisé de constater que les cartes venant de la surface de basses fréquences sont plus impactées. Cela s'explique simplement par le fait que les basses fréquences possèdent des pentes plus faibles que les hautes fréquences et sont donc plus sensibles lors de bruits similaires.

$$z = \frac{583401555 * (x-1)^{17}}{32768} - \frac{300540195 * (x-1)^{15}}{4096} + \frac{1017958725 * (x-1)^{13}}{8192} - \frac{456326325 * (x-1)^{11}}{4096} + \frac{929553625 * (x-1)^{9}}{16384} - \frac{66927861 * (x-1)^{7}}{4096} + \frac{20369349 * (x-1)^{5}}{8192} - \frac{692835 * (x-1)^{3}}{4096} + \frac{109395 * (x-1)}{32768} + \frac{5014575 * (y-1)^{14}}{2048} - \frac{16900975 * (y-1)^{12}}{2048} + \frac{22309287 * (y-1)^{10}}{2048} - \frac{14549535 * (y-1)^{8}}{2048} + \frac{4849845 * (y-1)^{6}}{2048} - \frac{765765 * (y-1)^{4}}{2048} + \frac{45045 * (y-1)^{2}}{2048} - \frac{429}{2048} \quad (11)$$

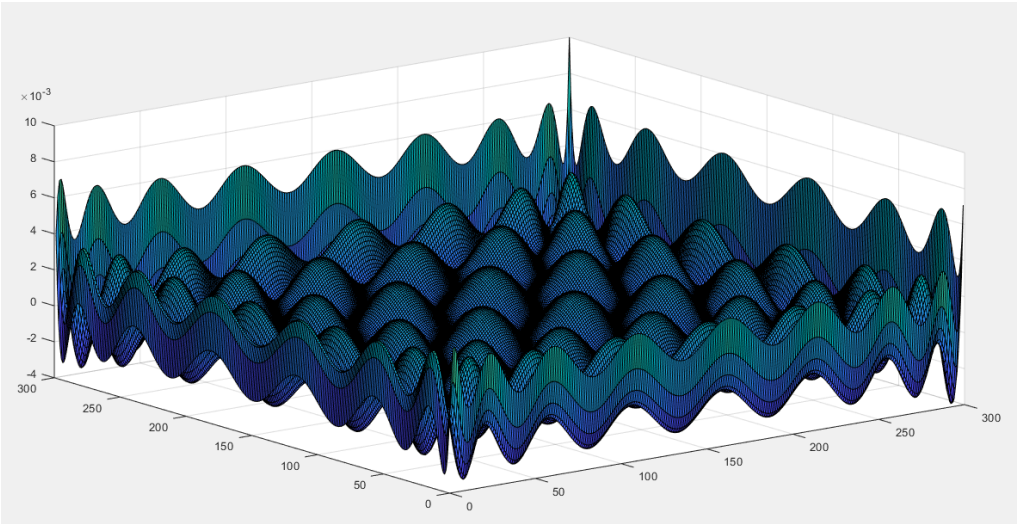


Figure 29 - Carte générée à partir de polynômes de Legendre

Maintenant que les cartes de surface et de leurs pentes sont générées, il est temps de se concentrer sur la présentation de l'algorithme qui va les reconstruire.

2. L'intégrateur de gradient

Dans la section « La structure du programme », il était expliqué que le programme d'intégration est divisé en trois parties : le masque d'intégration, l'intégration trapézoïdale et l'intégration de Southwell. Le code que j'ai construit dans ce but se retrouve dans l'Annexe 3.

Le premier module du script « Integrad_2D.m » a pour but de donner à l'utilisateur le choix de la zone de la surface à intégrer. Cette possibilité se fera à l'aide du « Masquage » défini par l'utilisateur. Il doit dans un premier temps définir la forme du masque qu'il souhaite réaliser à l'aide d'une fenêtre offrant les différentes propositions : un masque en forme d'ellipse ou de rectangle. Une fenêtre avec la surface générée par le script précédent apparaît ensuite pour permettre à l'utilisateur de sélectionner la zone à intégrer. Lorsque le masquage est réalisé, le programme demande à l'utilisateur s'il est satisfait. Le cas échéant, le programme continue. Sinon, le programme relance le module jusqu'à satisfaction de l'utilisateur.

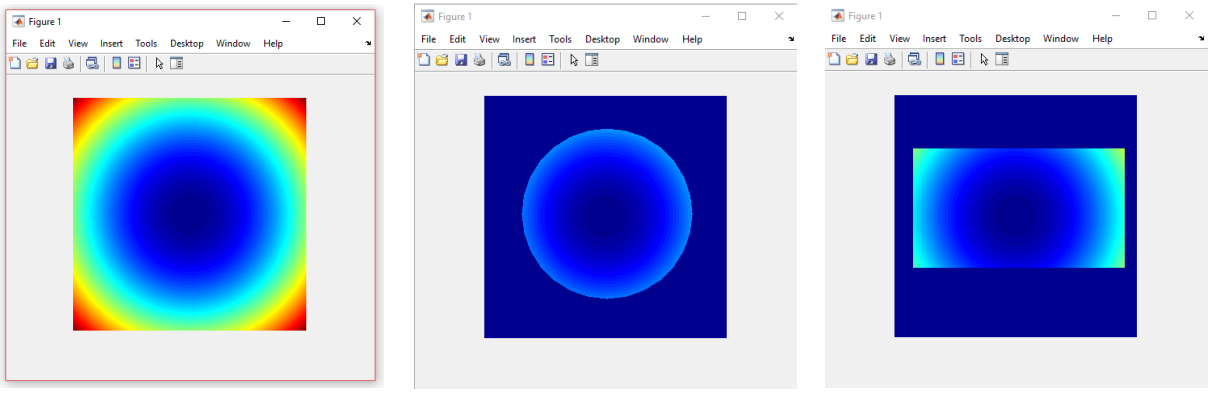


Figure 30 - Fenêtre montrant la surface générée sans masque, avec un masque circulaire et avec un masque rectangulaire

Le module « Cartes et paramètres » permet de préparer le terrain pour l'intégration trapézoïdale. Après avoir enregistré les paramètres relatifs aux dimensions des cartes de pente et des distances entre les pixels, le module crée la carte qui accueille la surface reconstruite et définit le milieu du masque comme la hauteur 0.

Il est utile de noter que cette valeur est exprimée en mm. Le programme de calcul de pentes utilisait exclusivement des distances en mètre. Ce choix de modification fut fait par souci de facilité par rapport aux attentes de la société AMOS qui utilise habituellement des distances en mm.

Le module suivant concerne l' « Intégration trapézoïdale ». Celle-ci démarre du pixel que le programme a défini comme le centre du masque et intègre vers les x+/y+ et vers les x-/y- le long d'une diagonale passant par ce pixel.

Pour la suite du module, j'ai dû recourir à une subtilité permettant l'intégration de toute la zone couverte par le masque. En effet, la logique veut que pour intégrer la surface entière, le programme reparte de chaque pixel de la diagonale définie et intègre vers les x-/y+ et x+/y-. Cependant, cela pose un problème pour les masques qui n'ont pas une longueur égale à leur largeur. En effet, comme cela est visible dans la surface suivante, certaines zones ne sont pas couvertes par les diagonales.

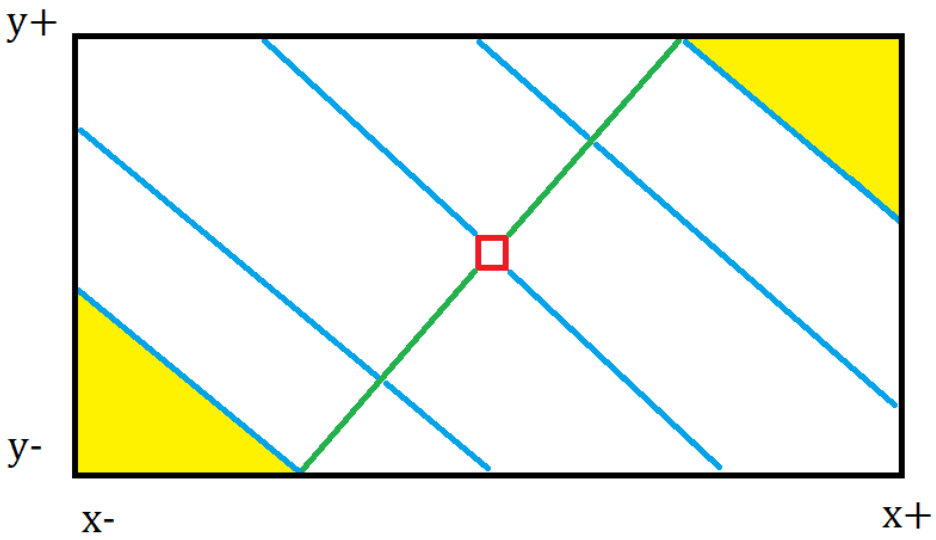


Figure 31 - Intégration depuis le pixel rouge le long des lignes vertes, puis le long des lignes bleues en ne passant pas dans les zones jaunes

Pour qu'aucune zone ne soit oubliée, le programme continue son intégration en repartant de la diagonale la plus proche de la zone non intégrée et poursuit cette manipulation jusqu'à atteindre chaque zone du masque.

Néanmoins, comme ces zones non couvertes ne se situent pas au même endroit dans un masque plus vertical que dans un masque horizontal, la méthode pour couvrir ces zones est différente :

- Pour un masque horizontal, la zone gauche sera intégrée par une alternance de direction vers les x-/y+ et x-/y- et la zone droite par une alternance d'intégration vers les x+/y- et x+/y+ ;
- Pour un masque vertical, la zone gauche sera intégrée vers les x-/y+ et x+/y+ alors que la zone droite le sera vers les x+/y- et x-/y-.

De cette manière, toute la zone masquée sera couverte par l'intégration trapézoïdale. Notons tout de même la petite nuance précisée dans la partie théorique qui disait que deux de ces intégrations étaient nécessaires, car la moitié des pixels étaient couverts par une intégration trapézoïdale. Ainsi, en début de module, mais uniquement lors de sa deuxième itération, le pixel voisin au pixel central est calculé par une seule itération de l'intégration de Southwell et le reste des pixels qui n'ont pas encore été intégrés le sont de la même manière que la première moitié de la carte.

Avant de continuer, il me semble utile de préciser que cette technique d'intégration de masque ne peut pas fonctionner pour des miroirs possédant un trou en leur centre. Il sera nécessaire à l'avenir de développer une nouvelle méthode pour réaliser une première carte de la surface en prenant en compte l'absence de surface en son centre.

Ensuite, le module « Paramètres pour Southwell » définit les points de la carte reconstruite hors du masque comme des NaN (Not a Number) et crée une carte identique à la surface reconstruite qui accueille les valeurs trouvées pour chaque pixel lors de l'intégration de Southwell.

Le module « Intégration de Southwell » calcule la valeur de chaque pixel sur base de l'Équation (4). Lorsque la carte complète est réalisée, l'ancienne carte est remplacée par la nouvelle et la nouvelle accueillera la carte suivante lors de la prochaine itération. Cela se produit autant de fois que le masque possède de pixels.

Au cours de l'exécution de ce module, il faut noter que :

- Les points hors du masque reçoivent la valeur NaN ;
- La carte du masque qui est binaire est utilisée pour définir le poids σ de chaque pixel dans l'Équation (4) ;
- Les pixels utilisés dans l'Équation (4) qui sont hors du masque reçoivent la valeur 0, car dans le cas contraire, la valeur finale du pixel calculé serait NaN même si leur poids est 0.

Enfin, le dernier module permet l'« Affichage » de la carte reconstruite.

3. Les résultats

L'analyse des résultats peut être divisée en 3 parties :

- La comparaison entre les cartes reconstruites par l'intégration trapézoïdale et les cartes réelles ;

- La comparaison entre les cartes reconstruites par l'intégration de Southwell et les cartes réelles ;
- Le temps nécessaire à la réalisation du programme.

L'intégration trapézoïdale est utilisée de manière très basique dans ce programme afin de définir une première carte de la surface et permettre une intégration de Southwell par la suite. Les résultats attendus ne devaient donc pas être révolutionnaires et c'est ce qui est constaté dans les deux figures suivantes. Celles-ci représentent la différence entre la carte d'origine et la carte reconstruite. Dans cette analyse de l'intégration trapézoïdale, les cartes avec bruit ne sont pas présentées. Le propos de cette partie de l'analyse du programme est de montrer l'évolution de la surface entre les deux types d'intégration.

Comme on peut le constater, la différence dans le cas de la surface du quatrième terme de Zernike est de 1,51 10^{-4} mm de PV sur la surface entière. L'erreur générée est donc de 1,51 % pour une surface qui a été généré sans bruit, cela est donc assez important.

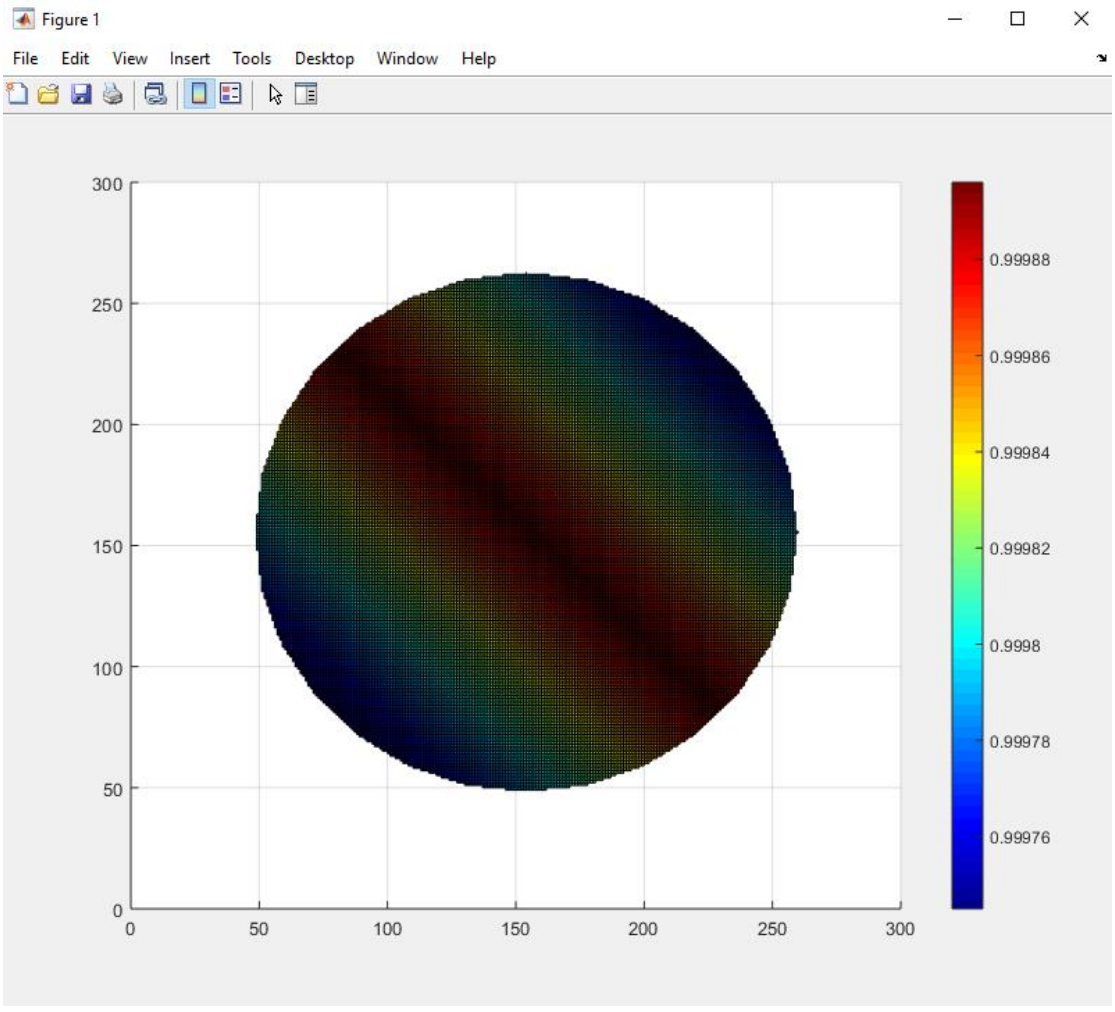


Figure 32 - Intégration trapézoïdale de la surface du quatrième terme de Zernike

Dans le cas de la reconstruction de la surface générée à partir de polynômes de Legendre, l'erreur est encore plus élevée. Dans ce cas, le PV de la carte représentant la différence entre la reconstruction et la carte de départ est de 2,36 10^{-4} mm. L'erreur sur la surface est donc doublée pour la même dimension de masque.

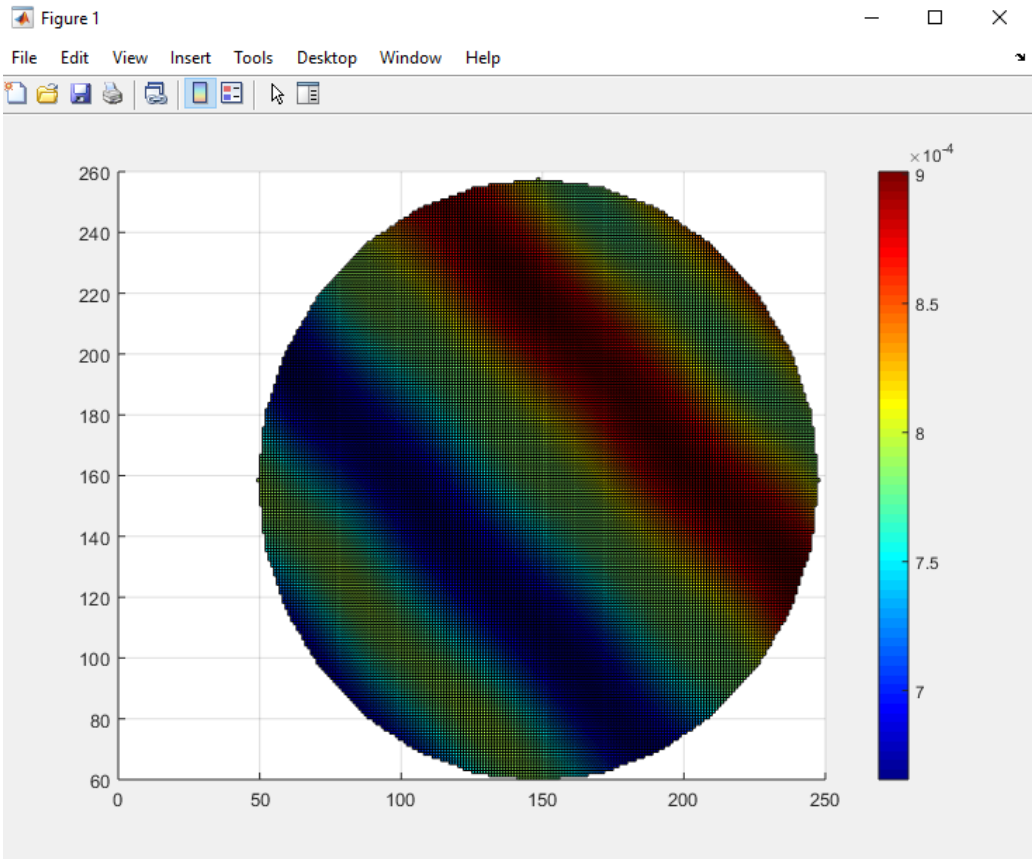


Figure 33 - Intégration trapézoïdale de la surface générée par des polynômes de Legendre

Comme attendu, la reconstruction sur base de l'intégration trapézoïdale n'est qu'une approximation de la surface idéale. Malgré son taux d'erreur, cette intégration constitue une base de travail de qualité pour l'intégration de Southwell.

Au travers de l'analyse du programme d'intégration que j'ai développé, nous allons constater de l'efficacité de l'intégration de Southwell pour la reconstruction d'une surface.

Tout d'abord, analysons la carte reconstruite de la surface représentant le focus. Comme cela est visible dans la première des quatre cartes suivantes, l'erreur entre la carte reconstruite et la carte de départ est largement plus petite que dans le cas de la simple intégration trapézoïdale ; elle est de l'ordre du nanomètre. On peut constater que l'erreur a été produite lors des deux intégrations trapézoïdales, car un pixel sur deux se situe sur le haut et un sur deux sur le bas de cette carte d'erreurs. On peut faire le lien avec la création du pixel voisin au pixel central et la possibilité d'une erreur de calcul lors de sa création.

La seconde des quatre images représente l'erreur produite lors de la reconstruction de cette même surface, lorsque du bruit a été ajouté aux cartes des pentes. Cette fois-ci, l'erreur est plus importante : 4,95 nm sur le PV de la surface. Cette erreur était attendue en raison de l'ajout du bruit, mais reste cependant plus qu'acceptable sur une surface reconstruite de 10 µm de PV.

La troisième image représente la surface reconstruite de la carte composée de deux polynômes de Legendre. Dans ce cas-ci, la valeur du PV de la carte d'erreur est de 1,92 10^{-5} mm. Cette erreur d'environ 20 nm reste dix fois inférieure à celle de l'intégration trapézoïdale. Cependant, elle reste aussi 20 fois plus importante que celle représentée dans la première image. Néanmoins, pour l'utilisation que la société AMOS fera de ce programme, ce niveau d'erreur reste recevable et acceptable.

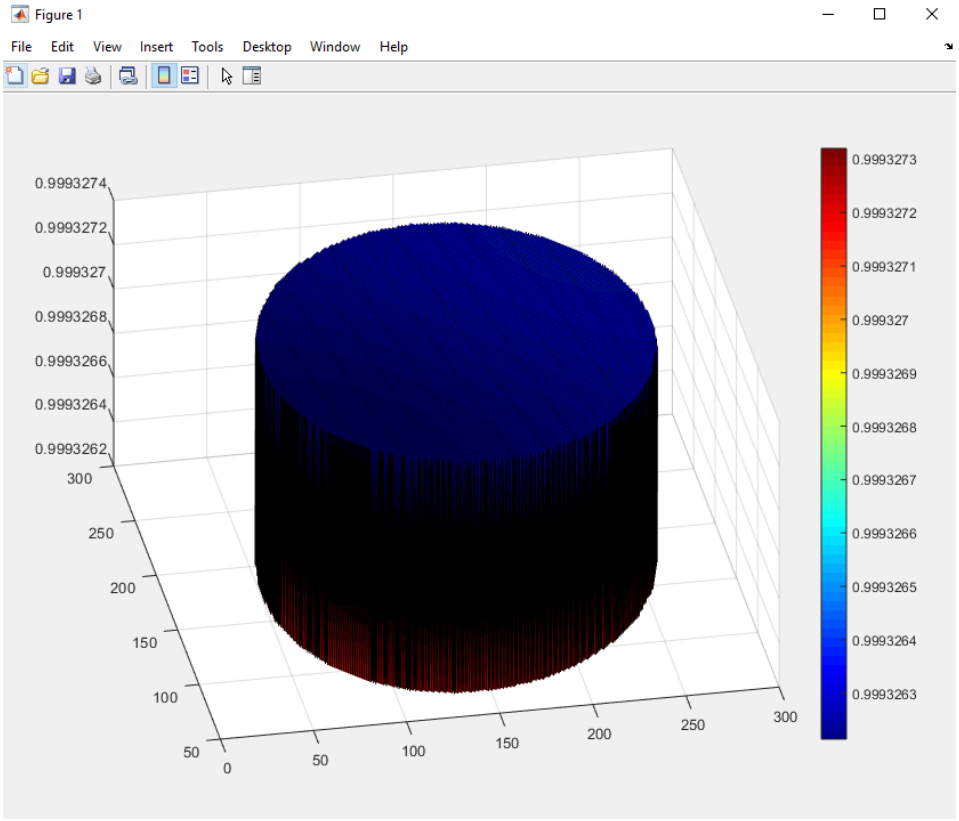


Figure 34 - Carte d'erreurs de la carte de focus reconstruite sans bruit

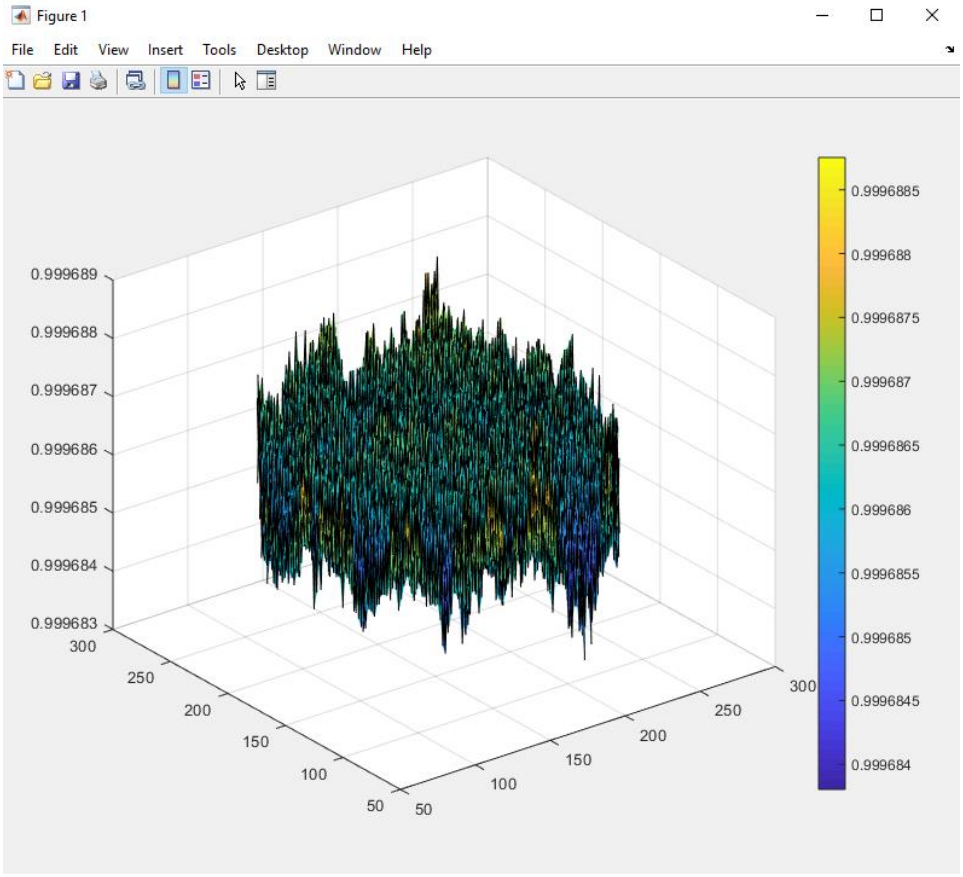


Figure 35 - Carte d'erreurs de la carte de focus reconstruite avec bruit

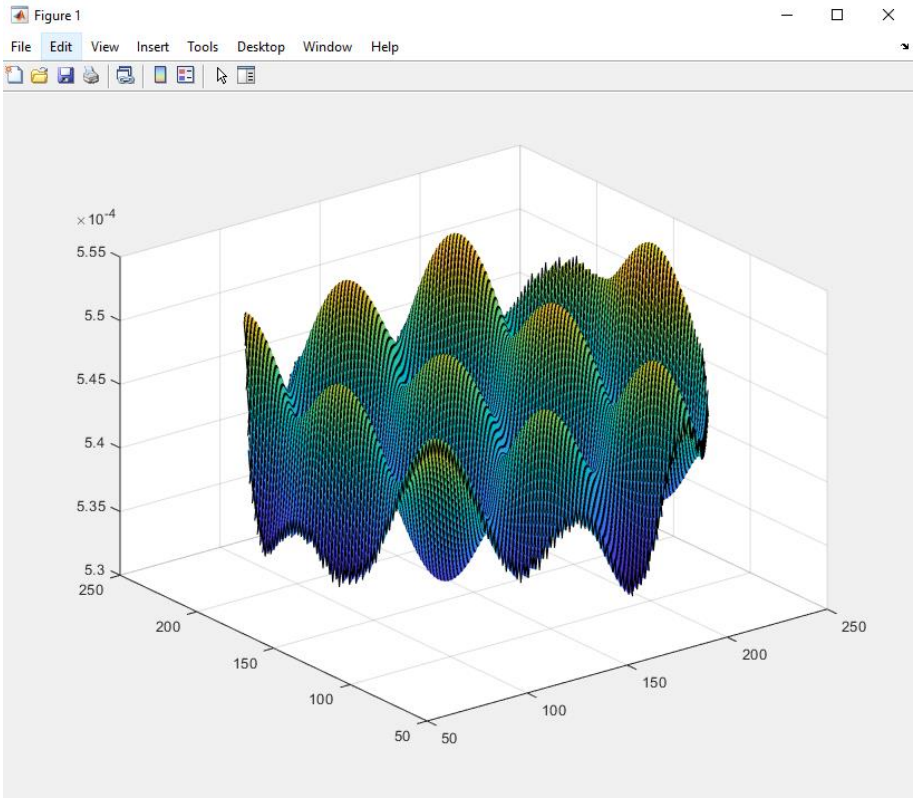


Figure 36 - Carte d'erreurs de la carte de polynômes de Legendre reconstruite sans bruit

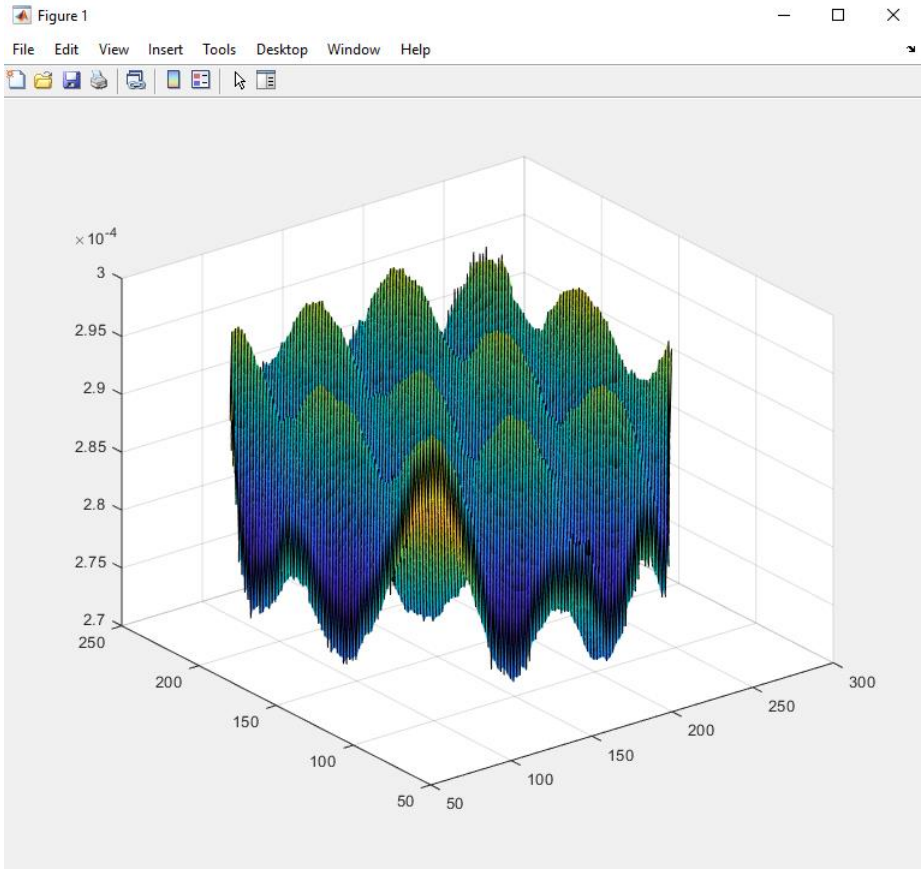


Figure 37 - Carte d'erreurs de la carte de polynômes de Legendre reconstruite avec bruit

Enfin, la dernière image représente la même surface reconstruite, mais avec du bruit dans la carte des pentes. Comme l'analyse visuelle de l'Annexe 3 le suggère, les pentes de cette surface ont été peu impactées par l'ajout d'un bruit. Cela se vérifie à nouveau ici. On peut observer que l'erreur due à la reconstruction est de 2,30 10^{-5} mm de PV. Cette valeur est presque similaire à celle obtenue juste avant. On peut donc conclure que l'ajout du bruit n'influence que très peu le résultat final lors de la reconstruction des hautes fréquences.

Le dernier point de cette analyse concerne le temps nécessaire au fonctionnement de ce programme. Pour les quatre cartes recréées ci-dessus, le zone de masquage était à peu de chose près identique et couvrait des surfaces d'environ 200 x 200 pixels. Le temps de travail a donc été environ le même et tournait autour des 150 secondes. Cependant, pour des cartes plus grandes, l'utilisateur devra s'attendre à un temps de travail plus long étant donné le caractère itératif de l'intégration de Southwell.

Il est tout de même nécessaire de préciser que l'ordinateur utilisé ne possède pas une grande puissance de calcul. Ainsi, un ordinateur plus puissant pourrait permettre un calcul de plus grandes cartes tout en prenant moins de temps. Si ce n'était pas le cas, il reste envisageable que le nombre d'itérations soit divisé pour gagner du temps, car, au vu des résultats obtenus, un surplus d'itérations n'est peut-être pas utile en fonction de l'attente de l'utilisateur en termes d'erreurs commises.

Programme complet de reconstruction par déflectométrie infrarouge

Dans cette section, je présente le programme qui résulte de la combinaison de mes algorithmes de calcul de pentes et d'intégration de gradient. Je survole la structure presque similaire à celle déjà décrite tout en m'attardant plus en détail sur les points qui ont été modifiés depuis leur première version. Dans la seconde partie de cette section, je présente comment le programme a évolué du point de vue des performances en temps, de l'interface, mais aussi des potentiels risques d'erreurs. Enfin, je présente les résultats obtenus à l'aide de mesures effectuées en partenariat avec Monsieur Baron.

1. Le programme

La présentation du programme se fera comme dans les sections précédentes, c'est-à-dire suivant la chronologie de son déroulement. Tous les scripts sont présentés dans l'Annexe 4. Comme remarque générale, il est utile de savoir que, à l'inverse de la première version du calcul des pentes, toutes les mesures de grandeurs sont exprimées en millimètres.

Tout d'abord, les deux premiers modules du script « Deflect.m » n'ont pas changé lors de la recombinaison des programmes. Le premier point qui diffère se situe dans le troisième module « Sélection du dossier avec les données ». Le module charge les données d'un fichier « infos.mat » situé dans le dossier des scripts. Il contient l'information sur le dossier contenant le dernier dossier traité par ce programme. Ainsi, l'utilisateur peut démarrer la recherche du dossier suivant dans ce qui peut être son espace de rangement de mesures au lieu du dossier des scripts. Cette modification est purement ergonomique.

Dans le module suivant, les deux lignes de code concernant la taille des dossiers JPEG ont été retirées. Celles-ci ont été ajoutées au nouveau script « Info_doss_im.m » qui, comme présenté plus loin, reprend et enregistre toutes les données concernant les dossiers JPEG et les images qu'ils contiennent. D'autres lignes de code ont été déplacées dans ce script comme décrit dans la suite de cette sous-section.

De plus, dans ce module, le programme nettoie l'espace de travail Matlab des variables qui ne sont plus utiles. Cette démarche sera dorénavant entreprise à la fin de chaque module de ce script. Enfin, le script « JPEG.m », que ce module appellait, n'a pas subi de modification.

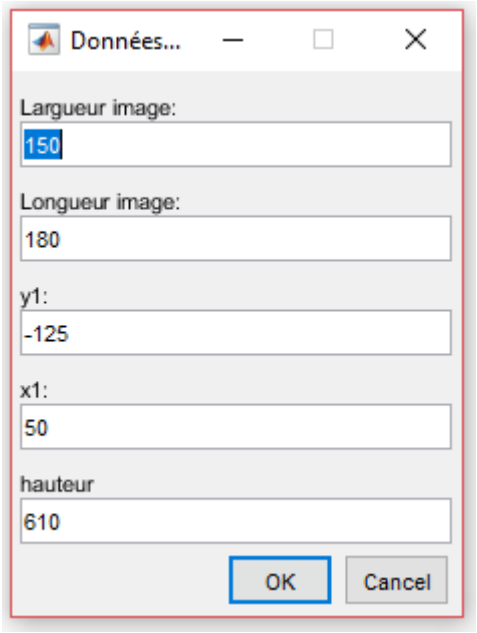
Ensuite, le module « Infos des dossiers et des images » appelle le nouveau script « Info_doss_im.m ». Comme déjà expliqué, ce script recense toutes les informations utiles au sujet des dossiers JPEG et des images qu'ils contiennent. Dans un premier module, le script détermine les chemins d'accès vers les dossiers d'images en JPEG. Il s'agit de lignes de codes déplacées depuis le module « Référence des meilleures intensités » de la première version. Le module suivant établit la liste des noms des images des deux dossiers et détermine leur taille. Ces lignes de codes sont les premières lignes de code déplacées évoquées antérieurement. Enfin, le dernier module détermine les dimensions des images sur base de la première image. Ces lignes de code se situaient précédemment dans le script « Intensite.m ».

Dans le script « Deflect.m », les deux modules suivants ont été inversés. Cette manœuvre n'influence en rien la réalisation du programme. Son seul but est de déterminer, dans un premier temps, tous les paramètres du système avant d'entamer l'analyse des images et de tous les calculs.

Le premier de ces deux modules, qui appelle le script « Coordonnees.m », n'a pas évolué. À l'inverse, le script, lui, a subi quelques modifications. Dans un premier temps, le module « Paramètres de la caméra » appelle le fichier « slope.mat » généré par l'instrument de Monsieur Baron. L'objectif est d'en retirer les informations concernant la position de la caméra lors de la collecte des données. Le second module offre à l'utilisateur une fenêtre contenant des paramètres par défaut au sujet de la prise de mesure à la hauteur de 610 mm. Ces paramètres mesurés à la main sur base de l'image captée par la caméra représentent :

- Les longueur et largeur de la surface observées ;
- La position du pixel de la surface en bas à gauche de l'image (situé dans le coin x-/y-) ;
- La hauteur, distance entre le miroir et le plan caméra/fil.

Pour une hauteur différente, ces valeurs doivent être mesurées à nouveau manuellement et inscrites dans la fenêtre apparue à l'écran avant d'être validées. Cependant, les mesures présentées dans cette section ont été réalisées à la hauteur de 610 mm, les paramètres n'ont donc pas dû être modifiés.



Lorsque les mesures ont été validées, la fin du module détermine la position de chaque pixel surface ainsi que leurs dimensions. À nouveau, cette partie du module a également été modifiée. En effet, suite à une différence d'axes entre Matlab et notre projet avec Monsieur Baron, les axes y entre nos images et les tableaux de données entrant les valeurs des pixels de l'image sont inversés. En d'autres

termes, notre pixel origine situé en x-/y- sur notre image correspond au pixel x-/y+ dans le tableau Matlab. Il a donc fallu redéfinir la manière de déterminer la position y de chaque pixel comme illustré dans l'Annexe 4. Cependant, cette différence d'axe n'impacte en rien la reconstruction de la surface. Étant donné que le calcul des pentes et l'intégration se font dans le même repère, aucune erreur n'est commise.

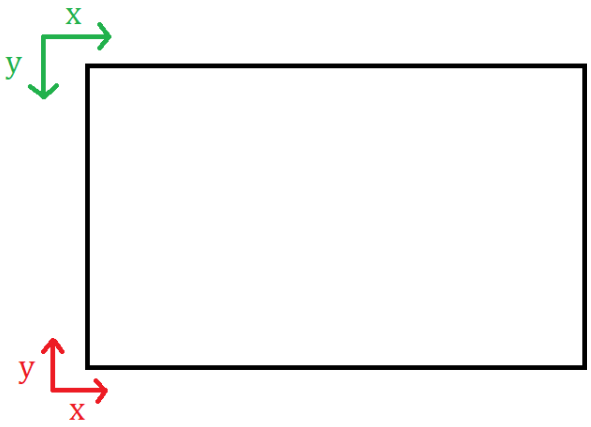


Figure 38 - Représentation de la différence d'axes entre Matlab et notre projet

De plus, comme nous pouvons le voir, la partie déterminant la position du fil a disparu de ce module. Monsieur Baron ayant entre-temps modifié son programme, le nom de chaque image correspond à l'itération d'acquisition de l'image le long d'un axe. Ainsi, dans le cas où la première acquisition se fait à la distance de 1 mm et que le pas entre chaque acquisition est de 1 mm, le nom de chaque image correspond à la position du fil en millimètres. Elles sont donc numérotées de 1 à 270, le scan parcourant totalement les axes de l'instrument en x et en y.

À l'aide de ces deux modules et grâce à l'amélioration de l'instrument de mesure de Monsieur Baron, les hypothèses et estimations effectuées lors de la première version ne sont plus utilisées.

Ensuite, le module « Références des meilleures intensités » du script « Deflect.m » n'a été modifié que des quelques lignes qui ont migré vers le script « Info_doss_im.m ». Le script « Intensite.m » que ce module appelle a été totalement modifié. En effet, ce dernier ne contient plus qu'un seul module qui permet le calcul du centre de masse du fil pour chaque pixel de la surface à l'aide de l'Équation (8). Pour déterminer cette valeur, un seuil minimum d'intensité est pris en compte afin que le bruit généré dans les images ne fausse pas la valeur obtenue. Cette valeur a été choisie en observant l'intensité reçue par plusieurs pixels situés sur la surface d'un miroir mesuré et vaut 60. Il est utile de noter que certains pixels sont complètement sous le seuil d'intensité tout au long d'un scan. Ils reçoivent donc la valeur de NaN. Enfin, le module crée aussi une image des intensités maximales en x et y qui sera utilisée lors du masquage pour l'intégration.

Les deux modules suivants qui concernent le calcul des pentes en x et en y, n'ont été modifiés que par la suppression des variables devenues inutiles au programme. Les deux scripts qu'ils appellent ont quant à eux été modifiés de la même manière : les premiers et derniers modules ont été supprimés, ne laissant que le module « Calcul de pentes ». Ces suppressions interviennent puisque la position de la caméra est déjà connue et qu'un affichage des cartes de pentes est devenu à présent inutile. Quant au module conservé, la seule modification importante porte sur la valeur de la pente (0) pour autant que la position du fil dans le pixel en cours de calcul soit NaN.

La fin du script « Deflect.m » est inédite à cette version. Le premier des deux modules qui constituent cette fin est l'« Intégration » du gradient. Ce module fait appel au script « Integrad_2D.m »

avant de supprimer les variables devenues inutiles. Au sein de ce script, une seule modification a été réalisée : elle concerne l'appel de l'image à masquer, devenue celle des intensités maximales. Cette image a été choisie, car elle permet de visualiser parfaitement la zone qui a été couverte par les scans.

Enfin, le dernier module a pour rôle de clôturer du programme. Il supprime dans un premier temps les dossiers d'images en JPEG afin de libérer de l'espace de mémoire sur l'ordinateur. Ensuite, il sauvegarde le dossier dans lequel se situait le dossier sélectionné en début de programme. Enfin, il sauvegarde les résultats dans un fichier « results.mat » dans ce même dossier.

2. Résultats obtenus

Nous pouvons analyser deux miroirs qui ont été reconstruits.

Le premier est un miroir en aluminium tourné diamant d'un diamètre de 163 mm. Il a été mesuré à une hauteur de 610 mm. On peut constater sur la figure suivante que ce miroir plat est reconstruit sous forme de selle de cheval avec une variation PV de 2,88 mm. En retirant le tilt de la surface reconstruite, la variation PV n'est plus que de 0,314 mm. Sachant que le miroir possède une planéité de 0,012 mm, la valeur obtenue reste tout de même 30 fois supérieure à la valeur réelle. Cette erreur de précision est probablement due au problème de reconstruction des basses fréquences lié à la déflectométrie.

L'analyse des hautes fréquences telle que l'équipe de scientifiques du projet SLOTS l'a réalisée est impossible dans le cadre de la reconstruction présentée ici, car les 10 premiers termes de Zernike n'ont pas été retirés aux cartes des pentes. Cependant, dans la prochaine et dernière version de ce programme, cette analyse peut être et est effectuée.

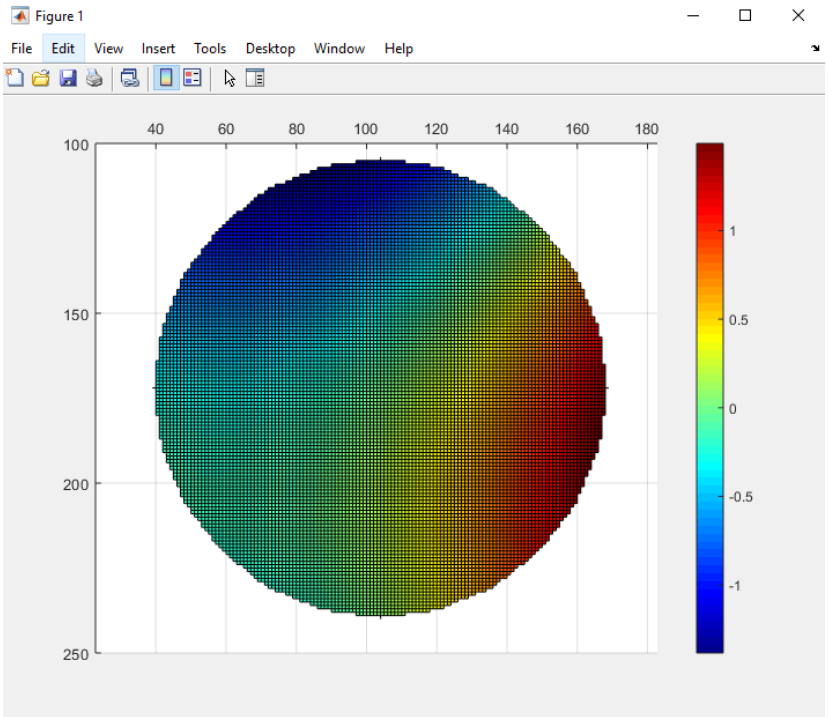


Figure 39 - Surface du miroir plat reconstruite

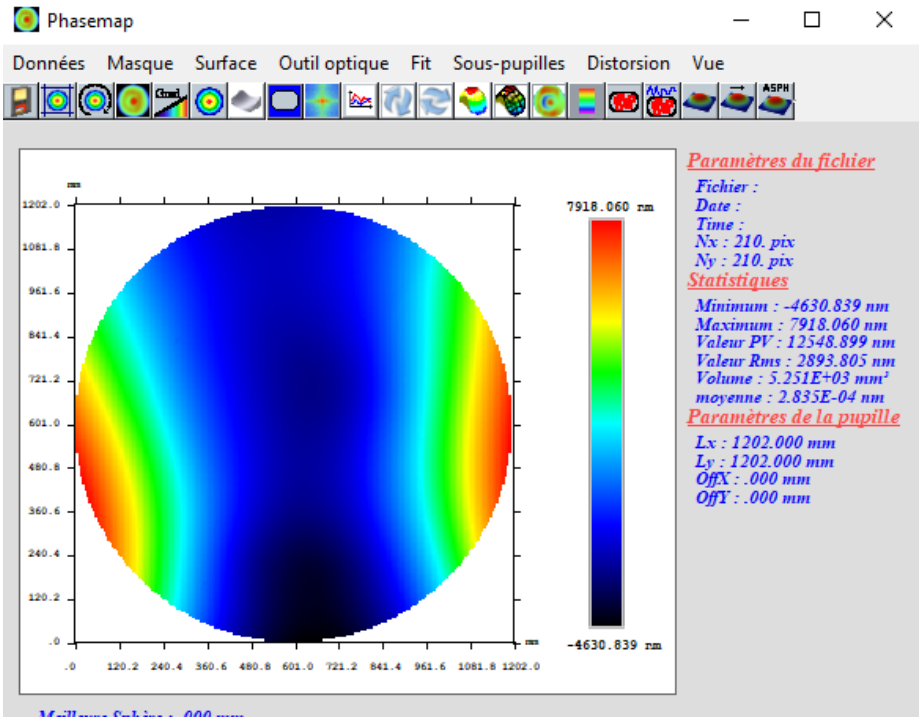


Figure 40 - Surface du miroir plat mesuré à l'interféromètre

Un second miroir concave en zérodur a ensuite été mesuré. Celui-ci possède un diamètre de 157 mm et un rayon de courbure de 600 mm. Il a été mesuré à une hauteur de 610 mm. Comme observé dans les figures suivantes, la reconstruction possède une variation de PV de 4,75 mm. Cette valeur semble proche des 5 mm de PV de la surface mesurée à l'interféromètre. Cependant, la surface reconstruite possède un tilt conséquent qui diminue considérablement la valeur du PV lorsqu'il est retiré mais aussi un aspect déformé en comparaison de la forme réelle de la surface.

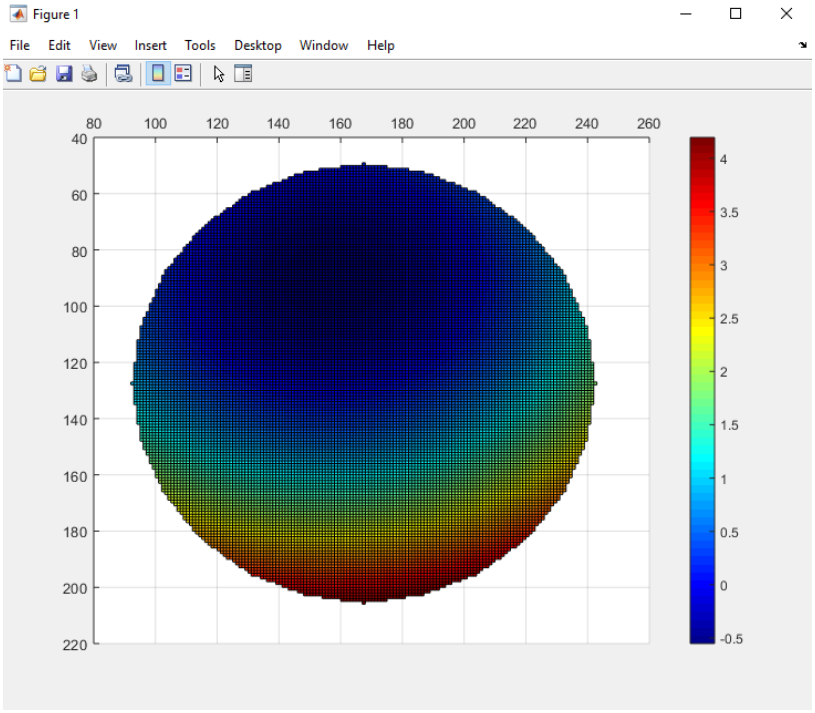


Figure 41 - Surface du miroir concave reconstruit

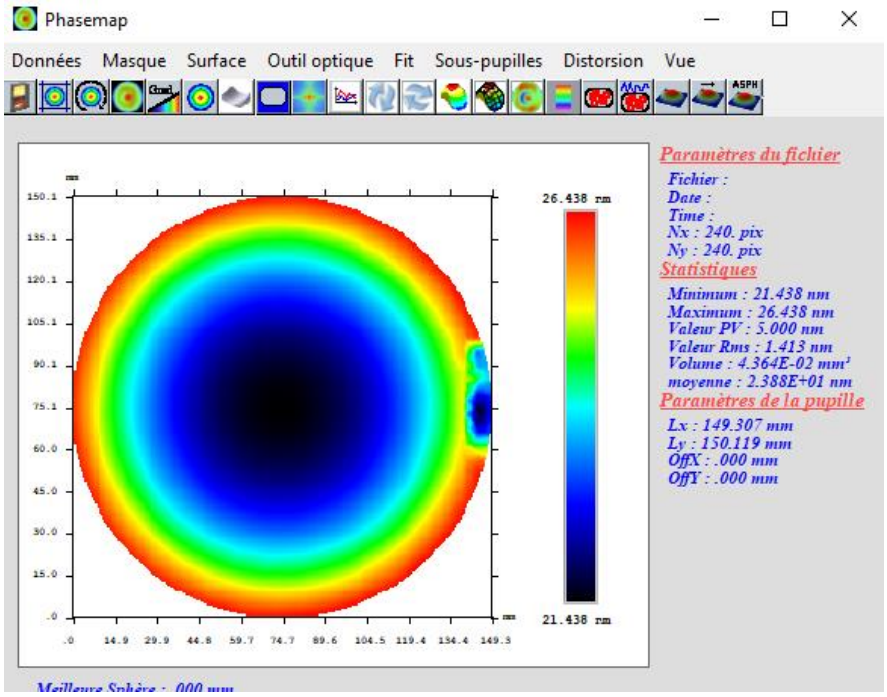


Figure 42 - Surface du miroir concave mesuré à la 3D

Pour tenter de palier à ces erreurs de surface, j'ai utilisé la reconstruction du miroir plat comme référence et je l'ai soustraite de la surface du miroir concave reconstruit. Les deux cartes n'étant pas centrées de la même manière, le résultat donne une carte ovale avec certains bords non considérés. Néanmoins, la surface obtenue possède une géométrie plus proche de la surface d'origine que la carte de départ. Au niveau du PV, celui-ci n'est pas comparable entre les deux cartes en raison du manque de certaines parties du miroir obtenu.

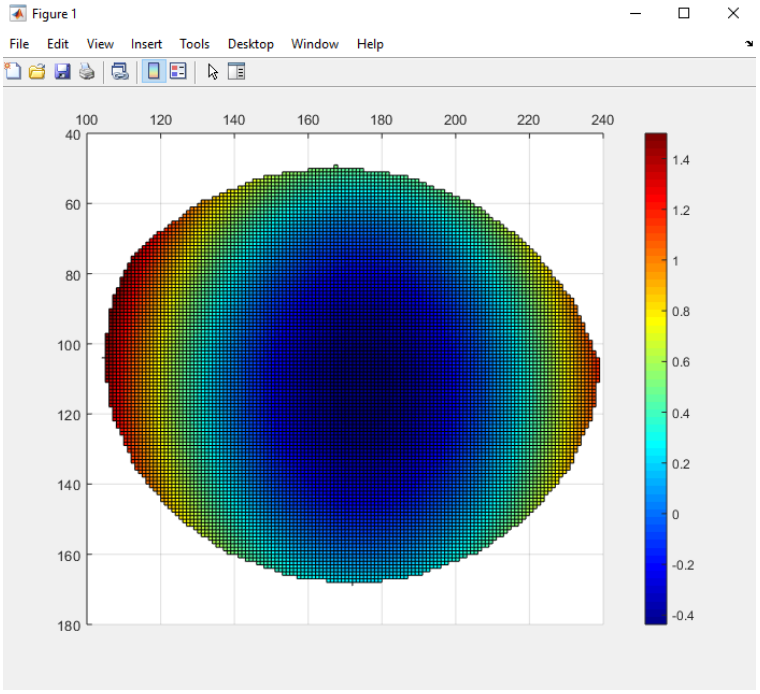


Figure 43 - Surface du miroir concave reconstruit en ayant retiré la reconstruction du miroir plat comme référence

À nouveau, l'analyse des hautes fréquences ne peut pas être discutée dans ce cas-ci, au regard des raisons évoquées ci-dessus.

Avant de conclure, un dernier point doit être discuté. Pour obtenir ces résultats, un facteur multiplicateur de -1 a dû être appliqué aux pentes de l'axe y afin de ne pas obtenir la surface en forme de selle de cheval présentée ci-dessous. L'hypothèse principale est que l'erreur est due au fait que la hauteur dépasse la distance de focus du miroir, et que seules les pentes de l'axe y seraient impactées en raison de la disposition de l'instrument.

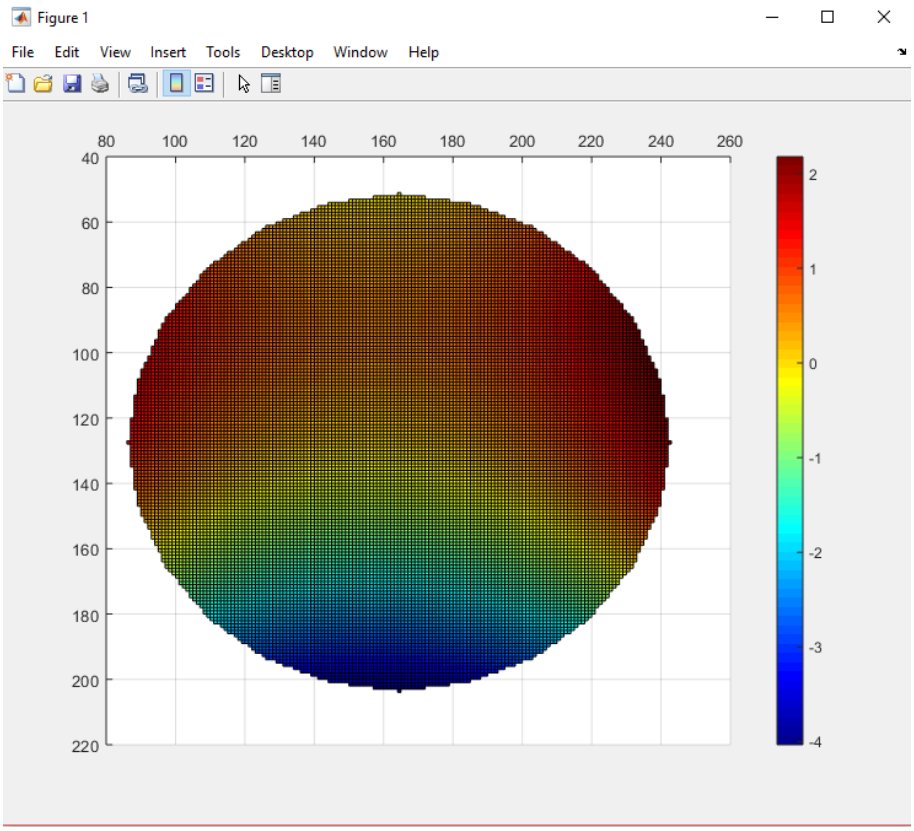


Figure 44 - Surface du miroir concave reconstruite sans le facteur -1

Pour conclure après observation de ce qui a été obtenu, nous pouvons constater que la reconstruction de basses fréquences par déflectométrie infrarouge est effectivement victime d'erreurs conséquentes comme introduit dans la section théorique de ce mémoire. De plus, aucune information concrète au sujet des hautes fréquences ne peut être retirée de ces cartes. Cependant, à l'aide de la dernière version du programme développé, l'analyse des hautes fréquences peut être faite et est discutée dans la prochaine section.

Le retrait d'une surface de référence se révèle être une idée intéressante. Cependant, comme nous l'avons vu, une référence trop petite ou centrée différemment ne couvre pas la surface mesurée. Il serait donc intéressant d'avoir accès à une surface plate couvrant la totalité du champ de vue afin de réaliser à nouveau ce genre de manipulations.

Enfin, une analyse des prochains résultats sur la nécessité du facteur multiplicateur dans le cas d'un dépassement du focus pourrait être intéressante afin de démontrer l'hypothèse présentée ci-dessus.

3. Analyse des performances

L'analyse des performances du programme est à nouveau divisée selon le temps de travail, l'interface et les erreurs dans le programme.

Le temps de travail du programme est équivalent à la somme de ceux présentés dans les deux sections précédentes (environ 160 secondes). Néanmoins, selon le choix du dossier, des paramètres sur l'interface présentée précédemment et le masquage de l'image pour l'intégration, le temps de travail peut prendre quelques secondes comme plusieurs minutes supplémentaires en fonction de la réactivité de l'utilisateur. Le programme reste néanmoins largement plus rapide qu'une mesure 3D de ces miroirs.

Concernant l'interface, maintenant que l'on sait que le programme fonctionne correctement, la seule fenêtre de résultats conservée reste celle du résultat final de la reconstruction. Cependant, au cours du programme, d'autres fenêtres apparaissent. Une première est l'interface de paramètres qui a été introduite et qui permet d'avoir la main sur la hauteur et le champ de vue de la caméra à cette même hauteur. Comme nous pouvons le constater, la valeur est toujours encodée manuellement. Cela s'explique par le fait que la sonde et sa mesure automatique n'ont été disponibles que lors du développement de la dernière version du programme. Évidemment, dans la prochaine section, son ajout automatique aux paramètres est présenté. Les autres fenêtres sont celles du choix du dossier et du masquage et ont déjà été présentées.

Enfin, concernant les erreurs dans le programme, une première erreur importante qui n'avait pas été repérée dans les erreurs des calculs de pente a été corrigée, rendant les résultats de reconstruction davantage plus solides. Il s'agit des positions y de chaque pixel de la surface qui étaient erronées suite aux axes différents entre Matlab et notre projet comme présenté antérieurement.

Une seconde source d'erreur qui a pu être éliminée est celle résultant des approximations et hypothèses émises quant au déplacement du fil, au champ de vue de l'image et à la hauteur. Ces hypothèses ont simplement été remplacées par des valeurs concrètes et mesurées manuellement.

Une troisième source d'erreur repérée au cours du développement de ce programme est que l'aspect RGB des images a complètement été oublié. Ainsi, seule une des trois composantes a été utilisée pour déterminer le centre de masse du fil sur chaque pixel de la surface. Cette erreur est évidemment rectifiée dans la version présentée dans la section suivante.

Enfin, une dernière potentielle source d'erreur reste une utilisation inappropriée du programme par l'utilisateur que ce soit au niveau des dossiers ou des images choisies comme présenté dans la section « Calcul des pentes ».

Octopus

Dans cette dernière section consacrée à la présentation de l'évolution de mon programme de reconstruction d'une mesure de surface par déflectométrie infrarouge, je présente la version finale de mon programme.

Afin de marquer le coup et, mais aussi afin que la société AMOS puisse employer un nom plus court que le titre de ce mémoire pour en parler, j'ai donné à cette dernière version le nom « Octopus ». Ce nom fait référence au personnage de comics Dr Octopus et à ses bras mécaniques, bras mécanique que l'on retrouve dans la structure de l'instrument développé par Monsieur Baron

Dans cette section comme dans les précédentes, il y a trois sous-sections. Tout d'abord, je présente le code réalisé pour cette dernière version. Ensuite, je présente les différents résultats que j'ai obtenus grâce aux nouvelles récoltes de données de Monsieur Baron. Enfin, je présente les performances d'Octopus.

1. Présentation du programme

Cette dernière version est différente de celles déjà présentées précédemment, car elle a dû être complètement restructurée afin qu'une interface puisse lui être associée. Pour cela, j'ai utilisé l'application GUIDE de Matlab qui permet la création de fenêtres d'interface. Comme vous pouvez le constater dans l'Annexe 5, chaque fenêtre de l'interface possède son propre script. La structure de base de ce script est présentée dans le premier point de cette annexe afin que le lecteur puisse se faire une idée des lignes de code qui ont été développées.

Cependant, lors de la création d'une interface à l'aide de GUIDE, un script de base est généré automatiquement. Ce dernier est composé de nombreuses fonctions qu'il faut compléter de nos lignes de code afin de rentre l'interface utile. Mais, le problème d'une fonction dans Matlab est qu'un espace de travail (là où sont enregistrées les données de travail) propre lui est attribué. Il a donc fallu tout au long du programme, que je présente ci-dessous, commencer par charger les données des fichiers « infos.mat » et « datas.mat » associés respectivement au dossier des scripts et du miroir reconstruit dans chaque nouvelle fonction activable.

L'interface que j'ai développée est une interface qui pourrait être qualifiée de linéaire. Cette interface démarre d'une première fenêtre qui est remplacée par une nouvelle fenêtre et ainsi de suite tout au long des calculs à réaliser. De plus, cette linéarité est divisée en deux parties. En effet, comme je le présente un peu plus loin, la première fenêtre du programme permet de réaliser soit le calcul des pentes sur base des données, soit la reconstruction de la surface sur base des pentes. Ce choix a été fait afin de permettre l'utilisation du programme NDA, utilisé couramment par la société AMOS, entre les

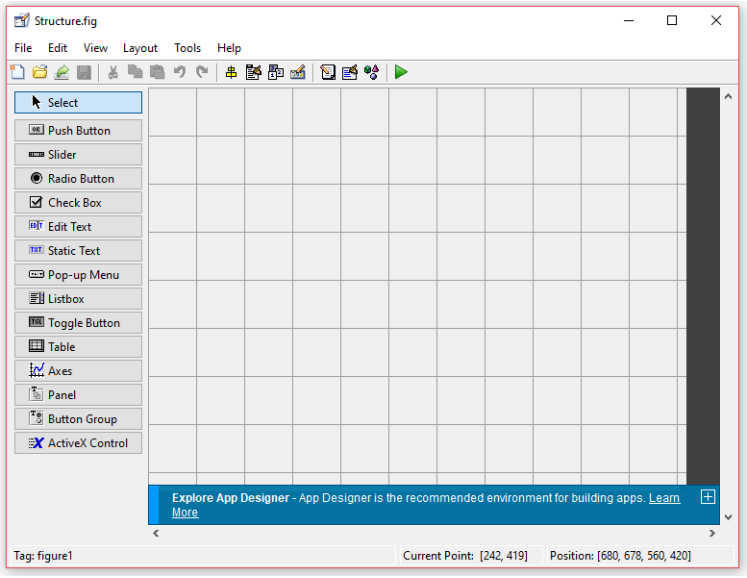


Figure 45 - Fenêtre de l'application GUIDE lors du démarrage de la réalisation d'une nouvelle interface

deux parties et ainsi permettre à l'utilisateur un traitement des images des pentes avant la reconstruction de la surface.

La première fenêtre mon programme est liée au script « Octopus.m ». Cette fenêtre de présentation permet à l'utilisateur de faire un choix entre le calcul des pentes d'un miroir, le calcul de la surface d'un miroir ou simplement de fermer le programme.

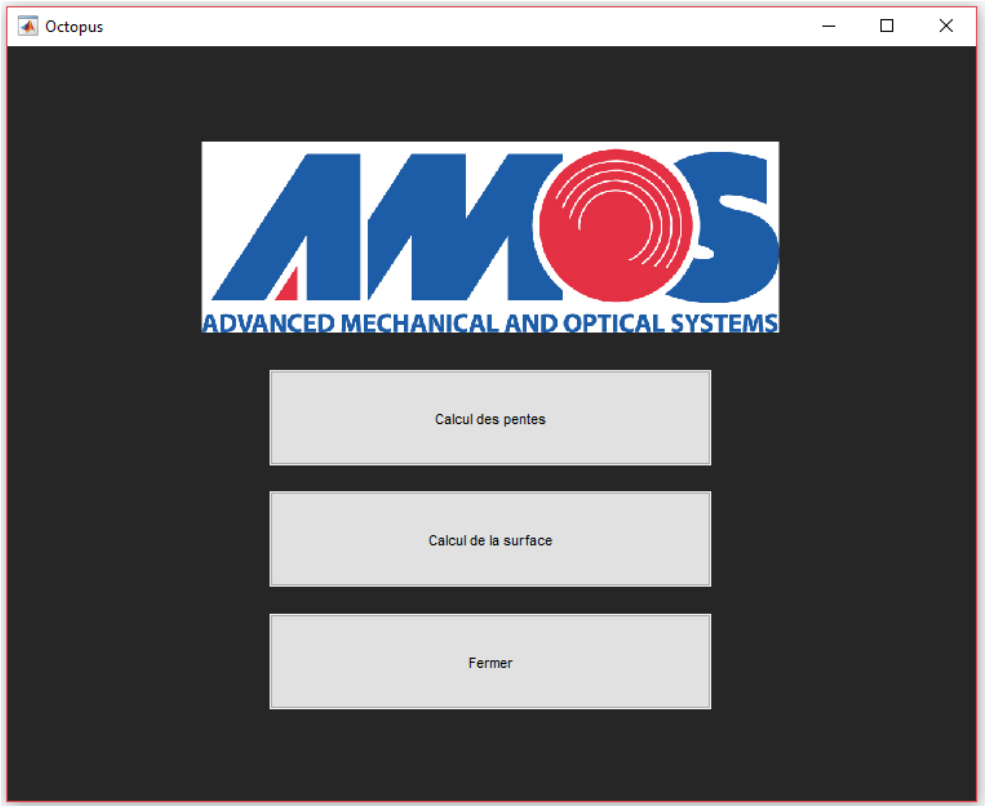


Figure 46 - Première fenêtre du programme liée au script "Octopus.m"

Comme vous pourrez le remarquer tout au long de la lecture de cette section, vous verrez que des éléments sont récurrents dans chaque script. Premièrement, le logo de la société AMOS est généré

automatiquement sur l'interface. Deuxièmement, chaque nouvelle fenêtre s'ouvre aux dimensions de la précédente excepté cette première fenêtre. Troisièmement, le bouton de fermeture de la fenêtre est codé de manière à fermer le programme le plus proprement possible afin de ne pas laisser trainer un fichier « datas.mat » inutile ou rempli de variables non nécessaires. Enfin, dans toutes les fenêtres (excepté celle-ci), un bouton « Précédent » est introduit et permet à l'utilisateur de revenir à la fenêtre précédente en conservant uniquement les données calculées jusqu'à cette précédente fenêtre. Les deux derniers boutons présentés sont accompagnés d'une fenêtre demandant à l'utilisateur une validation de l'opération à effectuer.

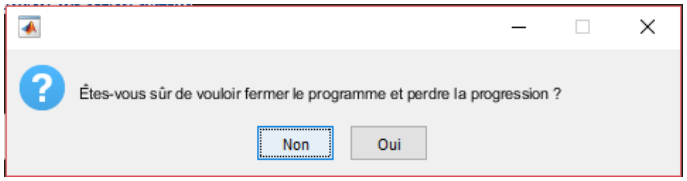


Figure 47 - Message de confirmation de la manipulation à effectuer

En choisissant l'option « Calcul des pentes », l'utilisateur est dirigé vers une nouvelle fenêtre liée au script « Ouvrir.m ». Cette fenêtre permet à l'utilisateur de sélectionner le dossier à traiter à l'aide du bouton « Sélectionner dossier images ». L'adresse du dossier s'affiche alors sur la fenêtre. Il ne reste plus qu'à sélectionner le bouton suivant pour passer à la fenêtre suivante.

Concernant le bouton « Suivant », il y a plusieurs choses à savoir. La première est que si le dossier sélectionné ne contient pas de dossier « x » et « y », un message d'erreur s'affiche et l'utilisateur est invité à sélectionner un autre dossier. Pareillement, si l'utilisateur venait à modifier le chemin du dossier image dans la barre éditable et que cette adresse n'existe pas, un message d'erreur s'affichera. Deuxièmement, entre la sélection du bouton suivant et l'affichage de la fenêtre suivante, le programme importe les informations des dossiers images et les sauvegarde dans un fichier « datas.mat ». Contrairement aux précédentes versions, les images ne sont plus converties en JPEG puis lues une à une pour déterminer le centre de masse du fil ; toutes les informations des images seront entrées dans deux matrices 3D (une pour les x, l'autre pour les y) dont le troisième indice correspond au numéro de l'acquisition de l'image lors du scan. Enfin, une petite barre de progression informe l'utilisateur sur l'avancée de l'importation des images dans les deux matrices.

La fenêtre suivante est liée au script « Parametres.m ». Sur cette fenêtre, le programme affiche automatiquement les paramètres liés à la position de la caméra, à la hauteur mesurée à l'aide de la sonde, aux largueur et longueur réelles de la surface, à la position réelle du pixel 1 de la surface (situé en bas à gauche de l'image), à la taille des pas, qui depuis la dernière version a été modifiée par Monsieur Baron, ainsi que les positions de départ du fil lors des différents scans. Le but de cette fenêtre est d'observer les différents paramètres de mesure et de les contrôler avant de lancer la suite du programme. Afin, que ces valeurs ne soient pas altérées par l'utilisateur, les cases d'édition ont été rendues inactivables.

En ce qui concerne les largeur, longueur et coordonnées du pixel 1, celles-ci sont calculées automatiquement sur base de la hauteur de prise des images. Pour ce faire, j'ai réalisé une mesure manuelle de la surface à une hauteur donnée. Les paramètres obtenus sont les suivants : une hauteur de 709 mm, une largeur de 309 mm, une longueur de 325 mm et les coordonnées du pixel 1 (-31 mm ; 0 mm). Ces valeurs permettent de déterminer l'orientation de la caméra et ainsi le champ de vue qu'elle observe pour d'autres hauteurs.

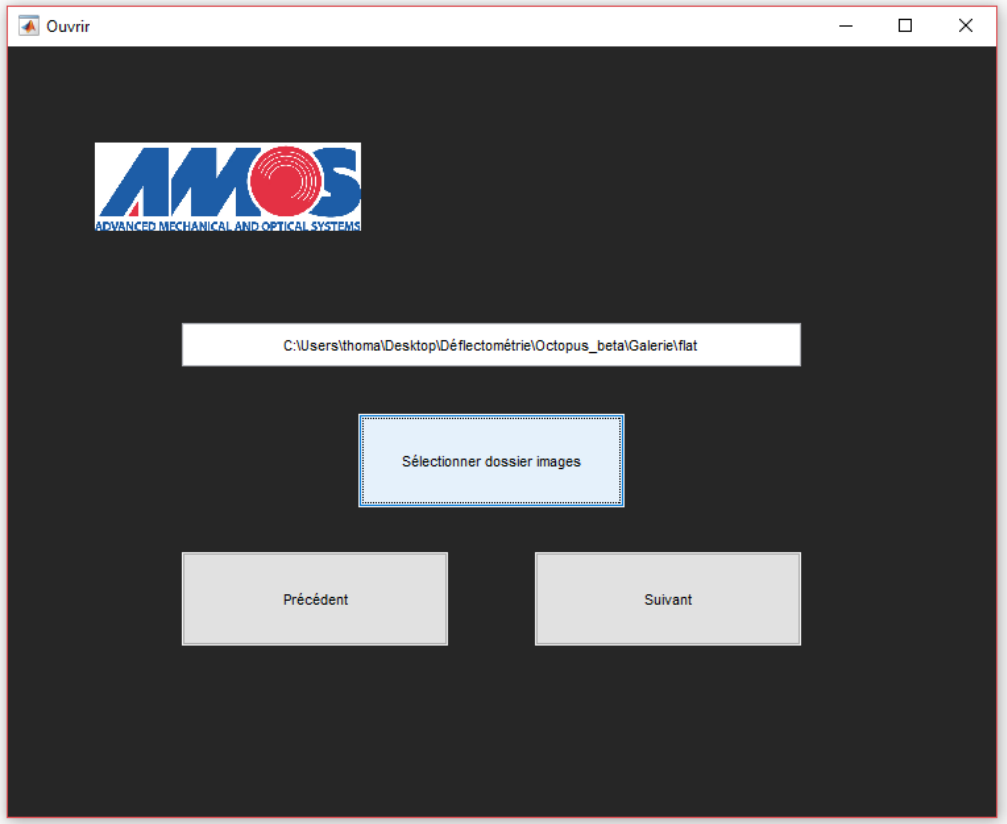


Figure 48 - Fenêtre du programme liée au script "Ouvrir.m"

Figure 49 - Fenêtre du programme liée au script "Parametres.m"

Le bouton « Suivant » de cette fenêtre permet, en plus de passer à la fenêtre suivante, de réaliser certains calculs. Tout d'abord, le programme réalise une mesure du centre de masse du fil pour chaque pixel avec une valeur seuil de l'intensité égale à zéro. Ensuite, le programme détermine un pixel de référence pour l'intensité. Ce pixel, dont l'écart entre la plus basse et la plus haute intensité qu'il contient est le plus grand de tous, permet, comme présenté à la fenêtre suivante, de modifier la valeur seuil du centre de masse et donc le centre de masse calculé. Enfin, les coordonnées de chaque pixel de la surface sont déterminées. L'évolution de ces différentes étapes est indiquée à l'utilisateur par une fenêtre de progression commentée.

La fenêtre « Centre_de_masse » est liée au script du même nom. Cette fenêtre a pour but de permettre à l'utilisateur d'influencer la valeur seuil du centre de masse. À l'aide du slider à droite de la fenêtre et du graphique d'intensité du pixel de référence aux cours des acquisitions, l'opérateur peut sélectionner la zone d'intérêt pour le futur calcul des pentes. L'image de gauche (centre de masse du fil le long du scan en x) et celle du centre (le long du scan en y) vont ainsi évoluer à chaque modification de la valeur. Cette valeur est affichée à titre indicatif sous le slider. Enfin, un bouton « Intensité maximum » permet d'afficher les cartes, non plus en fonction du centre de masse mais bien du maximum d'intensité dans chaque pixel.

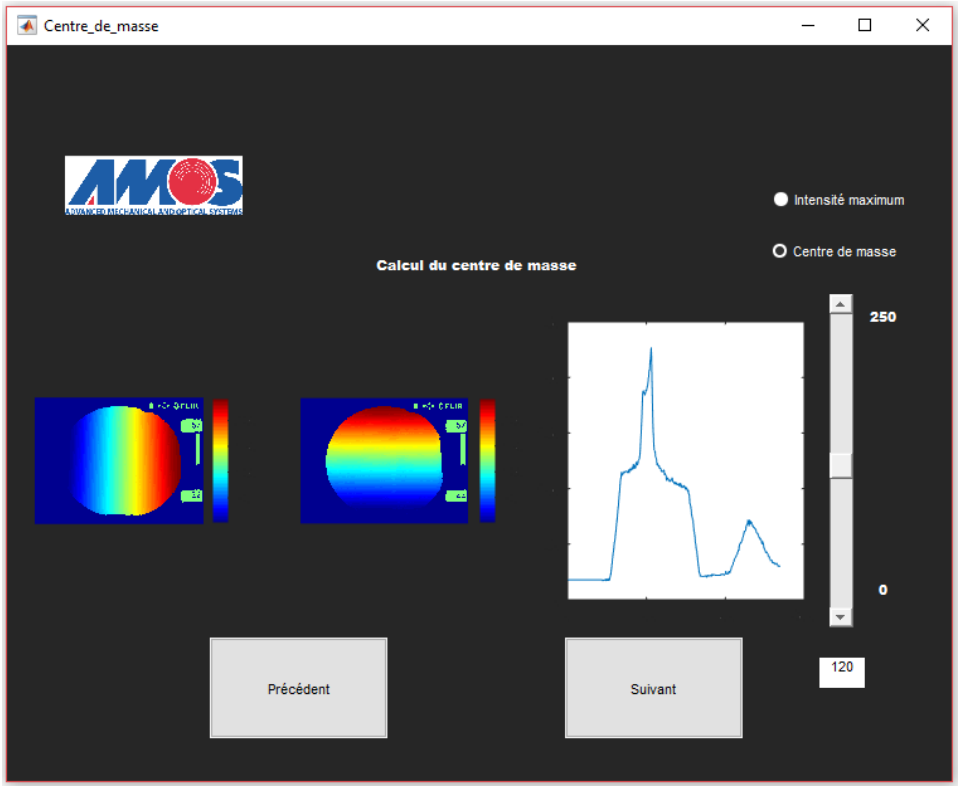


Figure 50 - Fenêtre du programme liée au script "Centre_de_masse.m"

Comme vous pouvez le constater dans les lignes de codes, un script identique est appelé lors d'un mouvement du slider et de la sélection du bouton « Centre de masse ». Ce script a été développé séparément du script « Centre_de_masse.m » afin de ne pas alourdir le programme de lignes de code répétées. Ce script est également présenté dans l'Annexe 5.

Enfin, en cliquant sur le bouton « Suivant », le programme réalise le calcul des pentes du miroir avant d'afficher la fenêtre suivante.

La dernière fenêtre du calcul des pentes, initié à la fenêtre de départ, présente les pentes calculées du miroir. Cette fenêtre affiche sur l'image de gauche les pentes calculées en x et, à droite, les

pentes calculées en y. Deux autres boutons importants sont situés sur la fenêtre. Le premier est le bouton « Distorsion ». Celui-ci permet d'estimer ou non les positions réelles des pixels de la surface suite à la distorsion de l'image sous la forme d'une approximation plus que basique en forme de trapèze. Le but de ce bouton est de corriger les erreurs liées à la distorsion à petite échelle pour le calcul des pentes. Bien que très basique, cette fonctionnalité n'est qu'un premier pas vers une amélioration prochaine du calcul de la distorsion. Le second bouton permet d'enregistrer les deux cartes de pentes au format TXT afin de les rendre lisibles avec le logiciel NDA précédemment évoqué.

En cliquant sur le bouton « Suivant », l'opérateur est redirigé vers la fenêtre d'accueil du programme. Le fichier « datas.mat » contenant toutes les données utilisées lors du calcul des pentes est nettoyé des informations qui ne sont plus utiles pour la suite du travail, à savoir la reconstruction de la surface.

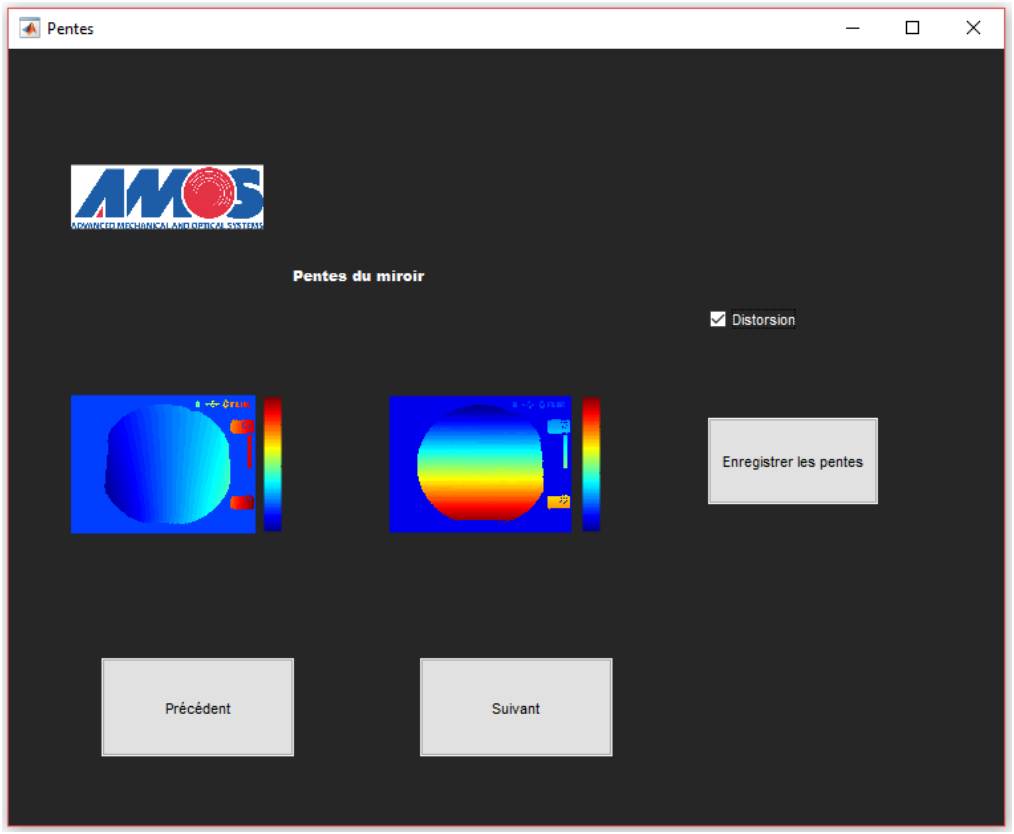


Figure 51 - Fenêtre du programme liée au script "Pentes.m"

Muni de ces deux cartes de pentes, l'utilisateur peut utiliser le programme NDA fréquemment utilisé par la société AMOS afin de réaliser un traitement de leurs images. Le traitement d'image utile à accomplir se divise en deux parties. La première consiste à réaliser un masquage des cartes afin de sélectionner la zone qui sera traitée par la suite. Cette zone sera aussi celle utilisée pour la reconstruction lors du retour sur le programme Octopus. La seconde partie du traitement d'image consiste à sélectionner les polynômes de Zernike qui sont retirés. Comme présenté antérieurement, afin de réaliser une analyse correcte des hautes fréquences comme l'équipe du projet SLOTS l'a réalisée, les 10 premiers termes de Zernike doivent être retirés sur les cartes des pentes. Cependant, il ne tient qu'à l'opérateur de modifier cette sélection de termes à retirer tant qu'il reste conscient de la qualité générale d'une mesure déflectométrique sur d'éventuelles basses fréquences conservées. Il est important de souligner que le logiciel NDA lit les valeurs en nanomètre. Cependant, comme il s'agit de valeurs de pente, ces unités sont erronées et ne doivent simplement pas être prises en compte.

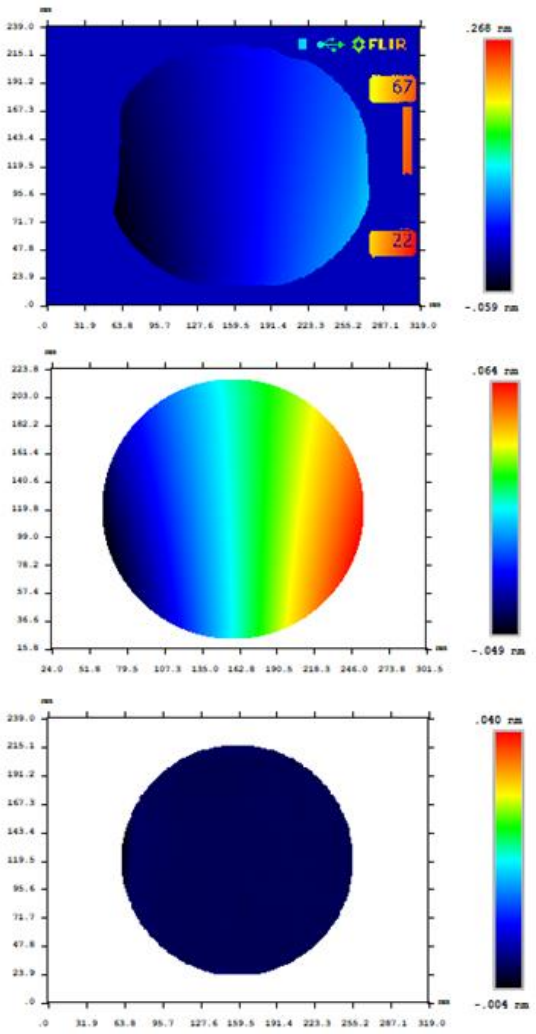


Figure 52 - Image des pentes en x affichée dans NDA, ensuite masquée et finalement avec les 10 premiers termes de Zernike retirés

Lorsque toutes les manipulations ont été réalisées et à nouveau sauvegardées en format TXT, l'opérateur peut retourner sur le programme Octopus afin de débuter la reconstruction de la surface en cliquant sur le bouton « Calcul de la surface » de la fenêtre d'accueil.

La première fenêtre affichée dans cette deuxième partie du programme est celle qui demande à l'opérateur de sélectionner les fichiers des pentes x et y ainsi que le fichier « datas.mat » correspondant. Cette fenêtre est liée au script « Ouvrir2.m » présenté en Annexe 5.

En cliquant sur le bouton « Suivant », le programme importe les données dans Octopus. Attention cependant que, si un ou plusieurs fichiers ne sont pas sélectionnés ou sont inexistants, Octopus affichera un message d'erreur invitant l'utilisateur à une vérification. De plus, un message d'erreur s'affiche également si les 3 fichiers ne se situent pas dans un même dossier. Cette précaution intervient afin que les données de l'utilisateur ne soient pas confondues avec les données d'une autre mesure.

La fenêtre suivante est liée au script « Pentes2.m » et permet la visualisation et la vérification des cartes des pentes sélectionnées par l'utilisateur pour la reconstruction.

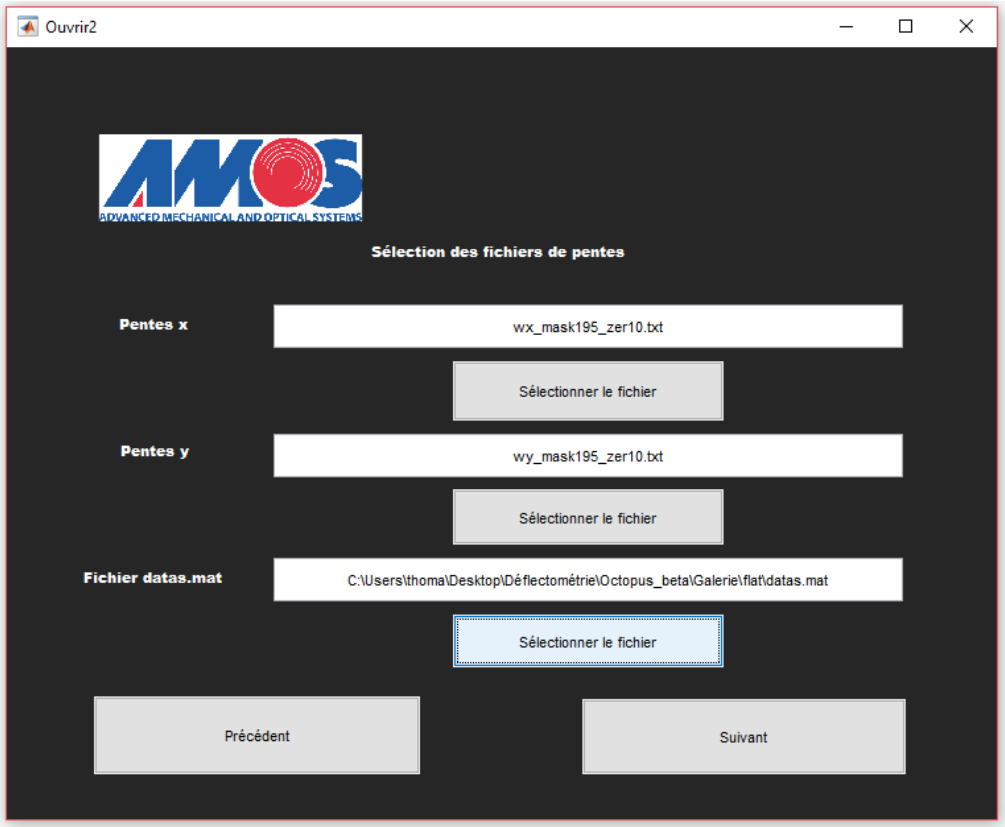


Figure 53 - Fenêtre du programme liée au script "Ouvrir2.m"

Deux menus déroulants sont proposés à l'opérateur. Le premier concerne le dépassement ou non du focus, déjà évoqué dans la section précédente. Afin de permettre à l'opérateur une reconstruction correcte de son miroir, celui-ci peut indiquer le dépassement potentiel. Le deuxième menu permet de régler la vitesse d'intégration. En effet, le temps de reconstruction est lié essentiellement à l'intégration de Southwell qui est itérative. Ainsi, en influant sur le nombre d'itérations, le temps d'intégration sera soit plus court, soit plus long. Trois options s'offrent à l'utilisateur : lente, moyenne ou rapide, options qui correspondent respectivement au nombre d'itérations divisé par 1, 10 ou 25.

Lorsque le choix des paramètres est effectué, l'utilisateur peut sélectionner le bouton « Suivant » qui initie la reconstruction de la surface. Contrairement aux versions précédentes à cette étape du travail, l'utilisateur n'est plus invité à réaliser un masquage de la surface à intégrer puisque les cartes des pentes ont déjà été masquées dans NDA. L'intégration se réalise donc sur base de ces masquages, et la reconstruction débute par la détermination de la zone à intégrer.

Un autre paramètre du programme a également été modifié ; il s'agit des distances entre les pixels. Celles-ci étaient considérées comme fixes dans les versions précédentes du programme. Suite à l'application d'une correction de distorsion, cette dernière affirmation devient fausse. Dorénavant, la distance entre deux pixels de la surface est calculée sur base de la différence de leurs coordonnées.

Enfin, afin de permettre à l'utilisateur de visualiser la progression de la reconstruction, une barre de progression s'affiche à l'écran.

La dernière fenêtre du programme propose la carte de la surface reconstruite. Cette fenêtre est liée au script « Surface_Finale.m ». Un bouton « Enregistrement de la surface » permet à l'utilisateur d'enregistrer la surface au format TXT afin de l'utiliser pour un traitement d'image dans le logiciel NDA.

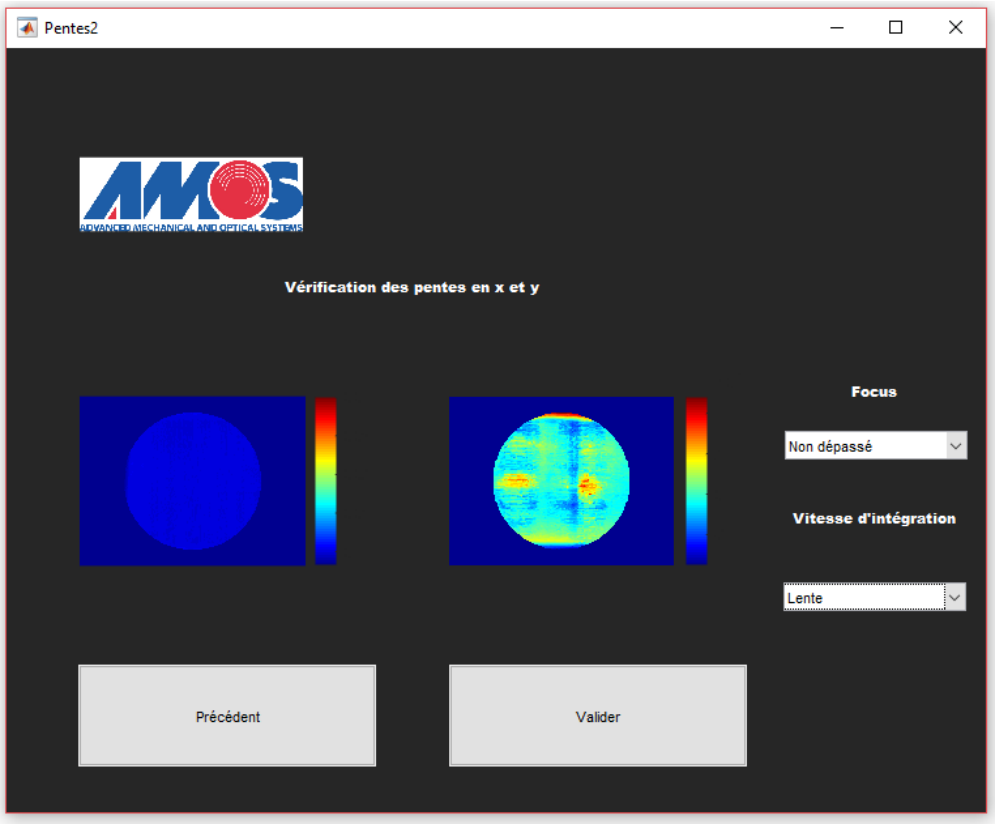


Figure 54 - Fenêtre du programme liée au script "Pentes2.m"

Enfin, le bouton « Fin de reconstruction » redirige l'utilisateur sur la fenêtre d'accueil d'Octopus tout en nettoyant à nouveau le fichier « datas.mat » des valeurs inutiles. Les seules informations conservées sont :

- Les dimensions des cartes ;
- Les coordonnées des pixels de la surface ;
- Les cartes de pentes masquées et non masquées ;
- La carte de la surface reconstruite.

Cette dernière étape clôture la présentation des capacités de la dernière version du programme Octopus réalisé dans le cadre de ce mémoire. Le travail d'analyse de l'image obtenue n'est pas terminé pour autant. En effet, alors que la carte est reconstruite, l'utilisateur a l'opportunité de l'ouvrir dans NDA pour effectuer le traitement d'image. Il peut à nouveau réaliser un masquage pour délimiter les zones à traiter et, par exemple, retirer à nouveau les 10 premiers polynômes de Zernike de la surface pour réaliser des analyses similaires à celles obtenues par le projet SLOTS.

2. Résultats obtenus

Les résultats présentés ci-dessous proviennent de la mesure de trois miroirs. Les deux premiers miroirs sont ceux qui ont déjà été mesurés lors de la section précédente. Le troisième miroir est un miroir en SiC (Carbure de silicium) sur lequel une TWF (une fonction d'usure) a été réalisée à l'aide d'un patin. Seule l'analyse de la fonction d'usure sera réalisée ici et non celle de la surface complète du miroir.

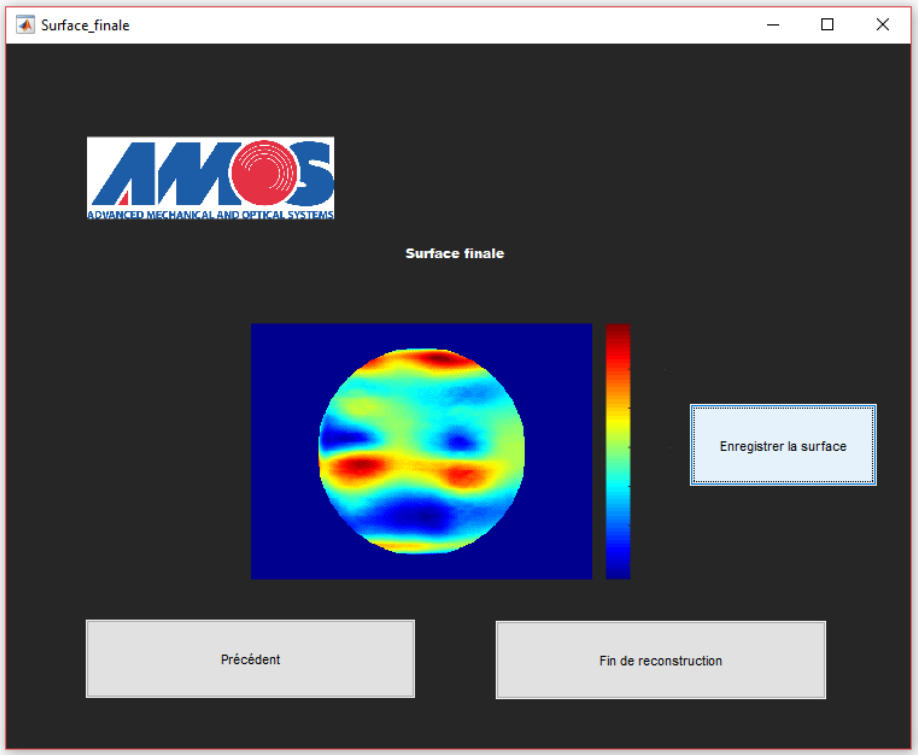


Figure 55 - Fenêtre du programme lié au script "Surface_Finale.m"

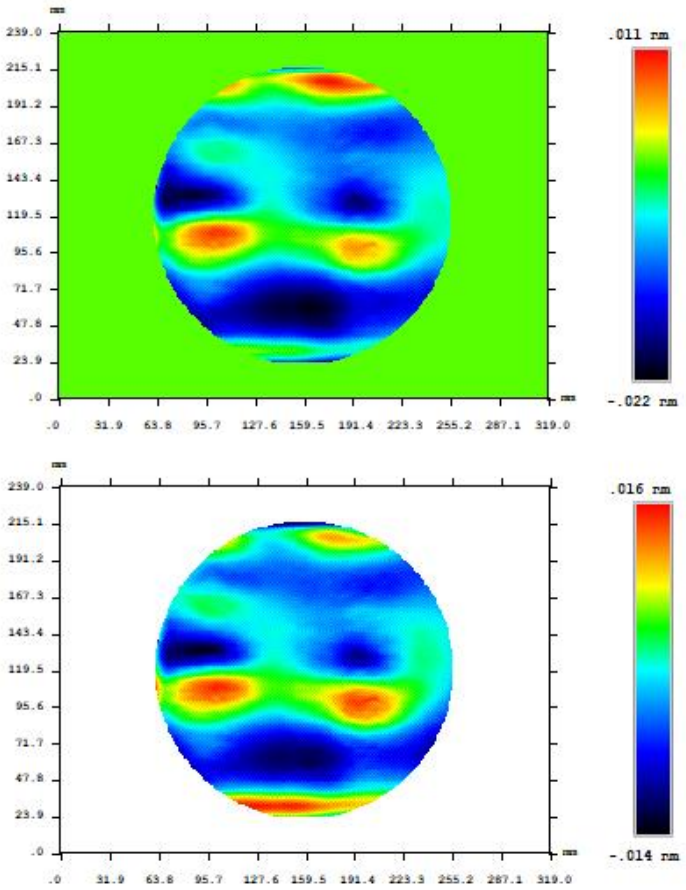


Figure 56 - Évolution de la carte de la surface avant et après le masquage et retrait des 10 premiers polynômes de Zernike sur NDA

En ce qui concerne la mesure du miroir plat en aluminium, je ne présente ici que les résultats liés à l'analyse des hautes fréquences. En effet, la reconstruction des basses fréquences à l'aide de ce nouveau programme est globalement similaire à celle de la reconstruction précédente car la méthode de travail n'a pas évolué. Cette démarche sera identique pour le miroir concave en zérodur.

Autre information utile avant de débuter la présentation des résultats, l'analyse des hautes fréquences a été réalisé d'une manière similaire à celle du projet SLOTS, à savoir le retrait des 10 premiers polynômes de Zernike sur les cartes des pentes et la surface reconstruite. Enfin, afin de comparer ce qui est comparable, ces mêmes polynômes ont été retirés aux cartes des surfaces de miroirs mesurées par interférométrie ou machine 3D.

En ce qui concerne l'analyse des résultats des mesures des hautes fréquences du miroir plat, on peut constater facilement que ces mesures présentent des différences. Dans le cas de la mesure interférométrique, le PV s'élève à 1,7 µm. Par contre, dans le cas de notre mesure, le PV s'élève à 30 µm. Cependant, on peut tout de même constater des similarités sur les cartes : on peut voir les deux bosses rouges situées approximativement aux mêmes endroits ainsi que d'autres plus petites similarités dans la structure des cartes.

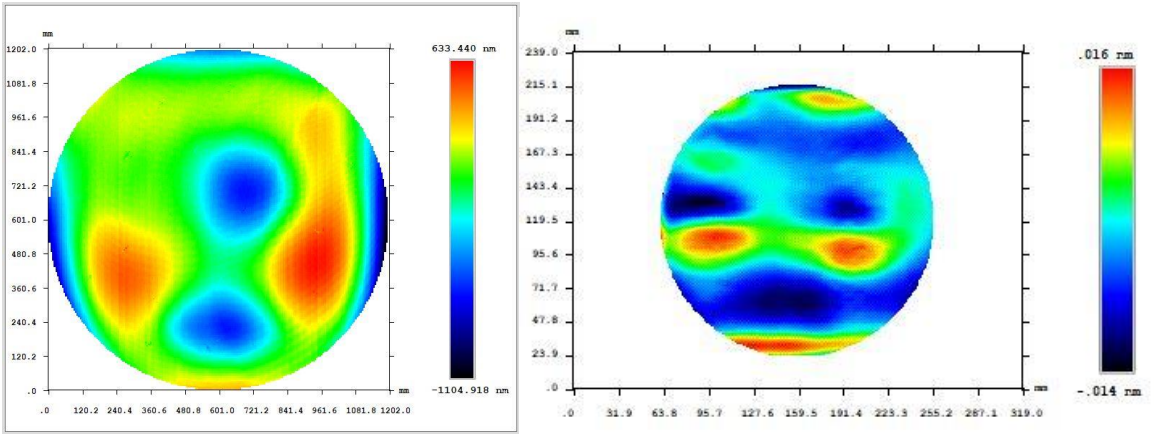


Figure 57 - Cartes des hautes fréquences du miroir plat par interférométrie (gauche) et déflectométrie infrarouge (droite)

Le résultat de l'analyse des hautes fréquences des mesures du miroir concave est visuellement nettement différent de celui du cas précédent. En effet, aucune structure commune n'est visible ou reconnaissable. Cependant, en analysant les valeurs obtenues plus attentivement, on constate que le PV en mesure 3D est de 79,7 µm alors qu'en déflectométrie infrarouge il est de 49 µm. En comparant l'écart obtenu ici avec celui obtenu sur le miroir plat, on constate que ceux-ci sont d'un ordre de grandeur similaire bien que opposés. Au vu de ces deux premiers résultats, l'erreur de mesure pourrait donc avoir une origine commune. La mesure du troisième miroir devrait nous indiquer si cette tendance et cette hypothèse se confirment.

En ce qui concerne précisément la mesure de ce troisième miroir ou plutôt de la TWF sur sa surface, l'analyse est présentée en deux parties. Dans un premier temps, les résultats de basses fréquences obtenus sont mis en relation avec ceux de la section précédente. La seconde partie présente les résultats des hautes fréquences.

En ce qui concerne les basses fréquences obtenues par 3D et par la mesure déflectométrique, la comparaison met en évidence la présence d'un facteur 20 entre les mesures. En effet, dans le cas de la mesure 3D, le PV est de 90,8 µm alors que, dans la mesure déflectométrique, le PV est de 1,922 mm. Cet ordre de grandeur n'est pas sans rappeler la mesure obtenue lors du calcul des basses fréquences du

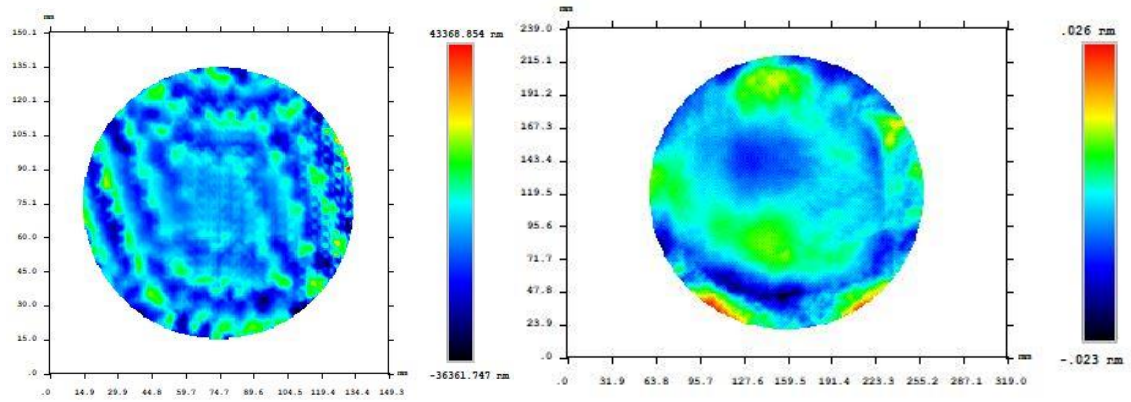


Figure 58 - Cartes des hautes fréquences du miroir concave par 3D (gauche) et déflectométrie infrarouge (droite)

miroir plat (pour rappel, le facteur était de 30). Le seul autre point commun qui relie ces deux surfaces est leur relative planéité. Dans le futur, il pourrait être intéressant de réaliser la mesure d'autres miroirs plans afin de vérifier si cette autre tendance se généralise. En effet, la mesure réalisée sur deux seuls miroirs ne permet pas d'émettre une règle générale mais plutôt des hypothèses.

L'autre fait observable est que les formes générales des deux mesures semblent plutôt similaires avec un creux au centre et certaines parties du bord relevées. La position des bords relevés est néanmoins décalée d'un angle de -45°. En combinant ce résultat avec celui du paragraphe précédent, on peut à nouveau en conclure que l'exactitude et la précision des résultats de basses fréquences obtenus en déflectométrie sont toutes relatives comme la théorie semblait l'indiquer.

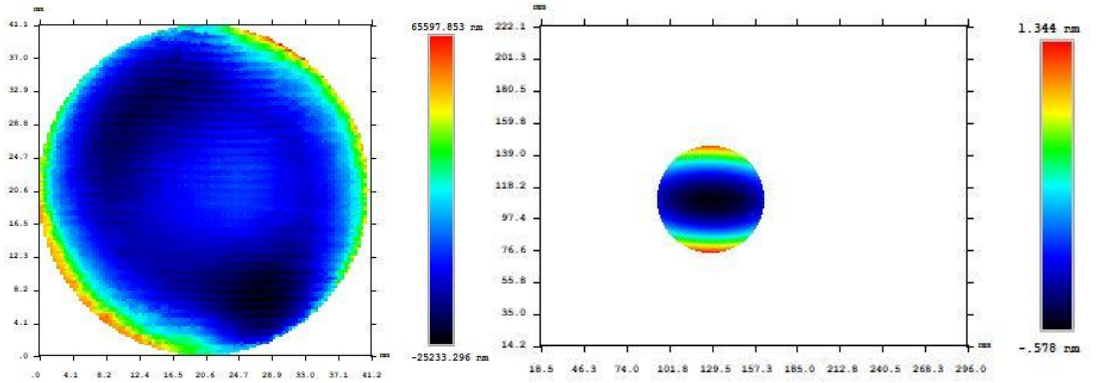


Figure 59 - Cartes des basses fréquences de la TWF sur le miroir en SiC par 3D (gauche) et déflectométrie infrarouge (droite)

Enfin, les derniers résultats de mesures que l'on peut extraire de celles réalisées avec Monsieur Baron concernent l'analyse des hautes fréquences sur le miroir en SiC. Comme présenté dans la figure suivante, les visuels des deux surfaces sont totalement différents. Néanmoins, en comparant les valeurs des PV, on peut à nouveau constater un écart de l'ordre de ceux déjà obtenus. En effet, comme le PV à la 3D est de 61,4 µm et que celui avec la déflectométrie est de 81 µm, on obtient un écart de 20,4 µm ce qui est de l'ordre de grandeur des 30 µm d'écart obtenus précédemment. Cette troisième mesure vient donc compléter la tendance observée qu'une source d'erreurs est présente à une étape du dispositif et qu'elle influe selon un ordre de grandeur identique sur les mesures des hautes fréquences. On peut constater également que l'écart observé dans le cas du miroir en SiC se produit dans le même sens que dans le cas du miroir plat, c'est-à-dire une mesure déflectométrique plus élevée que la mesure interférométrique ou 3D, contrairement au cas du miroir concave. Cette observation permet à nouveau de relier les deux miroirs qui n'ont en commun que leur forme plane. À nouveau, d'autres mesures de

miroirs plans pourraient permettre de constater la généralisation de ces tendances pour ce type de forme de surface.

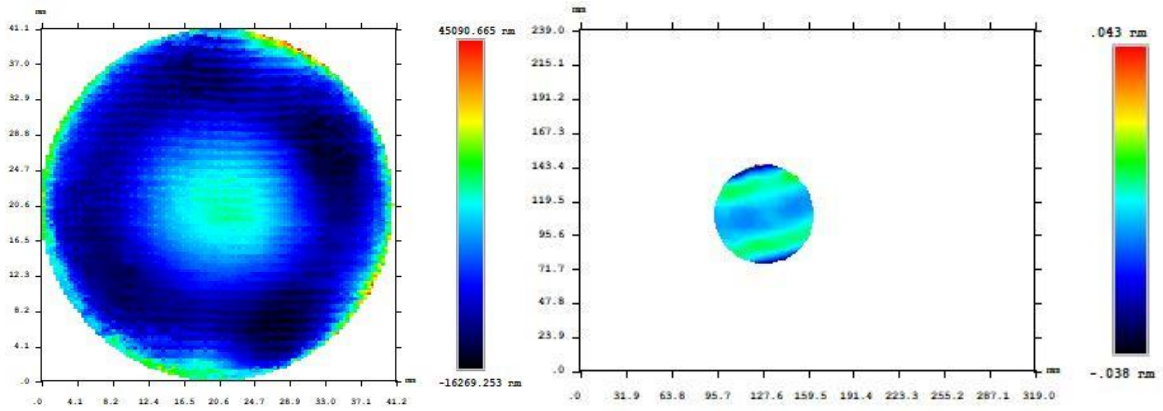


Figure 60 - Cartes des hautes fréquences de la TWF du miroir en SiC par 3D (gauche) et déflectométrie infrarouge (droite)

Nous avons discuté avec Monsieur Baron de la source d'erreur liée à cette tendance d'écart de 20-30 µm sur les PV des surfaces. Il m'a indiqué qu'il était possible que cette erreur provienne du déplacement du fil de son instrument comme il l'a indiqué dans son propre travail [5]. En effet, il a pu observer des erreurs de répétabilité dans le déplacement du fil chauffé dues à la présence d'hystérésis. Ces erreurs, qui se dessinent comme des erreurs de hautes fréquences, pourraient donc être effectivement à l'origine de cette erreur commune dans chacune des mesures des hautes fréquences. D'autre part, Monsieur Baron m'a indiqué que l'optique de la caméra n'est pas optimale. L'utilisation de cette caméra pourrait donc également être une source d'erreur.

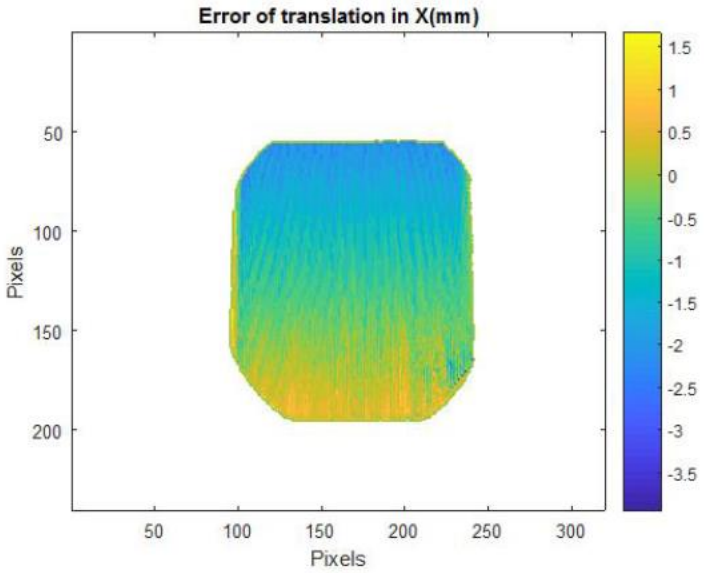


Figure 61 - Erreurs de répétabilité du déplacement du fil en x [5]

En conclusion de tous les résultats obtenus au cours de mon travail, il est intéressant de procéder à un récapitulatif de toutes les observations, ainsi que des hypothèses émises sur d'éventuelles tendances à investiguer plus en profondeur dans le futur.

Dans un premier temps, revenons sur les résultats des basses fréquences. Comme la théorie semblait l'indiquer, les basses fréquences sont considérablement affectées lors d'une mesure par

déflectométrie. Cependant, la mesure d'une référence plane assez grande qui pourrait être retirée aux futures mesures semblent être une piste à exploiter dans les prochains travaux réalisés avec cet instrument.

Ensuite, comme cela vient d'être présenté, une source d'erreur du même ordre est identifiée dans toutes les mesures par déflectométrie infrarouge des hautes fréquences des miroirs. Cette erreur pourrait provenir du déplacement du fil lors du scan et serait liée à sa répétabilité et donc à sa précision mais aussi à une qualité de caméra médiocre.

Enfin, des erreurs avec des ordres de grandeur similaires semblent apparaître sur des miroirs de même forme même s'ils sont de dimensions différentes. Il serait donc intéressant de mesurer d'autres miroirs plans afin de vérifier les tendances observées. D'autre part, il pourrait aussi être instructif d'observer si des miroirs de forme concave donnent des comportements similaires à celui mesuré par déflectométrique infrarouge.

3. Analyse des performances

L'analyse des performances du programme est à nouveau divisée en trois parties selon le temps d'exécution, l'interface et les potentielles sources d'erreurs.

Le temps de travail nécessaire à la réalisation de la reconstruction ne dépend principalement que de l'utilisateur. En effet, exception faite de l'intégration qui peut prendre de quelques secondes à plusieurs minutes, toutes les opérations entre les fenêtres de l'interface prennent moins de 10 secondes. Ainsi, le temps global de travail sur le programme Octopus dépendra principalement de l'utilisation que l'opérateur en fera.

En ce qui concerne l'interface, l'utilisateur du programme Octopus a maintenant beaucoup plus de possibilités de choix de travail en fonction de ses besoins. Dans les versions précédentes, seuls le réglage manuel et obligatoire de certains paramètres ainsi que le masquage de la surface étaient réalisables. L'opérateur a dorénavant la possibilité d'avoir un visuel sur l'avancement de chaque étape ainsi qu'un contrôle sur des paramètres comme la valeur seuil pour le centre de masse, une correction de distorsion, l'enregistrement des cartes en TXT, un retour en arrière suite à une erreur de manipulation, le réglage par rapport au dépassement ou non du focus pour l'intégration ou encore la vitesse d'intégration. Toutes ces options offrent à l'utilisateur une ergonomie nouvelle pour son travail.

Enfin, pour clôturer cette section, je présente les différentes erreurs dans le programme qui peuvent causer des inexactitudes voire une interruption du programme.

L'erreur observée lors de l'analyse des images en RGB décrite dans la section précédente a été corrigée dans cette dernière version. Ainsi, le risque d'erreurs sur les données est maintenant résolu à ce niveau.

L'utilisation du programme d'une manière inadaptée ou maladroite est de moins en moins un problème grâce aux contrôles de sécurité disposés tout au long du programme. En effet, la plupart des manipulations qui commandent la recherche d'un dossier ou d'un fichier dans l'ordinateur sont contrôlées un maximum afin d'éviter de potentielles erreurs, notifiées à l'utilisateur le cas échéant. De plus, un retour en arrière avec le bouton « Précédent » ou de fermeture de la fenêtre nettoie l'environnement ou les fichiers de toutes les informations obsolètes afin de ne conserver que l'essentiel et d'éviter toute confusion pour la suite.

Le bouton « Distorsion », présenté dans la fenêtre liée au script « Pentes.m », n'est qu'une ébauche d'un futur traitement d'image utile aux corrections des erreurs dues à la position et à

l'orientation de la caméra. Celui-ci n'est pas à prendre comme argent comptant mais comme le début d'une amélioration des résultats par la correction de cette source d'erreurs.

Enfin, la détermination automatique du champ de vue sur base des précédentes mesures et de la hauteur peut évidemment comporter des sources d'erreurs pour les calculs. Il serait donc intéressant de trouver à l'avenir une solution alternative basée sur une connaissance correcte de l'orientation, de la position et de l'ouverture angulaire de la caméra afin de déterminer correctement le champ de vue observé.

Améliorations, points d'attention et autres options envisageables

Dans cette dernière section, je présente toutes les pistes que j'ai pu découvrir au long de ces quelques mois de travail concernant le futur du projet que nous avons entamé avec Monsieur Baron. Dans les lignes qui suivent, je reviens donc sur des points qui ont été abordés précédemment mais j'aborde aussi de nouvelles idées qui pourraient être appliquées à notre instrument ou au programme Octopus.

1. Les points abordés précédemment

Le premier point présenté dans cette section porte sur l'introduction d'une mesure de référence dans les calculs de reconstruction comme évoqué précédemment. Rappelons-le, cette idée est née des suites de la mesure en basses fréquences du miroir concave. Après un premier résultat bien éloigné de la mesure 3D, j'ai tenté de retirer la mesure réalisée sur le miroir plat en aluminium en formulant l'hypothèse que sa reconstruction représentait la carte des erreurs commises. Visuellement, la forme du miroir semblait se rapprocher de la surface réelle. Cependant, en raison d'un manque de données et d'un centrage différent entre les cartes, le résultat obtenu est trop partiel.

Il pourrait donc être intéressant à l'avenir d'envisager de mesurer des miroirs ou des surfaces qui remplissent le champ de vue de la caméra. En comparant ces mesures avec des mesures 3D ou interférométriques, l'utilisateur pourra constater si une structure commune d'erreurs se dessine sur les cartes des pentes des miroirs mesurés. En effet, il est plus intéressant d'observer les erreurs sur les cartes des pentes, ces erreurs provenant davantage de la mesure déflectométrique des pentes que de la reconstruction. Cependant, si cette structure ne semble pas être similaire pour tous les miroirs mesurés, une analyse de l'évolution de la structure en fonction de la forme du miroir pourrait être envisageable. Ainsi, un modèle d'erreur pourrait être créé et appliqué aux cartes des pentes afin de les corriger.

À l'heure actuelle, cette analyse est irréalisable à cause du manque de miroirs remplissant le champ de vue de la caméra. En effet, comme nous l'avons vu précédemment, tous les miroirs mesurés ne remplissent pas entièrement les images prises par la caméra. Il faut donc attendre que de telles pièces soient disponibles.

Un second point à présenter dans cette section est l'analyse de la distorsion de la caméra. Comme évoqué précédemment, la position et l'orientation de la caméra rendent une image distordue de la surface mesurée. Dans ces conditions, il est impossible de réaliser une soustraction entre les cartes mesurées par interférométrie et celles mesurées par déflectométrie. Une carte d'erreurs ne peut donc pas être établie.

Comme présenté dans la section précédente, un premier dispositif a déjà été mis en place afin de pallier à ce problème. Cependant, celui-ci reste très insuffisant car il ne permet pas encore d'ajuster l'image sans distorsion mais simplement de donner une première évaluation basique des coordonnées de chaque pixel de la surface.

Une sérieuse option pour régler ce problème de distorsion pourrait être d'appliquer un écran d'Hartmann sur une surface remplissant le champ de vue afin de déterminer correctement la distorsion de l'image et, sur base de cette mesure, de corriger l'image. En effet, comme introduit précédemment, un écran d'Hartmann est un masque composé d'un quadrillage parfait de petits trous. En disposant une plaque chauffée derrière cet écran, la caméra infrarouge pourrait observer les différents trous présents dans son champ de vue. Ensuite, en comparant la mesure acquise avec la configuration réelle des trous, la distorsion de l'image pourrait être calculée. Enfin, sur base des résultats obtenus, il ne resterait plus qu'à appliquer la correction adéquate aux acquisitions de l'image afin d'obtenir des mesures sans distorsion.

Un troisième point consiste à déterminer le champ de vue observé par la caméra d'une manière plus propre que celle créée pour le programme Octopus. Ce point est lié au précédent par le fait qu'il s'agit de travailler sur l'analyse de l'image réalisée par la caméra infrarouge. Cependant, ici il ne s'agit pas de déterminer comment l'image est déformée mais plutôt quelle surface elle observe. Ainsi, en déterminant son champ de vue, les coordonnées de chaque pixel de la surface pourront être déterminées avec précision ce qui ne peut que rendre le calcul des pentes et l'intégration du gradient meilleurs.

Pour cartographier avec précision ce champ de vue, l'idée que je présente maintenant est basée sur un des motifs dynamiques présentés dans la section théorique de ce travail. En effet, le dispositif pourrait réaliser la mesure d'une plaque chauffée à l'aide d'un laser. Ainsi, en connaissant les positions exactes de la surface pointées par le laser et observées par chaque pixel de la caméra, il serait aisé de déterminer le champ de vue de l'image. D'autre part, cette idée pourrait aussi permettre de déterminer la distorsion de l'image acquise et d'évaluer la correction à appliquer.

Enfin, le dernier point à investiguer et déjà évoqué afin d'améliorer les résultats du dispositif est en train d'être mis en place : il s'agit de la deuxième configuration de l'instrument. Pour rappel, cette nouvelle configuration ne permet pas encore la prise d'acquisitions de miroirs suite à des problèmes liés à l'installation d'une nouvelle caméra et à un problème de sonde qui mesure la hauteur.

Effectivement, le calcul des pentes est réalisé à l'aide des Équations (9) comme nous l'avons vu dans la partie théorique. Il était aussi stipulé que ces équations étaient utilisées suite à la simplification des Équations (2) dans le cadre où la hauteur de la surface du miroir était largement inférieure à la distance entre le miroir et le plan caméra/fil. Sur base de ces affirmations, on peut donc supposer que les résultats obtenus à l'aide de cette approximation seraient meilleurs avec un dispositif où cette distance est plus grande.

Dans le premier dispositif, la distance entre le miroir et le plan fil/caméra était de l'ordre du décimètre alors que, dans le second, il est de l'ordre du mètre. Cette différence d'un ordre de grandeur pour réaliser des mesures similaires permettra de minimiser les erreurs obtenues lors des calculs des pentes lorsque le nouveau dispositif sera opérationnel. De plus, il pourrait être à nouveau intéressant de comparer les résultats des erreurs réalisées dans chacune des configurations afin de constater si un modèle se dessine en fonction de cette distance.

2. Les nouvelles idées

Dans cette sous-section, je présente de nouvelles idées qui n'ont pas encore été développées ou qui ont déjà commencé à être mises en place dans le cadre de notre projet avec Monsieur Baron.

La première possibilité d'amélioration pourrait être d'utiliser directement les Équations (2) avec des images prises par deux caméras au lieu des Équations (9) simplifiées. En effet, le problème de l'utilisation des Équations (2) est que celles-ci contiennent deux inconnues : la pente locale et la hauteur du miroir aux coordonnées (x_m, y_m). Cependant, si vous avez l'occasion de poser à un mathématicien ou à un physicien la question de savoir comment résoudre une équation à deux inconnues, celui-ci vous répondra que deux équations liant les inconnues sont nécessaires pour la résolution.

Comme les Équations (2) sont liées aux positions des pixels de la surface, à la position du fil chauffé et à la position de la caméra en plus de la pente locale et de la hauteur de la surface, l'idée est de collecter des données de scan du miroir à l'aide de deux caméras. De cette manière, les Équations (2) peuvent être appliquées sur les mesures de chaque caméra mais avec des paramètres différents. Ainsi, pour chaque pixel surface, il y aurait deux équations de calcul des pentes locales en x et en y. Il ne resterait donc plus qu'à égaliser les deux équations pour éliminer la variable liée à la pente locale en x ou en y et directement résoudre l'équation obtenue pour obtenir la valeur de la hauteur de la surface.

Cette idée repose cependant sur des points dont il faut absolument tenir compte afin de réaliser une mesure en stéréo comme je le propose ici. Le premier est que le champ de vue des deux caméras et leur position doivent être définis de manière exacte pour chacune d'elles afin de minimiser les erreurs lors de l'évaluation de la hauteur pour un même pixel de la surface.

Le second est que le coût d'une seconde caméra infrarouge peut être conséquent. Il a donc deux solutions à ce problème :

- Réaliser deux scans avec une même caméra placée à deux endroits différents ;
- Réaliser un scan avec la caméra qui se déplace à deux positions pour faire des acquisitions à chaque déplacement du fil.

Les deux solutions présentent cependant un problème de répétabilité de la position du fil dans la première et de position de la caméra dans la seconde. Néanmoins, si elles sont correctement caractérisées et définies, ces solutions devraient donner de bons résultats.

Enfin, le dernier point à soulever concerne les équations utilisées. En effet, comme expliqué ci-dessus, deux équations sont nécessaires pour déterminer la valeur de deux variables. Cependant, si ces équations sont multiples l'une de l'autre, leur détermination sera impossible. Il est donc impératif que la disposition des caméras ne rende pas ces équations multiples l'une de l'autre. Une analyse de ces équations peut donc se révéler utile afin de déterminer la meilleure position possible pour la caméra.

En ce qui concerne la seconde possibilité d'amélioration, celle-ci consiste en l'analyse du travail de Monsieur Alexis Bechet [12]. Ce dernier a réalisé un stage au Centre Spatial de Liège sur la déflectométrie dans le visible dans le cadre de ses études. Au cours de nos travaux qui ont été menés à la même période, nous avons eu l'occasion d'entrer en contact sur le conseil de Monsieur Marc Georges, employé du CSL, maître de stage de Monsieur Bechet et membre de mon jury pour ce mémoire.

Au cours de nos travaux, nous avons eu l'occasion à plusieurs reprises de parler déflectométrie et un des points de recherche de Monsieur Bechet a attiré mon attention. Au cours de ses recherches, il tentait de déterminer les chemins empruntés par les rayons pour impacter chaque pixel de la caméra. Son objectif était de démontrer que tous les chemins ne passent pas par le même point focal de la

caméra et que, sur base de cette connaissance, il est possible de déterminer les coordonnées des pixels de la surface impactés par chaque rayon.

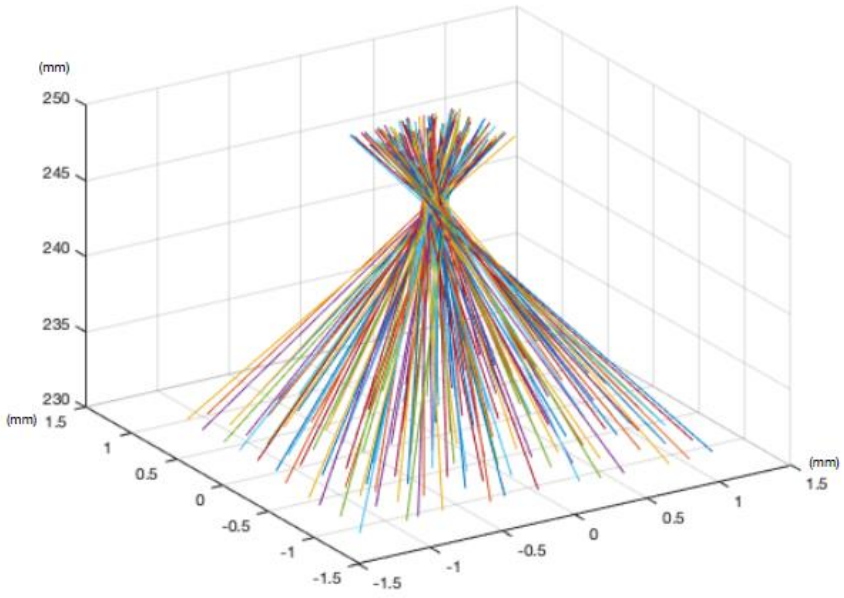


Figure 62 - Résultats obtenus par Monsieur Bechet concernant les chemins des rayons incident la caméra [12]

Comme cela est visible dans la figure ci-dessus, les résultats de Monsieur Bechet montrent que tous les rayons ne passent pas par le même point. Cependant, tout au long de mon travail, j'ai utilisé l'hypothèse qui dit que tous les rayons passent par le point focal de la caméra. Il serait donc intéressant de pouvoir caractériser la position du passage des rayons dans le plan de ce point pour chaque pixel afin de corriger les erreurs de position de la caméra puisque cette position est actuellement prise pour tous les pixels au niveau de son point focal.

Pour conclure cette section, on peut constater qu'un grand nombre d'idées d'amélioration du système peuvent être envisagées dans le futur. On peut également s'attendre à en découvrir de nouvelles en fonction de la progression de l'application de celles déjà présentées.

Conclusion

Pour conclure ce mémoire, revenons sur les objectifs de travail de départ afin d'évaluer comment ils ont été atteints.

Dans un premier temps, je me suis longuement attardé sur l'apprentissage et la compréhension de la déflectométrie dans l'infrarouge mais aussi dans le visible afin de maîtriser les tenants et aboutissants de cette technique de mesure. Au travers de l'analyse du projet SLOTS et du projet développé en 2013-2014 par la société AMOS, j'ai aussi pu découvrir la qualité des résultats de mesure envisageable pour notre projet commun avec Monsieur Baron.

Ensuite, j'ai eu l'occasion de collaborer avec Monsieur Baron sur ce projet dans le cadre du développement de nos travaux respectifs. Ce fût pour nous une belle opportunité de partage d'idées, d'analyses et d'entraide qui ont permis que notre projet commun évolue dans les meilleures conditions possibles, tant dans son aspect physique que numérique.

De plus, au fil de ces mois de travail, Octopus, mon programme de reconstruction d'une surface par déflectométrie infrarouge, s'est développé et a évolué en fonction des besoins pour l'obtention des meilleurs résultats possibles. De deux simples algorithmes de calcul de pentes et d'intégration de gradient, j'ai pu développer une première interface relativement complète au vu du temps qui m'était imparti et de mes connaissances de base en programmation sur Matlab. À l'heure actuelle, Octopus a permis les reconstructions de trois miroirs en basses et hautes fréquences. Ces résultats, malgré les erreurs observées et mesurées, ont malgré tout permis d'offrir de futures pistes d'amélioration pour les résultats de l'instrument.

Cependant, en raison de problèmes techniques qui rendent actuellement l'instrument de mesure inutilisable, la mesure de grands miroirs depuis le banc de test est impossible. Néanmoins, cette deuxième configuration devrait être rapidement rendue opérationnelle et rendre la mesure de grands miroirs par déflectométrie infrarouge une réalité pour la société AMOS.

Enfin, terminons en soulignant le fait que, maintenant que l'instrument et le programme sont implémentés, il reste néanmoins de nombreux points d'amélioration du système à envisager pour la suite. On peut donc s'attendre à ce que la déflectométrie en infrarouge nous dévoile encore de nombreux secrets.

Bibliographie

[1] J.H. Burge, P. Su, G. Butel, R. Huang, A. Maldonado et T. Su, *Measuring large mirrors using SCOTS, the Software Configurable Optical Test System*, Proc SPIE **9151**, 91510Z (2014).

[2] T. Su, S. Wang, R.E. Parks, P. Su et J.H. Burge, *Measuring rough optical surfaces using scanning long-wave optical test system. 1. Principle and implementation*, Appl. Opt. **52**, (29) 7117-7126 (2013).

[3] D.W. Kim, C.J. Oh, P. Su et J.H. Burge, *Advanced Technology Solar Telescope 4.2 m Off-axis Primary Mirror Fabrication*, OSA, paper OTh2B.3 (2014).

[4] AMOS, *Infrared Deflectometry*, AMOS (2014).

[5] M. Baron, *Automatic deflectometry measurement of a mirror shape during griding/polishing*, Université de Louvain (2019).

[6] M.C. Knauer, J. Kaminski et G. Häusler, *Phase Measuring Deflectometry: a new approach to measure specular free-form surfaces*, Proc SPIE **5457** (2004).

[7] P. Su, M.A.H. Khreishi, T. Su, R. Huang, M.Z. Dominguez, A. Maldonado, G. Butel, Y. Wang, R.E. Parks et J.H. Burge, *Aspheric and freeform surfaces metrology with software configurable optical test system: a computerized reverse Hartmann test*, Optical Engineering **53**, (3) 031305 (2014).

[8] C. Faber, E. Olesch, R. Krobot et G. Häusler, *Deflectometry challenges interferometry – the competition gets tougher!*, Proc SPIE **8493**, 84930R (2012).

[9] S. Höfer, J. Burke et M. Heizmann, *Infrared deflectometry for the inspection of diffusely specular surfaces*, Adv. Opt. Techn. **5**, (5-6) 377-387 (2016).

[10] D.W. Kim, T. Su, P. Su, C. Oh, L. Graves et J. Burge, *Accurate and rapid IR metrology for the manufacture of freeform optics*, SPIE Newsroom (2015).

[11] D. Malacara, *Optical Shop Testing*, John Wiley & Sons, 370-374 (2007).

[12] A. Bechet, *Rapport de stage pour le Centre Spatial de Liège : Déflectométrie*, Angleur (2019).

Annexe 1

1. Deflect.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Nettoyage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Nettoyage de la fenêtre de commande
clear;
% Nettoyage de l'espace de travail
clc;
% Fermeture des fenêtres déjà ouvertes
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% Lien du dossier contenant les scripts %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lien avec le dossier d'origine contenant les différents scripts
pth = cd;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% Sélection du dossier avec les données %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Demande du dossier de données à sélectionner
pth_r = uigetdir(pth,'Sélectionne ton dossier de données');

% Création des liens vers les dossiers x et y
pth_rx=[pth_r,'/x'];
pth_ry=[pth_r,'/y'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% Conversion des dossiers x et y en dossiers JPEG %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail d'abord dans le dossier x
pth_r = pth_rx;
dossier = 'x';

% Exécution de JPEG.m pour le dossier x
cd(pth); % Retour dans le dossier des scripts
run JPEG;
```

```
% Création du lien vers dossier JPEG/x
pth_xj = pth_w;
% Définition de la taille du dossier JPEG/x
size_jpeg(1) = size_fold(1);

% Travail ensuite dans le dossier y
pth_r = pth_ry;
dossier = 'y';

% Exécution de JPEG.m pour le dossier y
cd(pth); % Retour dans le dossier des scripts
run JPEG;

% Création du lien vers dossier JPEG/y
pth_yj = pth_w;
% Définition de la taille du dossier JPEG/y
size_jpeg(2) = size_fold(1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% Références des meilleures intensités %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Création d'un tableau avec les deux adresses des dossiers x et y
pth_xyj = [pth_xj; pth_yj];

% Mesure de la taille des adresses de dossiers pour pouvoir les réutiliser
plus tard
size_pth_xyj = size(pth_xyj(1,:)); % Est égal pour pth_xj et pth_yj, car le
chemin ne diffère pas en nombre de caractères

% Exécution de Intensite.m pour les 2 dossiers JPEG
cd(pth);
run Intensite;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Calcul des coordonnées %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cd(pth);
% Exécution de Coordonnes.m
run Coordonnees;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes x %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cd(pth);
% Exécution de Pentes_X.m
run Pentes_X;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cd(pth);
% Exécution de Pentes_Y.m
run Pentes_Y;
```

2. JPEG.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Noms des images %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier actuel
cd(pth_r);

% Sélection des fichiers bmp, png, gif, hdf, bpm, pgm, ppm, jpg
bmp = dir('*.bmp');
png = dir('*.png');
gif = dir('*.gif');
hdf = dir('*.hdf');
bpm = dir('*.bpm');
pgm = dir('*.pgm');
ppm = dir('*.ppm');
jpg = dir('*.jpg');

% Création d'un tableau avec tous les fichiers trouvés
fil = [bmp; png; gif; hdf; bpm; pgm; ppm; jpg];

% Taille du dossier actuel
size_fold = size(fil);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Création du dossier JPEG %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lien du dossier de sauvegarde des images JPEG
pth_w = [pth_r(1:end-2),'/JPEG/', dossier];

%Si ce dossier n'existe pas, le dossier est créé
if ~exist(pth_w, 'dir')
  mkdir(pth_w);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% Enregistrement des images en JPEG %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier JPEG
cd(pth_w);

% Boucle de l'image 1 à la dernière qui a le numéro de la taille du dossier
for i = 1:size_fold

    % Extraction des infos de l'image i
    I = imread([pth_r,'/',fil(i) .name]);

    % Création du nom de la nouvelle image JPEG
    filname = fil(i).name;

    % Retrait du format initial et ajout du .jpg au nom de l'image
    filname = [filname(1:end-4),'.jpg'];

    % Enregistrement de l'image
    imwrite(I, filname, 'jpg');
```

```
end
```

3. Intensite.m

```
% Lancement d'une boucle pour faire le dossier JPEG x puis le y
for h = 1:2

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%% Dossier itéré %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Placement dans le dossier itéré
    cd(pth_xyj(h,1:size_pth_xyj(2)));

    % Liste de ce que le dossier actuel contient
    liste = dir('*.jpg');

    % Taille de la liste
    size_fold = size(liste);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%% Préparation à la cartographie %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Ouverture de la première image du dossier en rgb
    rgb_scale = imread([pth_xyj(h,1:size_pth_xyj(2)),'/',liste(1) .name]);

    % Conversion de la première image en niveau de gris
    gray_scale = rgb2gray(rgb_scale);

    % Dimensions de l'image en gris (supposées être identiques pour toutes
les images du dossier)
    dim_im = size(gray_scale);

    % Nombre de pixels de l'image
    taille_im = dim_im(1)*dim_im(2);

    % Création d'une carte des intensité max vierge de la taille de l'image
en gris
    intensite = zeros(dim_im(1),dim_im(2));

    % Création d'une carte des références des images pour les maximums
d'intensité (taille de l'image en gris)
    ref = ones(dim_im(1),dim_im(2));

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%% Réalisation des 2 cartographies %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Ouverture des images du dossier une à une
    for i = 1:size_fold

        % Conversion de l'image rgb ouverte en niveau de gris
        gray   =   rgb2gray(imread([pth_xyj(h,1:size_pth_xyj(2)),'/',liste(i)
.name]));

        % Analyse de chaque pixel de l'image
        for j = 1:taille_im
```

```
            % Comparaison de l'image avec la carte des maximums
            % Si l'intensité est plus grande dans l'image ouverte que sur la
carte
            if gray(j) > intensite(j)

                % Changement de la valeur du maximum dans la carte
                intensite(j) = gray(j);
                % Changement de la référence
                ref(j) = i;

            % Si l'intensité égale ou inférieure, rien ne change
            end
        end
    end

    % Création des cartes de références distinctes pour chaque dossier
    if h == 1
        ref_x = ref;
    elseif h == 2
        ref_y = ref;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%% Affichage des deux cartes %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Création de l'image des intensités maximums
    image_fin = uint8(intensite);

    % Ouverture de cette image avec une barre de couleur
    figure, imshow(image_fin), colorbar;

    % Création de l'image des références
    image_ref = uint8(ref);

    % Ouverture de cette image avec barre de couleur
    figure, imshow(image_ref), colorbar;

end
```

4. Coordonnees.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Définition des distances estimées %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition des déplacements totaux du fil
fil_dist = ([]);
fil_dist(1) = 0.5;
fil_dist(2) = 0.5;

% Définition de la mesure réelle de l'image en x
xreel = 0.5;

% Préparation de la variable qui correspondra aux longueur et largeur de la
surface
distreel = ([]);

% Boucle pour chaque dossier
for h = 1:2

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%% Calcul de la position du fil %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Calcul de l'avancée du fil à chaque itération de la machine
    pas = fil_dist(h)/size_jpeg(h);

    % Calcul de la position du fil à chaque image
    s(1)= pas/2;
    for i = 2:size_jpeg(h)
        s(i) = s(i-1) + pas;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%% Calcul de la position des pixels de la surface %%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Calcul des dimensions réelles de l'image
    % En x
    if h == 1
        distreel(1) = xreel;
    % En y
    elseif h == 2
        distreel(2) = (xreel/dim_im(1))*dim_im(2);
    end

    % Calcul de la dimension réelle du pixel
    pix = distreel(h)/dim_im(h);

    % Calcul de la position exacte de chaque pixel
    m(1)= pix/2;
    for i = 2:dim_im(h)
        m(i)= m(i-1) + pix;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%% Enregistrement des données respectives %%%%%%%%%%%%%%%%%%
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Pour x
    if h == 1
        pas_x = pas;
        xs = s;
        pix_x = pix;
        xm = m;

    % Pour y
    elseif h == 2
        pas_y = pas;
        ys = s;
        pix_y = pix;
        ym = m;
    end

end
```

5. Pentes_X.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% Paramètres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Position x de la caméra
xc = 0.135;
% Position z de la caméra
zc = 1.60;
% Position z du fil
zf = 1.60;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Tableau avec pente de chaque pixel
wx = ([]);

% Balayage ses x
for i = 1:dim_im(1)
    % Balayage des y
    for j = 1:dim_im(2)
        % Calcul de la pente de chaque pixel
        wx(i,j) = (1/2)*(((xm(i)-xs(ref_x(i,j)))/zf)+((xm(i)-xc)/zc));
        % Calcul de la valeur max et la valeur min de la pente
        if i==1 && j==1
            wx_max = wx(i,j);
            wx_min = wx(i,j);
        else
            if wx(i,j)> wx_max
                wx_max = wx(i,j);
            elseif wx(i,j) < wx_min
                wx_min = wx(i,j);
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Affichage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure, imshow(wx, [wx_min wx_max]), colorbar;
```

6. Pentes_Y.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Paramètres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Position x de la caméra
yc = -0.235;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Tableau avec pente de chaque pixel
wy = ([]);

% Balayage des x
for i = 1:dim_im(1)
    % Balayage des y
    for j = 1:dim_im(2)
        % Calcul de la pente de chaque pixel
        wy(i,j) = (1/2)*(((ym(i)-ys(ref_y(i,j)))/zf)+((ym(i)-yc)/zc));
        % Calcul de la valeur max et la valeur min de la pente
        if i==1 && j==1
            wy_max = wy(i,j);
            wy_min = wy(i,j);
        else
            if wy(i,j)> wy_max
                wy_max = wy(i,j);
            elseif wy(i,j) < wy_min
                wy_min = wy(i,j);
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Affichage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure, imshow(wy, [wy_min wy_max]), colorbar;
```

Annexe 2

1. Fct_pour_int2D.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Choix de la carte %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Nettoyage l'environnement Matlab
clear all;clc;

% Demande du choix de carte à l'utilisateur
choix = menu('Choisis une fonction','Zernike focus' ...
    ,'Zernike focus avec bruit','Legendre 17°x14°', 'Legendre 17°x14° avec
bruit');
reponse = [1, 2, 3, 4]


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Zernike focus %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if reponse(choix) == 1

    % Création de la carte de la surface vierge
    image = zeros(300);

    % Création de la carte de focus centrée
    for i = 1:300
        for j = 1:300
            image(i,j) = 2*0.000001*((j-150)^2+(i-150)^2)-1;
        end
    end

    % Création des cartes de pentes
    [pente_x, pente_y] = gradient(image);

    % Dimensions des cartes
    dim_im = size(image);

    % Maximum et minimum de la carte
    max_image = max(max(image));
    min_image = min(min(image));

    % Distances entre les pixels
```

```
    distreel = [1 1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Zernike focus plus bruit %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

elseif reponse(choix) == 2

    % Création de la carte de la surface vierge
    image = zeros(300);

    % Création de la carte de focus centrée
    for i = 1:300
        for j = 1:300
            image(i,j) = 2*0.000001*((j-150)^2+(i-150)^2)-1;
        end
    end

    % Création des cartes de pentes
    [pente_x, pente_y] = gradient(image);

    % Dimensions des cartes
    dim_im = size(image);

    % Maximum et minimum de la carte
    max_image = max(max(image));
    min_image = min(min(image));

    % Distances entre les pixels
    distreel = [1 1];

    % Ajout d'un bruit à la carte x
    for i = 1:300
        for j = 1:300
            pente_x(i,j) = pente_x(i,j) + (0.000002*(rand-0.5));
        end
    end

    % Ajout d'un bruit à la carte y
    for i = 1:300
        for j = 1:300
            pente_y(i,j) = pente_y(i,j) + (0.000002*(rand-0.5));
        end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Legendre 17°x 14°y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

elseif reponse(choix) == 3

    % Création de la carte de la surface vierge
    image = zeros(300);

    % Création de la carte avec les polynômes de Legendre
    for i = 1:300
        for j = 1:300
            image(i,j) = 0.005*((583401555*((j/150)-1)^17)/32768 -
(300540195*((j/150)-1)^15)/4096 ...
```

```
                + (1017958725*((j/150)-1)^13)/8192 - (456326325*((j/150)-
1)^11)/4096 ...
                + (929553625*((j/150)-1)^9)/16384 - (66927861*((j/150)-
1)^7)/4096 ...
                + (20369349*((j/150)-1)^5)/8192 - (692835*((j/150)-
1)^3)/4096 ...
                + (109395*((j/150)-1))/32768 + (5014575*((i/150)-
1)^14)/2048 ...
                - (16900975*((i/150)-1)^12)/2048 + (22309287*((i/150)-
1)^10)/2048 ...
                - (14549535*((i/150)-1)^8)/2048 + (4849845*((i/150)-
1)^6)/2048 ...
                - (765765*((i/150)-1)^4)/2048 + (45045*((i/150)-1)^2)/2048
...
                - 429/2048);
        end
    end

    % Création des cartes de pentes
    [pente_x, pente_y] = gradient(image);

    % Dimensions des cartes
    dim_im = size(image);

    % Maximum et minimum de la carte
    max_image = max(max(image));
    min_image = min(min(image));

    % Distances entre les pixels
    distreel = [1 1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% Legendre 17°x 14°y plus bruit %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    elseif reponse(choix) == 4

    % Création de la carte de la surface vierge
    image = zeros(300);

    % Création de la carte avec les polynômes de Legendre
    for i = 1:300
        for j = 1:300
            image(i,j) = 0.005*((583401555*((j/150)-1)^17)/32768 -
(300540195*((j/150)-1)^15)/4096 ...
                + (1017958725*((j/150)-1)^13)/8192 - (456326325*((j/150)-
1)^11)/4096 ...
                + (929553625*((j/150)-1)^9)/16384 - (66927861*((j/150)-
1)^7)/4096 ...
                + (20369349*((j/150)-1)^5)/8192 - (692835*((j/150)-
1)^3)/4096 ...
                + (109395*((j/150)-1))/32768 + (5014575*((i/150)-
1)^14)/2048 ...
                - (16900975*((i/150)-1)^12)/2048 + (22309287*((i/150)-
1)^10)/2048 ...
                - (14549535*((i/150)-1)^8)/2048 + (4849845*((i/150)-
1)^6)/2048 ...
                - (765765*((i/150)-1)^4)/2048 + (45045*((i/150)-1)^2)/2048
...
                - 429/2048);
```

```
        end
    end

    % Création des cartes de pentes
    [pente_x, pente_y] = gradient(image);

    % Dimensions des cartes
    dim_im = size(image);

    % Maximum et minimum de la carte
    max_image = max(max(image));
    min_image = min(min(image));

    % Distances entre les pixels
    distreel = [1 1];


     % Ajout d'un bruit à la carte x
    for i = 1:300
        for j = 1:300
            pente_x(i,j) = pente_x(i,j) + (0.000002*(rand-0.5));
        end
    end

    % Ajout d'un bruit à la carte y
    for i = 1:300
        for j = 1:300
            pente_y(i,j) = pente_y(i,j) + (0.000002*(rand-0.5));
        end
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Affichage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Affichage de la carte générée
surf(image);
```

2. Cartes des pentes de chaque surface avec et sans bruit

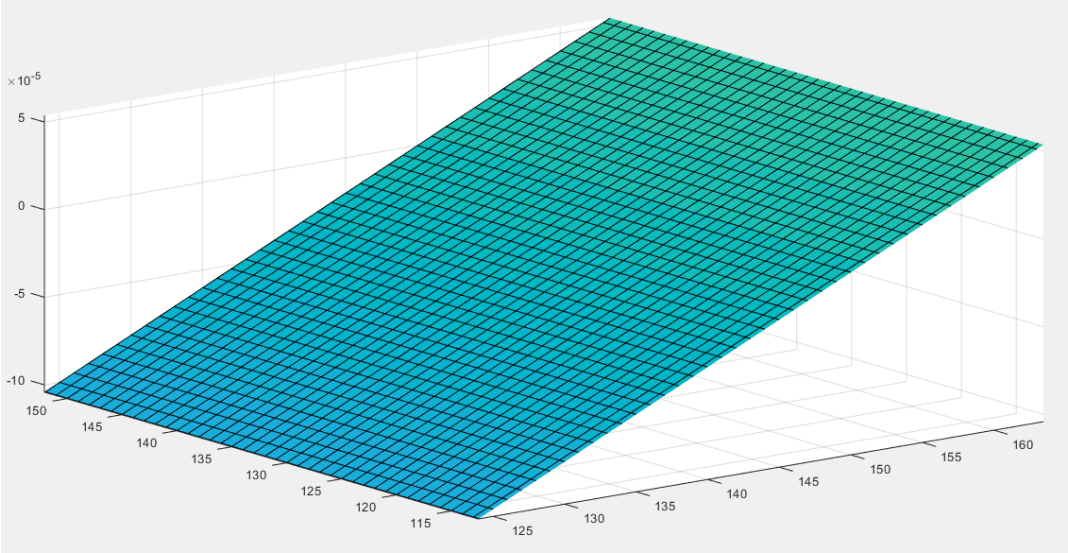


Figure 63 – Zoom sur la carte des pentes x de la surface du quatrième terme de Zernike sans bruit

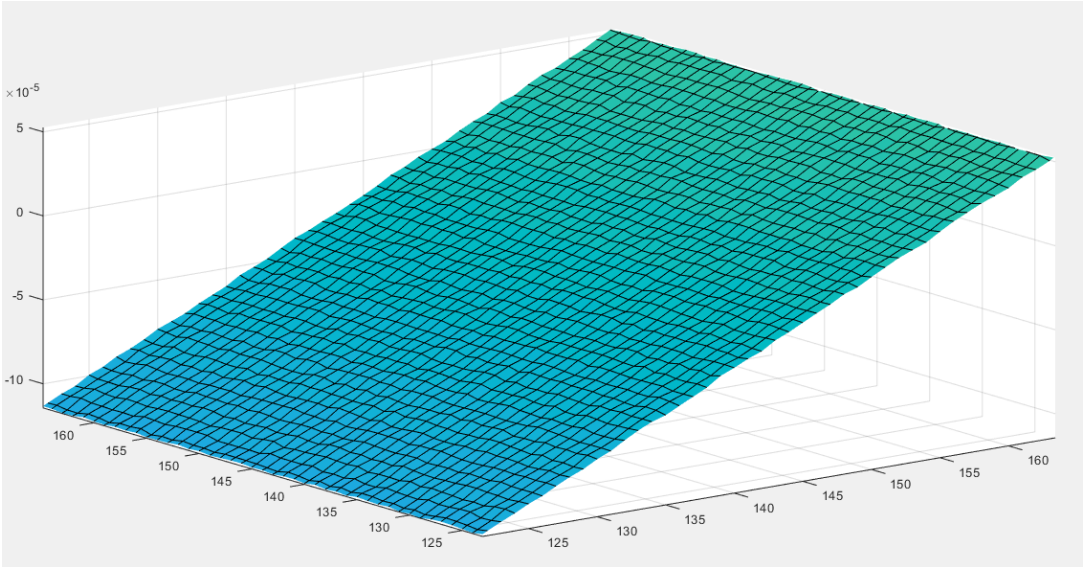


Figure 64 – Zoom sur la carte des pentes y de la surface du quatrième terme de Zernike avec bruit

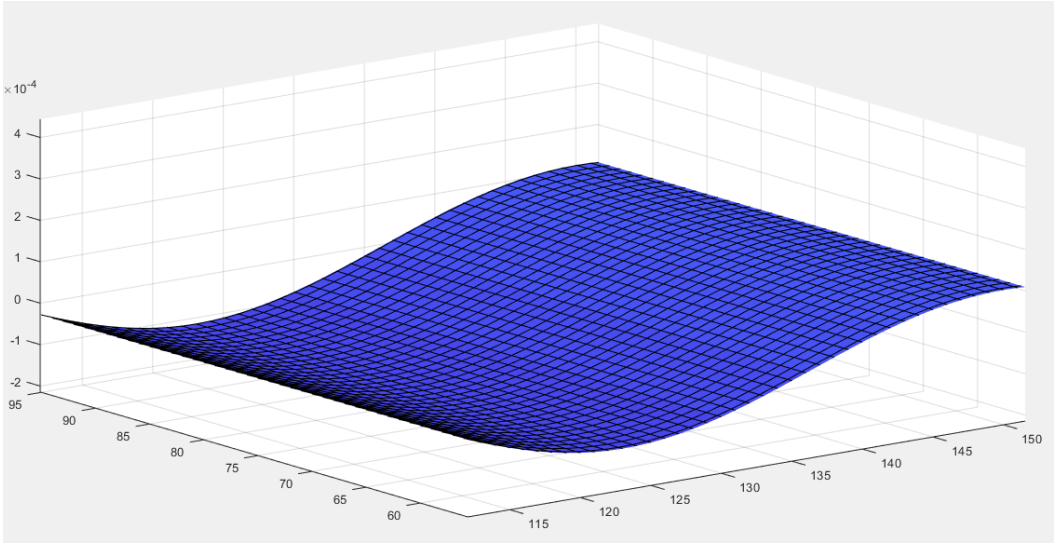


Figure 65 – Zoom sur la carte des pentes x de la surface en polynôme de Legendre sans bruit

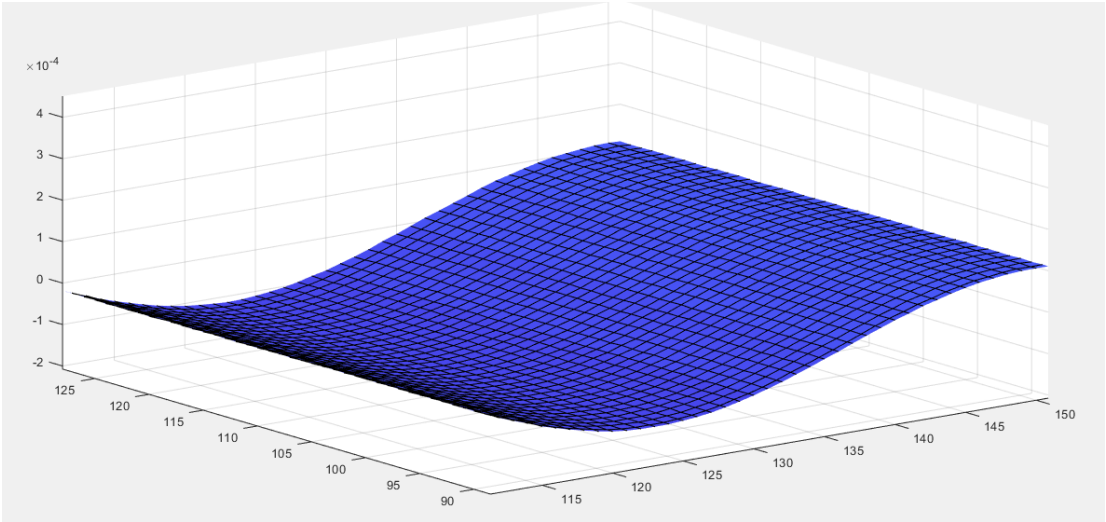


Figure 66 – Zoom sur la carte des pentes y de la surface en polynôme de Legendre avec bruit

Annexe 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Masquage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la variable satisfaction du masque
satisfaction = 0;

% Demande de la zone à intégrer
while satisfaction == 0

    % Fermeture de l'image précédente
    close(gcf);

    % Création d'un tableau qui servira d'image masquée
    intensite_mask = image;

    % Affichage de l'image de base
    figure, imshow(intensite_mask, [min_image max_image]), colormap('jet');

    % Demande du type de masque voulu
    answer1 = questdlg('Quel type de masque veux-tu ?', ...
        'Intensité minimum', ...
        'Ellipse','Rectangle','Ellipse');

    % Effet suivant la réponse
    switch answer1
        case 'Ellipse'
            e = imellipse;
        case 'Rectangle'
            e = imrect;
    end

    % Création d'une carte binaire avec ce qui est dans et hors du masque
    mirror_ok = createMask(e);

    % Coordonnées du masque
    pos = getPosition(e);

    % NaN sur la carte d'intensité masqué hors du masque
    for i = 1:(dim_im(1)*dim_im(2))
        if mirror_ok(i) == 0
            intensite_mask(i) = NaN;
        end
    end
```

```
    % Affichage de l'image masquée
    imshow(intensite_mask, [min_image max_image]), colormap('jet');

    % Demande de satisfaction
    answer2 = questdlg('Ok?', ...
        'Intensité minimum', ...
        'Oui','Non','Oui');

    % Effet en fonction de la réponse
    switch answer2
        case 'Oui'
            satisfaction = 1;
        case 'Non'
            satisfaction = 0;
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% Cartes et paramètres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Tableau des pentes en x et y
X = pente_x;
Y = pente_y;

% Distance réelle entre chaque pixel
dx = distreel(2);
dy = distreel(1);

% Définition d'un point de référence
z_centre = 0;

% Tableau des hauteurs (toutes égales à 0 au début)
Z = zeros(dim_im(1), dim_im(2));

% Définition du centre du masque
y0 = round(pos(2)) + round(pos(4)/2);
x0 = round(pos(1)) + round(pos(3)/2);

% Définition de la hauteur de référence au centre de l'image
Z(y0,x0) = z_centre;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% Intégration trapézoïdale %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for h = 0:1

    y0 = y0+h;

    % Calcul du point à côté du centre du masque pour la seconde
intégration trapézoïdale
    if h == 1
        Z(y0,x0) = ((mirror_ok(y0,x0-1)*Z(y0,x0-1)...
                    + mirror_ok(y0,x0+1)*Z(y0,x0+1) + mirror_ok(y0-
1,x0)*Z(y0-1,x0) ...
```

```
                    + mirror_ok(y0+1,x0)*Z(y0+1,x0))/(mirror_ok(y0,x0-1) +
mirror_ok(y0,x0+1)...
                    + mirror_ok(y0-1,x0) + mirror_ok(y0+1,x0)))...
                    + ((mirror_ok(y0,x0-1)*((dx/2)*(X(y0,x0-1) +
X(y0,x0)))...
                    - mirror_ok(y0,x0+1)*((dx/2)*(X(y0,x0+1) +
X(y0,x0)))...
                    + mirror_ok(y0-1,x0)*((dy/2)*(Y(y0-1,x0) +
Y(y0,x0)))...
                    - mirror_ok(y0+1,x0)*((dy/2)*(Y(y0+1,x0) +
Y(y0,x0))))...
                    / (mirror_ok(y0,x0-1) + mirror_ok(y0,x0+1)...
                    + mirror_ok(y0-1,x0) + mirror_ok(y0+1,x0)));
    end

    % Intègration vers les X+/Y+
    for i = 1:max(dim_im)
        % Si hors masque, stop
        if mirror_ok(y0+i,x0+i) == 0
            break;
        % Si dans le masque, intégration
        else
            Z(y0+i,x0+i) = Z(y0+i-1,x0+i-1)...
                + (dx/2)*(X(y0+i-1,x0+i-1)+X(y0+i,x0+i-1))...
                + (dy/2)*(Y(y0+i-1,x0+i)+Y(y0+i,x0+i));
            max_diag = i;
        end
    end

    % Intègration vers les X-/Y-
    for i = 1:max(dim_im)
        if  mirror_ok(y0-i,x0-i) == 0
            break;
        % Si dans le masque, intégration
        else
            Z(y0-i,x0-i) = Z(y0-i+1,x0-i+1)...
                - (dx/2)*(X(y0-i+1,x0-i+1)+X(y0-i,x0-i+1))...
                - (dy/2)*(Y(y0-i+1,x0-i)+Y(y0-i,x0-i));
            min_diag = i;
        end
    end


    % Si le masque est orienté horizontalement
    if pos(3)>pos(4)
        % Définition du nombre de vagues d'intégrations à gauche et à
droite
        nb_int = ceil(pos(3)/pos(4));
        % Définition du sens de départ pour intégrer côté droit
        sens = 1;
        % Définition des x et y de départ
        x = x0-min_diag;
        y = y0-min_diag;
        % Les vagues d'intégrations à droite
        for i = 1:nb_int
            % Pour intégrer dans le sens X+/Y-
            if sens == 1
                % Pour s'assurer que le point de départ est dans le masque
                while mirror_ok(y,x) == 1
                    % Intégration à partir de (y,x)
                    for j = 1:max(dim_im)
```

```
                % Si le point suivant à intégrer est hors du masque
                if  mirror_ok(y-j,x+j) == 0
                    break;
                % Si il est dans le masque, intégration
                else
                    Z(y-j,x+j) = Z(y-j+1,x+j-1)...
                        + (dx/2)*(X(y-j+1,x+j-1)+X(y-j+1,x+j))...
                        - (dy/2)*(Y(y-j+1,x+j)+Y(y-j,x+j));
                end
            end
            % Définition du point suivant à partir duquel intégrer
            x = x+1;
            y = y+1;
        end
        % Passage au sens suivant
        sens = 2;
        % Définition du point de départ du sens suivant
        x = x-1;
        y = y-1;
    % Pour intégrer dans le sens X+/Y+, avec les mêmes commentaires
    elseif sens == 2
        while mirror_ok(y,x) == 1
            for j = 1:max(dim_im)
                if  mirror_ok(y+j,x+j) == 0
                    break;
                else
                    Z(y+j,x+j) = Z(y+j-1,x+j-1)...
                        + (dx/2)*(X(y+j-1,x+j-1)+X(y+j-1,x+j))...
                        + (dy/2)*(Y(y+j-1,x+j)+Y(y+j,x+j));
                end
            end
            x = x+1;
            y = y-1;
        end
        sens = 1;
        x = x-1;
        y = y+1;
    end
end
% Passage à l'intégration du côté gauche
sens = 1;
% Définition du point de départ
x = x0+max_diag;
y = y0+max_diag;
for i = 1:nb_int
    % Intégration vers les X-/Y+ avec les mêmes commentaires
    if sens == 1
        while mirror_ok(y,x) == 1
            for j = 1:max(dim_im)
                if  mirror_ok(y+j,x-j) == 0
                    break;
                else
                    Z(y+j,x-j) = Z(y+j-1,x-j+1)...
                        - (dx/2)*(X(y+j-1,x-j+1)+X(y+j-1,x-j))...
                        + (dy/2)*(Y(y+j-1,x-j)+Y(y+j,x-j));
                end
            end
            x = x-1;
            y = y-1;
        end
        sens = 2;
```

```
            x = x+1;
            y = y+1;
        % Intégration vers les X-/Y-, avec les mêmes commentaires
        elseif sens == 2
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y-j,x-j) == 0
                        break;
                    else
                        Z(y-j,x-j) = Z(y-j+1,x-j+1)...
                            - (dx/2)*(X(y-j+1,x-j+1)+X(y-j+1,x-j))...
                            - (dy/2)*(Y(y-j+1,x-j)+Y(y-j,x-j));
                    end
                end
                x = x-1;
                y = y+1;
            end
            sens = 1;
            x = x+1;
            y = y-1;
        end
    end

% Dans le cas où le masque est vertical
elseif pos(4)>= pos(3)
    % Définition du nombre de vagues
    nb_int = ceil(pos(4)/pos(3));
    % Définition du sens de départ pour intégrer vers le haut
    sens = 1;
    % Définition du point de départ
    x = x0-min_diag;
    y = y0-min_diag;
    % Les vagues d'intégration vers le haut
    for i = 1:nb_int
        % Intégration vers les X-/Y+
        if sens == 1
            % Vérification que le point de départ est dans le masque
            while mirror_ok(y,x) == 1
                % Intégration à partir de ce point
                for j = 1:max(dim_im)
                    % Si le point suivant est hors du masque
                    if  mirror_ok(y+j,x-j) == 0
                        break;
                    % S'il est dans le masque, intégration
                    else
                        Z(y+j,x-j) = Z(y+j-1,x-j+1)...
                            - (dx/2)*(X(y+j-1,x-j+1)+X(y+j-1,x-j))...
                            + (dy/2)*(Y(y+j-1,x-j)+Y(y+j,x-j));
                    end
                end
                % Définition du point suivant
                x = x+1;
                y = y+1;
            end
            % Passage au sens suivant
            sens = 2;
            % Définition du point de départ
            x = x-1;
            y = y-1;
        % Intégration dans le sens X+/Y+, mêmes commentaires
        elseif sens == 2
```

```
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y+j,x+j) == 0
                        break;
                    else
                        Z(y+j,x+j) = Z(y+j-1,x+j-1)...
                            + (dx/2)*(X(y+j-1,x+j-1)+X(y+j-1,x+j))...
                            + (dy/2)*(Y(y+j-1,x+j)+Y(y+j,x+j));
                    end
                end
                x = x-1;
                y = y+1;
            end
            sens = 1;
            x = x+1;
            y = y-1;
        end
    end
    % Passage à l'intégration vers le bas
    sens = 1;
    x = x0+max_diag;
    y = y0+max_diag;
    for i = 1:nb_int
        % Intégration dans le sens X+/Y-, mêmes commentaires
        if sens == 1
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y-j,x+j) == 0
                        break;
                    else
                        Z(y-j,x+j) = Z(y-j+1,x+j-1)...
                            + (dx/2)*(X(y-j+1,x+j-1)+X(y-j+1,x+j))...
                            - (dy/2)*(Y(y-j+1,x+j)+Y(y-j,x+j));
                    end
                end
                x = x-1;
                y = y-1;
            end
            sens = 2;
            x = x+1;
            y = y+1;
        % Intégration dans le sens X-/Y-, mêmes commentaires
        elseif sens == 2
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y-j,x-j) == 0
                        break;
                    else
                        Z(y-j,x-j) = Z(y-j+1,x-j+1)...
                            - (dx/2)*(X(y-j+1,x-j+1)+X(y-j+1,x-j))...
                            - (dy/2)*(Y(y-j+1,x-j)+Y(y-j,x-j));
                    end
                end
                x = x+1;
                y = y-1;
            end
            sens = 1;
            x = x-1;
            y = y+1;
        end
    end
```

```
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Paramètres de Southwell %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NaN partout où l'intégration trapézoïdale n'est pas passée
for i = 1:(dim_im(1)*dim_im(2))
    if mirror_ok(i) == 0
        Z(i) = NaN;
    end
end

% Création d'un tableau vide pour l'intégration de Southwell
Z_southwell = ([]);

% Définition de la dimension à ce tableau en le rendant égal à Z
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        Z_southwell(i,j) = Z(i,j);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Intégration de Southwell %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition du nombre d'itérations
for h = 1:(round((pos(3)*pos(4)))/1)
    % Pour les Y dans le masque
    for i = round(pos(2)):round(pos(2)+pos(4))
        % Pour les X dans le masque
        for j = round(pos(1)):round(pos(1)+pos(3))
            % NaN si pas dans le masque
            if mirror_ok(i,j) == 0
                Z_southwell(i,j) = NaN;

            else
                % Si dans le masque, mais qu'un pixel autour vaut NaN,
pixel autour égal à 0
                if mirror_ok(i,j-1) == 0
                    Z(i,j-1) = 0;
                end
                if mirror_ok(i,j+1) == 0
                    Z(i,j+1) = 0;
                end
                if mirror_ok(i-1,j) == 0
                    Z(i-1,j) = 0;
                end
                if mirror_ok(i+1,j) == 0
                    Z(i+1,j) = 0;
                end
                % Réalisation de l'intégration de Southwell
                Z_southwell(i,j) = ((mirror_ok(i,j-1)*Z(i,j-1)...
                    + mirror_ok(i,j+1)*Z(i,j+1) + mirror_ok(i-1,j)*Z(i-1,j)
...
                    + mirror_ok(i+1,j)*Z(i+1,j))/(mirror_ok(i,j-1) +
mirror_ok(i,j+1)...
                    + mirror_ok(i-1,j) + mirror_ok(i+1,j)))...
```

```
                    + ((mirror_ok(i,j-1)*((dx/2)*(X(i,j-1) + X(i,j)))...
                    - mirror_ok(i,j+1)*((dx/2)*(X(i,j+1) + X(i,j)))...
                    + mirror_ok(i-1,j)*((dy/2)*(Y(i-1,j) + Y(i,j)))...
                    - mirror_ok(i+1,j)*((dy/2)*(Y(i+1,j) + Y(i,j))))...
                    / (mirror_ok(i,j-1) + mirror_ok(i,j+1)...
                    + mirror_ok(i-1,j) + mirror_ok(i+1,j)));
            end
        end
    end
    % Définition de Z avec les valeurs de Z_southwell pour pouvoir réitérer
    for i = 1:dim_im(1)
        for j = 1:dim_im(2)
            Z(i,j) = Z_southwell(i,j);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Affichage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

surf(Z(1:dim_im(1),1:dim_im(2)));
```

Annexe 4

1. Deflect.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Nettoyage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Nettoyage de la fenêtre de commande
clear;
% Nettoyage de l'espace de travail
clc;
% Fermeture des fenêtres déjà ouvertes
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% Lien du dossier contenant les scripts %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lien avec le dossier d'origine contenant les différents scripts
pth = cd;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% Sélection du dossier avec les données %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Chargement du dernier dossier visité
load([pth, '\infos.mat']);

% Demande du dossier de données à sélectionner
pth_r = uigetdir(pth,'Sélectionne ton dossier de données');
pth_d = pth_r ;

% Création des liens vers les dossiers x et y
pth_rx=[pth_r,'/x'];
pth_ry=[pth_r,'/y'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% Conversion des dossiers x et y en dossiers JPEG %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail d'abord dans le dossier x
pth_r = pth_rx;
dossier = 'x';
```

```
% Exécution de JPEG.m pour le dossier x
cd(pth); % Retour dans le dossier des scripts
run JPEG;

% Création du lien vers dossier JPEG/x
pth_xj = pth_w;

% Travail ensuite dans le dossier y
pth_r = pth_ry;
dossier = 'y';

% Exécution de JPEG.m pour le dossier y
cd(pth); % Retour dans le dossier des scripts
run JPEG;

% Création de lien vers dossier JPEG/y
pth_yj = pth_w;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear bmp bpm dossier fil filname gif h hdf...
      I i jpg pgm png ppm pth_w size_fold...
      pth_r pth_rx pth_ry;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Infos des dossiers et des images %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exécution du script Info_doss_im.m
cd(pth);
run Info_doss_im;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear pth_xj pth_yj h rgb_scale gray_scale image_1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Calcul des coordonnées %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exécution le script Coordonnees.m
cd(pth);
run Coordonnees;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear dxo dyo h i dims dpix image_infos m prompt dlgtitle definput;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Références des meilleures intensités %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exécution du script Intensite.m
cd(pth);
run Intensite;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear h intensite s_centro num den k image nb i j intensite_x
intensite_y...
    liste_1 liste_2 pth_xyj size_jpeg size_pth_xyj;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes x %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exécution du script Pentes_X.m
cd(pth);
run Pentes_X;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear wx_max wx_min i j;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes y %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exécution du script Pentes_Y.m
cd(pth);
run Pentes_Y;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear wy_max wy_min i j xc xm xs yc ym ys zc zf;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Intégration %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exécution du script Integrad_2D.m
cd(pth);
run Integrad_2D;

% Nettoyage de toutes les variables qui ne sont plus utiles
clear satisfaction intensite_mask image_fin answer1 e mirror_ok pos i...
    answer2 X Y z_centre y0 x0 h max_diag min_diag nb_int sens x y...
    Z_southwell j;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Fin %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cd(pth);

% Suppression des dossiers JPEG
rmdir([pth_d, '/JPEG'], 's');

% Définition du dossier dans lequel se situait le dossier sélectionné en
début de programme
cd(pth_d);
cd ../;
dern_pth_d = cd;

% Sauvegarde du dossier retourné et les résultats dans deux fichiers .mat
séparés
save([pth,'\infos.mat'],'dern_pth_d');
save([pth_d,'\results.mat'], 'dim_im', 'dx', 'dy', 'wx', 'wy', 'Z');
```

2. JPEG.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Noms des images %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier actuel
cd(pth_r);

% Sélection des fichiers bmp, png, gif, hdf, bpm, pgm, ppm, jpg
bmp = dir('*.bmp');
png = dir('*.png');
gif = dir('*.gif');
hdf = dir('*.hdf');
bpm = dir('*.bpm');
pgm = dir('*.pgm');
ppm = dir('*.ppm');
jpg = dir('*.jpg');

% Création d'un tableau avec tous les fichiers trouvés
fil = [bmp; png; gif; hdf; bpm; pgm; ppm; jpg];

% Taille du dossier actuel
size_fold = size(fil);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Création du dossier JPEG %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lien du dossier de sauvegarde des images JPEG
pth_w = [pth_r(1:end-2),'/JPEG/', dossier];

% Si ce dossier n'existe pas, le dossier est créé
if ~exist(pth_w, 'dir')
  mkdir(pth_w);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% Enregistrement des images en JPEG %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier JPEG
cd(pth_w);

% Boucle de l'image 1 à la dernière qui a le numéro de la taille du dossier
for i = 1:size_fold

    % Extraction des infos de l'image i
    I = imread([pth_r,'/',fil(i) .name]);

    % Création du nom de la nouvelle image JPEG
    filname = fil(i).name;

    % Retrait du format initial et ajout du .jpg au nom de l'image
    filname = [filname(1:end-4),'.jpg'];

    % Enregistrement de l'image
```

```
    imwrite(I, filname, 'jpg');
end
```

3. Info_doss_im.m.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Chemins d'accès aux dossiers jpeg %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Création d'un tableau avec les deux adresses de dossiers
pth_xyj = [pth_xj; pth_yj];

% Mesure de la taille des adresses de dossiers pour pouvoir les réutiliser
plus tard
size_pth_xyj = size(pth_xyj(1,:)); % Est égal pour pth_xj et pth_yj, car le
chemin ne diffère pas en nombre de caractères

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Création de la liste des deux dossiers jpeg et de leur taille %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lancement d'une boucle pour faire le dossier JPEG x puis le y
for h = 1:2

    % Ouverture du chemin vers le dossier itéré
    cd(pth_xyj(h,1:size_pth_xyj(2)));

    % Liste de ce que le dossier actuel contient
    if h == 1
        liste_1 = dir('*.jpg');
        size_jpeg(1) = size(liste_1,1);
    elseif h == 2
        liste_2 = dir('*.jpg');
        size_jpeg(2) = size(liste_2,1);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% Recherche de la dimension des images %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Ouverture de la première image
image_1 = imread([pth_xyj(1,1:size_pth_xyj(2)),'/',liste_1(1) .name]);

if ndims(image_1) == 3
    % Définition de la première image du dossier en rgb
    rgb_scale = image_1;
    % Conversion de la première image en gris
    gray_scale = rgb2gray(rgb_scale);
elseif ndims(image_1) == 2
    % Définition de la première image du dossier en gris
    gray_scale = image_1;
end

% Définition des dimensions de l'image en gris (supposées être identique
pour toutes les images du dossier)
dim_im = size(gray_scale);
```

4. Coordonnees.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Paramètres de la caméra %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Appel du fichier slope.mat
cd(pth_d);
load('slope.mat', 'dxo', 'dyo');

% Position x de la caméra
xc = dxo;
% Position y de la caméra
yc = dyo;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Position réelle de chaque pixel %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Mesures réelles de l'image
prompt = {'Largueur image:','Longueur image:','y1:','x1:','hauteur'};
dlgtitle = 'Données images';
dims = [1 35];
definput = {'150','180','-125','50','610'};
image_infos = str2double(inputdlg(prompt,dlgtitle,dims,definput));

% Position z de la caméra
zc = image_infos(5);
% Position z du fil
zf = image_infos(5);

for h = 1:2

    % Calcul de la dimension réelle du pixel
    dpix = image_infos(h)/dim_im(h);

    % Enregistrement des données en fonction de leur axe
    if h == 1
        % Calcul de la position exacte de chaque pixel
        m(1) = image_infos(h+2) + image_infos(h) - (dpix/2);
        for i = 2:dim_im(h)
            m(i)= m(i-1) - dpix;
        end
        ym = m;
        dy = dpix;
    elseif h == 2
        % Calcul de la position exacte de chaque pixel
        m(1) = image_infos(h+2) + (dpix/2);
        for i = 2:dim_im(h)
            m(i)= m(i-1)+ dpix;
        end
        xm = m;
        dx = dpix;
    end

end
```

5. Intensite.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% Calcul de la position du fil avec le centre de masse %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lancement d'une boucle pour faire le dossier JPEG x puis le y
for h = 1:2

    intensite = zeros(dim_im(1),dim_im(2));
    s_centro = zeros(dim_im(1),dim_im(2));
    num = zeros(dim_im(1),dim_im(2));
    den = zeros(dim_im(1),dim_im(2));

    % Lecture de chaque image
    for k = 1:270
        % Ouverture de l'image
        image = imread([pth_xyj(h,1:size_pth_xyj(2)),'/',liste_1(k)
.name]);
        % Valeur de la position du fil lors de l'acquisition
        nb = str2double(liste_1(k).name(1:end-4));
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                % Pour une intensité supérieure à 60
                if image(i,j) > 60
                    % Calcul du numérateur et dénominateur
                    num(i,j) = double(num(i,j)) +
double(nb)*double(image(i,j));
                    den(i,j) = double(den(i,j)) + double(image(i,j));
                end
                % Réalisation de la carte des intensités maximales
                if image(i,j) > intensite(i,j)
                    intensite(i,j) = image(i,j);
                end
            end
        end
    end

    % Calcul du centre de masse
    for i = 1:dim_im(1)
        for j = 1:dim_im(2)
            s_centro(i,j) = double(num(i,j))/double(den(i,j));
        end
    end

    % Sauvegarde des paramètres dans des variables distinctes
    if h == 1
        intensite_x = intensite;
        xs = s_centro;
    elseif h == 2
        intensite_y = intensite;
        ys = s_centro;
    end
end

% Création de l'image des intensités maximales en x et y
intensite = (intensite_x + intensite_y)/2;
image_fin = uint8(intensite);
```

6. Pentes_X.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Tableau avec pente de chaque pixel
wx = ([]);
wx_max = 0;
wx_min = 0;

%Balayage des x
for i = 1:dim_im(1)
    %Balayage des y
    for j = 1:dim_im(2)
        if isnan(xs(i,j)) == 0
            %Calcul de la pente de chaque pixel
            wx(i,j) = (1/2)*(((xm(j)-xs(i,j))/zf)+((xm(j)-xc)/zc));
            %Calcul de la valeur max et la valeur min de la pente
            if wx(i,j) > wx_max
                wx_max = wx(i,j);
            elseif wx(i,j) < wx_min
                wx_min = wx(i,j);
            end
        elseif isnan(xs(i,j)) == 1
            wx(i,j) = 0;
        end
    end
end
```

7. Pentes_Y.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Calcul des pentes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Tableau avec pente de chaque pixel
wy = ([]);
wy_max = 0;
wy_min = 0;

%Balayage des x
for i = 1:dim_im(1)
    %Balayage des y
    for j = 1:dim_im(2)
        if isnan(ys(i,j)) == 0
            %Calcul de la pente de chaque pixel
            wy(i,j) = (1/2)*(((ym(i)-ys(i,j))/zf)+((ym(i)-yc)/zc));
            %Calcul de la valeur max et la valeur min de la pente
            if wy(i,j) > wy_max
                wy_max = wy(i,j);
            elseif wy(i,j) < wy_min
                wy_min = wy(i,j);
            end
        elseif isnan(ys(i,j)) == 1
            wy(i,j) = 0;
        end
    end
end
```

8. Integrad_2D.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% Masquage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la variable satisfaction du masque
satisfaction = 0;

% Demande de la zone à intégrer
while satisfaction == 0

    % Fermeture de l'image précédente
    close(gcf);

    % Création d'un tableau qui servira d'image masquée
    intensite_mask = image_fin;

    % Affichage de l'image de base
    figure, imshow(intensite_mask), colormap('jet');

    % Demande du type de masque voulu
    answer1 = questdlg('Quel type de masque veux-tu ?', ...
        'Intensité minimum', ...
        'Ellipse','Rectangle','Ellipse');

    % Effet suivant la réponse
    switch answer1
        case 'Ellipse'
            e = imellipse;
        case 'Rectangle'
            e = imrect;
    end

    % Création d'une carte binaire avec ce qui est dans et hors du masque
    mirror_ok = createMask(e);

    % Définition des coordonnées du masque
    pos = getPosition(e);

    % NaN sur la carte d'intensité masqué hors du masque
    for i = 1:(dim_im(1)*dim_im(2))
        if mirror_ok(i) == 0
            intensite_mask(i) = NaN;
        end
    end

    % Affichage de l'image masquée
    imshow(intensite_mask), colormap('jet');

    % Demande de satisfaction
    answer2 = questdlg('Ok?', ...
        'Intensité minimum', ...
        'Oui','Non','Oui');

    % Effet en fonction de la réponse
    switch answer2
        case 'Oui'
```

```
            satisfaction = 1;
        case 'Non'
            satisfaction = 0;
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Cartes et paramètres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Tableau des pentes en x et y
X = wx;
Y = wy;

% Définition d'un point de référence
z_centre = 0;

% Tableau des hauteurs (toutes égales à 0 au début)
Z = zeros(dim_im(1), dim_im(2));

% Définition du centre du masque
y0 = round(pos(2)) + round(pos(4)/2);
x0 = round(pos(1)) + round(pos(3)/2);

% Définition de la hauteur de référence au centre de l'image
Z(y0,x0) = z_centre;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Intégration trapézoïdale %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for h = 0:1

    y0 = y0+h;

    % Calcul du point à côté du centre du masque pour la seconde
intégration trapézoïdale
    if h == 1
        Z(y0,x0) = ((mirror_ok(y0,x0-1)*Z(y0,x0-1)...
                    + mirror_ok(y0,x0+1)*Z(y0,x0+1) + mirror_ok(y0-
1,x0)*Z(y0-1,x0) ...
                    + mirror_ok(y0+1,x0)*Z(y0+1,x0))/(mirror_ok(y0,x0-1) +
mirror_ok(y0,x0+1)...
                    + mirror_ok(y0-1,x0) + mirror_ok(y0+1,x0)))...
                    + ((mirror_ok(y0,x0-1)*((dx/2)*(X(y0,x0-1) +
X(y0,x0)))...
                    - mirror_ok(y0,x0+1)*((dx/2)*(X(y0,x0+1) +
X(y0,x0)))...
                    + mirror_ok(y0-1,x0)*((dy/2)*(Y(y0-1,x0) +
Y(y0,x0)))...
                    - mirror_ok(y0+1,x0)*((dy/2)*(Y(y0+1,x0) +
Y(y0,x0))))...
                    / (mirror_ok(y0,x0-1) + mirror_ok(y0,x0+1)...
                    + mirror_ok(y0-1,x0) + mirror_ok(y0+1,x0)));
    end

    % Intègration vers les X+/Y+
    for i = 1:max(dim_im)
```

```
        % Si hors masque, stop
        if mirror_ok(y0+i,x0+i) == 0
            break;
        % Si dans le masque, intégration
        else
            Z(y0+i,x0+i) = Z(y0+i-1,x0+i-1)...
                + (dx/2)*(X(y0+i-1,x0+i-1)+X(y0+i,x0+i-1))...
                + (dy/2)*(Y(y0+i-1,x0+i)+Y(y0+i,x0+i));
            max_diag = i;
        end
    end

    % Intégration vers les X-/Y-
    for i = 1:max(dim_im)
        if  mirror_ok(y0-i,x0-i) == 0
            break;
        % Si dans le masque, intégration
        else
            Z(y0-i,x0-i) = Z(y0-i+1,x0-i+1)...
                - (dx/2)*(X(y0-i+1,x0-i+1)+X(y0-i,x0-i+1))...
                - (dy/2)*(Y(y0-i+1,x0-i)+Y(y0-i,x0-i));
            min_diag = i;
        end
    end


    % Si le masque est orienté horizontalement
    if pos(3)>pos(4)
        % Définition du nombre de vagues d'intégrations à gauche et à
droite
        nb_int = ceil(pos(3)/pos(4));
        % Définition du sens de départ pour intégrer côté droit
        sens = 1;
        % Défini les x et y de départ
        x = x0-min_diag;
        y = y0-min_diag;
        % Les vagues d'intégrations à droite
        for i = 1:nb_int
            % Pour intégrer dans le sens X+/Y-
            if sens == 1
                % Pour s'assurer que le point de départ est dans le masque
                while mirror_ok(y,x) == 1
                    % intégration à partir de (y,x)
                    for j = 1:max(dim_im)
                        % Si le point suivant à intégrer est hors du masque
                        if  mirror_ok(y-j,x+j) == 0
                            break;
                        % S'il est dans le masque, intégration
                        else
                            Z(y-j,x+j) = Z(y-j+1,x+j-1)...
                                + (dx/2)*(X(y-j+1,x+j-1)+X(y-j+1,x+j))...
                                - (dy/2)*(Y(y-j+1,x+j)+Y(y-j,x+j));
                        end
                    end
                    % Définition du point suivant à partir duquel intégrer
                    x = x+1;
                    y = y+1;
                end
                % Passage au sens suivant
                sens = 2;
                % Définition du point de départ du sens suivant
```

```
            x = x-1;
            y = y-1;
        % Pour intégrer dans le sens X+/Y+, avec les mêmes commentaires
        elseif sens == 2
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y+j,x+j) == 0
                        break;
                    else
                        Z(y+j,x+j) = Z(y+j-1,x+j-1)...
                            + (dx/2)*(X(y+j-1,x+j-1)+X(y+j-1,x+j))...
                            + (dy/2)*(Y(y+j-1,x+j)+Y(y+j,x+j));
                    end
                end
                x = x+1;
                y = y-1;
            end
            sens = 1;
            x = x-1;
            y = y+1;
        end
    end
    % Passage à l'intégration du côté gauche
    sens = 1;
    % définition du point de départ
    x = x0+max_diag;
    y = y0+max_diag;
    for i = 1:nb_int
        % Intégration vers les X-/Y+ avec les mêmes commentaires
        if sens == 1
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y+j,x-j) == 0
                        break;
                    else
                        Z(y+j,x-j) = Z(y+j-1,x-j+1)...
                            - (dx/2)*(X(y+j-1,x-j+1)+X(y+j-1,x-j))...
                            + (dy/2)*(Y(y+j-1,x-j)+Y(y+j,x-j));
                    end
                end
                x = x-1;
                y = y-1;
            end
            sens = 2;
            x = x+1;
            y = y+1;
        % Intégration vers les X-/Y-, avec les mêmes commentaires
        elseif sens == 2
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y-j,x-j) == 0
                        break;
                    else
                        Z(y-j,x-j) = Z(y-j+1,x-j+1)...
                            - (dx/2)*(X(y-j+1,x-j+1)+X(y-j+1,x-j))...
                            - (dy/2)*(Y(y-j+1,x-j)+Y(y-j,x-j));
                    end
                end
                x = x-1;
                y = y+1;
            end
```

```
            sens = 1;
            x = x+1;
            y = y-1;
        end
    end

% Dans le cas où le masque est vertical
elseif pos(4)>= pos(3)
    % Définition du nombre de vagues
    nb_int = ceil(pos(4)/pos(3));
    % Définition du sens de départ pour intégrer vers le haut
    sens = 1;
    % Définition du point de départ
    x = x0-min_diag;
    y = y0-min_diag;
    % Les vagues d'intégration vers le haut
    for i = 1:nb_int
        % Intégration vers les X-/Y+
        if sens == 1
            % Vérification que le point de départ est dans le masque
            while mirror_ok(y,x) == 1
                % Intégration à partir de ce point
                for j = 1:max(dim_im)
                    % Si le point suivant est hors du masque
                    if  mirror_ok(y+j,x-j) == 0
                        break;
                    % S'il est dans le masque, intégration
                    else
                        Z(y+j,x-j) = Z(y+j-1,x-j+1)...
                            - (dx/2)*(X(y+j-1,x-j+1)+X(y+j-1,x-j))...
                            + (dy/2)*(Y(y+j-1,x-j)+Y(y+j,x-j));
                    end
                end
                % Définition du point suivant
                x = x+1;
                y = y+1;
            end
            % Passage au sens suivant
            sens = 2;
            % Définition du point de départ
            x = x-1;
            y = y-1;
        % Intégration dans le sens X+/Y+, mêmes commentaires
        elseif sens == 2
            while mirror_ok(y,x) == 1
                for j = 1:max(dim_im)
                    if  mirror_ok(y+j,x+j) == 0
                        break;
                    else
                        Z(y+j,x+j) = Z(y+j-1,x+j-1)...
                            + (dx/2)*(X(y+j-1,x+j-1)+X(y+j-1,x+j))...
                            + (dy/2)*(Y(y+j-1,x+j)+Y(y+j,x+j));
                    end
                end
                x = x-1;
                y = y+1;
            end
            sens = 1;
            x = x+1;
            y = y-1;
        end
```

```
        end
        % Passage à l'intégration vers le bas
        sens = 1;
        x = x0+max_diag;
        y = y0+max_diag;
        for i = 1:nb_int
            % Intégration dans le sens X+/Y-, mêmes commentaires
            if sens == 1
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y-j,x+j) == 0
                            break;
                        else
                            Z(y-j,x+j) = Z(y-j+1,x+j-1)...
                                + (dx/2)*(X(y-j+1,x+j-1)+X(y-j+1,x+j))...
                                - (dy/2)*(Y(y-j+1,x+j)+Y(y-j,x+j));
                        end
                    end
                    x = x-1;
                    y = y-1;
                end
                sens = 2;
                x = x+1;
                y = y+1;
            % Intégration dans le sens X-/Y-, mêmes commentaires
            elseif sens == 2
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y-j,x-j) == 0
                            break;
                        else
                            Z(y-j,x-j) = Z(y-j+1,x-j+1)...
                                - (dx/2)*(X(y-j+1,x-j+1)+X(y-j+1,x-j))...
                                - (dy/2)*(Y(y-j+1,x-j)+Y(y-j,x-j));
                        end
                    end
                    x = x+1;
                    y = y-1;
                end
                sens = 1;
                x = x-1;
                y = y+1;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Paramètres de Southwell %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NaN partout où l'intégration trapézoïdale n'est pas passée
for i = 1:(dim_im(1)*dim_im(2))
    if mirror_ok(i) == 0
        Z(i) = NaN;
    end
end

% Création d'un tableau vide pour l'intégration de Southwell
Z_southwell = ([]);
```

```
% Définition de la dimension à ce tableau en le rendant égal à Z
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        Z_southwell(i,j) = Z(i,j);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Intégration de Southwell %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition du nombre d'itérations
for h = 1:(round((pos(3)*pos(4)))/1)
    % Pour les Y dans le masque
    for i = round(pos(2)):round(pos(2)+pos(4))
        % Pour les X dans le masque
        for j = round(pos(1)):round(pos(1)+pos(3))
            % NaN si pas dans le masque
            if mirror_ok(i,j) == 0
                Z_southwell(i,j) = NaN;

            else
                % Si dans le masque, mais qu'un pixel autour vaut NaN, le
pixel autour égal à 0 pour pouvoir calculer
                if mirror_ok(i,j-1) == 0
                    Z(i,j-1) = 0;
                end
                if mirror_ok(i,j+1) == 0
                    Z(i,j+1) = 0;
                end
                if mirror_ok(i-1,j) == 0
                    Z(i-1,j) = 0;
                end
                if mirror_ok(i+1,j) == 0
                    Z(i+1,j) = 0;
                end
                % Réalisation de l'intégration de Southwell
                Z_southwell(i,j) = ((mirror_ok(i,j-1)*Z(i,j-1)...
                    + mirror_ok(i,j+1)*Z(i,j+1) + mirror_ok(i-1,j)*Z(i-1,j)
...
                    + mirror_ok(i+1,j)*Z(i+1,j))/(mirror_ok(i,j-1) +
mirror_ok(i,j+1)...
                    + mirror_ok(i-1,j) + mirror_ok(i+1,j)))...
                    + ((mirror_ok(i,j-1)*((dx/2)*(X(i,j-1) + X(i,j)))...
                    - mirror_ok(i,j+1)*((dx/2)*(X(i,j+1) + X(i,j)))...
                    + mirror_ok(i-1,j)*((dy/2)*(Y(i-1,j) + Y(i,j)))...
                    - mirror_ok(i+1,j)*((dy/2)*(Y(i+1,j) + Y(i,j))))...
                    / (mirror_ok(i,j-1) + mirror_ok(i,j+1)...
                    + mirror_ok(i-1,j) + mirror_ok(i+1,j)));
            end
        end
    end
    % Définition de Z avec les valeurs de Z_southwell pour pouvoir réitérer
    for i = 1:dim_im(1)
        for j = 1:dim_im(2)
            Z(i,j) = Z_southwell(i,j);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%% Affichage %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

surf(Z(1:dim_im(1),1:dim_im(2)));
```

Annexe 5

1. Structure de base de la fonction GUIDE

```
function varargout = Structure(varargin)
% STRUCTURE MATLAB code for Structure.fig
%      STRUCTURE, by itself, creates a new STRUCTURE or raises the existing
%      singleton*.
%
%      H = STRUCTURE returns the handle to a new STRUCTURE or the handle to
%      the existing singleton*.
%
%      STRUCTURE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STRUCTURE.M with the given input
arguments.
%
%      STRUCTURE('Property','Value',...) creates a new STRUCTURE or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Structure_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Structure_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 13-Aug-2019 11:50:39

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

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


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

% Choose default command line output for Structure
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

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


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

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

2. Octopus.m

```
function varargout = Octopus(varargin)
% OCTOPUS MATLAB code for Octopus.fig
%      OCTOPUS, by itself, creates a new OCTOPUS or raises the existing
%      singleton*.
%
%      H = OCTOPUS returns the handle to a new OCTOPUS or the handle to
%      the existing singleton*.
%
%      OCTOPUS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in OCTOPUS.M with the given input arguments.
%
%      OCTOPUS('Property','Value',...) creates a new OCTOPUS or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Octopus_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Octopus_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 21-Jun-2019 08:58:18

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

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


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

```
% Choose default command line output for Octopus
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Vérification infos.mat %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Vérification de l'existence du fichier infos.mat, sinon recréation d'un
de base
if ~exist('infos.mat')
    % Lien avec le dossier d'origine avec les différents scripts
    pth = cd;
    % Adresse du dernier fichier visité
    dern_pth_d = cd;
    % Sauvegarde de tout dans un fichier infos.mat
    save('infos.mat', 'pth', 'dern_pth_d');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

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


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

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Calculer les pentes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

% Définition de la position de la fenêtre avant de la sauver
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de la fenêtre actuelle
delete(gcf);
```

```
% Lancement de la fenêtre 'Ouvrir'
run Ouvrir;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Calculer la surface %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton3_Callback(hObject, eventdata, handles)

% Définition de la position de la fenêtre avant de la sauver
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de la fenêtre actuelle
delete(gcf);

% Lancement de la fenêtre 'Ouvrir'
run Ouvrir2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Fermer %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

% Définition de la position de la fenêtre avant de la sauver
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de tout
clear; clc; delete(gcf);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Définition de la position de la fenêtre avant de la sauver
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de tout
clear; clc; delete(gcf);
```

3. Ouvrir.m

```
function varargout = Ouvrir(varargin)
% OUVRIR MATLAB code for Ouvrir.fig
%      OUVRIR, by itself, creates a new OUVRIR or raises the existing
%      singleton*.
%
%      H = OUVRIR returns the handle to a new OUVRIR or the handle to
%      the existing singleton*.
%
%      OUVRIR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in OUVRIR.M with the given input arguments.
%
%      OUVRIR('Property','Value',...) creates a new OUVRIR or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Ouvrir_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Ouvrir_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 21-Jun-2019 10:44:36

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

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


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

```
% Choose default command line output for Ouvrir
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

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


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

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


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Sélectionner dossier images %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

% Chargement du fichier infos.mat
load('infos.mat');

% Demande de sélection du dossier image
```

```
pth_r = uigetdir(dern_pth_d,'Sélectionne ton dossier');

% En fonction du choix de l'opérateur
if pth_r == 0 %Si clic sur annuler

    % Rien dans la barre d'édition
    set(handles.edit1, 'String', '');

else %Si clic sur un dossier et ouvrir

    % Écriture de l'adresse dans la barre d'édition
    set(handles.edit1, 'String', pth_r);

    % Travail dans le dossier choisi
    cd(pth_r);

    % Dossier parent
    cd ../;

    % Enregistrement du dossier comme dernier dossier dans lequel on a
choisi un dossier
    dern_pth_d = cd;

    % Retour dans le dossier avec les scripts
    cd(pth);

    % Sauvegarde de l'adresse du dossier choisi et celui dans lequel il se
trouve
    save('infos.mat', 'dern_pth_d', 'pth_r', '-append');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Suivant %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton3_Callback(hObject, eventdata, handles)

% Lecture de l'adresse inscrite dans le cadre
pth_r = get(handles.edit1, 'String');

% Vérification de si cette adresse existe
if exist(pth_r) % Si oui

    % Définition la position de la fenêtre et sauvegarde
    pos = get(gcf, 'Position');
    save('infos.mat', 'pos', 'pth_r', '-append');

    % Vérification de si le sous-dossier x existe
    if exist([pth_r, '/x']) % Si oui

        % Vérification de si le sous-dossier y existe
        if exist([pth_r, '/y']) % Si oui

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%% Importation des données des images %%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            % Importation des infos d'Octopus
```

```
            load('infos.mat');

            % Création des chemins vers les dossiers x et y du dossier
image
            pth_rx=[pth_r,'/x'];
            pth_ry=[pth_r,'/y'];

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Enregistrement des images du dossier X dans une matrice 3D %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            % Démarrage de la barre d'état d'avancement de l'importation
            time = waitbar(0,'Ouverture du dossier x');

            % Travail dans le dossier x
            cd(pth_rx);

            % Sélectionn des fichiers bmp, png, gif, hdf, bmp, pgm, ppm,
jpg
            bmp = dir('*.bmp');
            png = dir('*.png');
            gif = dir('*.gif');
            hdf = dir('*.hdf');
            bpm = dir('*.bpm');
            pgm = dir('*.pgm');
            ppm = dir('*.ppm');
            jpg = dir('*.jpg');

            % Création d'un tableau avec tous les fichiers trouvés
            fil = [bmp; png; gif; hdf; bpm; pgm; ppm; jpg];

            % Taille du dossier actuel
            size_fold = size(fil);

            % Avancement de la barre d'état
            waitbar(0.2,time,'Importation des données x');

            % Boucle qui importe les infos des images dans un matrice 3D
            for i = 1:size_fold

                % Extraction des infos de l'image i
                image = imread([pth_rx,'/',fil(i) .name]);

                % Définition du nom (qui est un chiffre) de l'image
                nb = str2double(fil(i).name(1:end-4));

                % Conversion de l'image en niveau de gris pour avoir une
image 2D
                % Si format rgb
                if ndims(image) == 3
                    % Ouverture de la première image du dossier en rgb
                    rgb_scale = image;
                    % Conversion de la première image en gray
                    gray_scale = rgb2gray(rgb_scale);
                    % Si format gray
                elseif ndims(image) == 2
                    % Enregistrement de l'image
                    gray_scale = image;
                end
```

```
                % Enregistrement de la dimension 2D de l'image
                dim_im = size(gray_scale);

                % Double boucle pour enregistrer les infos de l'image 2D à
sa position en 3D
                for j = 1:dim_im(1)
                    for k = 1:dim_im(2)
                        X(j,k,nb) = gray_scale(j,k);
                    end
                end
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Enregistrement des images du dossier Y dans une matrice 3D %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            % Avancement de la barre d'état
            waitbar(0.4,time,'Ouverture du dossier y');

            % Travail dans le dossier y
            cd(pth_ry);

            % Sélection des fichiers bmp, png, gif, hdf, bmp, pgm, ppm, jpg
            bmp = dir('*.bmp');
            png = dir('*.png');
            gif = dir('*.gif');
            hdf = dir('*.hdf');
            bpm = dir('*.bpm');
            pgm = dir('*.pgm');
            ppm = dir('*.ppm');
            jpg = dir('*.jpg');

            % Création d'un tableau avec tous les fichiers trouvés
            fil = [bmp; png; gif; hdf; bpm; pgm; ppm; jpg];

            % Taille du dossier actuel
            size_fold = size(fil);

            % Avancement de la barre d'état
            waitbar(0.6,time,'Importation des données y');

            % Boucle qui importe les infos des images dans un matrice 3D
            for i = 1:size_fold

                % Extraction des infos de l'image i
                image = imread([pth_ry,'/',fil(i) .name]);

                % Définition du nom (qui est un chiffre) de l'image
                nb = str2double(fil(i).name(1:end-4));

                % Conversion de l'image en niveau de gris pour avoir une
image 2D
                % Si format rgb
                if ndims(image) == 3
                    % Ouverture de la première image du dossier en rgb
                    rgb_scale = image;
                    % Conversion de la première image en gray
                    gray_scale = rgb2gray(rgb_scale);
```

```
                    % Si format gray
                elseif ndims(image) == 2
                    % Enregistrement de l'image
                    gray_scale = image;
                end

                % Enregistrement de la dimension 2D de l'image
                dim_im = size(gray_scale);

                % Double boucle pour enregistrer les infos de l'image 2D à
sa position en 3D
                for j = 1:dim_im(1)
                    for k = 1:dim_im(2)
                        Y(j,k,nb) = gray_scale(j,k);
                    end
                end
            end

            % Avancement de la barre d'état
            waitbar(0.8,time,'Importation des données dans Octopus');

            % Travail dans le dossier image
            cd(pth_r);

            % Sauvegarde des matrices 3D X et Y et des dimensions des
images
            save('datas.mat','X','Y', 'dim_im');

            % Avancement de la barre d'état
            waitbar(1,time,'Données importées');

            % Pause pour marquer la fin de la manipulation et fermeture de
la barre d'état
            pause(1);
            close(time);

            % Travail dans le dossier des scripts
            cd(pth);
            % Fermeture de la fenêtre actuelle
            delete(gcf);

            % Ouverture de la fenêtre 'Paramètres'
            run Parametres;

        else % Si non
            % Explication de l'erreur
            msg = msgbox('Il manque le dossier x et/ou y dans ce dossier.
Veuillez vérifier que le dossier sélectionné contient bien les dossiers
nécessaires.');
        end

    else % Si non
        % Explication de l'erreur
        msg = msgbox('Il manque le dossier x et/ou y dans ce dossier.
Veuillez vérifier que le dossier sélectionné contient bien les dossiers
nécessaires.');
    end

else % Si non
```

```
    % Explication que l'adresse n'existe pas
    msg = msgbox('Attention : ce dossier n''existe pas. Veuillez vérifier
l''adresse exacte ou la sélectionner avec le bouton ci-dessus.');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton4_Callback(hObject, eventdata, handles)

% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir perdre la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de ce qui est utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        delete(gcf);
        % Ouverture de la fenêtre 'Octopus'
        run Octopus;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer2

    case 'Oui'
        % Ouverture de infos.mat et ne sauve que ce qui est utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        % Fermeture de tout
        clear; clc; delete(gcf);

    case 'Non'
        % Aucune action
```

```
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Fonctions non-utilisées %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Barre d'édition 1
function edit1_Callback(hObject, eventdata, handles)
```

4. Parametres.m

```
function varargout = Parametres(varargin)
% PARAMETRES MATLAB code for Parametres.fig
%      PARAMETRES, by itself, creates a new PARAMETRES or raises the
existing
%      singleton*.
%
%      H = PARAMETRES returns the handle to a new PARAMETRES or the handle
to
%      the existing singleton*.
%
%      PARAMETRES('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PARAMETRES.M with the given input
arguments.
%
%      PARAMETRES('Property','Value',...) creates a new PARAMETRES or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Parametres_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Parametres_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 24-Jun-2019 11:35:34

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

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


% --- Executes just before Parametres is made visible.
function Parametres_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Parametres (see VARARGIN)

% Choose default command line output for Parametres
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Affichage des paramètres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier images
cd(pth_r);

% Positions x et y de la caméra
load('slope.mat');
set(handles.edit1, 'String', dxo);
set(handles.edit2, 'String', dyo);

% Chargement des données utiles au calcul de la position du fil et des
pentes
load('mirror.mat');
load('datas.mat');

% Changement du nom de la variable step
st = eval('step');

% Affichage de la hauteur et la taille de pas du fil
set(handles.edit3, 'String', h, 'Enable', 'inactive');
set(handles.edit8, 'String', st, 'Enable', 'inactive');

% Calcul des angles extrêmes en y
alphay1 = atan((0-(-235))/709);
alphay_end = atan((0-(-235)+309)/709);

% Calcul du pas d'angle en y
d_alphay = (alphay_end - alphay1)/dim_im(1);

% Détermination des positions extrêmes en y
ym1 = dyo + h * tan(alphay1 + (d_alphay*(0.5)));
ym_end = dyo + h * tan(alphay1 + (d_alphay*(dim_im(1)-1+0.5)));

% Calcul des angles extrêmes en x
alphax1 = atan(((-31)-135)/sqrt((709^2)+((0-ym1)^2)));
alphax_end = atan(((-31)+325-135)/sqrt((709^2)+(((-235)-ym1)^2)));
```

```
% Calcul du pas d'angle en x
d_alphax = (alphax_end - alphax1)/dim_im(2);

% Détermination des positions extrêmes en x
xm1 = dxo + (sqrt(((ym1-dyo)^2)+(h^2))*tan(alphax1+...
    ((0.5)*d_alphax)));
xm_end = dxo + (sqrt(((ym1-dyo)^2)+(h^2))*tan(alphax_end));

% Affichage de toutes les valeurs qu'il reste à présenter
set(handles.edit5, 'String', ym_end-ym1, 'Enable', 'inactive');
set(handles.edit4, 'String', xm_end-xm1, 'Enable', 'inactive');
set(handles.edit7, 'String', xm1, 'Enable', 'inactive');
set(handles.edit6, 'String', ym1, 'Enable', 'inactive');
set(handles.edit9, 'String', x1, 'Enable', 'inactive');
set(handles.edit10, 'String', y1, 'Enable', 'inactive');

% Travail dans le dossier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

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


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

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

% Chargement du fichier infos.mat
load('infos.mat');

% Travail dans le dossier image
cd(pth_r);

% Chargement du fichier datas.mat
load('datas.mat');

% Création des variables à l'aide de chaque case d'édition
xc = str2double(get(handles.edit1, 'String'));
yc = str2double(get(handles.edit2, 'String'));
zc = str2double(get(handles.edit3, 'String'));
```

```
zf = str2double(get(handles.edit3, 'String'));
larg = str2double(get(handles.edit5, 'String'));
long = str2double(get(handles.edit4, 'String'));
x1 = str2double(get(handles.edit7, 'String'));
y1 = str2double(get(handles.edit6, 'String'));
st = str2double(get(handles.edit8, 'String'));
filx = str2double(get(handles.edit9, 'String'));
fily = str2double(get(handles.edit10, 'String'));

% Ajout des variables au fichier datas.mat
save('datas.mat', 'xc', 'yc', 'zc', 'zf', 'larg', 'long', 'x1',...
    'y1', 'st', 'filx', 'fily', '-append');

% Calcul du centre de masse pour la matrice X et puis Y
for h = 1:2

    % Dans la première, par rapport à X
    if h == 1
        % Chargement des données X dans la matrice datas
        datas = X;
        % Lancement d'une barre de progression
        time = waitbar(0,'Calcul du centre de masse pour x');
        % Dans la deuxième, par rapport à Y
    elseif h == 2
        % Chargement des données Y dans la matrice datas
        datas = Y;
        % Évolution de la barre de progression
        waitbar(0.3,time,'Calcul du centre de masse pour y');
    end

    % Création des matrices centre de masse, numérateur et dénominateur de
la taille d'une image
    s_centro = zeros(dim_im(1),dim_im(2));
    num = zeros(dim_im(1),dim_im(2));
    den = zeros(dim_im(1),dim_im(2));

    % Nécessaire pour avoir la 3ème dimension
    dim_im = size(datas);

    % Calcul du numérateur et du dénominateur de chaque pixel
    for k = 1:dim_im(3)
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                num(i,j) = double(num(i,j)) +
double(k)*double(datas(i,j,k));
                den(i,j) = double(den(i,j)) + double(datas(i,j,k));
            end
        end
    end

    % Calcul du centre de masse de chaque pixel par rapport à la position
de départ en valeur numéro d'image
    for i = 1:dim_im(1)
        for j = 1:dim_im(2)
            s_centro(i,j) = double(num(i,j))/double(den(i,j));
        end
    end

    % Calcul de la position x exacte du fil par centre de masse
    if h == 1
```

```
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                xs(i,j) = filx + (s_centro(i,j)*st);
            end
        end

        % Calcul de la position y exacte du fil par centre de masse
    elseif h == 2
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                ys(i,j) = fily + (s_centro(i,j)*st);
            end
        end
    end
end

% Évolution de la barre d'état
waitbar(0.6,time,'Détermination du pixel de référence');

% Détermination du pixel avec le plus grand écart entre son min et max
d'intensité
ecart_max = 0;
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        ecart = max(datas(i,j,:) - min(datas(i,j,:)));
        if ecart_max < ecart
            i_max = i;
            j_max = j;
            ecart_max = ecart;
        end
    end
end
for i = 1:dim_im(3)
    pixel_int(i) = datas(i_max, j_max, i);
end

% Évolution de la barre d'état
waitbar(0.8,time,'Calcul des tailles et positions de chaque pixel');

% Détermination de la taille d'un pixel
dx = long/dim_im(2);
dy = larg/dim_im(1);

% Détermination de la position de chaque pixel en x en 2D
xm(1:dim_im(1),1) = x1 + (dx/2);
for i = 2:dim_im(2)
    for j = 1:dim_im(1)
        xm(j,i) = xm(j,i-1) + dx;
    end
end

% Détermination de la position de chaque pixel en y en 1D
ym(1) = y1 + larg - (dy/2);
for i = 2:dim_im(1)
    ym(i) = ym(i-1) - dy;
end

% Sauvegarde des données utiles
save('datas.mat', 'xs', 'ys', 'pixel_int', 'dx', 'dy',...
    'xm', 'ym', '-append');
```

```
% Évolution de la barre d'état
waitbar(1,time,'Données calculées');

% Pause pour marquer la fin de la manipulation et fermeture de la barre
d'état
pause(1);
close(time);

% Travail dans le dossier des scripts
cd(pth);

% Définition de la position de la fenêtre et sauvegarde
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de la fenêtre actuelle
delete(gcf);

% Lancement de la fenêtre 'Centre de masse'
run Centre_de_masse;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir retourner à la sélection du
dossier ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        % Nettoyage de ce qu'il s'est passé entre les deux fenêtres
        cd(pth_r);
        delete('datas.mat');
        cd(pth);
        % Fermeture de la fenêtre
        delete(gcf);
        % Ouverture de la fenêtre 'Ouvrir'
        run Ouvrir;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer2

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        % Suppression du fichier datas.mat
        cd(pth_r);
        delete('datas.mat');
        cd(pth);
        % Fermeture de tout
        clear; clc; delete(gcf);

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Fonctions inutilisées %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Barre d'édition 1
function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 2
function edit2_Callback(hObject, eventdata, handles)

function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 3
function edit3_Callback(hObject, eventdata, handles)

function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% Barre d'édition 4
function edit4_Callback(hObject, eventdata, handles)

function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 5
function edit5_Callback(hObject, eventdata, handles)

function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 6
function edit6_Callback(hObject, eventdata, handles)

function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 7
function edit7_Callback(hObject, eventdata, handles)

function edit7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 8
function edit8_Callback(hObject, eventdata, handles)

function edit8_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 9
function edit9_Callback(hObject, eventdata, handles)

function edit9_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 10
function edit10_Callback(hObject, eventdata, handles)

function edit10_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

5. Centre_de_masse.m

```
function varargout = Centre_de_masse(varargin)
% CENTRE_DE_MASSE MATLAB code for Centre_de_masse.fig
%      CENTRE_DE_MASSE, by itself, creates a new CENTRE_DE_MASSE or raises
the existing
%      singleton*.
%
%      H = CENTRE_DE_MASSE returns the handle to a new CENTRE_DE_MASSE or
the handle to
%      the existing singleton*.
%
%      CENTRE_DE_MASSE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in CENTRE_DE_MASSE.M with the given input
arguments.
%
%      CENTRE_DE_MASSE('Property','Value',...) creates a new
CENTRE_DE_MASSE or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Centre_de_masse_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Centre_de_masse_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 24-Jun-2019 14:31:50

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

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


% --- Executes just before Centre_de_masse is made visible.
```

```
function Centre_de_masse_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Centre_de_masse (see VARARGIN)

% Choose default command line output for Centre_de_masse
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%% Affichage des cartes etde l valeur de slider %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier image
cd(pth_r);

% Chargement du fichier datas.mat
load('datas.mat');

% Travail dans le dossier des scripts
cd(pth);

% Affichage des centres de masse x et y
axes(handles.axes2);
imshow(xs, [min(min(xs)) max(max(xs))]), colormap('jet'), colorbar;
axes(handles.axes3);
imshow(ys, [min(min(ys)) max(max(ys))]), colormap('jet'), colorbar;

% Affichage du pixel de référence
axes(handles.axes4);
plot(1:size(pixel_int, 2), pixel_int);

% Affichage du slider et de sa valeur
if exist('value')
    set(handles.slider1, 'Value', value);
end
set(handles.text5, 'String', round(get(handles.slider1,'Value')));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

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

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

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Slider %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function slider1_Callback(hObject, eventdata, handles)

% Valeur du slider
value = round(get(handles.slider1,'Value'));

% Valeur à la case numérique
set(handles.text5, 'String', value);

% Chargement de infos.mat et travail dans le dossier image
load('infos.mat');
cd(pth_r);

% Sauvegarde de la valeur du slider
save('datas.mat', 'value', '-append');

% Travail dans le dossier des scripts
cd(pth);

% Lancement d'un calcul du centre de masse, mais avec une valeur limite
run Centre_de_masse_boutons;

% Affichage des centres de masse
axes(handles.axes2);
imshow(xs, [min(min(xs)) max(max(xs))]), colormap('jet'), colorbar;
axes(handles.axes3);
imshow(ys, [min(min(ys)) max(max(ys))]), colormap('jet'), colorbar;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Intensités maximales %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function radiobutton2_Callback(hObject, eventdata, handles)

% Décochage du bouton 'Centre de masse'
set(handles.radiobutton3, 'Value', 0.0);

% Bouton 'Intensité maximum' inactivable
set(handles.radiobutton2, 'Enable', 'inactive');

% Slider inactif
set(handles.slider1, 'Enable', 'inactive');
```

```
% Bouton 'Centre de masse' actif
set(handles.radiobutton3, 'Enable', 'on');

% Chargement des fichier infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Calcul du centre de masse pour la matrice X et puis Y
for h = 1:2

    % Dans la première, par rapport à X
    if h == 1
        % Chargement des données X dans la matrice datas
        datas = X;
        % Dans la deuxième, par rapport à Y
    elseif h == 2
        % Chargement des données Y dans la matrice datas
        datas = Y;
    end

    % Création des matrices centre de masse et intensité max
    max_int = zeros(dim_im(1),dim_im(2));
    s_centro = zeros(dim_im(1),dim_im(2));

    % Nécessaire pour avoir la 3ème dimension
    dim_im = size(datas);

    % Calcul de la position par rapport à l'intensité max
    for k = 1:dim_im(3)
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                if datas(i,j,k) >= max_int(i,j)
                    max_int(i,j) = datas(i,j,k);
                    s_centro(i,j) = k;
                end
            end
        end
    end

    % Calcul de la position x exacte du fil par centre de masse
    if h == 1
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                xs(i,j) = filx + (s_centro(i,j)*st);
            end
        end

        % Calcul de la position y exacte du fil par centre de masse
    elseif h == 2
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                ys(i,j) = fily + (s_centro(i,j)*st);
            end
        end
    end
end

% Sauvegarde des données modifiées
save('datas.mat', 'xs', 'ys', '-append');
```

```
% Travail dans le dossier des scripts
cd(pth);

% Affichage des centres de masse x et y
axes(handles.axes2);
imshow(xs, [min(min(xs)) max(max(xs))]), colormap('jet'), colorbar;
axes(handles.axes3);
imshow(ys, [min(min(ys)) max(max(ys))]), colormap('jet'), colorbar;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Centre de masse %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function radiobutton3_Callback(hObject, eventdata, handles)

% Décochage du bouton 'Intensité maximum'
set(handles.radiobutton2, 'Value', 0.0);

% Bouton 'Centre de masse' inactif
set(handles.radiobutton3, 'Enable', 'inactive');

% Slider actif
set(handles.slider1, 'Enable', 'on');

% Bouton 'Intensité maximum' actif
set(handles.radiobutton2, 'Enable', 'on');

% Valeur du slider et sauvegarde
value = round(get(handles.slider1,'Value'));
load('infos.mat');
cd(pth_r);
save('datas.mat', 'value', '-append');

% Travail dans le dossier des scripts
cd(pth);

% Calcul du centre de masse
run Centre_de_masse_boutons;

% Affichage des centres de masse x et y
axes(handles.axes2);
imshow(xs, [min(min(xs)) max(max(xs))]), colormap('jet'), colorbar;
axes(handles.axes3);
imshow(ys, [min(min(ys)) max(max(ys))]), colormap('jet'), colorbar;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Suivant %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

% Chargement des fichier infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Création du tableau des pentes x de chaque pixel
```

```
wx = ([]);

% Balayage des x
for i = 1:dim_im(1)
    % Balayage des y
    for j = 1:dim_im(2)
        % Calcul de la pente avec vérification que xs ne soit pas un NaN
        if isnan(xs(i,j)) == 0
            wx(i,j) = (1/2)*(((xm(i,j)-xs(i,j))/zf)+((xm(i,j)-xc)/zc));
        elseif isnan(xs(i,j)) == 1
            wx(i,j) = 0;
        end
    end
end

% Détermination des valeurs max et min du tableau
wx_max = max(max(wx));
wx_min = min(min(wx));

% Création du tableau des pentes y de chaque pixel
wy = ([]);

% Balayage des x
for i = 1:dim_im(1)
    % Balayage des y
    for j = 1:dim_im(2)
        % Calcul de la pente avec vérification que ys ne soit pas un NaN
        if isnan(ys(i,j)) == 0
            wy(i,j) = (1/2)*(((ym(i)-ys(i,j))/zf)+((ym(i)-yc)/zc));
        elseif isnan(ys(i,j)) == 1
            wy(i,j) = 0;
        end
    end
end

% Détermination des valeurs max et min du tableau
wy_max = max(max(wy));
wy_min = min(min(wy));

% Sauvegarde des données utiles
save('datas.mat', 'wx', 'wx_max', 'wx_min', 'wy', 'wy_max',...
    'wy_min', '-append');

% Travail dans le dossier des scripts
cd(pth);

% Définition de la position de la fenêtre et sauvegarde
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de la fenêtre actuelle
delete(gcf);

% Lancement de la fenêtre 'Pentes'
run Pentes;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function pushbutton1_Callback(hObject, eventdata, handles)

% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir retourner aux paramètres ?',
...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth', 'pth_r');
        % Nettoyage de ce qu'il s'est passé entre les deux fenêtres
        cd(pth_r);
        load('datas.mat');
        save('datas.mat','X','Y', 'dim_im');
        cd(pth);
        % Fermeture de la fenêtre
        delete(gcf);
        % Ouverture de la fenêtre 'Parametres'
        run Parametres;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer2

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        % Suppression du fichier datas.mat
        cd(pth_r);
        delete('datas.mat');
        cd(pth);
        % Fermeture de tout
        clear; clc; delete(gcf);
```

```
    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Fonctions non-utilisées %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Slider 1
function slider1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

6. Centre_de_masse_boutons.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Calcul de la position du fil par centre de masse avec valeur seuil %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Chargement des fichier infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Calcul du centre de masse pour la matrice X et puis Y
for h = 1:2

    % Dans la première, par rapport à X
    if h == 1
        % Chargement des données X dans la matrice datas
        datas = X;
    % Dans la deuxième, par rapport à Y
    elseif h == 2
        % Chargement des données Y dans la matrice datas
        datas = Y;
    end

    % Création des matrices centre de masse, numérateur et dénominateur de
la taille d'une image
    s_centro = zeros(dim_im(1),dim_im(2));
    num = zeros(dim_im(1),dim_im(2));
    den = zeros(dim_im(1),dim_im(2));

    % Nécessaire pour avoir la 3ème dimension
    dim_im = size(datas);

    % Calcul du numérateur et du dénominateur de chaque pixel
    for k = 1:dim_im(3)
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                if datas(i,j,k) >= value
                    num(i,j) = double(num(i,j)) +
double(k)*double(datas(i,j,k));
                    den(i,j) = double(den(i,j)) + double(datas(i,j,k));
                end
            end
        end
    end

    % Calcul du centre de masse de chaque pixel
    for i = 1:dim_im(1)
        for j = 1:dim_im(2)
            s_centro(i,j) = double(num(i,j))/double(den(i,j));
        end
    end

     % Calcul de la position x exacte du fil par centre de masse
    if h == 1
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                xs(i,j) = filx + (s_centro(i,j)*st);
```

```
            end
        end

    % Calcul de la position y exacte du fil par centre de masse
    elseif h == 2
        for i = 1:dim_im(1)
            for j = 1:dim_im(2)
                ys(i,j) = fily + (s_centro(i,j)*st);
            end
        end
    end
end

% Sauvegarde des données modifiées
save('datas.mat', 'xs', 'ys', '-append');

% Travail dans le dossier des scripts
cd(pth);
```

7. Pentes.m

```
function varargout = Pentes(varargin)
% PENTES MATLAB code for Pentes.fig
%      PENTES, by itself, creates a new PENTES or raises the existing
%      singleton*.
%
%      H = PENTES returns the handle to a new PENTES or the handle to
%      the existing singleton*.
%
%      PENTES('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PENTES.M with the given input arguments.
%
%      PENTES('Property','Value',...) creates a new PENTES or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Pentes_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Pentes_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 24-Jun-2019 15:43:20

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

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


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

```
% Choose default command line output for Pentes
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Affichage des cartes de pentes %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Chargement des fichiers infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Affichage des pentes en x et en y
axes(handles.axes2);
imshow(wx, [wx_min wx_max]), colormap('jet'), colorbar;
axes(handles.axes3);
imshow(wy, [wy_min wy_max]), colormap('jet'), colorbar;

% Travail dans le dossier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

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


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

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Enregistrer les pentes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton3_Callback(hObject, eventdata, handles)%
```

```
% Chargement des fichiers infos.mat et datas.mat et travail dans le dossier
images
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Transposition du tableau
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        wx_save(j,dim_im(1)-i+1) = wx(i,j);
    end
end

% Sauvegarde du tableau dans un fichier txt
fileID = fopen('wx.txt','w');
fprintf(fileID,'Nx:\t\t\t %d \n',size(X,2));
fprintf(fileID,'Ny:\t\t\t %d \n',size(X,1));
fprintf(fileID,'dx:\t\t\t %d \n',1);
fprintf(fileID,'dy:\t\t\t %d \n',1);
fprintf(fileID,[repmat(' %f ', 1, size(X,1)), '\n'],wx_save);
fclose(fileID);

% Transposition du tableau
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        wy_save(j,dim_im(1)-i+1) = wy(i,j);
    end
end

% Sauvegarde du tableau dans un fichier txt
fileID = fopen('wy.txt','w');
fprintf(fileID,'Nx:\t\t\t %d \n',size(Y,2));
fprintf(fileID,'Ny:\t\t\t %d \n',size(Y,1));
fprintf(fileID,'dx:\t\t\t %d \n',1);
fprintf(fileID,'dy:\t\t\t %d \n',1);
fprintf(fileID,[repmat(' %f ', 1, size(Y,1)), '\n'],wy_save);
fclose(fileID);

% Message qui informe de l'enregistrement des tableaux
msg = msgbox('Pentes enregistrées !');
pause(1.5);
close(msg);

% Travail dans le dossier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Distorsion %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function checkbox2_Callback(hObject, eventdata, handles)

% Valeur du bouton 'Distorsion'
distor = get(hObject,'Value');

% Chargement du fichier infos.mat
load('infos.mat');

% Travail dans le dossier images
```

```
cd(pth_r);

% Sauvegarde de la valeur du bouton 'Distorsion'
save('datas.mat', 'distor', '-append');

% Travail dans le dossier des scripts
cd(pth);

% Calcul des pentes avec ou sans distorsion
run Pentes_bouton_dist;

% Travail dans le dossier images
cd(pth_r);

% Chargement du fichier datas.mat
load('datas.mat');

% Affichage des nouvelles cartes des pentes
axes(handles.axes2);
imshow(wx, [wx_min wx_max]), colormap('jet'), colorbar;
axes(handles.axes3);
imshow(wy, [wy_min wy_max]), colormap('jet'), colorbar;

% Travail dans le dossier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Suivant %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

% Chargement des fichiers infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Sauvegarde des valeurs utiles de datas.mat
save('datas.mat', 'wx', 'wy', 'xm', 'ym', 'dim_im');

% Travail dans le dossier des scripts
cd(pth);

% Sauvegarde de la position de la fenêtre
pos = get(gcf, 'Position');
save('infos.mat', 'pos', 'pth', 'dern_pth_d');

% Fermeture de la fenêtre
delete(gcf);

% Ouverture de la fenêtre 'Octopus'
run Octopus;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)
```

```
% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir retourner au centre de masse
?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth', 'pth_r');

        % Nettoyage de ce qu'il s'est passé entre les deux fenêtres
        cd(pth_r);
        distor = 0; % Ces quatre lignes ralentissent le retour en arrière
        save('datas.mat', 'distor', '-append'); %
        cd(pth); %
        run Pentes_bouton_dist; %
        cd(pth_r)
        load('datas.mat');
        save('datas.mat','X','Y', 'dim_im', 'dx', 'dy', 'filx', 'fily',...
            'larg', 'long', 'pixel_int', 'st', 'x1', 'xc', 'xm', 'xs',...
            'y1', 'yc', 'ym', 'ys', 'zc', 'zf');
        if exist('value')
            save('datas.mat', 'value', '-append');
        end
        cd(pth);

        % Fermeture de la fenêtre
        delete(gcf);
        % Ouverture de la fenêtre 'Ouvrir'
        run Centre_de_masse;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer2

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
```

```
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        % Suppression du fichier datas.mat
        cd(pth_r);
        delete('datas.mat');
        cd(pth);
        % Fermeture de tout
        clear; clc; delete(gcf);

    case 'Non'
         % Aucune action
end
```

8. Ouvrir2.m

```
function varargout = Ouvrir2(varargin)
% OUVRIR2 MATLAB code for Ouvrir2.fig
%      OUVRIR2, by itself, creates a new OUVRIR2 or raises the existing
%      singleton*.
%
%      H = OUVRIR2 returns the handle to a new OUVRIR2 or the handle to
%      the existing singleton*.
%
%      OUVRIR2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in OUVRIR2.M with the given input arguments.
%
%      OUVRIR2('Property','Value',...) creates a new OUVRIR2 or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Ouvrir2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Ouvrir2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Ouvrir2

% Last Modified by GUIDE v2.5 25-Jun-2019 10:23:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @Ouvrir2_OpeningFcn, ...
    'gui_OutputFcn',  @Ouvrir2_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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


% --- Executes just before Ouvrir2 is made visible.
function Ouvrir2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Ouvrir2 (see VARARGIN)
```

```
% Choose default command line output for Ouvrir2
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Ouvrir2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Ouvrir2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Sélectionner le fichier #1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

% Chargement du fichier infos.mat
load('infos.mat');

% Si un fichier a déjà été sélectionné
if exist('dern_d')
    % Sélection du fichier
    [fichier, chemin] = uigetfile(dern_d, 'Sélectionne le fichier');

    % Si c'est le premier fichier sélectionné
elseif ~exist('dern_d')
    % Sélection du fichier
    [fichier, chemin] = uigetfile(dern_pth_d, 'Sélectionne le fichier');
end

% Si clic sur annuler
if fichier == 0
    % Aucune action
```

```
    %Si clic sur un fichier et ouvrir
else
    % Définition du dossier comme le dernier visité
    dern_d = chemin;

    % Travail dans le dossier visité
    cd(dern_d);

    % Si le fichier datas.mat n'existe pas
    if ~exist("datas.mat")
        % Message d'erreur
        msg = errordlg('ATTENTION: le fichier doit être accompagné d''un
fichier datas.mat');

        % Travail dans le dossier des scripts
        cd(pth);

        % Si le fichier datas.mat existe
    elseif exist("datas.mat")
        % Définition du chemin du fichier
        pth_x = chemin;

        % Définition du nom du fichier
        wx_file = fichier;

        % Travail dans le dossier des scripts
        cd(pth);

        % Sauvegarde des données utiles
        save('infos.mat', 'pth_x', 'wx_file', 'dern_d', '-append');

        % Écriture du nom du fichier dans la barre éditable
        set(handles.edit1, 'String', fichier);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Sélectionner le fichier #2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

% Chargement du fichier infos.mat
load('infos.mat');

% Si un fichier a déjà été sélectionné
if exist('dern_d')
    % Sélection du fichier
    [fichier, chemin] = uigetfile(dern_d, 'Sélectionne le fichier');

    % Si c'est le premier fichier sélectionné
elseif ~exist('dern_d')
    % Sélection du fichier
    [fichier, chemin] = uigetfile(dern_pth_d, 'Sélectionne le fichier');
end

% Si clic sur annuler
if fichier == 0
    % Aucune action
```

```
    %Si clic sur un fichier et ouvrir
else
    % Définition du dossier comme le dernier visité
    dern_d = chemin;

    % Travail dans le dossier visité
    cd(dern_d);

    % Si le fichier datas.mat n'existe pas
    if ~exist("datas.mat")
        % Message d'erreur
        msg = errordlg('ATTENTION: le fichier doit être accompagné d''un
fichier datas.mat');

        % Travail dans le dossier des scripts
        cd(pth);

        % Si le fichier datas.mat existe
    elseif exist("datas.mat")

        % Définition du chemin du fichier
        pth_y = chemin;

        % Définition du nom du fichier
        wy_file = fichier;

        % Travail dans le dossier des scripts
        cd(pth);

        % Sauvegarde des données utiles
        save('infos.mat', 'pth_y', 'wy_file', 'dern_d', '-append');

        % Écriture du nom du fichier dans la barre éditable
        set(handles.edit2, 'String', fichier);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Sélectionner le fichier #3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton3_Callback(hObject, eventdata, handles)

% Chargement du fichier infos.mat
load('infos.mat');

% Si un fichier a déjà été sélectionné
if exist('dern_d')
    % Sélection du fichier
    [fichier, chemin] = uigetfile(dern_d, 'Sélectionne le fichier');

    % Si c'est le premier fichier sélectionné
elseif ~exist('dern_d')
    % Sélection du fichier
    [fichier, chemin] = uigetfile(dern_pth_d, 'Sélectionne le fichier');
end
```

```
% Si clic sur annuler
if fichier == 0
    % Aucune action

    %Si clic sur un fichier et ouvrir
else
    % Définition du dossier comme le dernier visité
    dern_d = chemin;

    % Définition du chemin du fichier
    pth_r = chemin;

    % Sauvegarde des données utiles
    save('infos.mat', 'dern_d', 'pth_r', '-append');

    % Écriture du nom du fichier et son chemin dans la barre éditable
    set(handles.edit3, 'String', [chemin, fichier]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton4_Callback(hObject, eventdata, handles)

% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir perdre la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        delete(gcf);
        % Ouverture de la fenêtre 'Octopus'
        run Octopus;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Suivant %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton5_Callback(hObject, eventdata, handles)

% Chargement du fichier infos.mat
load('infos.mat');

% Vérification de si les 3 fichiers existent et que leur dossier correspond
if exist('pth_x') && exist('pth_y') && exist('pth_r')
    if pth_x == pth_y
```

```
        if pth_x == pth_r

            % Travail dans le dossier des fichiers
            cd(pth_r);

            % Double boucle, une pour wx et une pour wy
            for h = 1:2
                % Pour wx
                if h == 1
                    % Ouverture du fichier
                    fileID = fopen([pth_r, wx_file]);

                    % Début de la barre d'état
                    time = waitbar(0,'Importation des pentes x');

                    % Pour wy
                elseif h == 2
                    % Ouverture du fichier
                    fileID = fopen([pth_r, wy_file]);

                    % Évolution de la barre d'état
                    waitbar(0.5,time,'Importation des pentes y');
                end

                % Extraction du contenu du fichier
                C = textscan(fileID,'%s');

                % Définition des dimensions du tableau
                Nx = str2num(char(C{1}(3)));
                Ny = str2num(char(C{1}(6)));

                % Définition d'une matrice pente
                w = zeros(Nx, Ny);

                % Inscription des infos du fichier dans la matrice des
pentes
                for i = 13:size(C{1},1)
                    w(i-12) = str2num(char(C{1}(i)));
                end

                % Le pixel 1 étant toujours masqué, il contient la valeur
de défaut
                ref = w(1);

                % NaN pour tous les pixels qui ont la valeur référence
                for i = 1:(size(w,1)*size(w,2))
                    if w(i) == ref
                        w(i) = NaN;
                    end
                end

                % Définition d'une matrice qui sera un flip des y (dû à
NDA)
                w_flip = zeros(Ny, Nx);

                % Flip de la matrice
                for i = 1:Nx
                    for j = 1:Ny
                        w_flip(Ny-j+1,i) = w(i,j);
```

```
                end
            end

            % En fonction de la boucle, définition de wx ou wy
            if h == 1
                wx_mask = w_flip;
            elseif h == 2
                wy_mask = w_flip;
            end

        end

        % Évolution de la barre d'état
        waitbar(1,time,'Données importées');
        pause(1);
        close(time);

        % Sauvegarde des données utiles
        save('datas.mat', 'wx_mask', 'wy_mask', '-append');

        % Travail dans le dossier des scripts
        cd(pth);

        % Définition du dernier fichier dans lequel on a travaillé
        cd(pth_r);
        cd ../;
        dern_pth_d = cd;

        % Travail dans le dossier des scripts
        cd(pth);

        % Définition de la position de la fenêtre et sauvegarde
        pos = get(gcf, 'Position');
        save('infos.mat', 'dern_pth_d', 'pth', 'pth_r', 'pos');

        % Fermeture de la fenêtre actuelle
        delete(gcf);

        % Lancement de l'affichage des pentes
        run Pentes2;

    else
        % Si deux des dossiers ne correspondent pas
        m = msgbox('Veuillez vérifier que les fichiers correspondent
bien au même projet. Si c''est le cas, veuillez les mettre dans un même
dossier avant de continuer.');
    end

else
    % Si deux des dossiers ne correspondent pas
    m = msgbox('Veuillez vérifier que les fichiers correspondent bien
au même projet. Si c''est le cas, veuillez les mettre dans un même dossier
avant de continuer.');
end

else
    % Si au moins une des cases est vide
    m = msgbox('Un ou plusieurs fichiers sont manquants ! Veuillez les
sélectionner avant de continuer !');
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)
% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');
% Effet suivant la réponse
switch answer2
    case 'Oui'
        % Ouverture de infos.mat et sauvegarde de l'utile
        load('infos.mat');
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
        % Fermeture de tout
        clear; clc; delete(gcf);
    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Fonctions non-utilisées %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Barre d'édition 1
function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 2
function edit2_Callback(hObject, eventdata, handles)

function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Barre d'édition 3
function edit3_Callback(hObject, eventdata, handles)

function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

9. Pentes2.m

```
function varargout = Pentes2(varargin)
% PENTES2 MATLAB code for Pentes2.fig
%      PENTES2, by itself, creates a new PENTES2 or raises the existing
%      singleton*.
%
%      H = PENTES2 returns the handle to a new PENTES2 or the handle to
%      the existing singleton*.
%
%      PENTES2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PENTES2.M with the given input arguments.
%
%      PENTES2('Property','Value',...) creates a new PENTES2 or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Pentes2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Pentes2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Pentes2

% Last Modified by GUIDE v2.5 25-Jun-2019 14:46:21

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Pentes2_OpeningFcn, ...
                   'gui_OutputFcn',  @Pentes2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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


% --- Executes just before Pentes2 is made visible.
function Pentes2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Pentes2 (see VARARGIN)
```

```
% Choose default command line output for Pentes2
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Affichage des cartes de pentes %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier des fichiers
cd(pth_r);

% Chargement du fichier datas.mat
load('datas.mat');

% Affichage des pentes
axes(handles.axes2);
imshow(wx_mask, [min(min(wx_mask)) max(max(wx_mask))]), colormap('jet'),
colorbar;
axes(handles.axes3);
imshow(wy_mask, [min(min(wy_mask)) max(max(wy_mask))]), colormap('jet'),
colorbar;

% Travail dans le dossier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Pentes2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Pentes2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function pushbutton1_Callback(hObject, eventdata, handles)

% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir retrouner à la sélection des
fichiers ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Chargement des fichiers infos.mat et datas.mat
        load('infos.mat');
        cd(pth_r);
        load('datas.mat');

        % Sauvegarde de ce qui doit être sauvé
        save('datas.mat', 'wx', 'wy', 'xm', 'ym', 'dim_im');

        % Travail dans le dossier des scripts
        cd(pth);

        % Sauvegarde de la position de la fenêtre
        pos = get(gcf, 'Position');
        save('infos.mat', 'pos', 'pth', 'dern_pth_d');

        % Fermeture de la fenêtre actuelle
        delete(gcf);

        % Ouverture de la fenêtre 'Ouvrir2'
        run Ouvrir2;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Suivant %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

% Chargement des fichiers infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Début de la barre d'état
time = waitbar(0,'Première intégration');

% Tableau des pentes en y à inverser
wy_mask = wy_mask*((-1)^(get(handles.popupmenu1,'Value')+1));

% Création d'un masque équivalent à celui appliqué sur l'image des pentes
mirror_ok = zeros(dim_im(1),dim_im(2));
for i = 1:dim_im(1)
```

```
    for j = 1:dim_im(2)
        if isnan(wx_mask(i,j)) == 0
            if isnan(wy_mask(i,j)) == 0
                mirror_ok(i,j) = 1;
            end
        end
    end
end

% Définition des dimensions du masque (xmin, ymin, width, height)
mask = zeros(1,4);

% xmin
i = 1;
while max(mirror_ok(:,i)) ~= 1
    i = i + 1;
end
mask(1) = i;

% ymin
i = 1;
while max(mirror_ok(i,:)) ~= 1
    i = i + 1;
end
mask(2) = i;

% width
i = dim_im(2);
while max(mirror_ok(:,i)) ~= 1
    i = i - 1;
end
mask(3) = i - mask(1) + 1;

% height
i = dim_im(1);
while max(mirror_ok(i,:)) ~= 1
    i = i - 1;
end
mask(4) = i - mask(2) + 1;

% NaN de la carte des pentes égal 0 pour éviter des erreurs de calcul
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        if isnan(wx_mask(i,j)) == 1
            wx_mask(i,j) = 0;
        end
        if isnan(wy_mask(i,j)) == 1
            wy_mask(i,j) = 0;
        end
    end
end

% Définition d'un point de référence
z_centre = 0;

% Tableau des hauteurs (toutes égales à 0 au début)
Z = zeros(dim_im(1), dim_im(2));

% Définition du centre du masque
y0 = round(dim_im(1)/2);
```

```
x0 = round(dim_im(2)/2);

% Définition de la hauteur de référence au centre de l'image
Z(y0,x0) = z_centre;

% Intégration trapézoïdale
for h = 0:1
    y0 = y0+h;

    % Calcul du point à côté du centre du masque pour la seconde
intégration trapézoïdale
    if h == 1
        Z(y0,x0) = ((mirror_ok(y0,x0-1)*Z(y0,x0-1)...
            + mirror_ok(y0,x0+1)*Z(y0,x0+1) + mirror_ok(y0-1,x0)*Z(y0-1,x0)
...
            + mirror_ok(y0+1,x0)*Z(y0+1,x0))/(mirror_ok(y0,x0-1) +
mirror_ok(y0,x0+1)...
            + mirror_ok(y0-1,x0) + mirror_ok(y0+1,x0)))...
            + ((mirror_ok(y0,x0-1)*((abs(xm(y0,x0-1)-
xm(y0,x0))/2)*(wx_mask(y0,x0-1) + wx_mask(y0,x0)))...
            - mirror_ok(y0,x0+1)*((abs(xm(y0,x0+1)-
xm(y0,x0))/2)*(wx_mask(y0,x0+1) + wx_mask(y0,x0)))...
            + mirror_ok(y0-1,x0)*((abs(ym(y0-1)-ym(y0))/2)*(wy_mask(y0-
1,x0) + wy_mask(y0,x0)))...
            - mirror_ok(y0+1,x0)*((abs(ym(y0+1)-
ym(y0))/2)*(wy_mask(y0+1,x0) + wy_mask(y0,x0))))...
            / (mirror_ok(y0,x0-1) + mirror_ok(y0,x0+1)...
            + mirror_ok(y0-1,x0) + mirror_ok(y0+1,x0)));
    end

    % Intégration vers les X+/Y+
    for i = 1:max(dim_im)
        % Si hors masque, stop
        if mirror_ok(y0+i,x0+i) == 0
            break;
            % Si dans le masque, intégration
        else
            Z(y0+i,x0+i) = Z(y0+i-1,x0+i-1)...
                + (abs(xm(y0+i-1, x0+i-1)-xm(y0+i,x0+i-
1))/2)*(wx_mask(y0+i-1,x0+i-1)+wx_mask(y0+i,x0+i-1))...
                + (abs(ym(y0+i-1)-ym(y0+i))/2)*(wy_mask(y0+i-
1,x0+i)+wy_mask(y0+i,x0+i));
            max_diag = i;
        end
    end

    % Intégration vers les X-/Y-
    for i = 1:max(dim_im)
        if  mirror_ok(y0-i,x0-i) == 0
            break;
            % Si dans le masque, intégration
        else
            Z(y0-i,x0-i) = Z(y0-i+1,x0-i+1)...
                - (abs(xm(y0-i+1,x0-i+1)-xm(y0-i,x0-i+1))/2)*(wx_mask(y0-
i+1,x0-i+1)+wx_mask(y0-i,x0-i+1))...
                - (abs(ym(y0-i+1)-ym(y0-i))/2)*(wy_mask(y0-i+1,x0-
i)+wy_mask(y0-i,x0-i));
            min_diag = i;
        end
    end
```

```
    % Si le masque est orienté horizontalement
    if mask(3)>mask(4)
        % Définition du nombre de vagues d'intégrations à gauche et à
droite
        nb_int = ceil(mask(3)/mask(4));
        % Définition du sens de départ pour intégrer côté droit
        sens = 1;
        % Définition des x et y de départ
        x = x0-min_diag;
        y = y0-min_diag;
        % Les vagues d'intégrations à droite
        for i = 1:nb_int
            % Pour intégrer dans le sens X+/Y-
            if sens == 1
                % Pour s'assurer que le point de départ est dans le masque
                while mirror_ok(y,x) == 1
                    % Intégration à partir de (y,x)
                    for j = 1:max(dim_im)
                        % Si le point suivant à intégrer est hors du masque
                        if  mirror_ok(y-j,x+j) == 0
                            break;
                            % S'il est dans le masque, intégration
                        else
                            Z(y-j,x+j) = Z(y-j+1,x+j-1)...
                                + (abs(xm(y-j+1,x+j-1)-xm(y-
j+1,x+j))/2)*(wx_mask(y-j+1,x+j-1)+wx_mask(y-j+1,x+j))...
                                - (abs(ym(y-j+1)-ym(y-j))/2)*(wy_mask(y-
j+1,x+j)+wy_mask(y-j,x+j));
                        end
                    end
                    % Définition du point suivant à partir duquel intégrer
                    x = x+1;
                    y = y+1;
                end
                % Passage au sens suivant
                sens = 2;
                % Définition du point de départ du sens suivant
                x = x-1;
                y = y-1;
                % Pour intégrer dans le sens X+/Y+, avec les mêmes
commentaires
            elseif sens == 2
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y+j,x+j) == 0
                            break;
                        else
                            Z(y+j,x+j) = Z(y+j-1,x+j-1)...
                                + (abs(xm(y+j-1,x+j-1)-xm(y+j-
1,x+j))/2)*(wx_mask(y+j-1,x+j-1)+wx_mask(y+j-1,x+j))...
                                + (abs(ym(y+j-1)-ym(y+j))/2)*(wy_mask(y+j-
1,x+j)+wy_mask(y+j,x+j));
                        end
                    end
                    x = x+1;
                    y = y-1;
                end
                sens = 1;
                x = x-1;
```

```
                y = y+1;
            end
        end
        % Passage à l'intégration du côté gauche
        sens = 1;
        % Définition du point de départ
        x = x0+max_diag;
        y = y0+max_diag;
        for i = 1:nb_int
            % Intégration vers les X-/Y+ avec les mêmes commentaires
            if sens == 1
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y+j,x-j) == 0
                            break;
                        else
                            Z(y+j,x-j) = Z(y+j-1,x-j+1)...
                                - (abs(xm(y+j-1,x-j+1)-xm(y+j-1,x-
j))/2)*(wx_mask(y+j-1,x-j+1)+wx_mask(y+j-1,x-j))...
                                + (abs(ym(y+j-1)-ym(y+j))/2)*(wy_mask(y+j-
1,x-j)+wy_mask(y+j,x-j));
                        end
                    end
                    x = x-1;
                    y = y-1;
                end
                sens = 2;
                x = x+1;
                y = y+1;
                % Intégration vers les X-/Y-, avec les mêmes commentaires
            elseif sens == 2
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y-j,x-j) == 0
                            break;
                        else
                            Z(y-j,x-j) = Z(y-j+1,x-j+1)...
                                - (abs(xm(y-j+1,x-j+1)-xm(y-j+1,x-
j))/2)*(wx_mask(y-j+1,x-j+1)+wx_mask(y-j+1,x-j))...
                                - (abs(ym(y-j+1)-ym(y-j))/2)*(wy_mask(y-
j+1,x-j)+wy_mask(y-j,x-j));
                        end
                    end
                    x = x-1;
                    y = y+1;
                end
                sens = 1;
                x = x+1;
                y = y-1;
            end
        end

        %Dans le cas où le masque est vertical
    elseif mask(4)>= mask(3)
        % Définition du nombre de vagues
        nb_int = ceil(mask(4)/mask(3));
        % Définition du sens de départ pour intégrer vers le haut
        sens = 1;
        % Définition du point de départ
        x = x0-min_diag;
        y = y0-min_diag;
```

```
        % Les vagues d'intégration vers le haut
        for i = 1:nb_int
            % Intégration vers les X-/Y+
            if sens == 1
                % Vérification que le point de départ est dans le masque
                while mirror_ok(y,x) == 1
                    % Intégration à partir de ce point
                    for j = 1:max(dim_im)
                        % Si le point suivant est hors du masque
                        if  mirror_ok(y+j,x-j) == 0
                            break;
                            % S'il est dans le masque, intégration
                        else
                            Z(y+j,x-j) = Z(y+j-1,x-j+1)...
                                - (abs(xm(y+j-1,x-j+1)-xm(y+j-1,x-
j))/2)*(wx_mask(y+j-1,x-j+1)+wx_mask(y+j-1,x-j))...
                                + (abs(ym(y+j-1)-ym(y+j))/2)*(wy_mask(y+j-
1,x-j)+wy_mask(y+j,x-j));
                        end
                    end
                    % Définition du point suivant
                    x = x+1;
                    y = y+1;
                end
                % Passage au sens suivant
                sens = 2;
                % Définition du point de départ
                x = x-1;
                y = y-1;
                % Intégration dans le sens X+/Y+, mêmes commentaires
            elseif sens == 2
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y+j,x+j) == 0
                            break;
                        else
                            Z(y+j,x+j) = Z(y+j-1,x+j-1)...
                                + (abs(xm(y+j-1,x+j-1)-xm(y+j-
1,x+j))/2)*(wx_mask(y+j-1,x+j-1)+wx_mask(y+j-1,x+j))...
                                + (abs(ym(y+j-1)-ym(y+j))/2)*(wy_mask(y+j-
1,x+j)+wy_mask(y+j,x+j));
                        end
                    end
                    x = x-1;
                    y = y+1;
                end
                sens = 1;
                x = x+1;
                y = y-1;
            end
        end
        % Passage à l'intégration vers le bas
        sens = 1;
        x = x0+max_diag;
        y = y0+max_diag;
        for i = 1:nb_int
            % Intégration dans le sens X+/Y-, mêmes commentaires
            if sens == 1
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y-j,x+j) == 0
```

```
                            break;
                        else
                            Z(y-j,x+j) = Z(y-j+1,x+j-1)...
                                + (abs(xm(y-j+1,x+j-1)-xm(y-
j+1,x+j))/2)*(wx_mask(y-j+1,x+j-1)+wx_mask(y-j+1,x+j))...
                                - (abs(ym(y-j+1)-ym(y-j))/2)*(wy_mask(y-
j+1,x+j)+wy_mask(y-j,x+j));
                        end
                    end
                    x = x-1;
                    y = y-1;
                end
                sens = 2;
                x = x+1;
                y = y+1;
                % Intégration dans le sens X-/Y-, mêmes commentaires
            elseif sens == 2
                while mirror_ok(y,x) == 1
                    for j = 1:max(dim_im)
                        if  mirror_ok(y-j,x-j) == 0
                            break;
                        else
                            Z(y-j,x-j) = Z(y-j+1,x-j+1)...
                                - (abs(xm(y-j+1,x-j+1)-xm(y-j+1,x-
j))/2)*(wx_mask(y-j+1,x-j+1)+wx_mask(y-j+1,x-j))...
                                - (abs(ym(y-j+1)-ym(y-j))/2)*(wy_mask(y-
j+1,x-j)+wy_mask(y-j,x-j));
                        end
                    end
                    x = x+1;
                    y = y-1;
                end
                sens = 1;
                x = x-1;
                y = y+1;
            end
        end
    end
end

% NaN partout où l'intégration trapézoïdale n'est pas passée
for i = 1:(dim_im(1)*dim_im(2))
    if mirror_ok(i) == 0
        Z(i) = NaN;
    end
end

% Création d'un tableau vide pour l'intégration de Southwell
Z_southwell = ([]);

% Définition des dimensions à ce tableau en le rendant égal à Z
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        Z_southwell(i,j) = Z(i,j);
    end
end

% Intégration de Southwell

% Choix du diviseur d'itération
```

```
if get(handles.popupmenu2,'Value') == 1
    div = 1;
elseif get(handles.popupmenu2,'Value') == 2
    div = 10;
elseif get(handles.popupmenu2,'Value') == 3
    div = 25;
end

% Définition du nombre d'itérations
for h = 1:(round((mask(3)*mask(4)))/div)

    % Évolution de la barre d'état
    waitbar(0.1 + (h*0.89/(round((mask(3)*mask(4)))/1)), time,'Intégration
en cours...');

    % Pour les Y dans le masque
    for i = round(mask(2)):round(mask(2)+mask(4)-1)
        % Pour les X dans le masque
        for j = round(mask(1)):round(mask(1)+mask(3)-1)
            % NaN si pas dans le masque
            if mirror_ok(i,j) == 0
                Z_southwell(i,j) = NaN;

            else
                % Si dans le masque, mais qu'un pixel autour vaut NaN,
pixel autour égal à 0
                if mirror_ok(i,j-1) == 0
                    Z(i,j-1) = 0;
                end
                if mirror_ok(i,j+1) == 0
                    Z(i,j+1) = 0;
                end
                if mirror_ok(i-1,j) == 0
                    Z(i-1,j) = 0;
                end
                if mirror_ok(i+1,j) == 0
                    Z(i+1,j) = 0;
                end
                % Réalisation de l'intégration de Southwell
                Z_southwell(i,j) = ((mirror_ok(i,j-1)*Z(i,j-1)...
                    + mirror_ok(i,j+1)*Z(i,j+1) + mirror_ok(i-1,j)*Z(i-1,j)
...
                    + mirror_ok(i+1,j)*Z(i+1,j))/(mirror_ok(i,j-1) +
mirror_ok(i,j+1)...
                    + mirror_ok(i-1,j) + mirror_ok(i+1,j)))...
                    + ((mirror_ok(i,j-1)*((abs(xm(i,j-1)-
xm(i,j))/2)*(wx_mask(i,j-1) + wx_mask(i,j)))...
                    - mirror_ok(i,j+1)*((abs(xm(i,j+1)-
xm(i,j))/2)*(wx_mask(i,j+1) + wx_mask(i,j)))...
                    + mirror_ok(i-1,j)*((abs(ym(i-1)-ym(i))/2)*(wy_mask(i-
1,j) + wy_mask(i,j)))...
                    - mirror_ok(i+1,j)*((abs(ym(i+1)-
ym(i))/2)*(wy_mask(i+1,j) + wy_mask(i,j))))...
                    / (mirror_ok(i,j-1) + mirror_ok(i,j+1)...
                    + mirror_ok(i-1,j) + mirror_ok(i+1,j)));
            end
        end
    end
    % Définition de Z avec les valeurs de Z_southwell pour pouvoir réitérer
    for i = 1:dim_im(1)
        for j = 1:dim_im(2)
```

```
            Z(i,j) = Z_southwell(i,j);
        end
    end
end

% Évolution de la barre d'état
waitbar(1,time,'Intégration finie');
pause(1);
close(time);

% Sauvegarde des valeurs utiles
save('datas.mat', 'Z', '-append');

% Travail dans le dossier des scripts
cd(pth);

% Sauvegarde de la position de la fenêtre
pos = get(gcf, 'Position');
save('infos.mat', 'pos', '-append');

% Fermeture de la fenêtre actuelle
delete(gcf);

% Ouverture de la fenêtre 'Surface finale'
run Surface_finale;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer2

    case 'Oui'
        % Chargement des fichiers infos.mat et datas.mat
        load('infos.mat');
        cd(pth_r);
        load('datas.mat');

        % Sauvegarde de ce qui doit être sauvé
        save('datas.mat', 'wx', 'wy', 'xm', 'ym', 'dim_im');

        % Travail dans le dossier des scripts
        cd(pth);

        % Sauvegarde de la position actuelle de la fenêtre
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');
```

```
        % Fermeture de tout
        clear; clc; delete(gcf);

    case 'Non'
         % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Fonctions non-utilisées %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Menu 1
function popupmenu1_Callback(hObject, eventdata, handles)

function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Menu 2
function popupmenu2_Callback(hObject, eventdata, handles)

function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

10. Surface_Finale.m

```
function varargout = Surface_finale(varargin)
% SURFACE_FINALE MATLAB code for Surface_finale.fig
%      SURFACE_FINALE, by itself, creates a new SURFACE_FINALE or raises
the existing
%      singleton*.
%
%      H = SURFACE_FINALE returns the handle to a new SURFACE_FINALE or the
handle to
%      the existing singleton*.
%
%      SURFACE_FINALE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SURFACE_FINALE.M with the given input
arguments.
%
%      SURFACE_FINALE('Property','Value',...) creates a new SURFACE_FINALE
or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Surface_finale_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Surface_finale_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 25-Jun-2019 13:35:25

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

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


% --- Executes just before Surface_finale is made visible.
function Surface_finale_OpeningFcn(hObject, eventdata, handles, varargin)
```

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

% Choose default command line output for Surface_finale
handles.output = hObject;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Paramètres de la fenêtre %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Définition de la position de la fenêtre précédente
load('infos.mat');
set(gcf, 'Position', pos);

% Affichage du logo AMOS sur la fenêtre
axes(handles.axes1);
imshow('amos_logo.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Affichage de la surface %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Travail dans le dossier des fichiers
cd(pth_r);

% Chargement du fichier 'datas.mat'
load('datas.mat');

% Affichage de la surface reconstruite
axes(handles.axes2);
imshow(Z, [min(min(Z)) max(max(Z))]), colormap('jet'), colorbar;

% Travail dans le dossier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Update handles structure
guidata(hObject, handles);

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


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

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%% Enregistrement de la surface %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton3_Callback(hObject, eventdata, handles)

% Chargement des fichiers infos.mat et datas.mat
load('infos.mat');
cd(pth_r);
load('datas.mat');

% Remplacement des NaN par 0 et flip du tableau selon y
for i = 1:dim_im(1)
    for j = 1:dim_im(2)
        if isnan(Z(i,j)) == 1
            Zbis(j,i) = 0;
        else
            Zbis(j,dim_im(1)-i+1) = Z(i,j);
        end
    end
end

% Sauvegarde le tableau en TXT
fileID = fopen('Z.txt','w');
fprintf(fileID,'Nx:\t\t\t %d \n',size(Z,2));
fprintf(fileID,'Ny:\t\t\t %d \n',size(Z,1));
fprintf(fileID,'dx:\t\t\t %d \n',1);
fprintf(fileID,'dy:\t\t\t %d \n',1);
fprintf(fileID,[repmat(' %f ', 1, size(Z,1)), '\n'],Zbis);
fclose(fileID);

% Annonce de la sauvegarde est effectuée
msg = msgbox('Surface enregistrée !');
pause(1.5);
close(msg);

% Travail dans le fichier des scripts
cd(pth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Précédent %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

% Demande de la perte de la progression
answer1 = questdlg('Êtes-vous sûr de vouloir retrouner à l''affichage des
pentes ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer1

    case 'Oui'
        % Chargement des fichiers infos.mat et datas.mat
        load('infos.mat');
        cd(pth_r);
        load('datas.mat');
```

```
        % Sauvegarde de ce qui doit être sauvé
        save('datas.mat', 'wx', 'wy', 'xm', 'ym', 'dim_im', 'wx_mask',...
            'wy_mask');

        % Travail dans le dossier des scripts
        cd(pth);

        % Sauvegarde de la position de la fenêtre
        pos = get(gcf, 'Position');
        save('infos.mat', 'pos', 'pth', 'dern_pth_d', 'pth_r');

        % Fermeture de la fenêtre actuelle
        delete(gcf);

        % Ouverture de la fenêtre 'Ouvrir2'
        run Pentes2;

    case 'Non'
        % Aucune action
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Fin de reconstruction %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

% Chargement des fichiers infos.mat et datas.mat
load('infos.mat');

% Sauvegarde de la position de la fenêtre
pos = get(gcf, 'Position');

% Sauvegarde des valeurs utiles de infos.mat
save('infos.mat', 'pth', 'pos', 'dern_pth_d');

% Fermeture de la fenêtre
delete(gcf);

% Ouverture de la fenêtre 'Octopus'
run Octopus;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Bouton de fermeture de la fenêtre %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function figure1_CloseRequestFcn(hObject, eventdata, handles)

% Demande de la perte de la progression
answer2 = questdlg('Êtes-vous sûr de vouloir fermer le programme et perdre
la progression ?', ...
    '',...
    'Non','Oui','Non');

% Effet suivant la réponse
switch answer2

    case 'Oui'
```

```
        % Chargement des fichiers infos.mat et datas.mat
        load('infos.mat');
        cd(pth_r);
        load('datas.mat');

        % Sauvegarde de ce qui doit être sauvé
        save('datas.mat', 'wx', 'wy', 'xm', 'ym', 'dim_im');

        % Travail dans le dossier des scripts
        cd(pth);

        % Sauvegarde de la position actuelle de la fenêtre
        pos = get(gcf, 'Position'); % pos après load pour ne pas sauver la
pos enregistrée avant
        save('infos.mat', 'dern_pth_d', 'pos', 'pth');

        % Fermeture de tout
        clear; clc; delete(gcf);

    case 'Non'
         % Aucune action
end
```