

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Etude et Implémentation sur FPGA d'algorithmes de segmentation d'images médicales en utilisant les Contours Actifs et les Courbes de Niveau

Bouchra BOUZID & Sabrina AIT BELKACEM

Sous la direction de Mme. Latifa HAMAMI Prof. ENP, Alger et de Mme. Fatima-Zohra
HAMIDATOU MAA. ENSA, Alger

Présenté et soutenu publiquement le 23/06/2024 auprès des membres du jury :

Président	M. Mourad	ADNANE	Prof.	ENP, Alger
Promotrice	Mme. Latifa	HAMAMI	Prof.	ENP, Alger
Promotrice	Mme. Fatima-Zohra	HAMIDATOU	MAA.	ENSA, Alger
Examinatrice	Mme. Fatiha	LANI	MAA.	ENP, Alger
Invitée	Mme. Radia	BENYAHIA	Prof.	CPMC, Alger

ENP 2024

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Etude et Implémentation sur FPGA d'algorithmes de segmentation d'images médicales en utilisant les Contours Actifs et les Courbes de Niveau

Bouchra BOUZID & Sabrina AIT BELKACEM

Sous la direction de Mme. Latifa HAMAMI Prof. ENP, Alger et de Mme. Fatima-Zohra
HAMIDATOU MAA. ENSA, Alger

Présenté et soutenu publiquement le 23/06/2024 auprès des membres du jury :

Président	M. Mourad	ADNANE	Prof.	ENP, Alger
Promotrice	Mme. Latifa	HAMAMI	Prof.	ENP, Alger
Promotrice	Mme. Fatima-Zohra	HAMIDATOU	MAA.	ENSA, Alger
Examinatrice	Mme. Fatiha	LANI	MAA.	ENP, Alger
Invitée	Mme. Radia	BENYAHIA	Prof.	CPMC, Alger

ENP 2024

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

ملخص

يعتمد هذا المشروع على استخدام الحدود النشطة والمنحنيات المستوية في تجزئة الصور الطبية، حيث يبرز أهميتهما في مجالات المتابعة و التشخيص الطبي. يتضح من خلال التحليل الدقيق لخوارزميات التجزئة فعاليتها وقابليتها للتكيف، مما يوفر حلولاً في مجال الرعاية الصحية. يُعرض النهج الخاص بتنفيذ الخوارزميات على FPGA كحل واعد لمعالجة الصور الطبية في الوقت الفعلي، مبرزاً مزايا هذه التقنية. يُسلط الضوء على أهمية تحسين الموارد وإدارة الوقت في إطار هذا المشروع من خلال عملية تصميم وتنفيذ الخوارزميات. وأخيراً، يؤكد تقييم أداء التنفيذ، الذي تم باستخدام بيانات صور طبية حقيقية، فعاليته وإمكانيته في مجال الطب.

كلمات مفتاحية : الحدود النشطة، المنحنيات المستوية، التجزئة، الصور الطبية، التنفيذ، FPGA، الوقت الفعلي.

Abstract

This project explores the use of Snakes and Level Sets in medical image segmentation, highlighting their crucial importance for medical diagnosis and monitoring. An in-depth analysis of segmentation algorithms highlights their recognised efficiency and adaptability in this field. The FPGA implementation approach is presented, highlighting the advantages of this technology for real-time image processing. The detailed process for designing and implementing the algorithms highlights the importance of optimising resources and management of computing time. Performance evaluation of the implementation, carried out with real medical image data, confirms its efficiency.

Keywords : Snakes, Level Sets, Segmentation, medical images, implémentation, FPGA, real time.

Résumé

Ce projet explore l'utilisation des contours actifs et des courbes de niveaux dans la segmentation d'images médicales, soulignant leur importance cruciale pour le diagnostic et le suivi médical. Une analyse approfondie des algorithmes de segmentation met en évidence leur efficacité reconnue et leur adaptabilité dans ce domaine. L'implémentation sur circuit FPGA est présentée, mettant en avant les avantages de cette technologie pour le traitement d'images en temps réel. Le processus détaillé de conception et de mise en œuvre des algorithmes souligne l'importance de l'optimisation des ressources et de la gestion du temps de calcul. L'évaluation des performances de l'implémentation, réalisée avec des données d'images médicales réelles, confirme son efficacité.

Mots clés : Contours actifs, Courbes de niveau, Segmentation, images médicales, implémentation, FPGA, temps réel.

Dédicace

À mes chers parents,

Aucune dédicace ne pourrait exprimer pleinement le respect, l'amour éternel et la gratitude que j'ai pour les sacrifices que vous avez consentis pour mon éducation et mon bien-être. Je vous remercie pour votre soutien et l'amour incommensurable que vous m'avez prodigué depuis ma naissance.

Maman, merci pour ta tendresse infinie et tes encouragements constants qui ont illuminé mon chemin tout au long de ma vie.

Papa, merci pour ta sagesse et ta force qui m'ont toujours inspiré à persévérer et à donner le meilleur de moi-même.

Que ce modeste travail soit l'accomplissement de vos vœux tout formulés, le fruit de vos innombrables sacrifices. Puisse Dieu le Très-Haut vous accorder santé, bonheur et longue vie.

À ma chère soeur "Nesrine",

À toi, ma deuxième maman, ma meilleure amie, ma confidente, ma sœur. Tu as toujours été mon étoile guide, éclairant les chemins les plus sombres de ma vie. Ta présence rassurante et tes conseils avisés sont comme un phare dans la tempête. Merci pour chaque sourire partagé, chaque larme séchée, chaque conseil sage et chaque étreinte réconfortante. Sans toi, ma vie n'aurait pas la même douceur ni la même profondeur. Que notre lien continue de grandir, remplissant nos vies de bonheur et de complicité à jamais.

À mes chers frères "Mohamed" et "Ali",

Vous êtes les héros de mon enfance, les chevaliers de mes rêves. Ensemble, nous avons exploré des mondes imaginaires et construit des châteaux dans les nuages. Votre présence solide et votre amitié fidèle sont un roc sur lequel je m'appuie dans les moments de doute et de triomphe. Merci pour vos encouragements sincères et votre optimisme contagieux. Que nos aventures ensemble continuent à enrichir nos vies de magie et de complicité.

À mes grands-parents,

Ceux qui m'ont quitté trop tôt, j'espère sincèrement que j'ai pu vous rendre fiers de moi. Les valeurs que vous avez incarnées et l'amour que vous m'avez transmis continuent d'illuminer ma route chaque jour.

À mes belles-soeurs "Nesrine" et "Leticia",

Pour votre accueil chaleureux et votre gentillesse. Vous êtes devenus une partie intégrante de ma vie et vous avez ajouté une dimension de bonheur et de soutien que je chéris profondément. Merci pour vos encouragements et pour être toujours là, que ce soit dans les moments de fête ou de défi.

À mon cher neveu "Hani" et à ma chère nièce "Joury",

Vous êtes une source de joie et de motivation constante. Vos sourires radieux et vos rires contagieux ont égayé même les moments les plus stressants de ma vie. Que votre innocence pétillante et votre joie de vivre continue à illuminer ma vie comme des feux d'artifice dans une nuit étoilée!

À mes soeurs de coeur "Chaima", "Lydia", "Ferial", "Wissal" et "Souhila",

Pour votre amitié précieuse et votre soutien indéfectible, vous êtes mes compagnes d'aventure les plus intrépides. Ensemble, nous avons surmonté des défis et tissé des souvenirs inoubliables. Merci d'être toujours là, pour les moments les plus fous et les plus magiques de ma vie.

À mes camarades,

Pour ces années partagées, les moments de complicité, d'entraide et de soutien mutuel. Ensemble, nous avons traversé des épreuves et célébré nos réussites.

À ma binôme "Sabrina",

Pour ces cinq dernières années incroyables passées ensemble. Nous avons vécu tous les moments, du plus difficile au plus joyeux. Ta patience, ta collaboration et ton soutien infailible ont été essentiels. Ton amitié a enrichi ma vie au-delà des études. Merci pour tout ce que nous avons accompli ensemble :)*

À tous ceux qui me sont chers,

Je vous aime et je vous dédie cet humble travail.

- Bouchra

Dédicace

Lorsque j'ai obtenu mon BAC, je t'avais demandé ce que tu souhaiterais que je devienne à l'avenir et tu m'avais répondu "ingénieure". J'espère sincèrement avoir pu te rendre fière. Je te dédie ce travail Mani, Allah yerahmek.

Je dédie également ce travail à mes parents, sans qui rien de tout cela n'aurait été possible. Mama, tu as toujours été présente pour moi, trouvant toujours les mots justes lorsque j'en avais besoin. Tu as su me soutenir dans les moments les plus difficiles, et pour cela, je ne te remercierai jamais assez.

Papa, tu as toujours tout fait pour que je ne manque de rien, pour que je me sente bien. Tu t'es démené pour me faciliter la vie. Je te remercie pour tes petites attentions qui me réconfortent toujours. Merci pour tout.

Je suis heureuse de vous voir aujourd'hui fiers de moi.

À toi, ma sœur, Amel, au fil des années, tu es devenue ma meilleure amie. Tu crois en moi, même lorsque moi-même doute de mes capacités. Tu es mon pilier et mon exemple. Je sais que tu seras toujours là pour moi. Je te remercie infiniment pour tout ce que tu fais constamment pour moi.

J'aimerais aussi remercier mes sœurs de cœur, Warda et Rima. Merci d'être à mes côtés après toutes ces années, merci d'être les personnes bienveillantes, généreuses et dévouées que vous êtes.

À mes amis : Romaiissa, Salem, Abdou, Meriem, Ferial, Wissal. Vous avez toujours été présents pour moi lorsque j'en ai eu besoin, et pour cela, je vous remercie.

À ma promotion, nous avons partagé trois belles années ensemble. Merci pour tous ces souvenirs que nous avons créés.

Enfin, je dédie ce travail à Bouchra, ma binôme. Pas seulement pour ce PFE mais ma complice pendant ces cinq dernières années. On se sait ;) Merci.

Je vous aime.

- Sabrina

Remerciements

Nous tenons à exprimer nos sincères reconnaissance et gratitude envers nos promotrices, Pr. Latifa HAMAMI et Mme. Fatima-Zohra HAMIDATOU. Leur soutien indéfectible, leurs conseils avisés et leur présence constante tout au long de notre projet ont été d'une valeur inestimable. Leur expertise, leur passion pour le traitement d'images, ainsi que leur nature bienveillante et encourageante, ont été une source d'inspiration pour nous.

Nous tenons également à remercier chaleureusement Pr. BENYAHIA du service d'imagerie médicale à l'hôpital Mustapha Bacha pour son aide précieuse quant à la mise à notre disposition des images médicales nécessaires à notre projet.

De même, nous exprimons notre gratitude envers les membres du jury, le président de jury, Pr. Mourad ADNANE et l'examinatrice Mme. Fatiha LANI. Nous apprécions profondément leurs analyses constructives et leur évaluation rigoureuse de notre travail.

Nous souhaitons également exprimer notre reconnaissance envers l'École Nationale Polytechnique pour avoir financé ce projet, ainsi qu'à tous les enseignants et professeurs qui ont croisé notre chemin depuis nos premières années d'études. Chacun d'entre eux a joué un rôle important dans notre formation et développement, nous leur sommes profondément reconnaissantes.

Bouchra & Sabrina.

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

Introduction générale	18
1 Segmentation d'images	20
1.1 Introduction	21
1.2 Généralités sur le traitement d'images	21
1.3 Pré-traitement	21
1.4 Segmentation d'images	21
1.4.1 Définition générale	21
1.4.2 Buts de la segmentation d'image	22
1.4.3 Principe	22
1.5 Segmentation basée sur les contours	23
1.5.1 Méthodes surfaciques	23
1.5.2 Méthodes dérivatives [1]	24
1.5.2.1 Opérateurs dérivatifs du premier ordre	25
1.5.2.2 Opérateurs dérivatifs du deuxième ordre	25
1.5.2.3 Filtrage optimal	26
1.5.3 Méthodes morphologiques	26
1.5.3.1 Notions et opérateurs morphologiques	26
1.5.3.2 Détection de contours en utilisant le gradient morphologique	27
1.6 Segmentation basée sur les régions	28
1.6.1 Méthodes de classification	28
1.6.1.1 Méthodes unidimensionnelles (Seuillage)	29
1.6.1.2 Méthodes multidimensionnelles	29
1.6.2 Méthodes markoviennes	30
1.6.3 Méthodes structurales	30
1.6.3.1 Agrégation de pixels	31
1.6.3.2 Segmentation par Corrélation d'Histogrammes Locaux	31
1.6.3.3 Méthodes de Division et de Fusion	32
1.7 Conclusion	33
2 Etat de l'Art sur les Contours Actifs et les Courbes de Niveau	34

2.1	Introduction	35
2.2	Contours Actifs Paramétriques (Snakes)	35
2.2.1	Contours Actifs Paramétriques Traditionnels	36
2.2.1.1	Énergies	37
2.2.1.2	Minimisation de l'équation d'énergies	38
2.2.2	Méthodes d'optimisation	40
2.2.2.1	Programmation Dynamique	40
2.2.2.2	Algorithme de Greedy	41
2.2.2.3	Gradient Vector Flow (GVF)	42
2.3	Courbes de niveau (Level sets)	44
2.3.1	Principe de base	45
2.3.2	Méthode des courbes de niveau traditionnelle	45
2.3.2.1	Initialisation de la fonction de distance	46
2.3.2.2	Evolution de la courbe	46
2.3.3	Formulation variationnelle des courbes de niveau sans la réinitiali- sation de la fonction de distance	52
2.3.3.1	Sélection du pas relatif au temps Δt	54
2.3.3.2	Initialisation flexible de la fonction des courbes de niveau	54
2.3.4	Classification des modèles de courbes de niveau	54
2.3.4.1	Modèles de courbes de niveau basés sur les contours	54
2.3.4.2	Modèles de courbes de niveau basés sur les régions	54
2.3.5	Algorithmes des modèles de courbes de niveau	55
2.3.5.1	Modèle d'évolution des courbes de niveau régularisé en fonction de la distance (DRLSE)	55
2.3.5.2	Modèle d'ajustement local d'image (LIF)	56
2.3.5.3	Modèle d'évolution des courbes de niveau avec estimation du champ de biais (LSE-BFE)	57
2.4	Conclusion	61
3	Simulation sur MATLAB	62
3.1	Introduction	63
3.2	Logiciel MATLAB	63
3.3	Imagerie médicale	63
3.3.1	Segmentation des images médicales	63
3.3.2	IRM du Sein	64
3.3.2.1	Récupération des images	64
3.3.2.2	Utilisation d'un logiciel pour la visualisation et l'exportation	65
3.3.3	IRM du cerveau	66
3.3.4	Pré-traitement de l'image	67
3.3.4.1	Mécanisme du filtrage anisotrope	67
3.3.4.2	Résultats de l'application du filtre	68
3.3.4.3	Interprétation des résultats de l'application du filtre	69
3.4	Simulation de la méthode des contours actifs sur MATLAB	69
3.4.1	Description de l'algorithme de la méthode GVF	70
3.4.1.1	Initialisation du contour actif (snake)	71
3.4.1.2	Carte de contours	71

3.4.1.3	Forces externes u et v	72
3.4.1.4	Déformation du contour actif	72
3.4.2	Résultats sur des images synthétiques	73
3.4.2.1	Initialisation	76
3.4.2.2	Avec et Sans interpolation	76
3.4.2.3	Réglages des paramètres μ , α , β et tol	77
3.4.3	Résultats sur des images médicales	79
3.4.3.1	Sur l'IRM d'un cerveau	79
3.4.3.2	Sur l'IRM d'un sein	80
3.5	Simulation des modèles des Courbes de niveau sur MATLAB	80
3.5.1	Description de l'algorithme du modèle DRLSE	81
3.5.1.1	Initialisation du champ de niveau de gris (LSF)	82
3.5.1.2	Boucle principale des itérations externes	82
3.5.1.3	Boucle interne d'itération	82
3.5.1.4	Choix de la fonction potentielle	83
3.5.2	Résultats du modèle DRLSE sur des images synthétiques	84
3.5.3	Résultats du modèle DRLSE sur des images médicales	86
3.5.3.1	Réglages des paramètres α , λ et μ	86
3.5.3.2	Sur l'IRM d'un cerveau	88
3.5.3.3	Sur l'IRM d'un sein	88
3.5.4	Description de l'algorithme du modèle LSE-BFE	89
3.5.4.1	Initialisation du champ de niveau de gris (LSF) et du biais (b)	90
3.5.4.2	Boucle principale des itérations externes	90
3.5.4.3	Boucle interne d'itération	91
3.5.4.4	Segmentation de l'image en régions distinctes	91
3.5.5	Résultats du modèle LSE-BFE sur des images synthétiques	92
3.5.6	Résultats du modèle LSE-BFE sur des images médicales	92
3.5.6.1	Sur l'IRM d'un cerveau	93
3.5.6.2	Sur l'IRM d'un sein	93
3.6	Conclusion	94
4	Implémentation sur FPGA	95
4.1	Introduction	96
4.2	Logiciels et matériels	96
4.3	Étapes de l'implémentation	96
4.3.1	Codage en C et simulation	96
4.3.2	Création des blocs IP (Intellectual Property)	96
4.3.3	Implémentation et Déploiement	97
4.4	Implémentation de l'algorithme GVF	97
4.4.1	Implémentation par parties	98
4.4.1.1	Codage en C	98
4.4.1.2	Création des blocs IP	100
4.4.1.3	Création du design par blocs	104
4.4.1.4	Programmation du CPU	111
4.4.2	Implémentation complète	116

4.4.2.1	Codage en C	116
4.4.2.2	Création des blocs IP	117
4.4.2.3	Création du design par blocs	118
4.4.2.4	Programmation du CPU	119
4.5	Implémentation de l'algorithme DRLSE	121
4.5.1	Implémentation par parties	121
4.5.1.1	Codage en C	121
4.5.1.2	Création des blocs IP	122
4.5.1.3	Création du design par blocs	123
4.5.1.4	Programmation du CPU	124
4.5.2	Implémentation complète	126
4.5.2.1	Codage en C	127
4.5.2.2	Création des blocs IP	127
4.5.2.3	Création du design par blocs	128
4.5.2.4	Programmation du CPU	129
4.6	Implémentation de l'algorithme LSE-BFE	130
4.6.1	Codage en C	130
4.6.2	Création des blocs IP	130
4.6.3	Création du design par blocs	132
4.6.4	Programmation du CPU	133
4.7	Résultats et interprétations	133
4.7.1	Algorithme GVF	133
4.7.2	Algorithme DRLSE	134
4.7.3	Algorithme LSE BFE	134
4.7.4	Discussion	134
4.8	Conclusion	136
Conclusion Générale et Perspectives		137
Bibliographie		139
Webographie		142
Annexes		144

Liste des tableaux

4.1	Temps d'exécution de chaque partie de l'algorithme GVF	116
4.2	Temps d'exécution de chaque partie de l'algorithme DRLSE	126
4.3	Utilisation des ressources matérielles et temps de calcul pour l'algorithme GVF	134
4.4	Utilisation des ressources matérielles et temps de calcul pour l'algorithme DRLSE	134
4.5	Utilisation des ressources matérielles et temps pour l'algorithme LSE-BFE	134

Table des figures

1.1	Courbes des dérivées première et seconde d'une transition "saut d'amplitude"	24
2.1	Principe des contours actifs	36
2.2	Illustration de la gestion des changements de topologie de l'implémentation en courbes de niveau : la fonction ϕ se déforme et, bien qu'elle ne change pas de topologie directement, le sous-ensemble observé (son intersection avec le plan de niveau zéro) change de topologie en donnant deux contours distincts	45
2.3	Utilisation de la fonction $\phi(x, y)$	46
2.4	Chevauchements des niveaux de la fonction de distance	51
2.5	Fonction ϕ après la réinitialisation	51
3.1	Les dossiers DICOM	64
3.2	Les fichiers .ima	65
3.3	Ouverture d'un dossier DICOM dans RadiAnt	65
3.4	Exportation des images depuis RadiAnt	66
3.5	Image d'une section présentant une anomalie	66
3.6	Images de la base de données "Brain Tumor Data"	67
3.7	Résultat de l'application du filtre anisotrope sur une image de la base de données "BRATS MICCAI brain tumor dataset" avec différentes valeurs de κ	68
3.8	Résultat de l'application du filtre anisotrope sur une image de l'IRM d'un sein avec différentes valeurs de κ	68
3.9	Organigramme de la méthode GVF	70
3.10	Résultat de la segmentation avec GVF sur une image synthétique "Puzzle", $\mu = 0.1, \alpha = 0.1, \beta = 0.052, tol = 0.04, it_{max} = 100$	73
3.11	Représentation des Forces Externes (u, v)	74
3.12	Résultat de la segmentation avec GVF sur une image synthétique U, $\mu = 0.15, \alpha = 0.1, \beta = 0.05, tol = 0.001, it_{max} = 500$	75
3.13	Résultat de la segmentation avec GVF sur une image synthétique contenant plusieurs formes géométriques, $\mu = 0.15, \alpha = 0.2, \beta = 0.05, tol = 0.005, it_{max} = 300$	75
3.14	Résultat de la segmentation avec GVF sur l'image "U"	76
3.15	Résultat de la segmentation avec GVF sur une image synthétique "puzzle" avec et sans interpolation des points du contour actif	77
3.16	Résultat du champs de forces externes avec 3 valeurs différentes de μ	77
3.17	Résultat de la segmentation avec GVF sur l'image "Puzzle" avec $\mu = 0.01$	78

3.18	Résultat de la segmentation avec GVF sur l'image "Puzzle" avec 3 valeurs différentes de α	78
3.19	Résultat de la segmentation avec GVF sur l'image "Puzzle" avec 3 valeurs différentes de β	79
3.20	Résultat de la segmentation avec GVF Sur l'IRM d'un cerveau, $\mu = 0.15, \alpha = 0.1, \beta = 0.06, tol = 0.004, it_{max} = 300$	80
3.21	Organigramme de la méthode DRLSE	81
3.22	Résultat de la segmentation avec l'algorithme DRLSE sur une image synthétique U'	84
3.23	Résultat de la segmentation avec l'algorithme DRLSE sur l'image d'un puzzle	85
3.24	Résultat du modèle DRLSE sur une image synthétique contenant plusieurs formes géométriques	85
3.25	Résultats de l'algorithme DRLSE sur une image médicale avec différentes valeurs de α avec 100 itérations	86
3.26	Résultats de l'algorithme DRLSE sur une image médicale avec différentes valeurs de λ avec 100 itérations	87
3.27	Résultats de l'algorithme DRLSE sur une image médicale avec différentes valeurs de μ avec 100 itérations	87
3.28	Résultats de l'algorithme DRLSE sur des IRM du cerveau $\alpha = 0.5, \lambda = 6$ et $\mu = 0.02$	88
3.29	Résultat de segmentation avec l'algorithme DRLSE sur l'IRM du sein avec $\alpha = 1.5, \lambda = 5$ et $\mu = 0.02$	89
3.30	Organigramme du modèle LSE-BFE	89
3.31	Résultat de la segmentation avec le modèle LSE-BFE sur des images synthétiques	92
3.32	Résultat de la segmentation avec le modèle LSE-BFE sur des IRM du cerveau	93
3.33	Résultat de la segmentation avec le modèle LSE-BFE sur des IRM du sein	93
4.1	Schéma synoptique simplifié de l'approche HLS	97
4.2	Code en C de la fonction carte de contours	98
4.3	Schéma de la partie 01 de l'algorithme GVF	99
4.4	Schéma de la partie 02 de l'algorithme GVF	99
4.5	Schéma de la partie 03 de l'algorithme GVF	99
4.6	Interface de Vitis HLS	100
4.7	Canaux de lecture et d'écriture AXI	101
4.8	Résumé de synthèse pour chaque partie	103
4.9	Graphique d'appel de la fonction principale "Déformation du contour" avec l'option de la carte thermique de BRAM	104
4.10	Interface Vivado	105
4.11	Bloc IP de la fonction "Carte de contours"	105
4.12	Ajout des différents blocs IP	106
4.13	Bloc design de la partie 01 - Carte de contours	107
4.14	Rapport d'utilisation de ressources résumé	108
4.15	Rapport d'utilisation de ressources détaillé	108
4.16	Rapport de consommation d'énergie	109

4.17	Résumé d'implémentation pour la fonction "Carte de contours"	110
4.18	Résumé d'implémentation pour la fonction "Champs de forces externes (u, v)"	111
4.19	Résumé d'implémentation pour la fonction "Déformation du contour" . . .	111
4.20	Création d'un overlay pour l'implémentation de la partie 1 - Carte de contours	112
4.21	Mappage du bloc Contours_Calcul pour l'implémentation de la partie 1 - Carte de contours	113
4.22	Allocation de mémoire pour le bloc Contours_Calcul pour l'implémentation de la partie 1 - Carte de contours	113
4.23	Exécution du code sur FPGA	114
4.24	Création du fichier .csv pour l'output_fx	114
4.25	Valeurs de fx sur MATLAB et sur notebook	115
4.26	Résultat de l'implémentation par parties de la méthode GVF	116
4.27	Schéma des différents blocs de l'implémentation complète de l'algorithme GVF	117
4.28	Résumé de synthèse pour chaque partie	117
4.29	Résumé de synthèse pour chaque partie	118
4.30	Design par blocs pour l'implémentation complète de l'algorithme des contours actifs par la méthode GVF	118
4.31	Résumé d'implémentation complète de l'algorithme des contours actifs par la méthode GVF	119
4.32	Valeurs de xf sur MATLAB et sur notebook	120
4.33	Valeurs de yf sur MATLAB et sur notebook	120
4.34	Résultat de l'implémentation complète de la méthode GVF	121
4.35	Schéma de la partie 01 de l'algorithme DRLSE	121
4.36	Schéma de la partie 02 de l'algorithme DRLSE	122
4.37	Résumé de synthèse pour chaque partie	122
4.38	Schéma de la partie 02 de l'algorithme DRLSE	123
4.39	Résumé d'implémentation de la partie 1 de l'algorithme DRLSE	123
4.40	Résumé d'implémentation de la partie 2 de l'algorithme DRLSE	124
4.41	Sorties des 2 parties de l'algorithme DRLSE sur le notebook	124
4.42	Sorties des 2 parties de l'algorithme DRLSE sur Matlab	125
4.43	Résultat de l'implémentation par parties de la méthode DRLSE	126
4.44	Schéma des différents blocs de l'implémentation complète de l'algorithme DRLSE	127
4.45	Résumé de synthèse pour chaque partie	128
4.46	Design par blocs pour l'implémentation complète de l'algorithme des courbes de niveau par l'approche contours DRLSE	128
4.47	Résumé d'implémentation complète de l'algorithme des courbes de niveau par l'approche contours DRLSE	129
4.48	Output_buffer5 (Output_phi)	129
4.49	Résultat de l'implémentation complète de la méthode DRLSE	129
4.50	Schéma de l'algorithme LSE-BFE	130
4.51	Rapport de synthèse de l'algorithme LSE-BFE	131
4.52	Design par blocs pour l'implémentation de l'algorithme LSE-BFE	132

4.53	Résumé d'implémentation de l'algorithme des courbes de niveau par l'approche régions LSE-BFE	132
4.54	Résultat de l'implémentation complète de la méthode LSE-BFE	133

Liste des abréviations

1D *One dimension.*

2D *Two dimensions.*

API *Application Programming Interface.*

AXI *Advanced eXtensible Interface.*

BD *Backward Differences.*

BRAM *Block Read Access Memory.*

CD *Centred Differences.*

CPU *Central Processing Unit.*

DICOM *Digital Imaging and Communication in Medicine.*

DRLSE *Distance Regularized Level Set Evolution*

DSP *Digital Signal Processor.*

FD *Forward Differences.*

FF *Flip-Flop.*

FPGA *Field-Programmable Gate Array.*

GVF *Gradient Vector Flow.*

HDL *Hardware Description Language.*

HLS *High-Level Synthesis.*

IRM *Imagerie par Résonance Magnétique.*

IP *Intellectual Property.*

LIF *Local image fitting.*

LSE-BFE *Level Set Evolution and Bias Field Estimation.*

LSF *Level Set Fonction.*

LUT *Look Up Table.*

MATLAB *MATrix LABoratory.*

PL *Programmable Logic.*

PS *Processing System.*

PYNQ *Python Productivity for ZYNQ.*

RTL *Register Transfer Level.*

SoC *System-on-Chip.*

TCL *Tool Command Language.*

tol *tolérence.*

VHDL *Very High Speed Integrated Circuit Hardware Description Language.*

Introduction Générale

La segmentation d'images informatisée est essentielle en imagerie médicale, utilisée pour des applications comme le diagnostic, la localisation de pathologies et la planification de traitements. Cependant, la tâche est complexe en raison de la variabilité des formes d'objets et de la qualité des images souvent affectées par le bruit.

Les modèles déformables, largement adoptés pour surmonter ces défis, sont des courbes ou surfaces qui se déplacent sous l'influence de forces internes et externes pour suivre les contours des objets dans une image. Ils offrent robustesse et précision, cruciales pour l'imagerie médicale.

Deux types principaux de modèles déformables existent : les contours actifs paramétriques, qui permettent une interaction directe mais sont moins adaptables topologiquement, et les courbes de niveau, qui gèrent naturellement les changements topologiques grâce à des représentations implicites.

En résumé, bien que les modèles déformables présentent des avantages significatifs pour la segmentation d'images médicales, leur complexité de calculs nécessite des optimisations avancées pour une utilisation efficace, notamment en temps réel et dans des environnements contraints en ressources.

Motivation

La motivation derrière la conduite de ce projet de fin d'études et la mise en œuvre des algorithmes de contours actifs et de courbes de niveau sur FPGA provient de la nécessité de solutions de segmentation d'images précises et efficaces dans des applications en temps réel. Les contours actifs et les courbes de niveau sont connus pour leur précision et leur adaptabilité, mais leur complexité de calculs représente un défi pour les systèmes embarqués. En tirant parti des capacités de traitement parallèle et de l'accélération matérielle des FPGA, nous visons à surmonter ces défis pour obtenir une segmentation d'images en temps réel et efficace.

Objectifs du projet

- **Adaptation d'Algorithmes :**
 - Tester et adapter des algorithmes de contours actifs (GVF) et de courbes de niveau (DRLSE et LSE BFE) pour la segmentation d'images.

- Explorer différentes techniques et architectures pour améliorer la précision et la robustesse de la segmentation.
- **Co-conception Matériel-Logiciel :**
 - Optimiser les algorithmes existants pour une implémentation sur FPGA.
 - Explorer les techniques d'accélération matérielle et les optimisations algorithmiques pour minimiser l'utilisation des ressources FPGA tout en maintenant une haute performance.
- **Déploiement et Validation :**
 - Valider le système implémenté en utilisant des images fournies par des partenaires médicaux.
 - Évaluer son efficacité en termes de précision de segmentation, temps de traitement et utilisation des ressources.

Structure du Travail

La résolution est abordée selon les étapes suivantes :

- Revue de la Littérature :
 - Conduire une revue approfondie de la littérature existante sur la segmentation d'images, les techniques de contours actifs et de courbes de niveau, et l'implémentation sur FPGA.
 - Identifier les concepts clés, algorithmes et méthodologies pertinents pour l'étude.
 - Analyser les forces et les limites des travaux précédents pour établir une base pour le projet.
- Prétraitement des Images :
 - Recueillir et prétraiter les images pour éliminer le bruit et normaliser les images.
- Simulation des Algorithmes sur MATLAB :
 - Simuler les algorithmes GVF, DRLSE et LSE BFE sur MATLAB en utilisant des images synthétiques et médicales.
 - Analyser les résultats des simulations pour évaluer la précision et l'efficacité de chaque algorithme.
- Implémentation des Algorithmes sur FPGA :
 - Optimiser les algorithmes pour une implémentation sur FPGA.
 - Déployer les algorithmes sur FPGA et évaluer leur performance en termes de temps de traitement et d'utilisation des ressources.

Chapitre 1

Segmentation d'images

1.1 Introduction

La segmentation d'images constitue une étape fondamentale dans le domaine du traitement d'images, offrant une approche cruciale pour comprendre et extraire des informations significatives à partir de données visuelles. Ce chapitre plonge dans cette discipline, couvrant les bases du traitement d'images, les outils mathématiques, le pré-traitement, et les diverses approches de segmentation.

Nous amorçons notre exploration en introduisant les concepts fondamentaux du traitement d'images. Par la suite, nous aborderons le rôle vital du pré-traitement dans la préparation des images pour la segmentation. La démarche se poursuit avec une exploration approfondie de la segmentation d'image, définissant son objectif et présentant les différentes méthodes utilisées.

Le chapitre explore deux grandes catégories de segmentation, basées sur les contours et les régions, en examinant diverses méthodes telles que les dérivatives, morphologiques, de classification, markoviennes et structurales.

1.2 Généralités sur le traitement d'images

Avant d'explorer les aspects techniques plus avancés, il est crucial de se familiariser avec les concepts de base du traitement d'images expliqués en **Annexe 1**.

1.3 Pré-traitement

La segmentation implique de diviser une image en régions distinctes, basées sur un ou plusieurs critères afin de créer des zones homogènes. Ces régions se distinguent par des différences significatives selon ces critères. Le prétraitement vise à faciliter cette segmentation en renforçant la similarité entre les pixels d'une même région ou en accentuant les différences entre les pixels de régions distinctes. Étant donné que les images contiennent souvent beaucoup de données, il est nécessaire d'appliquer des opérateurs locaux pour éviter des temps de traitement excessifs. Ces opérateurs se limitent à un petit voisinage de pixels autour du pixel courant pour leurs calculs.

Plusieurs méthodes de prétraitement sont disponibles, notamment la modification d'histogrammes, la réduction du bruit et le rehaussement de contraste [1].

1.4 Segmentation d'images

1.4.1 Définition générale

La segmentation d'images est une étape clé du processus de traitement des images numériques, elle permet d'extraire les objets d'intérêts, d'améliorer la précision des mesures et de faciliter l'interprétation [1]. C'est à partir de l'image segmentée que les paramètres

discriminants (caractéristiques spécifiques) sont extraits pour la classification ou l'interprétation. En résumé, la segmentation est un pilier fondamental pour une analyse d'image efficace et robuste.

1.4.2 Buts de la segmentation d'image

Parmi les buts de la segmentation d'images, nous citons [1] :

- Amélioration de la qualité de l'image :
 - Réduire le bruit et les artefacts,
 - Améliorer le contraste et la netteté de l'image.
- Décomposition et extraction d'entités :
 - Diviser l'image en zones homogènes.
 - Extraire et séparer les différentes entités présentes.
- Analyse et interprétation précises :
 - Déterminer les caractéristiques de chaque zone (forme, texture, etc.).
 - Appliquer des traitements spécifiques à chaque zone d'intérêt.
 - Interpréter le contour d'une image et localiser précisément les contours d'une zone.
- Applications spécifiques :
 - Faciliter la reconnaissance d'objets et la classification d'images.
 - Mesurer et analyser des structures dans des images scientifiques.
 - Améliorer la compression d'images et la transmission d'images.

1.4.3 Principe

Soit R la région spatiale entière occupée par une image. Nous pouvons considérer la segmentation d'une image comme un processus qui divise R en n sous-régions, R_1, R_2, \dots, R_n telles que [2] :

(a)

$$\bigcup_{i=1}^n R_i = R.$$

(b) R_i est un ensemble connecté, pour $i = 1, 2, \dots, n$.

(c) $R_i \cap R_j = \emptyset$, pour tous i et j , $i \neq j$.

(d) $Q(R_i) = \text{VRAI}$, pour $i = 1, 2, \dots, n$.

(e) $Q(R_i \cup R_j) = \text{FAUX}$, pour toutes les régions adjacentes R_i et R_j .

$Q(R_k)$ est un prédicat logique défini sur les points de l'ensemble R_k .

- La condition (a) indique que la segmentation doit être complète, en ce sens que chaque pixel doit se trouver dans une région.
- La condition (b) exige que les points d'une région soient connectés dans un sens prédéfini.
- La condition (c) stipule que les régions doivent être disjointes.
- La condition (d) concerne les propriétés qui doivent être satisfaites par les pixels d'une région segmentée - par exemple, $Q(R_i) = \text{VRAI}$ si tous les pixels de R_i ont la même intensité.
- Enfin, la condition (e) indique que deux régions adjacentes R_i et R_j doivent être différentes au sens du prédicat Q .

On voit donc que le problème fondamental de la segmentation est de diviser une image en régions satisfaisant les conditions précédentes. Les algorithmes de segmentation pour les images monochromes sont généralement basés sur l'une des deux catégories de base qui traitent des propriétés des valeurs d'intensité : la discontinuité et la similarité. Dans la première catégorie, nous supposons que les limites des régions soient suffisamment différentes les unes des autres et de l'arrière-plan pour permettre la détection des limites sur la base des discontinuités locales d'intensité. La segmentation basée sur les contours est la principale approche utilisée dans cette catégorie. Les approches de segmentation par région de la deuxième catégorie sont basées sur la partition d'une image en régions similaires selon un ensemble de critères prédéfinis [2].

1.5 Segmentation basée sur les contours

Dans une image, les variations d'intensité reflètent des changements dans les propriétés physiques ou géométriques de la scène, tels que des variations d'illumination, d'ombres, de changements d'orientation ou de distance à l'observateur, de réflectance de surface, et d'absorption des rayons. Ces variations sont souvent cruciales pour la segmentation, car elles définissent les frontières entre les régions et les objets de la scène .

Une approche consiste à considérer l'image numérique comme l'échantillonnage d'une fonction scalaire $A(x, y)$ à support borné et dérivable. Les méthodes surfaciques, dérivatives et morphologiques sont utilisées pour traiter ce problème, en particulier dans le domaine continu.

1.5.1 Méthodes surfaciques

Ces méthodes considèrent l'image des intensités comme une surface. La transition entre deux régions est modélisée par un gabarit utilisé pour détecter les contours. L'ap-

proximation par facettes de la surface fournit une équation analytique locale permettant un calcul précis (sub-pixel) de la position du contour et de ses caractéristiques.

1.5.2 Méthodes dérivatives [1]

Si on considère un signal continu monodimensionnel $A(x)$ qui présente une transition avec un saut d'amplitude en x_0 .

Donc, en $1D$ un contour correspond à un maximum de la dérivée première, c'est-à-dire à un passage par zéro de la dérivée seconde comme le montre la figure 1.1 [3] :

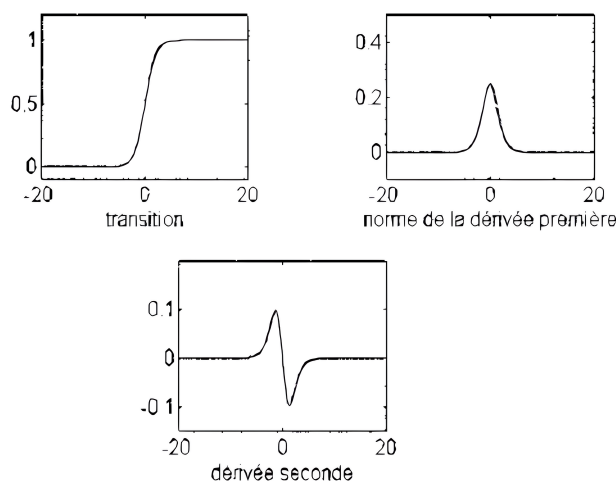


FIG. 1.1 : Courbes des dérivées première et seconde d'une transition "saut d'amplitude"

Pour retrouver cette propriété en $2D$, il faut se placer dans la direction du gradient. Les contours sont donc définis comme les maxima de la dérivée première dans la direction du gradient, c'est-à-dire des passages par zéros de la dérivée seconde dans la direction du gradient.

Donc dans le cas d'une image (cas bidimensionnel), on considère une fonction continue $A(x, y)$, tel que le vecteur gradient est donné par :

$$\vec{\nabla} A = \begin{pmatrix} \frac{\partial A}{\partial x} \\ \frac{\partial A}{\partial y} \end{pmatrix} \quad (1.1)$$

Le gradient $\vec{\nabla} A$ est caractérisé par son module et sa direction. Les expressions usuelles de ces grandeurs en norme euclidienne sont :

$$|\vec{\nabla} A| = \sqrt{\left(\frac{\partial A}{\partial x}\right)^2 + \left(\frac{\partial A}{\partial y}\right)^2} \quad (1.2)$$

$$\theta = \arg(\vec{\nabla} A) = \tan^{-1} \left(\frac{\partial A}{\partial y} / \frac{\partial A}{\partial x} \right) \quad (1.3)$$

Le module du vecteur gradient représente la pente de la surface image au point x_0 . La

présence locale d'un module élevé traduit une forte variation du niveau de gris autour de ce point.

Lorsqu'il y a un contour, c'est à dire une forte variation locale du niveau de gris, le vecteur gradient est perpendiculaire au contour. Étant donné que la dérivée directionnelle est nulle dans la direction perpendiculaire au contour, la variation de niveau de gris est nulle le long du contour [4].

1.5.2.1 Opérateurs dérivatifs du premier ordre

Les dérivées partielles sont approximées par différence finie telle que :

$$\frac{\partial A}{\partial y} \approx A_i(i, j) = A(i + 1, j) - A(i, j) \quad (1.4)$$

$$\frac{\partial A}{\partial x} \approx A_j(i, j) = A(i, j + 1) - A(i, j) \quad (1.5)$$

A_i et A_j sont respectivement un gradient vertical qui met en évidence les détails horizontaux de l'image, et un gradient horizontal qui met en évidence les détails verticaux de l'image.

Le module de ces deux gradients nous permet d'obtenir les contours de l'image.

En pratique, la plupart des opérateurs sont implémentés sous forme de masque, voici quelques exemples de masques couramment utilisés :

- **Opérateurs de Prewitt et de Sobel** : pour ces opérateurs, les dérivées directionnelles (gradients) horizontale et verticale s'expriment sous la forme :

$$\begin{cases} A_i(i, j) = h_i * A(i, j) \\ A_j(i, j) = h_j * A(i, j) \end{cases} \quad (1.6)$$

$$\text{Avec : } h_j = \begin{pmatrix} 1 & 0 & -1 \\ c & 0 & -c \\ 1 & 0 & -1 \end{pmatrix} \text{ et } h_i = \begin{pmatrix} 1 & c & -1 \\ 0 & 0 & 0 \\ -1 & -c & -1 \end{pmatrix}$$

Les matrices h_i et h_j , appelées aussi masques, sont les noyaux de convolution de filtres à réponse impulsionnelle finie.

Prewitt : $c = 1$ et Sobel : $c = 2$.

- Il existe d'autres opérateurs du premier ordre ; Kirsh, MDIF, NAGDIF et Roberts [1].

1.5.2.2 Opérateurs dérivatifs du deuxième ordre

Comme mentionné précédemment, les contours peuvent être localisés par les passages à zéro du Laplacien, celui-ci est donné par l'expression :

$$\Delta A(x, y) = \frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2} \quad (1.7)$$

- **Opérateur Laplacien sur voisinage réduit** : des approximations discrètes du Laplacien, calculées sur un voisinage 3x3, correspondent aux masques suivants :

$$\text{– 4-connexités : } \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\text{– 8-connexités : } \begin{pmatrix} 0 & 1 & 0 \\ 1 & -8 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

- **Opérateur Marr et Hildreth** : consiste à effectuer une convolution de l'image à l'aide d'une fonction gaussienne, suivie de l'application d'un filtre Laplacien. Ils suggèrent d'utiliser divers masques correspondant à différentes valeurs de la variance σ^2 de la gaussienne.
- **Opérateur Huertas-Médioni** : suggère une décomposition en filtres séparables de l'opérateur Laplacien d'une gaussienne.

1.5.2.3 Filtrage optimal

Dans le cas d'images complexes, les méthodes présentées ci-dessus ne sont pas très précises, en effet, les résultats obtenus sont très sensibles au bruit et engendrent souvent de fausses détections (de faux contours sont détectés dans les zones bruitées ou texturées et des contours significatifs dont le gradient est faible sont oubliés).

Pour résoudre les problèmes de précision, de localisation et d'efficacité de détection, il existe trois filtres optimaux : filtre de Canny, le filtre de Deriche et le filtre de Shen et Castan [1].

1.5.3 Méthodes morphologiques

La morphologie mathématique repose sur la comparaison entre l'image A à segmenter et un élément structurant appelé B . Les opérations employées dans ce contexte sont des opérations ensemblistes, à savoir la dilatation et l'érosion. À partir de ces opérations de base, émergent des outils plus sophistiqués tels que l'ouverture et la fermeture.

1.5.3.1 Notions et opérateurs morphologiques

Nous présentons les principales notions et opérateurs morphologiques [5] :

- **Élément structurant**

Un élément structurant représente un ensemble dont les caractéristiques sont définies ; sa forme et sa taille dépendent des objets à extraire de l'image étudiée. Lors d'une transformation morphologique, l'élément structurant est déplacé à travers l'image de manière à ce que son origine soit successivement positionnée sur chaque pixel de l'image. C'est cette interaction avec l'image qui engendre le résultat d'une opération morphologique.

- **Dilatation**

La dilatation, formulée selon la théorie des ensembles, est conceptualisée comme l'addition de Minkowski entre l'image A et l'élément structurant B . Ainsi, si A et B sont deux sous-ensembles de E , la dilatation de A par B , notée $A \oplus B$, est définie par l'ensemble des éléments c de E tels que $(a \in A \text{ et } b \in B)$ avec $c = a + b$.

En pratique, cette opération est réalisée en appliquant une fenêtre de taille fixe sur l'image et en effectuant un OU logique entre les pixels constituant la fenêtre pour chaque pixel de l'image, à l'exception du pixel traité (pixel central).

- **Érosion**

L'érosion, opération duale, correspond au complément de la somme de Minkowski entre le complément de l'image X et l'élément structurant B .

Si A et B sont deux sous-ensembles de E , alors l'érosion de A par B , notée $A \ominus B$, est définie par l'ensemble des éléments c de E tels que $c + b \in A$ pour tout $b \in B$.

En pratique, cette opération consiste à effectuer un ET logique entre les pixels contenus dans la fenêtre utilisée, à l'exception du pixel central.

- **Ouverture et fermeture**

L'ouverture et la fermeture sont des opérations morphologiques définies par une séquence spécifique d'érosion et de dilatation. L'ouverture consiste en une érosion suivie d'une dilatation, tandis que la fermeture implique une dilatation suivie d'une érosion.

L'ouverture d'une image A par un élément structurant B , notée $A \odot B$, est définie par la séquence d'opérations $(A \ominus B) \oplus B$. Cela équivaut à l'érosion de l'image A par l'élément structurant B , suivie de la dilatation du résultat obtenu.

De manière similaire, la fermeture d'une image A par un élément structurant B , notée $A \bullet B$, est définie par la séquence d'opérations $(A \oplus B) \ominus B$. Cela se traduit par la dilatation de l'image A par B , suivie de l'érosion du résultat.

En d'autres termes, l'ouverture a tendance à éliminer les petites structures indésirables tout en préservant les structures plus importantes, tandis que la fermeture a pour effet de remplir les lacunes dans les objets tout en maintenant leur forme générale. Ces opérations sont couramment utilisées en morphologie mathématique pour le filtrage.

Remarque

Les opérations de dilatation, érosion, ouverture et fermeture présentées précédemment sont les opérations binaires dans ce contexte. Lorsque nous les appliquons à des images en niveaux de gris, il est nécessaire de remplacer respectivement les concepts de ET et de OU logiques par ceux de maximum et de minimum (MAX et MIN).

1.5.3.2 Détection de contours en utilisant le gradient morphologique

La détection de contours peut être effectuée à l'aide du gradient morphologique ou du Laplacien [1], plus précisément en calculant la norme de ces derniers :

- **Gradient morphologique**

Le gradient morphologique se base sur les opérations d'érosion et de dilatation, pour

une image A , la norme du gradient s'obtient par l'opération :

$$\|\nabla A\| = \frac{(A \oplus B) - (A \ominus B)}{2} \quad (1.8)$$

- **Laplacien morphologique**

Le Laplacien est un opérateur différentiel :

- **Gradient par dilatation**

Pour une image A , la norme du gradient par dilatation s'obtient par l'opération :

$$\|\nabla A^+\| = (A \oplus B) - A \quad (1.9)$$

- **Gradient par érosion**

Pour une image A , la norme du gradient par érosion s'obtient par l'opération :

$$\|\nabla A^-\| = A - (A \ominus B) \quad (1.10)$$

Le module du Laplacien ΔA d'une image A est de ce fait la différence entre le module du gradient par dilatation et celui du gradient par érosion :

$$\|\Delta A\| = \|\nabla A^+\| - \|\nabla A^-\| \quad (1.11)$$

La sensibilité au bruit constitue un défi dans le calcul du gradient morphologique. Afin d'améliorer les résultats de la détection de contours lors de l'utilisation du gradient morphologique, il faut appliquer un filtrage morphologique préalable à la segmentation de l'image.

1.6 Segmentation basée sur les régions

Une région désigne un ensemble de pixels partageant des caractéristiques communes. L'approche par régions vise à décomposer une image en segments distincts, où chaque pixel appartient à une et une seule région.

Trois catégories principales de méthodes de segmentation par régions [3] existent ; méthodes de classification, méthodes markoviennes et méthodes structurales.

1.6.1 Méthodes de classification

Les méthodes de classification reposent sur l'évaluation de la similarité entre les individus d'une image, qui peuvent être définis comme des pixels, des sous-images de taille fixe ou des régions issues d'une première segmentation. La classification constitue une étape préliminaire à la segmentation, qui consiste à regrouper les pixels appartenant à une même classe.

La classification s'opère en utilisant divers attributs et critères dans l'image, tels que le niveau de gris des pixels obtenus à partir de l'histogramme ou les moments et indices de texture dans le voisinage du pixel considéré. Les méthodes de classification se divisent principalement en deux catégories : les méthodes unidimensionnelles, qui utilisent un seul attribut (comme le niveau de gris), et les méthodes multidimensionnelles, qui exploitent plusieurs attributs pour différencier les individus et former les classes.

1.6.1.1 Méthodes unidimensionnelles (Seuillage)

Le seuillage est une technique simple, non contextuelle, globale, qui repose sur une mesure quantitative d'une grandeur. Il permet de classer les pixels en deux catégories, ceux dont la mesure est inférieure au seuil et ceux dont la mesure excède ou égale le seuil.

$$g(x, y) = \begin{cases} 0 & \text{si } f(x,y) < S \\ 1 & \text{si } f(x,y) \geq S \end{cases} \quad (1.12)$$

- **Types de seuillage** : l'utilisation de l'histogramme se révèle être un outil essentiel pour la classification monodimensionnelle en déterminant des seuils basés sur les niveaux de gris de l'image. Trois approches principales peuvent être employées :
 - **Seuillage Global** : cette méthode, basée sur la recherche des minimums locaux de l'histogramme, est simple en l'absence de bruit. En présence de bruit, d'autres algorithmes, tels que la méthode de Fisher et la méthode de Bhattacharya[6], sont utilisés pour le calcul des seuils.
 - **Seuillage Local** : cette approche ouvre la voie à des méthodes locales itératives et adaptatives, offrant une précision et une robustesse accrues, particulièrement face à d'éventuels changements d'intensité dans l'image.
 - **Seuillage Dynamique** : les méthodes de Chow et Kaneko [1], ainsi que de Nakagawa, utilisent un seuillage dynamique. L'image est subdivisée en blocs, chaque bloc étant approximé par deux ou trois gaussiennes de paramètres connus. Si le bloc est bimodal, le seuil est choisi entre les moyennes des gaussiennes pour minimiser la somme des deux gaussiennes (ou par couples pour trois gaussiennes). Si le bloc n'est pas bimodal, une moyenne pondérée des seuils des voisins est attribuée au bloc, et pour chaque pixel, un seuil est déterminé par interpolation bilinéaire des seuils des centres des blocs environnants. Cette technique dynamique s'adapte aux variations non stationnaires d'intensité moyenne et d'amplitude similaires à celles des fluctuations.

1.6.1.2 Méthodes multidimensionnelles

Pour les méthodes multidimensionnelles, l'accent est mis sur l'utilisation de plusieurs attributs plutôt que sur un seul. Le choix de ces attributs peut être basé sur des connaissances préalables de l'image ou sur une sélection automatique en l'absence d'informations fiables. Les méthodes de classification multidimensionnelles sont généralement regroupées en deux catégories :

- **Classification non supervisée** : cette approche implique la découpe de l'espace de représentation en zones homogènes en fonction d'un critère de ressemblance entre les individus. Ce critère de ressemblance repose sur la proximité dans l'espace des attributs [1].
- **Classification supervisée** : également appelée classification avec apprentissage, cette méthode vise à construire une fonction d'identification ou de discrimination

pour les individus en se basant sur une classification préalablement connue d'un certain nombre d'individus. La fonction discriminante est établie à l'aide de méthodes établies, permettant ainsi de généraliser la classification aux autres individus de l'image [1].

1.6.2 Méthodes markoviennes

L'utilisation de modèles probabilistes et stochastiques en segmentation d'images découle de plusieurs observations clés.

- La modélisation de la texture d'une image n'est pas toujours précise en raison de son caractère aléatoire [7].
- La numérisation d'une image peut entraîner des déformations modélisées comme des phénomènes aléatoires.
- La théorie stochastique offre un cadre propice au développement d'algorithmes d'optimisation efficaces.

L'approche par modélisation probabiliste de la segmentation implique de considérer l'image $Y = (Y_s)$ et la carte des étiquettes $X = (X_s)$ (à construire) comme des variables aléatoires régies par une loi statistique π . Cette approche consiste à proposer une modélisation, c'est-à-dire définir une telle loi π . Étant donné que X et Y sont liées par la loi π , la reconstruction ou l'estimation de X , étant donné Y , se fait à l'aide de π et de Y .

Il est important de noter que la connaissance totale de la loi de formation d'image F , telle que $X = (X_s) \longrightarrow Y = (Y_s) = F(X)$, serait suffisante pour inverser F . Cependant, une telle fonction déterministe F est irréaliste en raison de la complexité du mécanisme de formation d'images et des perturbations liées au bruit. Ainsi, l'approche par modèles probabilistes utilise des lois statistiques conditionnelles pour définir les relations $X \longrightarrow Y$ et $Y \longrightarrow X$.

Les champs de Markov sont des exemples de telles lois conditionnelles et sont parmi les plus utilisés. Ils forment une famille étendue de modèles stochastiques intrinsèquement liés à la notion de système de voisinage, fondamentale en imagerie. Les champs Markoviens placent le problème de la segmentation dans un cadre stochastique, permettant la modélisation des textures, la représentation des déformations subies lors de l'acquisition, et l'utilisation d'algorithmes d'optimisation efficaces dont le comportement est théoriquement compris.

La segmentation Markovienne peut être considérée comme un processus d'étiquetage, où l'auto-organisation de la configuration des étiquettes est traitée par des modèles Markoviens, facilitant ainsi la modélisation des textures et des déformations, ainsi que l'application d'algorithmes d'optimisation bien définis [1].

1.6.3 Méthodes structurales

Ces approches pour la construction de régions reposent principalement sur des considérations algorithmiques. Elles se fondent sur des hypothèses relatives à l'homogénéité

des régions à extraire, ainsi que sur la structure sous-jacente du graphe, étendant ainsi le concept de voisinage pixel à celui d'adjacence entre groupements de points. Nous pouvons les classer en deux catégories distinctes [8] :

- **Stratégie Descendante : Division en Sous-Régions**

Cette approche démarre avec l'image entière et la divise en sous-régions selon leur homogénéité. Elle utilise un algorithme descendant pour construire les régions.

- **Stratégie Ascendante : Fusion depuis Pixels**

Cette méthode commence par une représentation élémentaire de l'image, souvent les pixels, et procède ensuite à des opérations de fusion pour construire des niveaux de regroupement plus élevés. Elle suit une approche ascendante qui permet de former progressivement les régions en agrégeant des éléments plus petits.

1.6.3.1 Agrégation de pixels

La segmentation par agrégation de pixels est une approche ascendante où l'image est initialement traitée comme un ensemble de pixels non regroupés. Le processus démarre à partir de pixels initiaux, et à chaque étape, des pixels voisins sont ajoutés à la région en cours de croissance en respectant les critères d'homogénéité et d'adjacence définis par un prédicat. Ce prédicat agit comme un mode de contrôle défini a priori et peut être de nature géométrique, radiométrique, ou une combinaison des deux. Les critères d'homogénéité peuvent inclure des mesures telles que la variance des niveaux de gris ou la proportion de points en dehors d'un intervalle spécifié. Le processus de croissance se poursuit jusqu'à ce que les pixels adjacents ne satisfont plus les conditions définies par le prédicat, marquant ainsi la fin de la segmentation de la région [9].

1.6.3.2 Segmentation par Corrélacion d'Histogrammes Locaux

La croissance des régions est orchestrée par la segmentation basée sur la corrélation d'histogrammes locaux. Ce processus capitalise sur la variation de la norme euclidienne des histogrammes locaux pour révéler les frontières entre des régions apparemment homogènes mais distinctes. En établissant un semi-ordre sur l'homogénéité des régions, on utilise les maxima de la norme comme points de départ initiaux (germes) pour la croissance des régions, cherchant ensuite à agréger les points avec des distributions similaires. La ressemblance entre les distributions est évaluée par le coefficient de corrélation [1].

Le processus de segmentation peut adopter deux approches :

- **Relaxation** : Après la sélection d'un germe, une première composante connexe est extraite autour de celui-ci avec un seuil T_1 faible. Les points de cette composante ayant des histogrammes locaux voisins du germe sont retenus. Ensuite, les points les plus corrélés dans cette composante forment une zone de référence avec un seuillage à T_2 élevé. La région finale R est obtenue à partir de la zone de référence et d'une autre zone constituée des pixels les plus proches de l'histogramme de référence, choisis selon un seuil T_3 , avec la condition :

$$0 < T_1 < T_3 < T_2 < 1 \quad (1.13)$$

- **Propagation :** La composante connexe est extraite, puis les points de la frontière de la composante qui ne correspondent pas à de vrais contours sont déterminés en vérifiant leur décorrélation du germe. Les points ainsi trouvés deviennent des points de propagation. La segmentation se poursuit en cherchant le prochain germe de norme maximale. Si ce nouveau point est un point de propagation, la région continue de croître en prenant ce point comme point de départ pour extraire une composante connexe qui prolonge la région. Si le point n'a jamais été exploré, il devient le premier germe d'une nouvelle région R_t , et le processus d'exploration de cette région commence. Le reste de la région est segmenté par l'extraction successive de composantes connexes autour de tous les points de propagation[6].

1.6.3.3 Méthodes de Division et de Fusion

Les méthodes de division et de fusion reposent sur l'éclatement et le regroupement de composantes d'image en manipulant le graphe d'adjacence et des prédicats d'uniformité. Le graphe d'adjacence, dénoté par $G = (V, E)$, représente les régions de l'image, où chaque sommet v_i est associé à une région R_i et chaque arête (v_j, v_k) connecte deux régions adjacentes.

Pour fusionner des régions, il est nécessaire de détecter toutes les possibilités de fusionnement en considérant les coûts et l'adjacence des régions. Les arêtes de moindre coût sont sélectionnées à chaque itération, suivant diverses approches [10] [11]. La fusion est réalisée, et le graphe est mis à jour pour la prochaine itération.

- **Partitionnement de Voronoï**

Dans le contexte de la segmentation de textures, le partitionnement de Voronoï associé à la triangulation de Delaunay est employé. Cette méthode cherche à partitionner l'image en régions homogènes de Voronoï. Les germes sont initialement choisis de manière aléatoire avec un processus de "Poisson". La phase de division intervient si des polygones non homogènes persistent, ajoutant de nouveaux germes et créant ainsi de nouvelles régions. La phase de fusion intervient lorsque de nouveaux germes ne peuvent plus être ajoutés, et les polygones de Voronoï voisins sont comparés pour éliminer les germes inutiles [12].

- **Quadtree**

La structure quadtree est utilisée pour découper récursivement l'image en carrés homogènes. Chaque carré est divisé en 4 s'il ne respecte pas le critère d'homogénéité. Cependant, cette approche présente des inconvénients, notamment la non-invariance en translation de l'image et une fondation exclusive sur des critères géométriques.

L'algorithme de segmentation par quadtree repose sur un prédicat P . Si P est faux pour une région, celle-ci est subdivisée en quadrants. La fusion intervient entre les régions adjacentes vérifiant P . Ce processus est répété jusqu'à ce qu'aucune opération supplémentaire ne soit possible [13].

- **Structure pyramidale**

La technique de structure pyramidale est employée pour segmenter une image texturée. Elle consiste à fusionner les régions par groupes en parallèle à chaque itération, créant ainsi une pyramide de graphes d'adjacence. Les fusions de petits groupes de régions sont provoquées plutôt que des fusions de couples de régions. Cette opération doit être parallèle pour tous les sommets d'un niveau, avec des traitements locaux pour chaque sommet. L'apex de la pyramide est atteint lorsqu'aucune réduction supplémentaire n'est possible [14].

1.7 Conclusion

En conclusion, ce chapitre a démystifié la segmentation d'images, mettant en avant les bases du traitement d'images, l'importance du pré-traitement, et les diverses approches de segmentation. La segmentation d'images, en constante évolution, offre un potentiel immense pour explorer et interpréter le monde visuel, ouvrant la voie à des découvertes novatrices dans divers domaines d'application.

À la suite de cette exploration, le prochain chapitre approfondira davantage notre compréhension des méthodes de segmentation, en se penchant spécifiquement sur les contours actifs (Snakes) et les courbes de niveaux (Level Sets) qui sont des méthodes variationnelles ou déformables. Ces approches avancées ouvrent de nouvelles perspectives dans le domaine, offrant des solutions plus dynamiques et précises pour la segmentation d'images complexes.

Chapitre 2

Etat de l'Art sur les Contours Actifs et les Courbes de Niveau

2.1 Introduction

Comme vu au chapitre précédent, dans le contexte de la segmentation d'images, une distinction essentielle est établie entre deux approches principales, à savoir la segmentation par région et la segmentation par contour. Alors que les méthodes basées sur les contours, telles que les méthodes dérivatives, offrent une simplicité d'implémentation, elles sont souvent sensibles au bruit et peuvent présenter des limitations dans la représentation des formes complexes. En contraste avec ces approches conventionnelles de détection de contours, les modèles déformables ou les méthodes variationnelles, notamment les contours actifs ou bien les courbes de niveau, intègrent une connaissance a priori sur la forme des objets à segmenter.

Cette connaissance préalable peut être dérivée d'une variété de sources, notamment des bases d'apprentissage où des formes de référence ou des prototypes sont établis, accompagnés de modes de déformation associés. Alternativement, elle peut consister en des propriétés plus générales telles que la continuité ou la régularité du contour de l'objet, représentant ainsi une forme de connaissance plus abstraite mais néanmoins efficace, et c'est dans ce cas qu'on parlera des contours actifs mais aussi des courbes de niveau [15].

Dans le présent chapitre, nous examinons en détail le principe fondamental des contours actifs "snakes", ainsi que les divers modèles et méthodes qui les sous-tendent, en mettant en lumière les principes et les techniques d'optimisation de l'équation d'énergie associée. Nous étudions également les courbes de niveau, connus sous le nom des "level sets", en exposant les fondements théoriques et les algorithmes pertinents.

2.2 Contours Actifs Paramétriques (Snakes)

Les contours actifs paramétriques, communément désignés sous le terme de "snakes", ont été conceptualisés par Kass, Witkin et Terzopoulos en 1988 [16].

Le principe fondamental des contours actifs réside dans leur capacité à diriger un contour initial vers une position d'équilibre autour de l'objet d'intérêt, en s'alignant avec les contours de l'objet à détecter.

Les contours actifs sont des modèles mathématiques utilisés pour segmenter des objets dans des images. Ils sont basés sur l'idée d'une courbe élastique qui se déforme pour s'adapter à la forme de l'objet. Le modèle de contours actifs se présente sous la forme d'une courbe fermée ou ouverte, à extrémités fixes ou libres. Les contours actifs sont initialement positionnés à proximité du contour souhaité et subissent une évolution contrôlée par un processus itératif de déformation, guidé par un critère de convergence.

La figure 2.1 [17] illustre le principe de convergence des contours actifs.

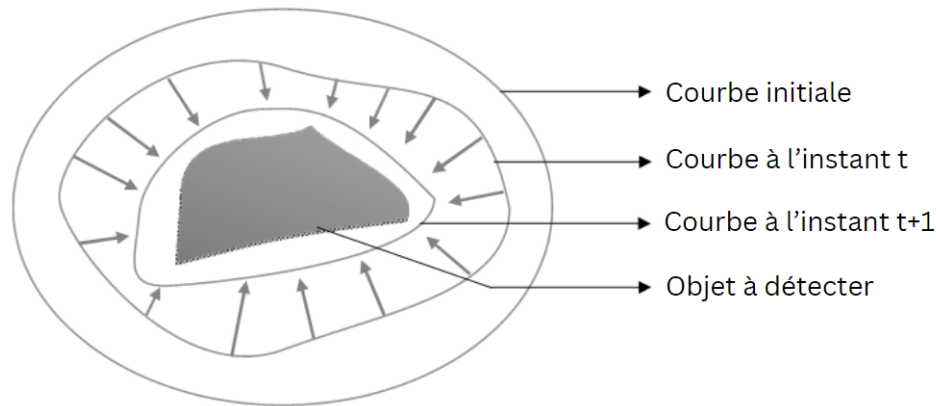


FIG. 2.1 : Principe des contours actifs

Durant chaque itération du processus, les points constituant le contour se déplacent en suivant une équation qui prend en compte les forces agissant sur le contour. Ces forces dépendent à la fois des données extraites de l'image, telles que le gradient et l'intensité, ainsi que des propriétés intrinsèques du contour, comme sa rigidité et son élasticité [18].

La position d'équilibre du contour est déterminée comme étant le minimum d'une fonction d'énergie, dont la dérivée par rapport aux déplacements des points du contour correspond aux forces à appliquer.

2.2.1 Contours Actifs Paramétriques Traditionnels

Le contour actif traditionnel, proposé par Kass [16], est une courbe placée sur l'image et est soumise à l'action de "forces externes" qui la déplacent et la déforment de sa position initiale pour l'adapter au mieux aux bords de l'objet que l'on souhaite segmenter.

Le contour est défini par une représentation paramétrique, il est assimilé à une courbe C représentée selon les notations [17] :

$$C = \{V(s, t) = [x(s, t), y(s, t)]\}, \forall (s, t) \in [0, 1] \times [0, \infty] \quad (2.1)$$

Avec :

- s : abscisse curviligne le long du contour.
- t : évolution temporelle de la courbe dans l'image.
- $V(s, t)$: position d'un point de la courbe C à un instant t .
- (x, y) : coordonnées cartésiennes d'un point de l'image.

Habituellement, dans la détection des contours, après avoir calculé le gradient de l'image, les maxima sont extraits, puis les contours sont reliés entre eux. Ici, nous procédons d'une autre manière ; nous partons d'un modèle de courbe continue et nous essayons de le localiser sur les maxima du gradient. Nous dessinons une courbe simple près des contours prévus et laissons l'action des forces de l'image pousser la courbe sur le reste

du chemin. La position finale correspond à l'équilibre atteint au minimum de l'énergie du modèle [19].

Avant d'examiner le mécanisme de déformation, il est impératif d'approfondir l'analyse des diverses énergies impliquées dans ce processus.

2.2.1.1 Énergies

La fonctionnelle d'énergie attachée au contour actif est composée de deux termes :

$$E_{\text{snake}}(v) \rightarrow \mathbf{E}_{\text{interne}}(v) + \mathbf{E}_{\text{externe}}(v) \quad (2.2)$$

Voici les détails de chacun de ces termes :

- **Énergie interne :**

L'énergie interne d'un contour actif régule l'aspect de la courbe en influençant sa cohérence et sa courbure. Elle assure la cohésion entre les points du contour et maintient la rigidité de sa courbure, ce qui constitue un élément de régularisation dans le processus de segmentation. Les caractéristiques intrinsèques de cette énergie prennent en considération les contraintes géométriques et structurales sur le contour, et elles permettent de contrôler sa géométrie [20].

$$E_{\text{int}}(v) = \frac{1}{2} \int_0^1 \left(\alpha(s) \cdot |v'(s)|^2 + \beta(s) \cdot |v''(s)|^2 \right) ds \quad (2.3)$$

Avec :

-

$$|v'(s)|^2 = \left(\frac{dx(s)}{ds} \right)^2 + \left(\frac{dy(s)}{ds} \right)^2 \quad (2.4)$$

- $v'(s)$: dérivée première de $v(s)$ par rapport à s .

- v'' : dérivée seconde de $v(s)$ par rapport à s .

- α : élasticité du contour.

- β : rigidité du contour.

Le terme du premier ordre, également connu sous le nom de tension, représente la force qui tend à allonger la courbe, prenant une valeur significative lorsque la courbe se distend. Lorsque le paramètre α est nul, ce terme n'intervient pas, ce qui peut entraîner des discontinuités dans la courbe, ce que l'on qualifie alors d'énergie de continuité.

Quant au terme du deuxième ordre, il correspond à la courbure de la courbe C . Sa valeur augmente lorsque la courbe s'incurve rapidement, comme lors de la formation de coins. Lorsque le paramètre β est nul, la courbe peut présenter une forte convexité. En revanche, si β est élevé, la courbe aura tendance à adopter la forme d'un cercle s'il s'agit d'une courbe fermée, ou celle d'une droite si elle est ouverte [18].

• **Énergie externe :**

Selon Kass, Witkin et Terzopoulos [16], l'énergie externe peut comprendre deux termes :

$$E_{\text{ext}}(v) = \mathbf{E}_{\text{image}}(v) + \mathbf{E}_{\text{contrainte}}(v) \quad (2.5)$$

- L'énergie de contrainte est définie par l'utilisateur selon les spécificités du problème. Par exemple, imposer une distance minimale ou maximale entre deux points consécutifs du contour [18]. Dans le cas d'absence de contraintes extérieures, cette énergie pourra être négligeable.
- L'énergie image d'un contour actif attire la courbe C vers les frontières de l'objet recherché. Cette énergie est déterminée par les caractéristiques de l'image, en tenant compte notamment des contours des formes recherchées.

$$E_{\text{image}}(v) = - \int_0^1 \lambda(s) \|\nabla(I(v(s)))\|^2 ds \quad (2.6)$$

Avec :

- * $\nabla(I(v(s)))$ représente le gradient de l'image.
- * $\lambda(s)$: facteur qui dépend de l'image I initiale.

Ainsi, les points de fort gradient ou présentant des propriétés de couleur spécifiques jouent un rôle prépondérant dans cette énergie externe, étant des indicateurs pertinents pour la localisation des contours désirés.

Très souvent, c'est le gradient gaussien qui est utilisé [17] :

$$E_{\text{image}}(v) = - \int_0^1 \|\nabla((G_\sigma * I)(v(s)))\|^2 ds \quad (2.7)$$

Avec :

- * G_σ : fonction gaussienne centrée d'écart type σ .

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp \left\{ -\frac{1}{2} \left[\frac{x^2 + y^2}{\sigma^2} \right] \right\} \quad (2.8)$$

- * σ : paramètre qui permet de définir l'étendue de l'attraction voulue, il peut causer des erreurs de localisation s'il est choisi trop grand.
- * * : opérateur de convolution.
- * ∇ : opérateur de gradient.

2.2.1.2 Minimisation de l'équation d'énergies

L'énergie totale du snake s'écrit donc comme suit :

$$E = \int_0^1 \left[\frac{1}{2} (\alpha |v'(s)|^2 + \beta |v''(s)|^2) + E_{\text{ext}}(v(s)) \right] ds \quad (2.9)$$

Minimiser l'équation 2.9 revient à résoudre l'équation d'Euler Lagrange suivante :

$$\alpha v^{(2)}(s) + \beta v^{(4)}(s) + \nabla E_{\text{ext}} = 0 \quad (2.10)$$

qui se traduit comme un équilibre de forces :

$$F_{int} + F_{ext} = 0 \quad (2.11)$$

Où :

$$\begin{cases} F_{int} = \alpha v^{(2)}(s) + \beta v^{(4)}(s) \\ F_{ext} = \nabla E_{ext} \end{cases} \quad (2.12)$$

Tel que :

- Des forces intérieures qui imposent une certaine régularité. Le coefficient α impose l'élasticité et le coefficient β impose la rigidité de la courbe.
- Une force extérieure – force d'image (terme de potentiel de l'équation de Lagrange) qui pousse la courbe vers les zones qui correspondent aux attributs recherchés. En effet, la courbe est attirée par le minimum local du potentiel, c'est-à-dire les maximas locaux du gradient, donc des contours.

Avec :

- $v^{(2)}$: dérivée seconde de $v(s)$ par rapport à s .
- $v^{(4)}$: 4ème dérivée de $v(s)$ par rapport à s .

$$\nabla E_{ext} = \frac{\partial E_{ext}}{\partial x} + \frac{\partial E_{ext}}{\partial y} \quad (2.13)$$

Le comportement du snake peut être modélisé par une fonction qui dépend du temps $v(s, t)$, ce qui donne la solution de l'équation 2.9:

$$\gamma \frac{\partial v(s, t)}{\partial t} = \alpha v^{(2)}(s, t) + \beta v^{(4)}(s, t) + \nabla E_{ext} \quad (2.14)$$

Où : γ est un coefficient d'amortissement (de viscosité), si γ est grand, le comportement du snake devient lent.

La solution de l'équation, quand v se stabilise, est la solution du problème statique. Donc le terme $\frac{\partial v(s, t)}{\partial t}$ doit être nul.

Pour arriver à la résolution, on doit passer par une discrétisation selon les différences finies, où les dérivées apparaissant dans le problème continu par des différences divisées :

$$\begin{cases} v'_{i,t} = v_{i,t} - v_{i-1,t} \\ v^{(2)}_{i,t} = v_{i+1,t} - 2v_{i,t} + v_{i-1,t} \\ v^{(4)}_{i,t} = v_{i+2,t} - 4v_{i+1,t} + 6v_{i,t} - 4v_{i-1,t} + v_{i-2,t} \\ \frac{\partial v_{i,t}}{\partial t} = v_{i,t} - v_{i,t-1} \end{cases} \quad (2.15)$$

On obtient la version discrète de l'équation 2.14 :

$$\begin{aligned} & \gamma (v_{i,t} - v_{i,t-1}) - \alpha (v_{i+1,t} - 2v_{i,t} + v_{i-1,t}) - \\ & \beta (v_{i+2,t} - 4v_{i+1,t} + 6v_{i,t} - 4v_{i-1,t} + v_{i-2,t}) = \\ & \frac{\partial E_{ext}(v_{i,t-1})}{\partial v} \end{aligned} \quad (2.16)$$

Qui a comme forme matricielle :

$$(K + \gamma I)V_t = \gamma V_{t-1} - \frac{\partial E_{ext}(V_{t-1})}{\partial v} \quad (2.17)$$

Avec :

- Courbe paramétrique à chaque itération t : $V_t = \{v_{i,t} = (x_{i,t}, y_{i,t}), i = 1..n\}$, n représentant le nombre de points dans le contour.
- I : matrice identité $n * n$.
- K : matrice $n * n$, les éléments de K sont déterminés par l'équation 2.16 et les conditions initiales.

Berger [21] présente les différents cas de la matrice K selon le type du contour actif, dans le cas d'un modèle fermé, elle se présente sous la forme suivante :

$$\begin{bmatrix} 2\alpha + 6\beta & -\alpha - 4\beta & \beta & 0 & \dots & 0 & \beta & -\alpha - 4\beta \\ -\alpha - 4\beta & 2\alpha + 6\beta & -\alpha - 4\beta & \beta & \dots & \dots & 0 & \beta \\ \beta & -\alpha - 4\beta & 2\alpha + 6\beta & -\alpha - 4\beta & \dots & \dots & \dots & 0 \\ 0 & \beta & -\alpha - 4\beta & 2\alpha + 6\beta & \dots & \dots & 0 & \dots \\ \dots & \dots & \dots & -\alpha - 4\beta & \dots & \dots & \beta & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & -\alpha - 4\beta & \beta \\ \beta & 0 & \dots & \dots & \dots & -\alpha - 4\beta & 2\alpha + 6\beta & -\alpha - 4\beta \\ -\alpha - 4\beta & \beta & 0 & \dots & 0 & \beta & -\alpha - 4\beta & 2\alpha + 6\beta \end{bmatrix}$$

Cette méthode d'implémentation des contours actifs traditionnels est largement répandue, permettant d'obtenir une solution à chaque itération par inversion de la matrice, accompagnée de l'ajustement du coefficient γ .

Toutefois, deux inconvénients majeurs sont associés à cette approche. Tout d'abord, la position initiale du contour actif doit être relativement proche du contour souhaité dans l'image, sinon le contour actif risque de converger vers des minima locaux au lieu de détecter les contours désirés. Ensuite, cette méthode présente une convergence peu satisfaisante dans les zones présentant une forte concavité ou des angles aigus.

Pour cela des méthodes d'optimisation ont été développées pour faciliter l'implémentation des snakes.

2.2.2 Méthodes d'optimisation

Plusieurs méthodes ont été proposées pour optimiser. Trois principales familles d'approches :

2.2.2.1 Programmation Dynamique

La programmation dynamique représente une approche bien établie pour la résolution de problèmes d'optimisation. Son adaptation aux contours actifs a été initiée par Amini,

Weymouth et Jain [22]. Cette méthode offre une alternative au calcul variationnel (méthode traditionnelle) dans le contexte de la segmentation d'images à l'aide des contours actifs.

On considère pour cela l'équation classique 2.9. La solution d'Euler - Lagrange donne :

$$F_v - \frac{\partial}{\partial s} F_{v_s} + \frac{\partial^2}{\partial s^2} F_{v_{ss}} = 0 \quad (2.18)$$

Où $F(s, v_s, v_{ss})$ représente la fonction à intégrer.

En discrétisant avec :

$$E_{\text{int}}(v_i) = (\alpha_i |v_i - v_{i1}|^2 + \beta_i |v_{i+1} - 2v_i + v_{i-1}|^2) / 2 \quad (2.19)$$

On obtient :

$$E_{\text{tot}} = \sum_{i=0}^{n-1} E_{\text{int}}(v_i) + E_{\text{ext}}(v_i) \quad (2.20)$$

La programmation dynamique peut être vue comme une méthode de prise de décision multi-niveaux pour minimiser l'énergie d'un contour. En partant d'un point initial du contour, on doit prendre des décisions pour chaque ensemble fini d'étapes (i_0, i_1, \dots, i_{n-1}) en choisissant parmi un ensemble fini de solutions potentielles [23].

L'énergie interne du contour est composée d'un terme de premier ordre et d'un terme de second ordre. Après la discrétisation, l'énergie interne implique un élément du contour, ainsi que son prédécesseur et son successeur dans le processus discret de résolution :

$$\begin{aligned} E_{\text{total}}(v_1, v_2, \dots, v_n) &= E_1(v_1, v_2, v_3) + E_2(v_2, v_3, v_4) + \dots + E_{n-2}(v_{n-2}, v_{n-1}, v_n) \\ \text{Où } E_{i-1}(v_{i-1}, v_i, v_{i+1}) &= E_{\text{ext}}(v_i) + E_{\text{int}}(v_{i-1}, v_i, v_{i+1}) \end{aligned} \quad (2.21)$$

L'expression ci-dessus est donc un problème d'optimisation d'une fonction numérique de plusieurs variables. Les variables étant les positions des différents points de la courbe.

La formulation standard sous forme récurrente de la programmation dynamique peut s'écrire :

$$S_i(v_{i+1}, v_i) = \min \{ S_{i-1}(v_i, v_{i-1}) + \alpha |v_i, v_{i-1}|^2 + \beta |v_{i+1} - 2v_i + v_{i-1}|^2 + E_{\text{ext}}(v_i) \} \quad (2.22)$$

Chaque itération de cette formulation produit un contour optimal. Bien que la convergence de la minimisation de l'énergie soit assurée, la complexité de ce processus demeure élevée. Pour un voisinage de taille m et un contour composé de n points, la complexité est de l'ordre de $O(nm^3)$ et la taille de la mémoire requise est de l'ordre de $O(nm^2)$. Cependant, il convient de noter que cette procédure est parallélisable. Par ailleurs, cette implémentation permet d'introduire diverses contraintes dans le processus de segmentation [18].

2.2.2.2 Algorithme de Greedy

Proposé par WILLIAMS et SHAH [24], l'algorithme de Greedy est une alternative fréquente à l'approche variationnelle. L'équation 2.9 est également discrétisée par différences

finies comme suit :

$$\begin{cases} E_{\text{Continuité}} : & \left\| \frac{dv_i}{ds} \right\|^2 = \|v_i - v_{i-1}\|^2 = (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ E_{\text{Courbure}} : & \left\| \frac{d^2v_i}{ds^2} \right\|^2 = \|v_{i-1} - 2v_i + v_{i+1}\|^2 = (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2 \end{cases}$$

Il est à noter que ces expressions supposent deux hypothèses. Premièrement, les points sont placés le long de la courbe à une distance unitaire les uns des autres. Deuxièmement, il est supposé que le paramètre s soit une longueur d'arc, permettant ainsi d'exprimer la courbure comme $|v''(s)|^2$.

L'algorithme de Greedy, similaire à la programmation dynamique, permet l'introduction d'autres contraintes. Sa vitesse d'exécution est supérieure à celle de la méthode précédente, s'exécutant en $O(nm)$ par itération au lieu de $O(nm^3)$ pour un contour de n points et un voisinage de taille m [18]. L'expression à minimiser, sans l'ajout de contraintes spécifiques, est :

$$E_{\text{tot}} = \int \alpha \cdot E_{\text{continuité}} + \beta \cdot E_{\text{courbure}} + \gamma \cdot E_{\text{image}}$$

L'algorithme est itératif. À chaque itération, le voisinage de chaque point est examiné, et le point du voisinage qui minimise l'énergie totale est sélectionné.

Après la discrétisation, la minimisation de la distance entre les points conduit naturellement à une rétraction du contour. Williams et Shah ont proposé une variante visant à atténuer cette rétraction excessive. Ils ont introduit l'utilisation de la différence entre la distance entre deux points $\|v_i - v_{i-1}\|$, et la distance moyenne entre les points du contour \bar{d} .

$$E_{\text{Continuité}} = \bar{d} - \|v_i - v_{i-1}\|$$

Avant la minimisation, les différents termes de l'expression sont normalisés.

Une adaptation visant à accélérer davantage l'algorithme a été proposée par Lam et Yan [25] (algorithme Fast Greedy). Elle consiste, par exemple, à limiter l'examen à quatre voisins parmi les huit disponibles pour un voisinage de 3×3 pixels. Si l'un de ces quatre voisins améliore l'énergie totale, il n'est alors pas nécessaire d'explorer les autres. Dans le cas contraire, les quatre voisins restants sont examinés. Cette approche augmente le nombre d'itérations nécessaires pour atteindre la convergence, mais réduit le temps de calcul de chaque itération [23].

2.2.2.3 Gradient Vector Flow (GVF)

Dans une démarche visant à améliorer la méthode de base, également connue sous le nom de snake classique, du point de vue de l'initialisation et de la convergence dans les concavités C.Xu et J.L Prince ont proposé la méthode du GVF snake [26]. Dans cette méthode, une nouvelle force extérieure appelée Gradient Vector Flow (GVF) a été introduite :

$$F_{\text{ext}} = \nabla(x, y) = [u(x, y), v(x, y)] \quad (2.23)$$

En comparaison avec les contours actifs traditionnels, le terme $\frac{\partial E_{\text{ext}}(V_{t-1})}{\partial v}$ est remplacé par les composantes du GVF.

On procède en deux étapes pour le calcul du champ GVF [17] :

• **Calcul de la carte du contour (Edge Map) :**

- La carte du contour $f(x, y)$ d'une image $I(x, y)$, représente la dérivée de l'image. Elle prend souvent la forme :

$$f(x, y) = |\nabla (G_\sigma(x, y) * I(x, y))|^2 \quad (2.24)$$

- Le gradient de la carte de contours ∇f a des vecteurs qui pointent vers les contours et sont normaux aux contours. Ces vecteurs ont généralement de forts modules seulement à proximité immédiate des contours. Dans les régions homogènes, ou $I(x, y)$ est presque constant, le gradient de la carte du contour est presque zéro.
- Grâce à cette première propriété, un snake initialisé près du contour converge vers une configuration stable à proximité de celui-ci. L'intérêt du GVF snake est de préserver cette propriété tout en étendant le champ de gradient plus loin des bords. Dans les régions homogènes, un processus de diffusion est utilisé pour générer des vecteurs qui convergent également vers les régions concaves.

• **Calcul de flux du vecteur gradient :**

- On définit le flux du vecteur gradient comme le champ de vecteur : $\nabla(x, y) = (u(x, y), v(x, y))$ qui minimise la fonctionnelle d'énergie suivante :

$$\varepsilon = \iint (\mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla f|^2 \cdot |V - \nabla f|^2) dx dy \quad (2.25)$$

Avec :

- * $u_x = \frac{\partial u}{\partial x}, u_y = \frac{\partial u}{\partial y}, v_x = \frac{\partial v}{\partial x}, v_y = \frac{\partial v}{\partial y}$
- * μ un paramètre de régularisation entre le premier et le second terme de l'intégrale.
- * pour les petites valeurs de ∇f , l'énergie est déterminée par la première partie de l'intégrale « La somme des carrés des dérivées partielles du vecteur champ », le résultat est un champ qui évolue très lentement. D'autre part, pour les grandes valeurs de ∇f , le second terme domine l'intégrale, il est minimisé lorsque $\nabla = \nabla f$, i.e. pour les grandes valeurs de ∇f , ∇ doit être au voisinage de ∇f , donc le champ varie lentement dans les régions homogènes [3].
- Le GVF peut être calculé en résolvant les équations d'Euler - Lagrange ci dessous :

$$\begin{cases} \mu \nabla^2 u - (u - f_x) (f_x^2 + f_y^2) = 0 \\ \mu \nabla^2 v - (v - f_y) (f_x^2 + f_y^2) = 0 \end{cases} \quad (2.26)$$

f_x et f_y sont les dérivées de f par rapport à x et y respectivement.

- On considère u et v comme des équations temporelles, étant donné que le contour varie en fonction du temps t , on obtient les équations de diffusion généralisées :

$$\begin{cases} u_t(x, y, t) \\ = \mu \nabla^2 u(x, y, t) - [u(x, y, t) - f_x(x, y)] * [f_x(x, y)^2 + f_y(x, y)^2] \\ v_t(x, y, t) \\ = \mu \nabla^2 v(x, y, t) - [v(x, y, t) - f_y(x, y)] * [f_x(x, y)^2 + f_y(x, y)^2] \end{cases} \quad (2.27)$$

Aussi écrites :

$$\begin{cases} u_t(x, y, t) = \mu \nabla^2 u(x, y, t) - b(x, y)u(x, y, t) + c_1(x, y) \\ v_t(x, y, t) = \mu \nabla^2 v(x, y, t) - b(x, y)v(x, y, t) + c_2(x, y) \end{cases} \quad (2.28)$$

Avec :

$$\begin{cases} b(x, y) = f_x(x, y)^2 + f_y(x, y)^2 \\ c_1(x, y) = b(x, y)f_x(x, y) \\ c_2(x, y) = b(x, y)f_y(x, y) \end{cases} \quad (2.29)$$

- **Numérisation :**

Les équations 2.26 ont été numérisées par Xu et Prince, en remplaçant x, y, t par les indices i, j, n respectivement :

$$\begin{aligned} u_t &= \frac{1}{\Delta t} (u_{i,j}^{n+1} - u_{i,j}^n) \quad \text{Et } v_t = \frac{1}{\Delta t} (v_{i,j}^{n+1} - v_{i,j}^n) \\ \nabla^2 u &= \frac{1}{\Delta x \Delta y} (u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4u_{i,j}) \\ \nabla^2 v &= \frac{1}{\Delta x \Delta y} (v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4v_{i,j}) \end{aligned}$$

A partir de ces dérivées partielles, on calcule les équations 2.27 comme suit :

$$\begin{aligned} u_{i,j}^{n+1} &= (1 - b_{i,j}\Delta t) u_{i,j}^n + r (u_{i+1,j}^n + u_{i,j+1}^n + u_{i-1,j}^n + u_{i,j-1}^n - 4u_{i,j}^n) + c_{i,j}^1 \Delta t \\ v_{i,j}^{n+1} &= (1 - b_{i,j}\Delta t) v_{i,j}^n + r (v_{i+1,j}^n + v_{i,j+1}^n + v_{i-1,j}^n + v_{i,j-1}^n - 4v_{i,j}^n) + c_{i,j}^2 \Delta t \end{aligned} \quad (2.30)$$

Les équations 2.30 sont stables pour $r = \frac{\mu \Delta t}{\Delta x \Delta y} \leq \frac{1}{4}$.

2.3 Courbes de niveau (Level sets)

La méthode des courbes de niveau est une technique numérique utilisée pour le suivi des interfaces et la modélisation des formes, principalement développée à partir du domaine de la propagation d'interface. Elle a été proposée pour la première fois en 1988 par Osher et Sethian [27]. Conçue initialement pour suivre les changements de forme des flammes selon l'équation thermodynamique, cette méthode repose sur l'idée de représenter implicitement une courbe bidimensionnelle comme l'ensemble de niveau zéro d'une surface fonctionnelle tridimensionnelle continue. En actualisant constamment la fonction des ensembles de niveau, la courbe peut évoluer de manière continue. Cette méthode résout essentiellement une équation aux dérivées partielles, et son principe théorique fondamental repose sur le concept de fonction implicite.

2.3.1 Principe de base

La méthode de segmentation par les courbes de niveau utilise une courbe paramétrique fermée $C(t)$ selon une équation d'ensembles de niveau [28] de la forme :

$$\frac{\partial C}{\partial t} = F \cdot N \quad (2.31)$$

Dans cette équation :

- t : temps.
- F : vitesse d'évolution.
- N : normale unitaire à la courbe.

Ainsi, chaque point de la courbe C évolue dans la direction normale à la courbe avec une vitesse F .

L'avantage principal de cette méthode est sa capacité à gérer automatiquement les changements de topologie pendant l'évolution de la courbe. Cela signifie que la courbe peut être divisée en deux ou trois courbes distinctes, ou inversement, plusieurs courbes peuvent fusionner pour devenir une seule (voir figure 2.2 [20]).

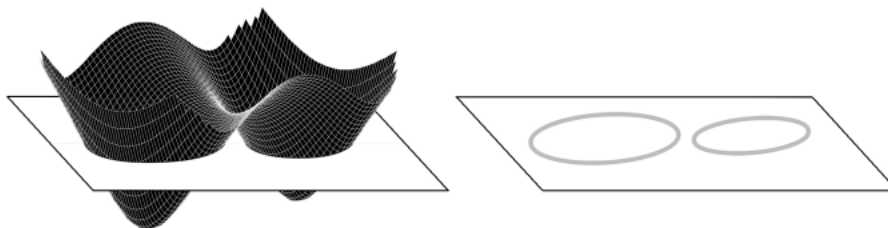


FIG. 2.2 : Illustration de la gestion des changements de topologie de l'implémentation en courbes de niveau : la fonction ϕ se déforme et, bien qu'elle ne change pas de topologie directement, le sous-ensemble observé (son intersection avec le plan de niveau zéro) change de topologie en donnant deux contours distincts

Il existe deux approches principales pour cette méthode :

- La première consiste à réinitialiser la fonction de distance à chaque n itérations.
- La deuxième fonctionne sans réinitialisation de cette dernière.

2.3.2 Méthode des courbes de niveau traditionnelle

Comme précédemment présenté, la méthode des courbes de niveau permet l'évolution d'une courbe paramétrique fermée $C(t)$ selon l'équation 2.31, également connue sous le nom d'équation d'ensembles de niveau. Chaque point de la courbe C évolue selon la direction normale à la courbe avec une vitesse F .

La fonction généralement utilisée pour représenter $C(t)$ est la distance algébrique à la courbe C . Ainsi, pour une courbe paramétrique plane, la fonction utilisée pour la représentation et l'évolution sera un plan $\phi(x, y)$ à valeurs réelles. L'évolution de la courbe implique la mise à jour de l'ensemble du tableau $\phi(x, y)$.

2.3.2.1 Initialisation de la fonction de distance

Initialement, les pixels correspondant au passage de la courbe C sont initialisés à zéro dans la matrice $\phi(x, y)$. Autrement dit, le long de la courbe C , la valeur de $\phi(x, y)$ est zéro [29], tandis que pour le reste des pixels, elle représente la distance euclidienne au point le plus proche de la courbe défini comme suit :

$$\phi_0(x, y) = \begin{cases} +d & \text{pour } (x, y) \in \Omega \\ 0 & \text{pour } (x, y) \in \partial\Omega = C(t) \\ -d & \text{pour } (x, y) \in \bar{\Omega} \end{cases} \quad (2.32)$$

Où :

- d : distance euclidienne.
- Ω : domaine de l'image I .

La figure 2.3 illustre l'utilisation de la distance signée $\phi(x, y)$.

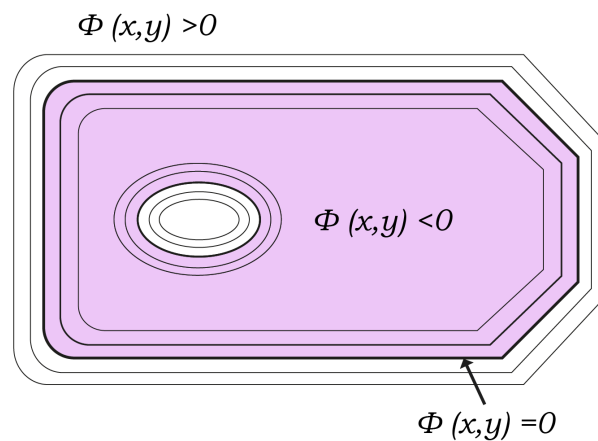


FIG. 2.3 : Utilisation de la fonction $\phi(x, y)$

2.3.2.2 Evolution de la courbe

Nous allons explorer les équations fondamentales régissant le mouvement des courbes de niveau, ainsi que la construction et la réinitialisation de la fonction de distance.

A) Équations fondamentales

Pour une courbe plane C , un tableau bidimensionnel de nombres réels $\phi(x, y)$ est utilisé. L'évolution de la fonction $\phi(x, y)$ suit l'équation des courbes de niveau :

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\partial C}{\partial t} = -\nabla \phi \cdot F \cdot \vec{N} = F \cdot |\nabla \phi| \quad (2.33)$$

Où : F est une fonction scalaire de vitesse qui dépend des propriétés locales de la courbe, des paramètres externes liés au gradient d'image, et des termes additionnels de propagation de la courbe.

Une approximation numérique convexe du premier ordre peut être utilisée pour estimer les dérivées spatiales de l'équation, donnant ainsi la fonction de vitesse F comme :

$$F = F_{\text{prop}} + F_{\text{curv}} + F_{\text{adv}} \quad (2.34)$$

- $F_{\text{prop}} = F_0$: vitesse de propagation de la courbe.
- $F_{\text{adv}} = \vec{U}(x, y, t) \cdot \vec{n}$: vitesse d'advection.
- $F_{\text{curv}} = -\epsilon k$: vitesse liée aux propriétés locales de la courbe.

Avec :

- $\vec{n} = \frac{\nabla \phi}{|\nabla \phi|}$: normale unitaire à la courbe.
- $\vec{U} = (u, v)$, avec u, v les dérivées spatiales de $g(|\nabla I|)$.
- k : courbure.

On peut exprimer l'équation des courbes de niveau de la manière suivante :

$$\frac{\partial \phi}{\partial t} + F_0 \cdot |\nabla \phi| + \vec{U}(x, y, t) \cdot \frac{\nabla \phi}{|\nabla \phi|} \cdot |\nabla \phi| = \epsilon \cdot k \cdot |\nabla \phi| \quad (2.35)$$

On obtient alors la relation suivante :

$$\frac{\partial \phi}{\partial t} + F_0 \cdot |\nabla \phi| + \vec{U}(x, y, t) \cdot \nabla \phi = \epsilon \cdot k \cdot |\nabla \phi| \quad (2.36)$$

- Le terme $F_0 |\nabla \phi|$ dans l'équation 2.35 décrit le mouvement des niveaux de contours dans la direction normale à la courbe. Il est influencé par le terme $g(|\nabla I|)$ [27] [30], qui représente une fonction décroissante de l'amplitude du gradient de l'image I . Cette relation peut être formulée comme suit :

$$g(|\nabla I|) = \frac{1}{1 + |\nabla G_\sigma * I|^p} \quad (2.37)$$

la fonction de l'équation 2.36 établit la condition d'arrêt de la fonction de distance. Elle est basée sur un opérateur dérivatif, tel que la norme du gradient, qui détecte les variations brusques d'amplitude dans l'image. Cette fonction décroît lorsque l'opérateur augmente et tend théoriquement vers zéro à mesure qu'il atteint l'infini. Où :

- $I(x, y)$: image.
- G : noyau Gaussien de variance σ .

Avec :

- $p = 1$ ou $p = 2$.

On peut exprimer le premier terme $F_0|\nabla\phi|$ sous cette forme :

$$F_0|\nabla\phi| = \alpha \left[\max(F_{0ij})\nabla^+ + \min(F_{0ij})\nabla^- \right] \quad (2.38)$$

Avec :

- α : coefficient de pondération.
 - * Si le contour initial se trouve à l'extérieur de l'objet d'intérêt, le coefficient α doit être positif.
 - * Si le contour initial se trouve à l'intérieur de l'objet d'intérêt, le coefficient α doit être négatif.
- F_{0ij} : terme qui dépend de $g(|\nabla I|)$.

– On peut exprimer ∇^+ sous la forme suivante :

$$\nabla^+ = \left[\max\left(D_{ij}^{-x}\phi, 0\right)^2 + \max\left(D_{ij}^{+x}\phi, 0\right)^2 + \min\left(D_{ij}^{-y}\phi, 0\right)^2 + \min\left(D_{ij}^{+y}\phi, 0\right)^2 \right] \quad (2.39)$$

– On peut exprimer ∇^- sous la forme suivante :

$$\nabla^- = \left[\max\left(D_{ij}^{+x}\phi, 0\right)^2 + \min\left(D_{ij}^{-x}\phi, 0\right)^2 + \max\left(D_{ij}^{+y}\phi, 0\right)^2 + \min\left(D_{ij}^{-y}\phi, 0\right)^2 \right] \quad (2.40)$$

Pour notre cas en 2D, nous utilisons l'implémentation numérique des dérivées en utilisant la méthode des différences finies du premier ordre [31].

- * D_{ij}^{+x}, D_{ij}^{+y} : Forward Differences (FD) de ϕ selon x et y , et sont définies comme suit :

$$D_{ij}^{+x} = D_{ij}^{+x}\phi = \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \quad (2.41)$$

$$D_{ij}^{+y} = D_{ij}^{+y}\phi = \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \quad (2.42)$$

- * D_{ij}^{-x}, D_{ij}^{-y} : Backward Differences (BD) de ϕ selon x et y , et sont définies comme suit :

$$D_{ij}^{-x} = D_{ij}^{-x}\phi = \frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x} \quad (2.43)$$

$$D_{ij}^{-y} = D_{ij}^{-y}\phi = \frac{\phi_{i,j} - \phi_{i,j-1}}{\Delta y} \quad (2.44)$$

Avec :

- $\phi_{i,j}$: valeur du tableau à la position $x = i$ et $y = j$.
- $\Delta x = \Delta y = 1$.

- Le terme $F_{\text{adv}}|\nabla\phi| = \vec{U}(x, y, t) \cdot \nabla\phi$ dans l'équation 2.35 correspond à la vitesse d'advection, il peut être exprimé sous la forme suivante :

$$F_{\text{adv}}|\nabla\phi| = \left[\max(u_{ij}, 0) D_{ij}^{-x} + \min(u_{ij}, 0) D_{ij}^{+x} + \max(v_{ij}, 0) D_{ij}^{-y} + \min(v_{ij}, 0) D_{ij}^{+y} \right] \quad (2.45)$$

- Le terme $F_{\text{curv}}|\nabla\phi| = -\epsilon k |\nabla\phi|$ dans l'équation 2.35 correspond à la vitesse liée aux propriétés locales de la courbe. il est donné par la relation suivante [3] :

$$F_{\text{curv}}|\nabla\phi| = -\epsilon k \left[\left(D_{ij}^{0x} \right)^2 + \left(D_{ij}^{0y} \right)^2 \right]^{1/2} \quad (2.46)$$

Où :

- K est calculée à partir de ϕ en utilisant des différences finies centrales [32] :

$$K = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|} \right) = \frac{\phi_{xx}\phi_y^2 - 2\phi_{xy}\phi_x\phi_y + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}} \quad (2.47)$$

Tel que :

- * ϕ_x : dérivée première de ϕ selon la direction de x .
- * ϕ_y : dérivée première de ϕ selon la direction de y .
- * ϕ_{xx} : dérivée seconde de ϕ selon la direction de x .
- * ϕ_{yy} : dérivée seconde de ϕ selon la direction de y .
- * ϕ_{xy} : dérivée seconde de ϕ selon la direction x puis y .
- D_{ij}^{0x} et D_{ij}^{0y} sont les Centered Differences (CD), elles sont définies comme suit :

$$D_{ij}^{0x} = D_{ij}^{0x}\phi = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \quad (2.48)$$

$$D_{ij}^{0y} = D_{ij}^{0y}\phi = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} \quad (2.49)$$

Et le terme $F_{\text{curv}}|\nabla\phi|$ devient :

$$F_{\text{curv}}|\nabla\phi| = -\epsilon \left[\frac{\phi_{xx}\phi_y^2 - 2\phi_{xy}\phi_x\phi_y + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}} \right] \left[\left(D_{ij}^{0x} \right)^2 + \left(D_{ij}^{0y} \right)^2 \right]^{1/2} \quad (2.50)$$

Pour faire évoluer la fonction de ϕ , nous utilisons la dérivée temporelle $\frac{\partial\phi}{\partial t}$:

$$\frac{\partial\phi}{\partial t} = \frac{\phi_{ij}^{k+1} - \phi_{ij}^k}{\Delta t} \quad (2.51)$$

Avec :

- * Δt : pas relatif du temps.
- * ϕ_{ij}^{k+1} : fonction de distance à l'itération $k + 1$.
- * ϕ_{ij}^k : fonction de distance à l'itération k .

Donc la fonction ϕ_{ij}^{k+1} s'écrit comme suit :

$$\begin{aligned} \phi_{ij}^{k+1} = & \phi_{ij}^k + \Delta t \left[-\alpha \left[\max(F_{0ij}) \nabla^+ + \min(F_{0ij}) \nabla^- \right] - \left[\max(u_{ij}, 0) D_{ij}^{-x} + \min(u_{ij}, 0) D_{ij}^{+x} \right. \right. \\ & \left. \left. + \max(v_{ij}, 0) D_{ij}^{-y} + \min(v_{ij}, 0) D_{ij}^{+y} \right] + \epsilon \left[\frac{\phi_{xx} \phi_y^2 - 2\phi_{xy} \phi_x \phi_y + \phi_{yy} \phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}} \right] \left[(D_{ij}^{0x})^2 + (D_{ij}^{0y})^2 \right]^{1/2} \right] \end{aligned} \quad (2.52)$$

B) Construction de la fonction de distance

La construction de la fonction de distance, ϕ , est cruciale dans la méthode des courbes de niveau. Cette fonction doit satisfaire la condition $|\nabla\phi| = 1$ tout en préservant sa configuration en tant que fonction de distance signée du front $C(t)$ tout au long de son évolution. Si le front initial C est un simple point, ϕ est construite de manière à ce que tous ses niveaux soient équidistants de C .

Lorsque la fonction de vitesse F est constante uniforme qui prend la valeur 1, le point initial augmenterait indéfiniment à chaque itération. Cette caractéristique est fondamentale dans la méthode des courbes de niveau. Ensuite, la fonction de vitesse est utilisée pour faire évoluer ϕ vers les contours souhaités [32].

Trois solutions sont envisagées pour satisfaire la condition $|\nabla\phi| = 1$ pendant l'évolution de la courbe :

1. La première solution implique d'interrompre périodiquement les calculs et de reconstruire ϕ selon la position actuelle de $C(t)$, tout en maintenant l'équidistance entre ses niveaux. Cependant, cette méthode est complexe et chronophage.
2. La deuxième solution propose d'évoluer une fonction de vitesse prolongée F_{ext} égale à F ainsi que ϕ . F_{ext} doit également satisfaire la condition spécifique suivante :

$$F_{\text{ext}} \nabla\phi = 0 \quad (2.53)$$

pour être une fonction de distance signée.

3. La troisième solution utilise la méthode Fast Marching [33] pour calculer ϕ en résolvant l'équation d'Eikonal particulière suivante :

$$|\nabla T| = 1 \quad (2.54)$$

- Pour chaque partie de l'interface [33], la condition de frontière $T = 0$ doit être vérifiée sur l'ensemble de niveau zéro de la fonction ϕ .
- La solution T sera ensuite notre fonction de distance signée temporaire ϕ^{temp} .

C) Réinitialisation de la fonction de distance [3]

La réinitialisation de la fonction de distance est abordée pour éviter les chevauchements de ses niveaux (voir la figure 2.4) pendant son évolution.

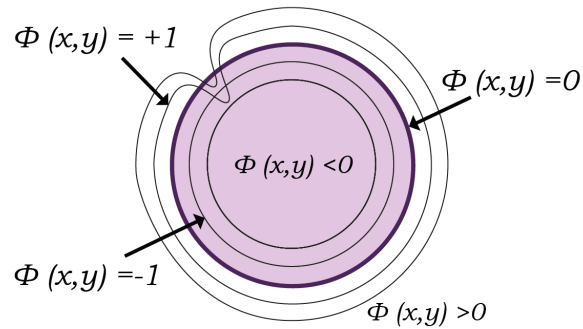


FIG. 2.4 : Chevauchements des niveaux de la fonction de distance

Cette réinitialisation est réalisée chaque "n" itérations pour maintenir la stabilité de la courbe tout en préservant l'équidistance entre ses niveaux ($\nabla\phi = 1$) (figure 2.5). Elle est effectuée en résolvant l'équation spécifique suivante :

$$\frac{\partial\phi}{\partial t} = \text{sign}(\phi_0)(1 - |\nabla\phi|) \quad (2.55)$$

qui donne de nouvelles valeurs à ϕ tout en respectant la condition de front particulière. Où :

- ϕ_0 : fonction à réinitialiser.
- $\text{sign}(\phi_0)$: signe de la fonction.

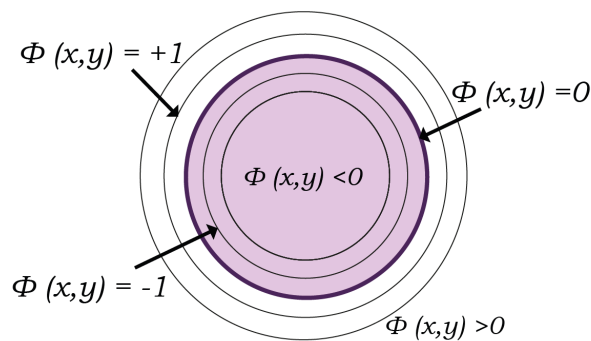


FIG. 2.5 : Fonction ϕ après la réinitialisation

2.3.3 Formulation variationnelle des courbes de niveau sans la réinitialisation de la fonction de distance

Pour garantir la stabilité de la courbe tout au long de son évolution, en particulier dans le voisinage du niveau zéro [34], il est crucial que la fonction ϕ satisfasse la condition :

$$|\nabla\phi| = 1$$

Pour éliminer le besoin de réinitialisation Chunming [34] a ainsi proposé l'intégrale suivante :

$$P(\phi) = \int_{\Omega} \frac{1}{2} (|\nabla\phi| - 1)^2 dx dy \quad (2.56)$$

L'énergie $P(\phi)$ représente l'énergie interne de ϕ , dépendant uniquement de ϕ , et agit comme une métrique pour évaluer le degré de convergence de ϕ vers la fonction de distance signée dans un domaine $\Omega \subset \mathbb{R}^2$ [34].

À partir de $P(\phi)$, la nouvelle formule d'énergie peut être exprimée comme suit :

$$\zeta(\phi) = \mu P(\phi) + \xi_{g,\lambda,\nu}(\phi) \quad (2.57)$$

Ici,

- μ représente le paramètre positif contrôlant l'écart de ϕ par rapport à la fonction de distance signée.
- L'énergie externe $\xi_{g,\lambda,\nu}(\phi)$, dépendant des données de l'image (comme F_{prop} dans le cas des level sets traditionnels), pousse l'ensemble de niveau zéro vers les objets désirés. Elle est définie comme suit :

$$\xi_{g,\lambda,\nu}(\phi) = \lambda L_g(\phi) + \nu A_g(\phi) \quad (2.58)$$

Où :

- λ : constante positive.
- ν : coefficient qui dépend de la position relative du contour initial par rapport à l'objet désiré.
 - si le contour initial est placé à l'extérieur de l'objet d'intérêt, ν devrait être positif, permettant ainsi au contour de se rétrécir plus rapidement.
 - En revanche, si le contour initial est situé à l'intérieur des contours de l'objet d'intérêt, ν devrait être négatif pour accélérer l'expansion du contour.

les termes $L_g(\phi)$ et $A_g(\phi)$ sont définis comme suit :

$$L_g(\phi) = \int_{\Omega} g \delta_{\epsilon} |\nabla\phi| dx dy \quad (2.59)$$

$$A_g(\phi) = - \int_{\Omega} gH(-\phi) dx dy \quad (2.60)$$

- La fonction d'énergie $L_g(\phi)$ calcule la longueur de la courbe de niveau zéro de la fonction ϕ selon la métrique euclidienne, définie par $ds = |C'(q)|dq$, où $\mathbf{C}(q) : [0, 1] \rightarrow \mathbb{R}^2$ est une courbe plane paramétrée [35]. Cette solution évalue le mouvement de chaque point du contour actif afin de minimiser la longueur de L_g .
- D'autre part, la fonction d'énergie $A_g(\phi)$ dans l'équation 2.58 est introduite pour accélérer l'évolution de la courbe.

Où :

- $H(-\phi)$: fonction heaviside.
- δ_{ϵ} : fonction dirac, définie comme suit :

$$\delta_{\epsilon}(x) = \begin{cases} 0 & |x| > \epsilon \\ \frac{1}{2\epsilon} [1 + \cos(\frac{\pi x}{\epsilon})] & |x| \leq \epsilon \end{cases} \quad (2.61)$$

Avec :

- ϵ : paramètre de régularisation.

Pour minimiser la fonction $\zeta(\phi)$ de l'équation 2.57, on utilise le gradient suivant :

$$\frac{\partial \phi}{\partial t} = - \frac{\partial \zeta}{\partial \phi} = - \frac{\partial (\mu P(\phi) + \xi_{g,\lambda,\nu}(\phi))}{\partial \phi} \quad (2.62)$$

En utilisant les calculs variationnels, la première dérivée de la fonction ζ peut être exprimée de la manière suivante :

$$\frac{\partial \zeta}{\partial \phi} = -\mu \left[\Delta \phi - \operatorname{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) \right] - \lambda \delta_{\epsilon}(\phi) \operatorname{div} \left(g \frac{\nabla \phi}{|\nabla \phi|} \right) - \nu g \delta_{\epsilon}(\phi) \quad (2.63)$$

Δ : opérateur Laplacien.

Il convient de noter que la dérivée temporelle $\frac{\partial \phi}{\partial t}$ est approximée par les FD comme il est mentionné dans l'équation 2.29. Donc l'équation d'évolution 2.11 peut être réécrite sous la forme suivante :

$$\frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta t} = \mu \left[\Delta \phi - \operatorname{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) \right] + \lambda \delta_{\epsilon}(\phi) \operatorname{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) + \nu g \delta_{\epsilon}(\phi) \quad (2.64)$$

Le terme $\left\{ \Delta \phi - \operatorname{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) \right\}$ de l'équation 2.64 correspond au "gradient flow" de la fonction d'énergie interne $\mu P(\phi)$. Son effet est de maintenir la stabilité de $|\nabla \phi| = 1$ tout au long de l'évolution de la fonction des courbes de niveau.

D'autre part, le terme $\left\{ \lambda \delta_{\epsilon}(\phi) \operatorname{div} \left(\frac{\nabla \phi}{|\nabla \phi|} \right) + \nu g \delta_{\epsilon}(\phi) \right\}$ de l'équation 2.64 correspond aux "gradient flows" des fonctions d'énergie $\lambda L_g(\phi)$ et $\nu A_g(\phi)$.

2.3.3.1 Sélection du pas relatif au temps Δt

Dans cette méthode, le pas du temps relatif peut être choisi de manière plus importante par rapport aux méthodes traditionnelles de courbes de niveau [3].

2.3.3.2 Initialisation flexible de la fonction des courbes de niveau

Dans le contexte de l'initialisation flexible de la fonction des courbes de niveau, Chunming [34] a proposé la formule suivante pour initialiser la fonction ϕ comme une fonction de distance signée :

$$\phi_0(x, y) = \begin{cases} -\rho & \text{si } (x, y) \in \Omega_0 - \partial\Omega_0 \\ 0 & \text{si } (x, y) \in \partial\Omega_0 \\ \rho & \text{si } (x, y) \in \Omega - \Omega_0 \end{cases} \quad (2.65)$$

Ici,

- Ω : domaine de l'image I .
- Ω_0 : sous-domaine de l'image I .
- $\partial\Omega_0$: points de contours Ω_0 .
- ρ : constante positive.

2.3.4 Classification des modèles de courbes de niveau

Selon l'article de référence [36], les méthodes des courbes de niveau existantes pour la segmentation d'images sont généralement classées en deux grandes catégories : les modèles basés sur les contours et les modèles basés sur les régions.

2.3.4.1 Modèles de courbes de niveau basés sur les contours

Ces modèles utilisent des informations de contour pour la segmentation d'images. Contrairement aux modèles basés sur les régions, ils ne supposent pas d'homogénéité d'intensité, ce qui les rend applicables aux images avec des inhomogénéités d'intensité. Des améliorations telles que le modèle **Distance Regularized Level Set Evolution "DRLSE"** ont été proposées par Li et al [37] pour résoudre ces problèmes.

2.3.4.2 Modèles de courbes de niveau basés sur les régions

Ces modèles visent à identifier chaque région d'intérêt en utilisant un descripteur de région spécifique pour guider le mouvement du contour actif. Cependant, définir un descripteur de région pour les images avec des **inhomogénéités d'intensité** est très difficile. Le modèle **Local image fitting "LIF"** proposé par Zhang et al [38] et **Level Set Evolution and Bias Field Estimation "LSE-BFE"** [39] offrent des alternatives basées sur les régions.

2.3.5 Algorithmes des modèles de courbes de niveau

Ce qui suit est une brève introduction à trois types de modèles de courbes de niveau, à savoir le modèle DRLSE, le modèle LIF et le modèle LSE-BFE.

2.3.5.1 Modèle d'évolution des courbes de niveau régularisé en fonction de la distance (DRLSE)

Le modèle DRLSE [37] introduit un terme de régularisation de distance pour stabiliser la fonction de contour niveau au cours du processus itératif. Cela permet d'améliorer la vitesse de segmentation et d'éviter la nécessité de réinitialiser la fonction de contour niveau.

$$\frac{\partial \phi}{\partial t} = \mu \operatorname{div}(d_p(|\nabla \phi|)\nabla \phi) - \frac{\partial \epsilon_{ext}}{\partial \phi} \quad (2.66)$$

$$\frac{\partial \phi}{\partial t} = \mu \operatorname{div}(d_p(|\nabla \phi|)\nabla \phi) + \lambda \delta_\epsilon(\phi) \operatorname{div}\left(g \frac{\nabla \phi}{|\nabla \phi|}\right) + \alpha g \delta_\epsilon(\phi) \quad (2.67)$$

L'équation 2.66 est une forme générale de DRLSE.

L'équation 2.67 est la forme lorsque DRLSE est appliqué à un modèle de contour actif basé sur les contours.

- $\mu \operatorname{div}(d_p(|\nabla \phi|)\nabla \phi)$: Le terme de régularisation de distance qui est associé à la fonction potentielle.
- $\lambda \delta_\epsilon(\phi) \operatorname{div}\left(g \frac{\nabla \phi}{|\nabla \phi|}\right)$ et $\alpha g \delta_\epsilon(\phi)$: Ces termes sont associés au terme de longueur pondéré et au terme de zone pondéré.
 - μ : paramètre de régularisation de distance.
 - λ : paramètre de régularisation de la longueur du contour.
 - α : paramètre de répulsion de la surface.
 - d_p est défini par :

$$d_p(s) \triangleq \frac{p'(s)}{s} \quad (2.68)$$

où, p est une **fonction potentielle** pour la régularisation de distance.

Une définition simple et directe de la fonction potentielle à simple puits p_1 est :

$$p = p_1(s) \triangleq \frac{1}{2}(s - 1)^2 \quad (2.69)$$

Il convient de noter que la fonction potentielle à double puits p_2 est une meilleure fonction potentielle pour la régularisation de distance comme suit :

$$p_2 = \begin{cases} \frac{1}{2\pi^2}(1 - \cos(2\pi s)), & \text{si } s \leq 1 \\ \frac{1}{2}(s - 1)^2, & \text{si } s \geq 1 \end{cases} \quad (2.70)$$

Les première et deuxième dérivées de p_2 sont données par :

$$p_2'(s) = \begin{cases} \frac{1}{2\pi}\sin(2\pi s), & \text{si } s \leq 1 \\ s - 1, & \text{si } s \geq 1 \end{cases}; \quad p_2''(s) = \begin{cases} \cos(2\pi s), & \text{si } s \leq 1 \\ 1, & \text{si } s \geq 1 \end{cases} \quad (2.71)$$

La fonction $d_p(s) = \frac{p_2'(s)}{s}$ satisfait :

$$|d_p(s)| < 1, \text{ pour tout } s \in (0, \infty) \text{ et } \lim_{s \rightarrow 0} d_p(s) = \lim_{s \rightarrow \infty} d_p(s) = 1 \quad (2.72)$$

- $\lambda > 0$ et $\alpha \in \mathbb{R}$ sont les coefficients du terme de contour et du terme de région.
- g est une fonction indicatrice de contour comme dans l'équation 2.37 avec $p = 2$.
- δ_ε (défini par l'équation 2.61).

Remarque : après le calcul de la Dirac, ce modèle suit les mêmes calculs que la formulation variationnelle des courbes de niveau sans réinitialisation.

2.3.5.2 Modèle d'ajustement local d'image (LIF)

Le modèle LIF [38] utilise une fonction énergétique basée sur l'ajustement d'image locale pour guider la segmentation. Cela lui permet de maintenir la précision sous-pixel et les propriétés de régularisation des limites tout en améliorant l'efficacité de calcul.

Zhang a défini une formulation d'image ajustée localement comme suit :

$$I^{LIF} = m_1 H_\varepsilon(\phi) + m_2 [1 - H_\varepsilon(\phi)] \quad (2.73)$$

où m_1 et m_2 sont définis comme suit :

$$\begin{cases} m_1 = \text{mean}(I \in (\{x \in \Omega | \phi(x) < 0\} \cap W_k(x))) \\ m_2 = \text{mean}(I \in (\{x \in \Omega | \phi(x) > 0\} \cap W_k(x))) \end{cases} \quad (2.74)$$

où, $W_\kappa(x)$ est une fonction de fenêtre rectangulaire telle qu'une fonction de fenêtre gaussienne tronquée ou une fonction de fenêtre constante. Ensuite, une fonction énergétique d'ajustement local d'image est :

$$E^{LIF}(\phi) = \frac{1}{2} \int |I(x) - I^{LFI}(x)|^2 dx, x \in \Omega \quad (2.75)$$

Enfin, Zhang a minimisé $E^{LFI}(\phi)$ par rapport à ϕ pour obtenir le flux de descente de gradient correspondant :

$$\frac{\partial \phi}{\partial t} = (I - I^{LFI})(m_1 - m_2)\delta_\varepsilon(\phi) = \left((I - m_1 H_\varepsilon(\phi)) - m_2 (1 - H_\varepsilon(\phi)) \right) (m_1 - m_2)\delta_\varepsilon(\phi) \quad (2.76)$$

2.3.5.3 Modèle d'évolution des courbes de niveau avec estimation du champ de biais (LSE-BFE)

La segmentation d'images est souvent compliquée par les inhomogénéités d'intensité. Pour relever ce défi, la méthode LSE-BFE [39] a été développée. Cette approche combine l'évolution des courbes de niveau avec l'estimation du champ de biais pour segmenter efficacement les images tout en tenant compte des variations d'intensité.

- **Modèle des images avec inhomogénéité d'intensité**

Le modèle utilisé pour cette méthode considère une image mesurée comme le produit d'une image réelle à restaurer, d'un champ de biais, et du bruit :

$$I = \tilde{b}\tilde{J} + n \quad (2.77)$$

où :

- I : image mesurée.
- \tilde{J} : image réelle.
- \tilde{b} : champ de biais.
- n : bruit additif.

- **Propriétés du modèle :**

Supposons que l'image réelle \tilde{J} et le champ de biais \tilde{b} ont les propriétés suivantes :

- **P(1)** le champ de biais \tilde{b} est lentement variable sur tout le domaine de l'image.
- **P(2)** les intensités de l'image réelle sont approximativement constantes dans chaque classe de tissu, cela signifie que :

$$\tilde{J}(x) \approx \tilde{c}_i \text{ pour } x \in \Omega_i \quad (2.78)$$

Avec :

- * $\{\tilde{\Omega}_i\}_{i=1}^N$: partition de Ω .
- * \tilde{c}_i : valeur spécifique de la propriété physique mesurée.

- **Formulation de l'énergie**

L'objectif principal est de séparer le domaine de l'image Ω en N régions disjointes $\tilde{\Omega}_i, i = 1, \dots, N$, en se basant sur l'image mesurée I . Cependant, en raison de l'inégalité d'intensité causée par le champ de biais \tilde{b} , les intensités mesurées ne sont pas séparables en utilisant des méthodes de classification traditionnelles basées sur l'intensité. Une nouvelle méthode pour la segmentation conjointe et la correction de biais a été proposée. Cette méthode est basée sur l'observation selon laquelle les intensités dans une région relativement petite sont séparables, ce qui peut être vérifié par les hypothèses **P(1)** et **P(2)** comme expliqué ci-dessus.

Considérons un voisinage circulaire avec un rayon relativement petit ρ centré sur chaque point x dans le domaine de l'image Ω , défini par :

$$O_x \triangleq \{y : |y - x| \leq \rho\} \quad (2.79)$$

La partition $\{\tilde{\Omega}_i\}_{i=1}^N$ induit une partition du voisinage O_x , c'est-à-dire $\{O_x \cap \tilde{\Omega}_i\}_{i=1}^N$.

Pour une fonction lisse \tilde{b} , les valeurs $\tilde{b}(y)$ pour tous les y dans le voisinage circulaire O_x peuvent être bien approximées par $\tilde{b}(x)$, qui est au centre de O_x . Par conséquent, les intensités $\tilde{b}(y)\tilde{J}(y)$ dans chaque sous-région $O_x \cap \tilde{\Omega}_i$ sont approximativement la constante $\tilde{b}(x)\tilde{c}_i$. On a donc l'approximation suivante :

$$\tilde{b}(y)\tilde{J}(y) \approx \tilde{b}(x)\tilde{c}_i, \text{ pour } y \in O_x \cap \tilde{\Omega}_i \quad (2.80)$$

- Les constantes $\tilde{b}(x)\tilde{c}_i$ peuvent être considérées comme les approximations des centres des clusters $\{I(y) : y \in O_x \cap \tilde{\Omega}_i\}$ dans le voisinage O_x . Par conséquent, les intensités dans le voisinage O_x se situent autour de N centres de clusters distincts $\tilde{m} \approx \tilde{b}(x)\tilde{c}_i$.
- Considérons la tâche consistant à classer les intensités $I(y)$ dans le voisinage O_x en N classes. Compte tenu de la séparabilité des intensités dans le voisinage O_x , cette tâche peut être réalisée à l'aide de la méthode standard de regroupement des K-moyennes basée sur la minimisation de la fonction objective pondérée suivante :

$$\varepsilon_x = \sum_{i=1}^N \int_{O_x \cap \tilde{\Omega}_i} K(x-y)|I(y) - b(x)c_i|^2 dy \quad (2.81)$$

Où :

- $b(x)c_i$ sont les centres des clusters à optimiser,
- $K(x-y)$ est une fonction de pondération non négative telle que $K(x-y) = 0$ pour $|x-y| > \rho$ et $\int_{O_x} K(x-y)dy = 1$,
- Bien que le choix de la fonction de pondération soit flexible, il est préférable d'utiliser une fonction de pondération $K(x-y)$ telle que des poids plus importants soient attribués aux données $I(y)$ pour y plus proches du centre x du voisinage O_x .
- La fonction de pondération K est choisie comme noyau gaussien tronqué, défini comme suit :

$$\begin{cases} K_\sigma(u) = \frac{1}{a} e^{-\frac{|u|^2}{2\sigma^2}} & \text{Pour } |u| \leq \rho \\ 0 & \text{Sinon} \end{cases} \quad (2.82)$$

Où :

- a est une constante telle que $K(u) = 1$.

La fonction objective ε_x de l'équation 2.81 peut être réécrite comme suit :

$$\varepsilon_x = \sum_{i=1}^N \int_{\tilde{\Omega}_i} K(x-y)|I(y) - b(x)c_i|^2 dy \quad (2.83)$$

En raison du fait que $K(x-y) = 0$ pour $y \notin O_x$.

Le but ultime est de trouver un ensemble optimal de partition du domaine d'image entier Ω , le champ de biais b et les constantes c_i . La minimisation d'une seule fonction objective ε_x , définie pour un point x , ne permet pas d'atteindre ce but. Alors il faut minimiser ε_x pour tous les points x . Ceci peut être réalisé en minimisant l'intégrale de ε_x sur Ω . Par conséquent, nous définissons une énergie $\varepsilon \triangleq \int \varepsilon_x dx$ c'est-à-dire :

$$\varepsilon = \int \left(\sum_{i=1}^N \int_{\Omega_i} K(x-y) |I(y) - b(x)c_i|^2 dy \right) dx \quad (2.84)$$

La minimisation directe de l'énergie avec la partition $\{\Omega_i\}_{i=1}^N$ comme variable n'est pas pratique. Donc l'utilisation d'une ou plusieurs fonctions de courbes de niveau pour représenter la partition est nécessaire. La minimisation de l'énergie peut donc être effectuée en résolvant une équation d'évolution de courbes de niveau.

La formulation des courbes de niveau de l'énergie pour les cas $N = 2$ et $N > 2$, appelées respectivement formulations en deux phases ou multiphase, sera donnée dans la suite.

• **Formulation des courbes de niveau**

Considérons d'abord le cas de $N = 2$. Dans ce cas, le domaine de l'image est divisé en deux régions $\{\Omega_i\}_{i=1}^2$. Ces deux régions peuvent être représentées par les régions séparées par le contour du niveau zéro d'une fonction ϕ , c'est-à-dire $\Omega_1 \triangleq \phi > 0$ et $\Omega_2 \triangleq \phi < 0$. En utilisant la fonction de Heaviside H , l'énergie ε dans l'équation 2.85 peut être exprimée comme une énergie en termes de ϕ , b et c comme suit :

$$\varepsilon(\phi, b, c) = \int \left(\sum_{i=1}^N \int_{\Omega_i} K(x-y) |I(y) - b(x)c_i|^2 M_i(\phi(y)) dy \right) dx \quad (2.85)$$

Où :

– M_i : fonctions d'appartenance.

Avec :

– $M_1(\phi(x)) = H(\phi(x))$,
 – $M_2(\phi(x)) = 1 - H(\phi(x))$.

En pratique, nous utilisons une fonction de Heaviside lissée donnée par l'équation suivante :

$$H_\varepsilon(x) = \frac{1}{2} \left[1 + \frac{2}{\pi} \arctan\left(\frac{x}{\varepsilon}\right) \right] \quad (2.86)$$

pour approximer la fonction de Heaviside originale H , avec $\varepsilon = 1$. Il est nécessaire d'ajouter un terme de régularisation $R(\phi)$ à l'énergie ci-dessus dans la fonctionnelle énergétique suivante :

$$F(\phi, b, c_1, c_2) \triangleq \varepsilon(\phi, b, c) + R(\phi) \quad (2.87)$$

Où :

- $R(\phi) \triangleq \mu \int |\nabla H(\phi)| dx + \nu \int (|\nabla \phi| - 1)^2 dx$.
- Le premier terme de R sert à régulariser le contour du niveau zéro de ϕ , comme dans les méthodes typiques des courbes de niveau.
- Le second terme régularise l'ensemble de la fonction des courbes de niveau ϕ en pénalisant son écart par rapport à la distance signée
- L'énergie $\varepsilon(\phi, b, c)$ est le terme de données dans notre cadre variationnel.

De même, nous pouvons utiliser plusieurs fonctions de courbes de niveau ϕ_1, \dots, ϕ_n pour représenter les régions $\{\Omega_i\}_{i=1}^N$ avec $N = 2^n$. Par commodité, nous utilisons une fonction vectorielle $\Phi = (\phi_1, \dots, \phi_n)$ pour représenter les fonctions ϕ_1, \dots, ϕ_n . L'énergie pour la formulation multiphase générale de notre méthode peut être définie comme suit :

$$F(\Phi, b, c) \triangleq \int \left(\sum_{i=1}^N \int_{\Omega_i} K(x-y) |I(y) - b(x)c_i|^2 M_i(\phi(y)) dy \right) dx + \sum_{i=1}^N R(\phi_i) \quad (2.88)$$

Où :

- $M_i(\Phi)$ sont des fonctions de Φ conçues de telle sorte que $\sum_{i=1}^N M_i(\Phi) = 1$.
- Pour $N = 3$ et deux fonctions de niveau ϕ_1 et ϕ_2 , la définition des fonction $M_i(\phi)$ sera :
 - * $M_1(\phi_1, \phi_2) = H(\phi_1)H(\phi_2)$,
 - * $M_2(\phi_1, \phi_2) = H(\phi_1)(1 - H(\phi_2))$,
 - * $M_3(\phi_1, \phi_2) = 1 - H(\phi_1)$.

pour obtenir une formulation à trois phases.

Décrivons ici que la minimisation de l'énergie pour le cas biphasé (le cas multiphasé peut être résolu avec la même procédure). Pour c et b fixes, la minimisation de $F(\phi, c, b)$ consiste à résoudre l'équation d'évolution des courbes de niveau comme l'équation de descente de gradient suivante :

$$\frac{\partial \phi}{\partial t} = - \frac{\partial F}{\partial \pi} \quad (2.89)$$

Où :

- $\frac{\partial F}{\partial \phi}$: La dérivée fonctionnelle du premier ordre de l'énergie F .

Dans la mise en œuvre numérique, à chaque itération selon l'équation 2.90, les variables c et b sont mises à jour selon la procédure suivante. Pour un ϕ et un c fixes, nous trouvons un champ de biais optimal \hat{b} qui minimise $F(\phi, c, b)$. On peut montrer que le minimiseur \hat{b} est :

$$\hat{b} = \frac{(IJ^{(1)}) * K}{J^{(2)} * K} \quad (2.90)$$

Avec :

- $J^{(1)} = \sum_{i=1}^N c_i M_i(\phi)$.
- $J^{(2)} = \sum_{i=1}^N c_i^2 M_i(\phi)$.
- Pour ϕ et b fixés, nous trouvons un \hat{c} optimal qui minimise $F(\phi, c, b)$. Par quelques manipulations de calcul, on peut montrer que le minimiseur $\hat{c} = (\hat{c}_1, \dots, \hat{c}_N)$ est :

$$\hat{c}_i = \frac{\int (b * K) I M_i(\phi) dx}{\int (b^2 * K) M_i(\phi) dx}, \quad i = 1, \dots, N. \quad (2.91)$$

2.4 Conclusion

Au terme de ce chapitre, nous avons exploré le concept des snakes traditionnels, mettant en lumière leurs principes fondamentaux ainsi que les différentes méthodes d'optimisation qui les accompagnent, telles que la programmation dynamique, Greedy et GVF. De plus, nous avons examiné la théorie des courbes de niveau, en étudiant les divers algorithmes qui lui sont associés. Dans le prochain chapitre, nous entreprendrons l'implémentation pratique du GVF et deux algorithmes de courbes de niveau LSE-BFE et DRLSE, en utilisant MATLAB. Cette phase pratique nous permettra de mettre en œuvre les concepts théoriques abordés dans ce chapitre et de mieux appréhender leur application dans des contextes concrets.

Chapitre 3

Simulation sur MATLAB

3.1 Introduction

En s'appuyant sur les concepts théoriques des algorithmes de contours actifs et de courbes de niveaux exposés au chapitre précédent et en considérant les études existantes dans ce domaine, notre choix de simulation sur MATLAB s'est porté sur le "GVF" comme méthode de contours actifs. Nous avons également décidé d'intégrer un algorithme basé sur les contours, le "DRLSE", ainsi qu'un algorithme basé sur les régions, le "LSF-BFE", pour les courbes de niveaux.

Pour chaque algorithme, nous présenterons un organigramme détaillé décrivant les différentes étapes, en expliquant les fonctions clés associées. Nous procéderons ensuite à l'étude de ces algorithmes en les appliquant d'abord sur des images de synthèse, puis sur des images médicales. Dans ce contexte, nous introduirons également les images médicales utilisées ainsi que le filtre appliqué pour les améliorer.

Tout au long de notre étude, nous comparerons et expliquerons les résultats de chaque algorithme, permettant ainsi une évaluation approfondie de leur performance dans le contexte de la segmentation d'images, notamment celles issues de domaine médical.

3.2 Logiciel MATLAB

Nous présentons le logiciel MATrix LABoratory, en **Annexe 2** en fournissant une définition, en détaillant ses principales fonctionnalités et en présentant son interface.

3.3 Imagerie médicale

L'imagerie médicale offre une multitude de techniques pour visualiser les structures anatomiques et les anomalies pathologiques du corps humain. La segmentation d'images est un processus clé dans le domaine de l'imagerie médicale, permettant de diviser une image en régions homogènes pour faciliter l'analyse et l'interprétation [40].

3.3.1 Segmentation des images médicales

La segmentation d'images médicales revêt une importance cruciale dans l'analyse et l'interprétation des données, offrant plusieurs avantages clés :

- L'utilisation de la segmentation dans l'imagerie médicale permet une meilleure visualisation et une analyse plus précise des structures anatomiques. En isolant les régions d'intérêt, la segmentation facilite la détection d'anomalies en contribuant ainsi à une interprétation plus approfondie des images médicales.

- Des travaux de recherche antérieurs ont exploré les applications de la segmentation dans le domaine médical. Ces études ont démontré l'efficacité de diverses méthodes de segmentation pour des tâches telles que la quantification des volumes tissulaires [41], la localisation de pathologies [42], la cartographie de structures anatomiques [43], la planification de traitement [44] et la chirurgie assistée par ordinateur [45]. Ces avancées ont ouvert la voie à de nouvelles approches et techniques pour améliorer la précision et l'efficacité de la segmentation d'images médicales.
- Contrairement à d'autres approches de segmentation présentées au chapitre 1, les contours actifs et les courbes de niveaux se distinguent par leur capacité à effectuer une segmentation ciblée d'objets spécifiques dans l'image. Plutôt que de segmenter tous les contours présents, ces méthodes permettent de délimiter précisément les contours des structures d'intérêt, offrant ainsi une segmentation plus précise et adaptée aux besoins cliniques [46].

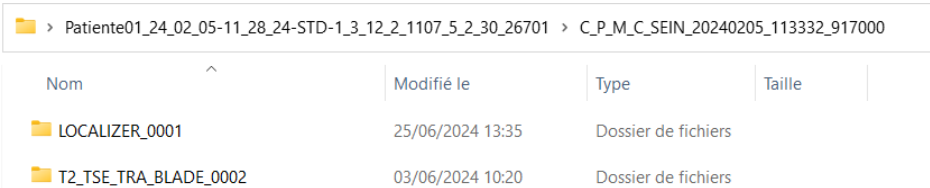
Dans les sections suivantes, nous examinerons deux types d'images médicales spécifiques qui sont ; des IRM (Imagerie par Résonance Magnétique) du cerveau ainsi que des IRM du sein, sur lesquelles nous appliquerons d'abord les algorithmes de segmentation par contour actif "GVF", puis les algorithmes de courbes de niveaux "DRLSE" et "LSF-BFE" :

3.3.2 IRM du Sein

L'IRM du sein est une technique d'imagerie médicale non invasive qui utilise des champs magnétiques et des ondes radio [47] pour créer des images détaillées de la structure interne du sein. Ces images sont souvent utilisées dans le dépistage précoce, le diagnostic et le suivi des maladies du sein, telles que le cancer du sein.

3.3.2.1 Récupération des images

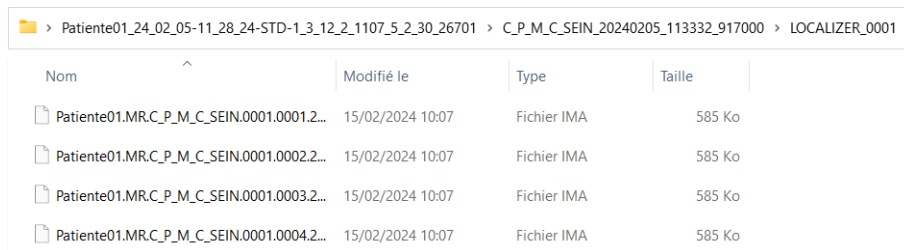
Les images IRM des seins ont été récupérées sous forme de dossiers **DICOM** contenant des fichiers **.ima** à partir du service de radiologie de l'hôpital Mustapha Bacha. Les fichiers **.ima** sont des images brutes qui sont stockées dans des dossiers DICOM avec d'autres informations de métadonnées médicales. La figure 3.1 montre le dossier DICOM tel qu'il apparaît sur un PC :



Nom	Modifié le	Type	Taille
LOCALIZER_0001	25/06/2024 13:35	Dossier de fichiers	
T2_TSE_TRA_BLADE_0002	03/06/2024 10:20	Dossier de fichiers	

FIG. 3.1 : Les dossiers DICOM

Les fichiers **.ima** sont des fichiers d'imagerie bruts qui font partie du format DICOM. La figure 3.2 illustre les fichiers **.ima** contenus dans le dossier DICOM :



Nom	Modifié le	Type	Taille
Patiente01.MRC_P_M_C_SEIN.0001.0001.2...	15/02/2024 10:07	Fichier IMA	585 Ko
Patiente01.MRC_P_M_C_SEIN.0001.0002.2...	15/02/2024 10:07	Fichier IMA	585 Ko
Patiente01.MRC_P_M_C_SEIN.0001.0003.2...	15/02/2024 10:07	Fichier IMA	585 Ko
Patiente01.MRC_P_M_C_SEIN.0001.0004.2...	15/02/2024 10:07	Fichier IMA	585 Ko

FIG. 3.2 : Les fichiers .ima

3.3.2.2 Utilisation d'un logiciel pour la visualisation et l'exportation

Étant donné que les fichiers .ima ne peuvent pas être lus directement sur un PC, nous avons utilisé le logiciel **RadiAnt** comme intermédiaire pour leur visualisation.

Remarque :

Ce logiciel offre 1 mois d'utilisation gratuite.

Pour la visualisation et l'exportation, voici comment nous avons procédé :

- **Importation des Dossiers DICOM dans RadiAnt :**

Nous avons importé les dossiers DICOM dans RadiAnt pour leur visualisation.

La figure 3.3 montre le processus d'ouverture d'un dossier DICOM dans RadiAnt :

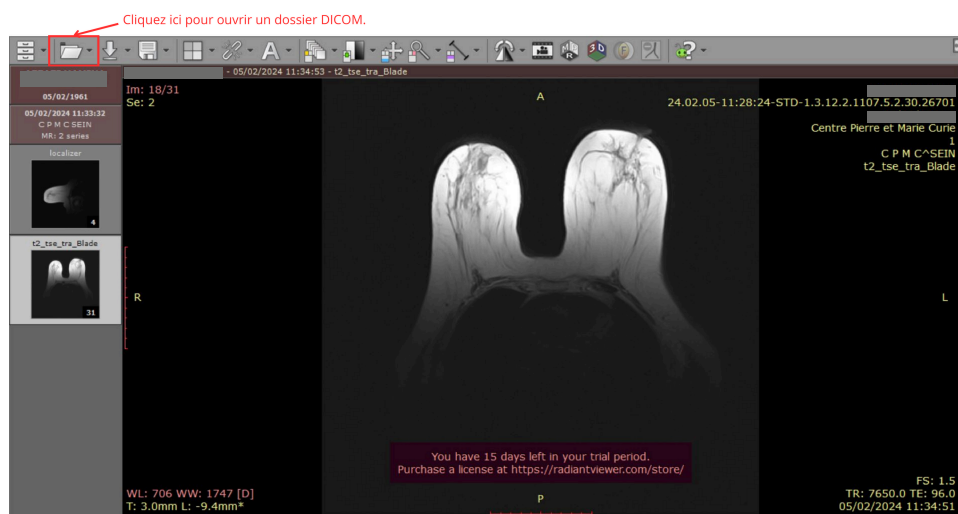


FIG. 3.3 : Ouverture d'un dossier DICOM dans RadiAnt

- **Visualisation et Sélection des Coupes dans RadiAnt :**

Après avoir importé les dossiers DICOM, nous avons utilisé RadiAnt pour visualiser les fichiers .ima et sélectionner les coupes qui convenaient le mieux à notre analyse.

- **Exportation des Images depuis RadiAnt :**

Une fois les coupes sélectionnées, nous avons exporté les images sous le format de fichier souhaité (.jpg, .png, .bmp ...etc) en ajustant les dimensions de l'image selon nos besoins.

La figure 3.4 montre le processus d'exportation des images depuis RadiAnt :

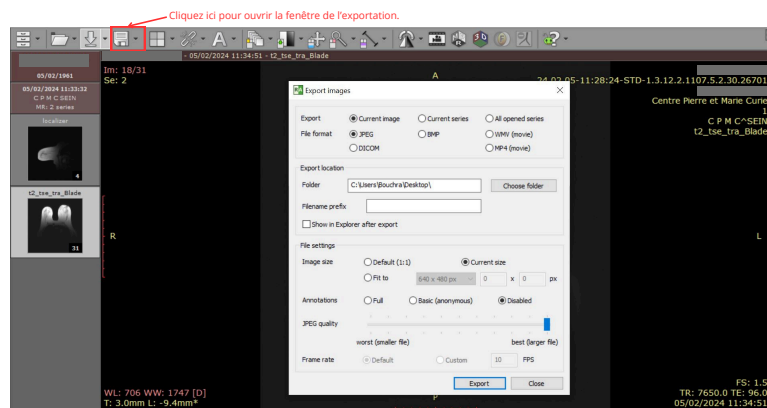


FIG. 3.4 : Exportation des images depuis RadiAnt

Cette procédure nous permet d'aboutir à des images représentant différentes coupes, fournissant ainsi une base précieuse pour notre analyse ultérieure.

D'après ces images, nous avons sélectionné les parties qui contiennent des pathologies.

La figure 3.5 illustre l'une de ces images :

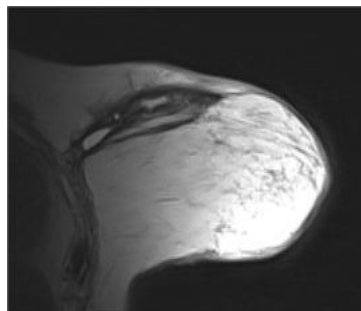


FIG. 3.5 : Image d'une section présentant une anomalie

3.3.3 IRM du cerveau

Dans cette section, nous examinons les images du "BRATS MICCAI brain tumor dataset" [48], une base de données renommée pour ses applications en apprentissage profond grâce à la diversité et la qualité de ses images de tumeurs cérébrales. En accédant à cette base, nous avons obtenu un ensemble varié d'images de différentes conditions et types de tumeurs cérébrales, essentielles pour notre étude sur la segmentation d'images médicales.

La figure 3.6 montre quelques images de la base de données :

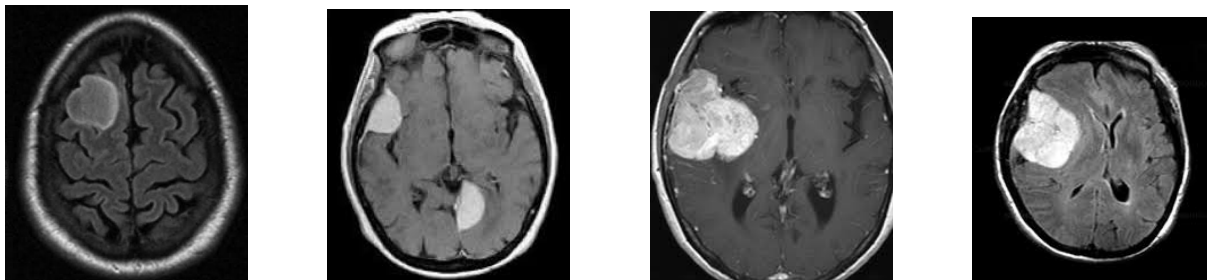


FIG. 3.6 : Images de la base de données "Brain Tumor Data"

3.3.4 Pré-traitement de l'image

Avant d'appliquer la méthode GVF et DRLSE pour la segmentation d'images, il est crucial de prétraiter l'image d'entrée afin d'améliorer la qualité des contours extraits. Dans ce contexte, le filtrage anisotrope se révèle être une technique efficace pour réduire le bruit tout en préservant les contours et les détails importants de l'image médicale. Dans cette section, nous détaillerons les différentes étapes du prétraitement de l'image en utilisant un filtre anisotrope, en expliquant en détail les mécanismes sous-jacents de cette technique sans inclure le code.

3.3.4.1 Mécanisme du filtrage anisotrope

Le filtre anisotrope est un type de filtre utilisé en traitement d'image pour réduire le bruit tout en préservant les contours et les bords importants de l'image [49]. Son mécanisme repose sur la diffusion anisotrope, où la diffusion du bruit est plus forte dans certaines directions que dans d'autres [50].

Voici le mécanisme général du filtre anisotrope :

1. **Calcul des gradients de l'image**

Les gradients de l'image sont calculés en utilisant la fonction "gradient()". Cela permet de mesurer les variations de luminosité entre les pixels dans les directions horizontale ($grad_x$) et verticale ($grad_y$).

2. **Calcul de la norme du gradient**

La norme du gradient est calculée en combinant les gradients horizontaux et verticaux à l'aide de la formule de la norme euclidienne :

$$grad_{norm} = \sqrt{grad_x^2 + grad_y^2} \quad (3.1)$$

Cela permet de quantifier la quantité de changement de luminosité à chaque pixel.

3. **Calcul de la diffusion anisotrope**

La diffusion anisotrope est calculée en fonction de la norme du gradient. La valeur κ est un paramètre de seuil qui contrôle la quantité de diffusion permise.

Plus la valeur de κ est grande, moins la diffusion est autorisée dans les régions où

le gradient est élevé, préservant ainsi les contours.

La fonction $c = \frac{1}{1 + \left(\frac{\text{gradnorm}}{\kappa}\right)^2}$ calcule le coefficient de diffusion pour chaque pixel en fonction de sa norme de gradient.

4. Mise à jour de l'image filtrée

L'image filtrée est mise à jour en utilisant la divergence des gradients pondérée par les coefficients de diffusion calculés précédemment. Cela aide à atténuer le bruit tout en préservant les contours. La fonction **"divergence()"** calcule la divergence d'un champ vectoriel bidimensionnel.

Ce processus est répété pour un nombre spécifié d'itérations afin d'améliorer progressivement la qualité de l'image filtrée .

3.3.4.2 Résultats de l'application du filtre

Les figures 3.7 et 3.8 présentent les résultats obtenus après l'application du filtre anisotrope à une IRM d'un cerveau et sur l'IRM d'un sein respectivement, illustrant l'impact de l'ajustement du paramètre κ sur la qualité de la réduction du bruit et la préservation des détails structuraux :

- Sur l'IRM d'un cerveau :

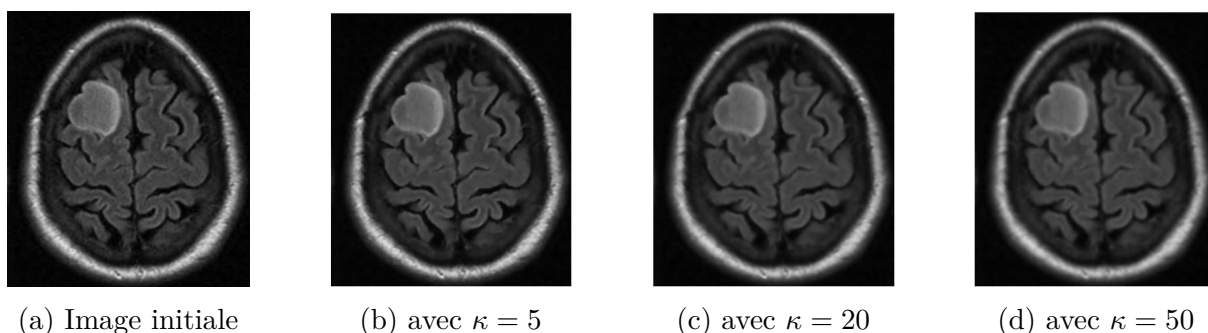


FIG. 3.7 : Résultat de l'application du filtre anisotrope sur une image de la base de données "BRATS MICCAI brain tumor dataset" avec différentes valeurs de κ

- Sur l'IRM d'un sein

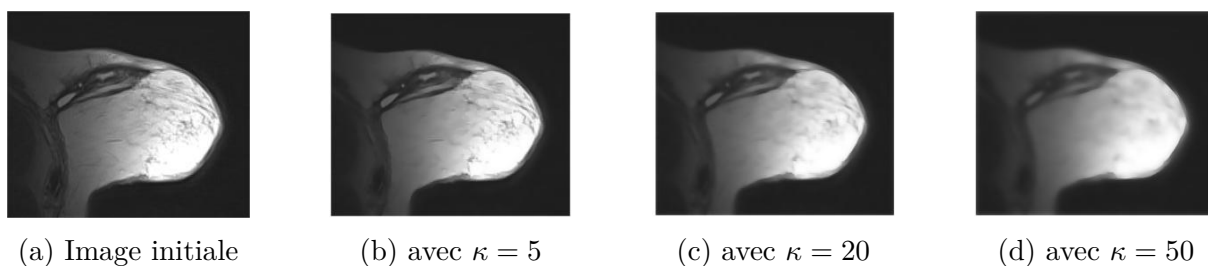


FIG. 3.8 : Résultat de l'application du filtre anisotrope sur une image de l'IRM d'un sein avec différentes valeurs de κ

3.3.4.3 Interprétation des résultats de l'application du filtre

- **Impact du paramètre κ sur le filtrage anisotrope :**
 - L'ajustement de κ a une influence significative sur les résultats du filtrage anisotrope.
 - Des valeurs plus élevées de κ entraînent une réduction plus marquée du bruit, conduisant à des images plus lisses mais avec une possible perte de détails fins.
 - À l'inverse, des valeurs plus faibles de κ préservent mieux les contours et les structures, tout en conservant une certaine quantité de bruit dans les régions de faible contraste.
- **Choix de κ pour des tâches de segmentation :**
 - Il est crucial de sélectionner avec soin la valeur optimale de κ en fonction des exigences spécifiques de l'application médicale.
 - Pour des tâches de segmentation ultérieures, telles que la méthode GVF ou celle des courbes de niveau "DRLSE", le choix de κ impacte directement la qualité de la segmentation en influençant la netteté et la précision des contours.
 - Un ajustement minutieux de κ permet d'assurer que les images filtrées répondent aux critères de qualité nécessaires pour des tâches de segmentation précises et fiables.

3.4 Simulation de la méthode des contours actifs sur MATLAB

Le choix d'implémenter la méthode des Forces de Gradient Vectorielles (GVF) pour la segmentation par contours actifs, plutôt que d'autres méthodes d'optimisation, repose sur plusieurs avantages démontrés par divers travaux de recherche [51]. Voici les raisons principales qui ont motivé cette décision :

- Contrairement aux snakes traditionnels, le GVF offre une flexibilité supérieure en termes d'initialisation. Il permet de placer le contour initial loin de l'objet à segmenter, sans nécessiter que ce contour entoure entièrement l'objet. De plus, le contour initial peut traverser l'objet, ce qui n'est pas possible avec les méthodes traditionnelles [52].
- Dans le cas du GVF, les forces externes sont calculées directement à partir de l'image à segmenter, au lieu d'être définies a priori comme dans les snakes traditionnels. Cette approche permet au contour de mieux s'adapter aux caractéristiques locales de l'image, améliorant ainsi la précision de la segmentation [15].
- La méthode GVF a été développée pour surmonter les limitations bien connues des snakes traditionnels, notamment les problèmes liés à l'initialisation et à la convergence. Les snakes traditionnels montrent souvent une convergence médiocre, particulièrement dans les régions concaves de l'objet à segmenter. En raison de la définition

a priori du champ de forces externes, les forces perpendiculaires nécessaires pour guider le contour vers l'intérieur des concavités sont souvent nulles, empêchant ainsi une segmentation correcte. Le GVF, en revanche, génère des forces qui permettent au contour de pénétrer efficacement dans les concavités [53].

- Étant donné notre objectif d'appliquer les contours actifs à des images médicales, la méthode GVF est particulièrement adaptée à la segmentation complexe requise pour ces types d'images. Des études comparatives entre le GVF et les snakes traditionnels ont démontré la supériorité du GVF dans le contexte de la segmentation d'images médicales [51].
- Le code utilisé pour l'algorithme GVF est tiré de la référence suivante [54].

3.4.1 Description de l'algorithme de la méthode GVF

Afin de visualiser de manière concise les différentes étapes impliquées dans la méthode de segmentation GVF, l'organigramme suivant (figure 3.9) détaille le flux de travail de l'algorithme, depuis le prétraitement de l'image jusqu'à l'obtention de l'image segmentée.

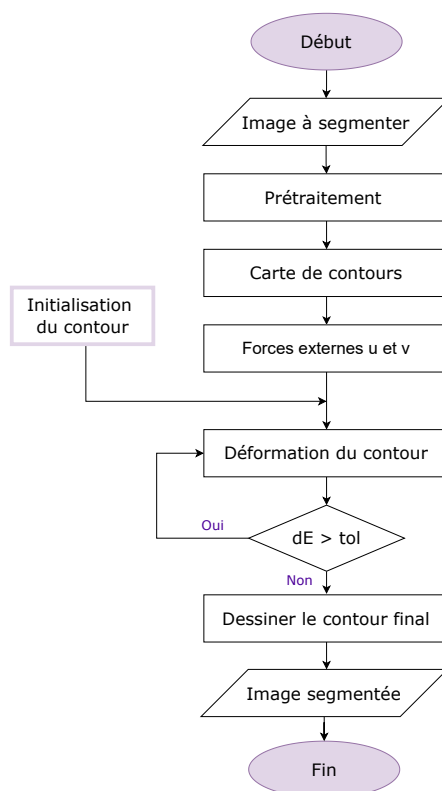


FIG. 3.9 : Organigramme de la méthode GVF

3.4.1.1 Initialisation du contour actif (snake)

L'algorithme d'initialisation du contour actif vise à créer un contour initial pour le processus de segmentation GVF.

$$C = \{V(i) = [x(i), y(i)]\}, \forall(i) \in [1, Num] \quad (3.2)$$

Avec : *Num* le nombre de points du snake, que nous choisissons.

Algorithm 1 Exemple d'initialisation du Contour Actif

```

1: procedure INITIALISATION_CONTOUR_ACTIF(N, M)
2:   cx ← ⌊ $\frac{N}{2}$ ⌋                                ▷ Coordonnée x du centre de l'image
3:   cy ← ⌊ $\frac{M}{2}$ ⌋                                ▷ Coordonnée y du centre de l'image
4:    $\rho$  ← min(cx, cy) - px                    ▷ Rayon
5:   Num ← 150                                    ▷ Nombre de points du contour
6:    $as$  ←  $\frac{2 \times \pi}{Num}$                           ▷ Pas d'angle
7:   for i ← 0 to Num - 1 do
8:     ang[i] ← -i × as                        ▷ Angles dans le sens horaire
9:   end for
10:  for i ← 0 to Num - 1 do
11:    x[i] ← cx +  $\rho \times \cos(\text{ang}[i])$         ▷ Coordonnées x des points du contour
12:    y[i] ← cy +  $\rho \times \sin(\text{ang}[i])$         ▷ Coordonnées y des points du contour
13:  end for
14: end procedure

```

- Cet algorithme initialise un contour actif sous forme d'un cercle, en effet, il peut être initialisé sous forme d'un rectangle ou carré ou autre.
- *cx* et *cy* peuvent être initialisés au centre de l'image comme dans l'algorithme ci-dessus (cette initialisation est utilisée pour les images synthétiques comme nous le verrons par la suite dans la section 3.4.2), ou saisis directement comme des coordonnées (comme nous le verrons par la suite dans la section 3.4.3 pour les images médicales).
- *px* est un paramètre utilisé pour rapprocher ou éloigner le contour actif initial de l'objet à segmenter.

3.4.1.2 Carte de contours

La carte de contours donnée par l'équation 2.24, est calculée sur MATLAB comme suit :

```

% Création d'un filtre Laplacien de Gaussien
Filtre = fspecial('log', [11, 11], 0.2);
% Calcul de la carte de contours : f = (conv(I, laplacian(G)))^2
f = imfilter(I, Filtre) .^ 2;

```

- L'équation 2.24 représente le carré de la norme du gradient de la convolution de l'image $I(i, j)$ par la fonction Gaussienne (vu dans l'équation 2.8) d'écart-type σ .
- La fonction **fspecial** crée un filtre Laplacien (11×11) d'une fonction Gaussienne d'écart-type $\sigma = 0.2$:
- La fonction **imfilter** applique la convolution entre le filtre et l'image I.

3.4.1.3 Forces externes u et v

Comme vu dans les équations 2.27, u et v sont calculés à partir de f_x et f_y :

```
% Création d'un masque_x et d'un masque_y
hx = -transpose( fspecial( 'sobel' ) ) / 4 ;
hy = -fspecial( 'sobel' ) / 4 ;
% Calcul de fx et fy
fx = imfilter( f, hx );
fy = imfilter( f, hy );
```

- **hx** et **hy** représentent les masques ou noyaux de convolution du filtre Sobel donnés dans l'équation 1.7.
- **hx** et **hy** sont utilisés pour calculer le gradient horizontal (f_x) et vertical (f_y) respectivement, de f .

Pour le calcul itératif de u et v (équations 2.27), nous passons également par le calcul du Laplacien de u et v :

```
% Création d'un masque 4-connexités
h = fspecial( 'laplacian', 0 );
% Calcul de Laplacien de u
Laplacien_u = imfilter( u, h );
Laplacien_v = imfilter( v, h );
```

- **h** représente le masque de l'opérateur Laplacien 4-connexités donné dans l'équation 1.8.

3.4.1.4 Déformation du contour actif

- **Calcul de x et y à chaque itération**

- Pour faciliter les calculs des équations 2.15, nous calculons d'abord 4 vecteurs intermédiaires :

```
x_b1 = x( [n, 1:n-1] );           % x[n-1]
x_b2 = x( [n-1, n, 1:n-2] );     % x[n-2]
x_a1 = x( [2:n, 1] );           % x[n+1]
x_a2 = x( [3:n, 1, 2] );       % x[n+2]
```

- Nous aurons par la suite les dérivées : première, seconde et 4ème de x :

$$dx2 = x_b1 - 2*x + x_a1; \quad \% x''(s)$$

$$dx4 = x_b2 - 4*x_b1 + 6*x - 4*x_a1 + x_a2; \quad \% x''''(s)$$

$$dx1 = x_a1 - x_b1; \quad \% x'(s)$$

- La mise à jour du contour se fait en suivant l'équation 2.9, sur MATLAB, cela revient à effectuer le calcul suivant :

$$x = x + dt * (\alpha * dx2 - \mathbf{beta} * dx4 + u);$$

Avec dt , un paramètre réglable.

- Nous effectuons les mêmes calculs pour la mise à jour de y .
- A chaque itération, nous calculons dE comme le montre la figure 2.10 et on le compare avec une tolérance tol que nous choisissons, le but étant de minimiser l'énergie totale (équation 2.9) que nous calculons sur MATLAB comme suit :

$$E = 0.5 * (\alpha * \mathbf{sum}(\mathbf{sqrt}(dx1.^2 + dy1.^2)) + \mathbf{beta} * \mathbf{sum}(\mathbf{sqrt}(dx2.^2 + dy2.^2))) + \mathbf{sum}(u) + \mathbf{sum}(v);$$

- **Interpolation des points de x et y après chaque déformation**

Cette étape est rajoutée lors de l'implémentation de la méthode GVF, elle n'existe pas dans la théorie mathématique de cette dernière.

L'objectif est de garder une distance moyenne entre chaque deux points du snake, pour cela il nous faut choisir une distance minimale (d_{min}) et distance maximale (d_{max}) qui seront l'intervalle de distance à respecter.

3.4.2 Résultats sur des images synthétiques

- Dans un premier temps, nous avons testé la méthode de segmentation par les contours actifs en utilisant le GVF sur une image synthétique représentant un "puzzle" avec plusieurs légères concavités.

La figure 3.10 illustre le résultat de la segmentation :

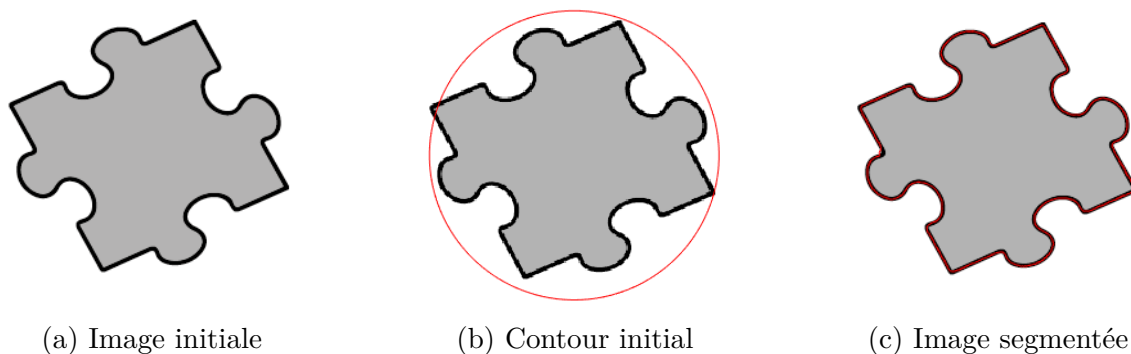


FIG. 3.10 : Résultat de la segmentation avec GVF sur une image synthétique "Puzzle", $\mu = 0.1$, $\alpha = 0.1$, $\beta = 0.052$, $tol = 0.04$, $it_{max} = 100$

En utilisant l'initialisation du snake vu dans l'algorithme (1), la méthode GVF a pu facilement segmenter l'objet, sans nécessiter plusieurs essais pour le choix des paramètres d'élasticité α et de rigidité β .

Remarques :

- Le réglage initial des paramètres α et β a été fait en se basant sur des expérimentations déjà faites sur la méthode GVF appliquée sur des images synthétiques, et puis de là, nous avons effectué un réglage manuel empirique.
- Le contour initial n'a pas besoin d'être proche de l'objet.
- Lors de la simulation sur MATLAB, en plus de la tolérance à laquelle on compare l'énergie totale à chaque itération, nous avons ajouté une autre condition d'arrêt à la segmentation avec la méthode GVF qui est un nombre d'itérations maximales it_{max} .

Pour comprendre comment le contour initial converge vers l'objet à segmenter, observons les forces externes u et v représentées dans la figure 3.11:

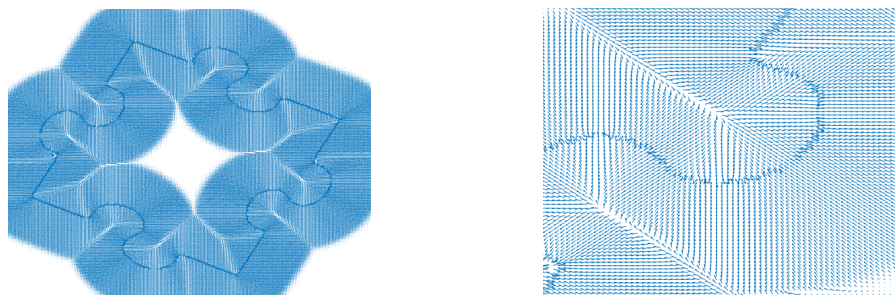


FIG. 3.11 : Représentation des Forces Externes (u, v)

Nous observons que l'ensemble des vecteurs (u, v) converge vers le contour de l'objet de chaque côté de celui-ci. Les valeurs de ces vecteurs sont importantes près du contour de l'objet et tendent vers zéro à mesure qu'on s'éloigne de celui-ci. Par conséquent, le contour initialisé converge vers les points de convergence des vecteurs (u, v) .

- La prochaine étape vise à démontrer la capacité de la méthode à segmenter des formes plus concaves par rapport aux contours actifs traditionnels, pour cela nous avons testé la méthode GVF sur une image synthétique représentant le caractère "U" qui a une concavité longue et étroite par rapport à l'image "puzzle".

La figure 3.12 illustre le résultat de la segmentation :

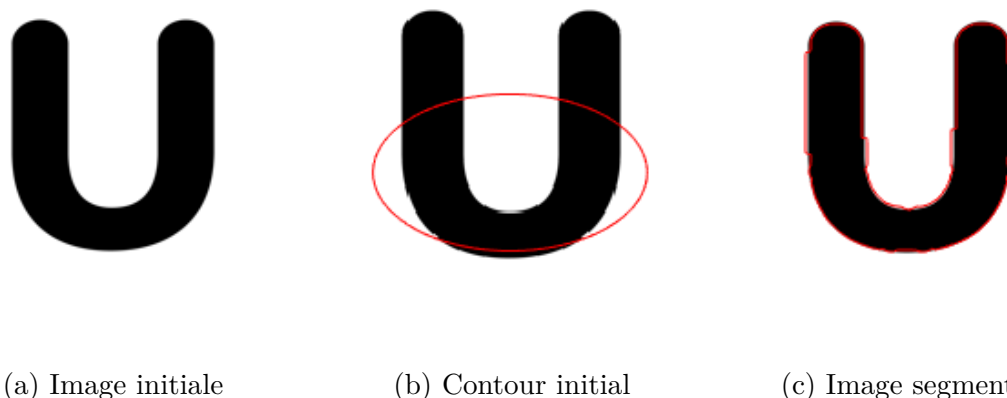


FIG. 3.12 : Résultat de la segmentation avec GVF sur une image synthétique U, $\mu = 0.15, \alpha = 0.1, \beta = 0.05, tol = 0.001, it_{max} = 500$

Avec un choix approprié des paramètres d'élasticité α et de rigidité β , et plus important encore, avec une initialisation correcte du contour, le GVF parvient à segmenter ce type de concavité.

Il convient de noter qu'il peut être quelque peu laborieux de déterminer la meilleure combinaison de paramètres et d'initialisation du contour pour ce type d'images contenant un objet avec une concavité longue et étroite.

- En voulant tester les limites de la méthode GVF, nous l'avons appliqué sur une image synthétique contenant cette fois-ci plusieurs objets distincts qui représentent différentes formes géométriques.

La figure 3.13 montre le résultat de la segmentation :

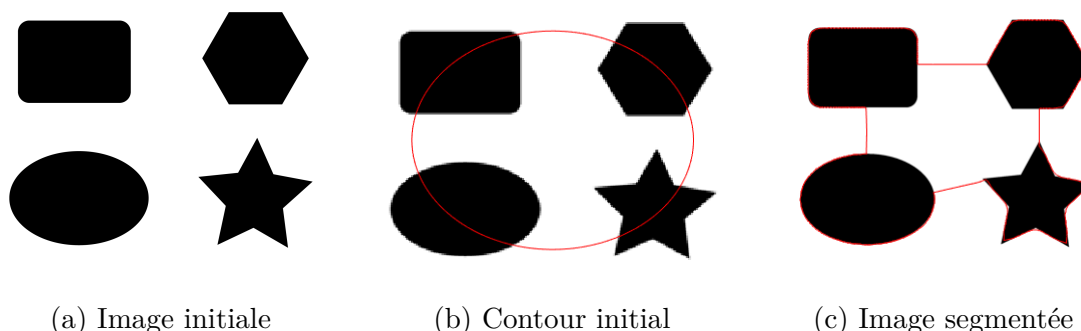


FIG. 3.13 : Résultat de la segmentation avec GVF sur une image synthétique contenant plusieurs formes géométriques, $\mu = 0.15, \alpha = 0.2, \beta = 0.05, tol = 0.005, it_{max} = 300$

La méthode GVF échoue à segmenter plusieurs objets, car elle repose sur l'initialisation d'un seul contour. Ce contour se déforme au fil des itérations pour approcher les contours d'un seul objet et ne se divise pas en plusieurs contours distincts.

3.4.2.1 Initialisation

Afin d'observer la dépendance de la méthode GVF à l'initialisation du contour actif, et par conséquent l'effet de cette initialisation sur le résultat de la segmentation, prenons l'exemple de l'image représentant le caractère "U".

En choisissant l'initialisation donnée dans l'algorithme (1), voici le résultat de la segmentation (figure 3.14) :



(a) Contour initial

(b) Image segmentée

FIG. 3.14 : Résultat de la segmentation avec GVF sur l'image "U"

En apportant quelques modifications à la forme et à l'emplacement du contour actif initial, on obtient le résultat de la segmentation de la figure 3.12.

Ces expérimentations menées sur l'image représentant le caractère "U" démontrent clairement l'impact significatif de l'initialisation du contour actif sur les résultats de la segmentation avec la méthode GVF. Les différentes configurations d'initialisation ont montré des variations notables dans la précision et l'efficacité de la segmentation. Ces observations soulignent l'importance cruciale d'une initialisation appropriée pour obtenir des résultats optimaux et fiables lors de l'utilisation de la méthode GVF pour la segmentation d'images complexes.

3.4.2.2 Avec et Sans interpolation

Pour montrer l'importance de l'ajout de la fonction d'interpolation, la figure 3.15 présente les résultats de la segmentation avec et sans interpolation en gardant le même jeu de paramètres utilisé pour la figure (3.10) :

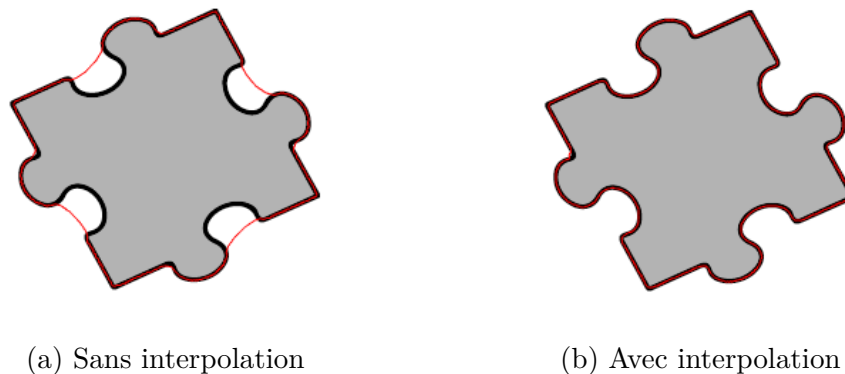


FIG. 3.15 : Résultat de la segmentation avec GVF sur une image synthétique "puzzle" avec et sans interpolation des points du contour actif

Les résultats obtenus montrent clairement que l'ajout de la fonction d'interpolation améliore significativement la qualité de la segmentation. Sans interpolation, les points du contour actif peuvent devenir irrégulièrement espacés, ce qui conduit à une segmentation moins précise et moins stable. En revanche, avec l'interpolation, les points du contour maintiennent une distance régulière, permettant ainsi une meilleure adaptation aux contours de l'objet à segmenter. Ces observations confirment l'importance de l'interpolation pour optimiser les performances de la méthode GVF.

3.4.2.3 Réglages des paramètres μ , α , β et tol

En gardant les paramètres de la figure (3.10), et en ne changeant qu'un paramètre à la fois voici les résultats :

- **Paramètre de régularisation μ :**

μ est le paramètre de régularisation pour le calcul de l'énergie externe donnée dans l'équation 2.25.

Lorsque μ est trop petit, le champ de forces externes se resserre et devient concentré sur le contour de l'objet à segmenter. À mesure que μ augmente, le champ de forces externes commence plus loin du contour de l'objet. Ceci est illustré dans la figure ci-dessous :

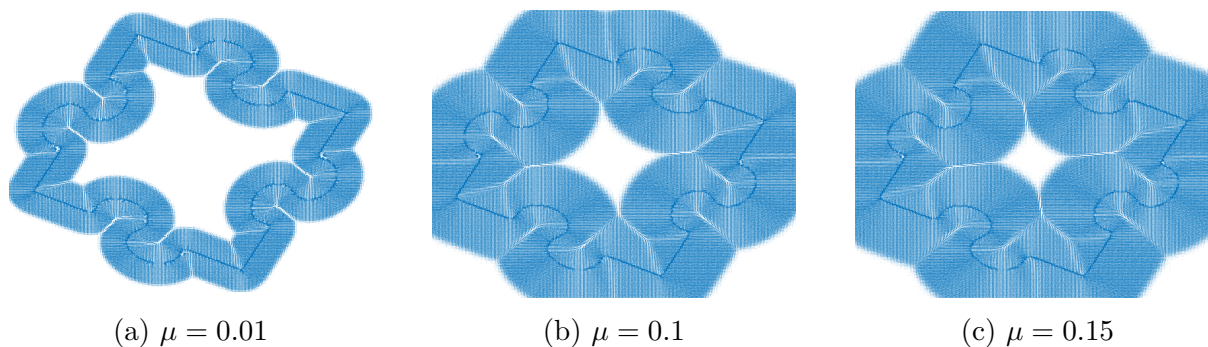


FIG. 3.16 : Résultat du champs de forces externes avec 3 valeurs différentes de μ

Comment cela affecte-t-il la segmentation ? Lorsque le champ de forces externes est concentré près de l'objet, initialiser un contour actif loin de l'objet ne permettra pas de segmenter l'objet souhaité correctement. En revanche, un champ de forces externes plus étendu, obtenu avec une valeur plus grande de μ , permet de mieux guider le contour actif vers l'objet même s'il est initialisé à une distance plus grande.

La figure 3.17 illustre le résultat de la segmentation avec $\mu = 0.01$:

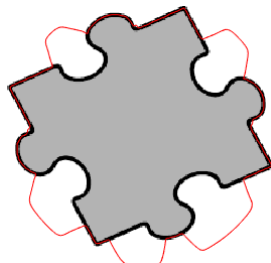
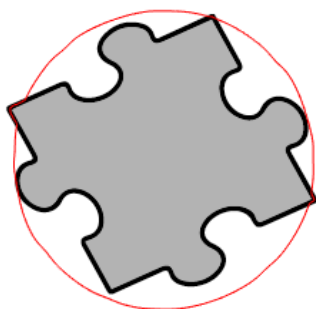


FIG. 3.17 : Résultat de la segmentation avec GVF sur l'image "Puzzle" avec $\mu = 0.01$

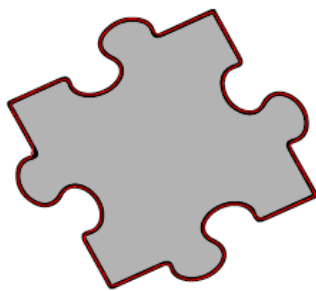
- **Paramètres α et β :**

Comme mentionné au chapitre 2, et comme le montre l'équation 2.3, la dérivée du premier ordre $v'(s)$ contrôle l'étirement du snake, tandis que la deuxième dérivée $v''(s)$ contrôle la flexion. Les paramètres α et β sont les poids qui contrôlent respectivement l'élasticité et la rigidité.

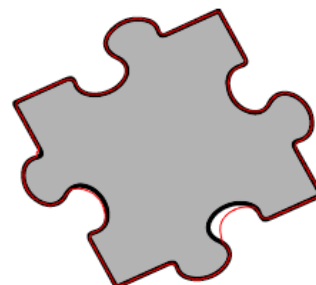
La figure 3.18 illustre les résultats de la segmentation pour différentes valeurs de α :



(a) $\alpha = 0.09$



(b) $\alpha = 0.1$



(c) $\alpha = 0.2$

FIG. 3.18 : Résultat de la segmentation avec GVF sur l'image "Puzzle" avec 3 valeurs différentes de α

Lorsque α est trop petit, le contour n'aura pas assez d'élasticité pour épouser le contour de l'objet à segmenter. À l'inverse, lorsque α est trop grand, le contour converge trop rapidement et ne suit plus correctement les forces externes.

La figure 3.19 illustre les résultats de la segmentation pour différentes valeur de β :

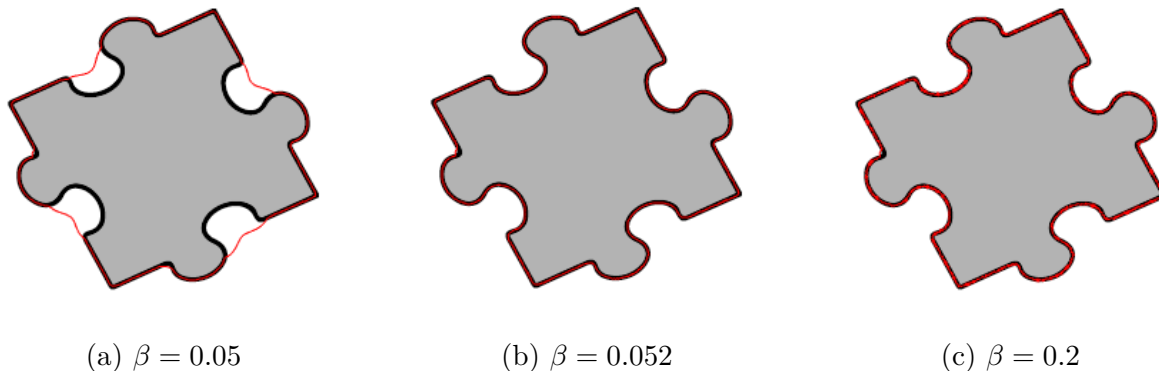


FIG. 3.19 : Résultat de la segmentation avec GVF sur l'image "Puzzle" avec 3 valeurs différentes de β

β est le paramètre à ajuster lorsque l'objet à segmenter possède des concavités et des angles. Lorsqu'il est trop petit, le contour n'aura pas assez de flexion pour segmenter les concavités ou les angles droits et peut demander plus d'itérations pour converger correctement. En revanche, lorsque β est trop grand, le contour converge vers l'objet, mais la ligne du contour actif devient brisée.

- **Tolérance tol :**

Pour des images synthétiques, le changement de la valeur de tol n'affecte pas le résultat final de la segmentation, mais bien le nombre d'itérations nécessaire pour la convergence.

Pour des images plus complexes, la valeur de la tolérance peut avoir un effet sur la précision de la segmentation, et nous le choisissons de manière empirique.

3.4.3 Résultats sur des images médicales

Après avoir bien compris l'importance du réglage des différents paramètres et de l'initialisation du contour actif pour la méthode GVF, nous appliquons dans cette section la segmentation par GVF aux images médicales présentées précédemment, à savoir les IRM du cerveau et du sein.

3.4.3.1 Sur l'IRM d'un cerveau

La figure 3.20 illustre les résultats obtenus en appliquant la méthode GVF sur des images médicales filtrées.

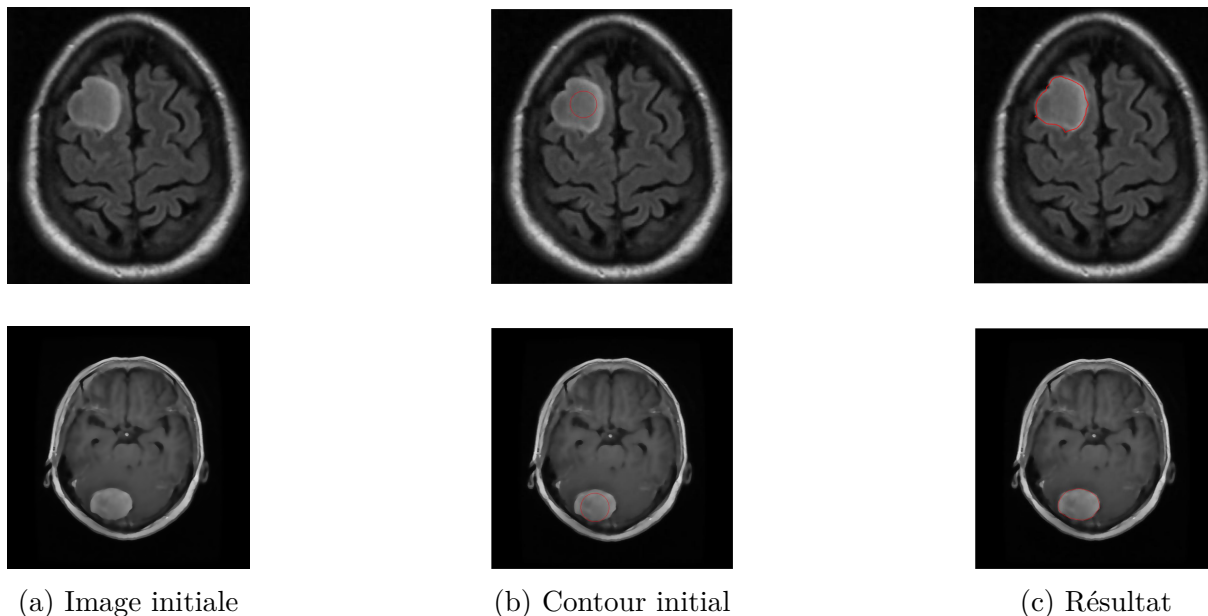


FIG. 3.20 : Résultat de la segmentation avec GVF Sur l'IRM d'un cerveau, $\mu = 0.15$, $\alpha = 0.1$, $\beta = 0.06$, $tol = 0.004$, $it_{max} = 300$

3.4.3.2 Sur l'IRM d'un sein

Les résultats de la segmentation de l'IRM du sein en utilisant la méthode GVF des contours actifs n'ont pas été concluants. Cela est dû à la complexité de l'image à segmenter et de l'objet d'intérêt. En effet, les IRM du sein présentent des variations d'intensité et des textures complexes qui rendent difficile l'application de la méthode GVF. De plus, la présence de bruit et de structures anatomiques similaires perturbe le champ de forces externes, entraînant une convergence incorrecte des contours actifs. En conséquence, les contours obtenus ne correspondent pas de manière précise aux limites réelles des structures d'intérêt.

3.5 Simulation des modèles des Courbes de niveau sur MATLAB

L'évolution des méthodes de segmentation d'images médicales a vu émerger une transition significative des contours actifs paramétriques (Snakes) vers les courbes de niveaux (Levels Set). Cette évolution est motivée par la nécessité de surmonter les limitations inhérentes aux contours actifs, notamment leur sensibilité à la position initiale du contour et leur difficulté à gérer les changements de topologie des courbes. Les courbes de niveaux offrent une approche géométrique robuste, permettant une représentation efficace des contours tout en résolvant le problème des changements de topologie. Cette évolution reflète ainsi une quête constante d'amélioration des techniques de segmentation pour répondre aux exigences croissantes de l'imagerie médicale.

- Le code utilisé pour l'algorithme DRLSE est tiré de la référence suivante [55].

- Le code utilisé pour l'algorithme LSE-BFE est tiré de la référence suivante [39].

3.5.1 Description de l'algorithme du modèle DRLSE

À travers cet organigramme (figure 3.21), nous allons résumer la théorie de l'algorithme DRLSE en décrivant son fonctionnement étape par étape.

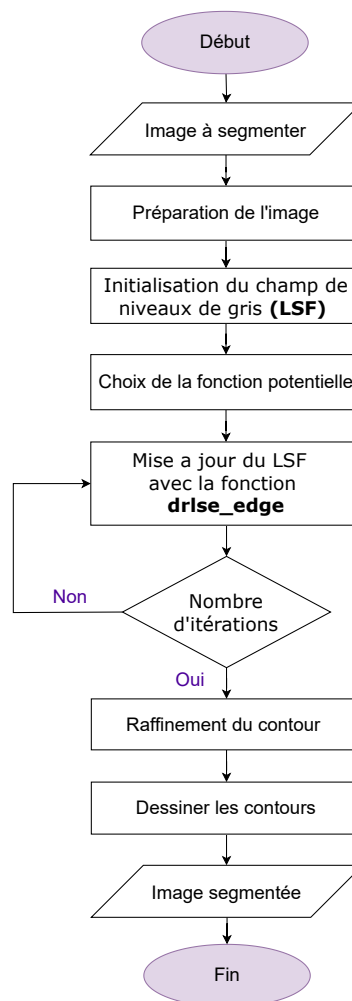


FIG. 3.21 : Organigramme de la méthode DRLSE

Après avoir présenté un organigramme du flux de travail, nous allons examiner les étapes importantes, notamment l'initialisation du champ de niveau (Level Sets Fonction, LSF), le choix de la fonction potentielle, la boucle principale d'itération externe et le raffinement des contours finaux. Chaque étape sera expliquée en mettant en évidence les fonctions utilisées et leur rôle dans le processus de segmentation.

3.5.1.1 Initialisation du champ de niveau de gris (LSF)

- La LSF est initialisée comme une fonction de niveau binaire, où une région rectangulaire est définie comme la région initiale à évoluer.
- **Possibilité d'initialiser plusieurs contours à la fois :**
 - On peut initialiser plusieurs contours en définissant plusieurs régions initiales avec différentes valeurs dans la matrice initialLSF.

```
c0 = 2;  
initialLSF = c0 * ones(size(Img));  
initialLSF(40:80, 40:150) = -c0;  
% Pour ajouter un deuxième contour, définir une autre région :  
initialLSF(100:150, 100:200) = -c0;  
phi = initialLSF;
```

3.5.1.2 Boucle principale des itérations externes

- Une boucle principale est utilisée pour faire évoluer la courbe de niveau sur un certain nombre d'itérations externes.
- À chaque itération, la fonction `drlse_edge` met à jour la LSF en utilisant les équations du modèle DRLSE, qui impliquent plusieurs termes d'énergie.
- Ces termes d'énergie comprennent le terme de longueur pondérée, le terme de régularisation de distance et le terme de surface pondérée.
- En combinant ces termes avec la fonction indicatrice de bord, la LSF est régularisée et évolue vers les contours des objets dans l'image.

```
for n = 1:iter_outer  
    phi = drlse_edge(phi, g, lambda, mu, alpha, epsilon, timestep,  
                    iter_inner, potentialFunction);  
end
```

- α est un paramètre clé qui influence la manière dont la courbe de niveau évolue dans le temps. En ajustant α , on peut contrôler la tendance de la courbe à s'étendre ou à se contracter, ce qui permet de capturer plus précisément les contours des objets dans l'image.

3.5.1.3 Boucle interne d'itération

- À l'intérieur de la fonction `drlse_edge`, une boucle interne itère plusieurs fois pour affiner la courbe de niveau.
- Cela permet d'améliorer la précision de la segmentation en capturant les détails fins et les contours complexes de l'objet.

3.5.1.4 Choix de la fonction potentielle

- Le choix de la fonction potentielle affecte la régularisation de la courbe de niveau.
- Le potentiel à puits simple est bon pour les modèles basés sur les régions.
- Le potentiel à double puits est efficace pour les modèles basés à la fois sur les bords et les régions.

```
potential = 2;
if potential == 1
    potentialFunction = 'single-well';
elseif potential == 2
    potentialFunction = 'double-well';
else
    potentialFunction = 'double-well';
end
```

Cette segmentation nécessite l'utilisation de diverses fonctions pour contrôler l'évolution du contour et évaluer la proximité des pixels par rapport à ce contour.

- La fonction **div** calcule la divergence d'un champ de vecteurs bidimensionnel. Voici le script pour cette fonction :

```
function f = div(nx, ny)
    [nxx, junk]=gradient(nx);
    [junk, nyy]=gradient(ny);
    f=nxx+nyy;
end
```

- La fonction **distReg_p2(phi)** calcule le terme de régularisation de distance utilisant le potentiel à double puits de l'équation 2.70. Voici le script pour cette fonction :

```
function f = distReg_p2(phi)
    [phi_x, phi_y]=gradient(phi);
    s=sqrt(phi_x.^2 + phi_y.^2);
    a=(s>=0) & (s<=1);
    b=(s>1);
    ps=a.*sin(2*pi*s)/(2*pi)+b.*(s-1);
    dps=((ps~=0).*ps+(ps==0))./((s~=0).*s+(s==0));
    f = div(dps.*phi_x - phi_x, dps.*phi_y - phi_y) +
    4*del2(phi);
end
```

- La fonction **Dirac** est utilisée pour quantifier la proximité des pixels par rapport au contour. Elle fournit une réponse brève à la localisation du contour en indiquant les changements brusques dans la fonction Heaviside, ce qui aide à déterminer les frontières des régions dans l'image (son expression est l'équation 2.61).

- La fonction **NeumannBoundCond** est utilisée pour imposer les conditions aux limites de Neumann sur le champ de niveaux de gris, garantissant ainsi la stabilité numérique du processus.

Voici le script pour cette fonction :

```
function g = NeumannBoundCond(f)
    [nrow, ncol] = size(f);
    g = f;
    g([1 nrow], [1 ncol]) = g([3 nrow-2], [3 ncol-2]);
    g([1 nrow], 2:end-1) = g([3 nrow-2], 2:end-1);
    g(2:end-1, [1 ncol]) = g(2:end-1, [3 ncol-2]);
end
```

3.5.2 Résultats du modèle DRLSE sur des images synthétiques

- **Segmentation d'un caractère concave**

Nous avons également testé le modèle DRLSE sur une image synthétique représentant le caractère "U" concave. Cette étape démontre la capacité de cette méthode à segmenter efficacement et avec plus de précision des formes concaves.

La figure 3.22 illustre le résultat de la segmentation :

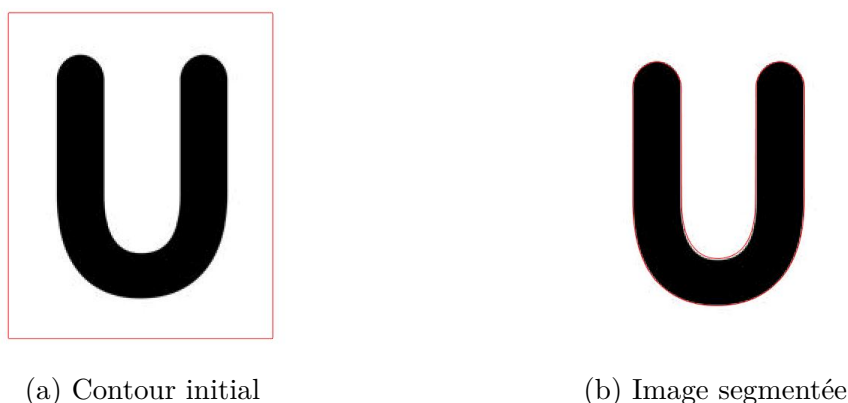


FIG. 3.22 : Résultat de la segmentation avec l'algorithme DRLSE sur une image synthétique U'

Le résultat de la segmentation de cette image démontre la supériorité du modèle DRLSE en termes de prise en compte des **concavités** par rapport aux différents modèles de contours actifs.

- **Segmentation d'une seule forme avec 2 contours initiaux**

Pour illustrer davantage la polyvalence de l'algorithme DRLSE, nous l'avons appliqué à une image représentant un puzzle. Dans ce cas, nous avons initialisé deux contours rectangulaires à l'intérieur du puzzle, correspondant à deux morceaux adjacents du puzzle. L'algorithme a fusionné ces contours pour segmenter l'ensemble du puzzle en fonction des similitudes de texture et de couleur entre les pièces adjacentes.

La figure 3.23 illustre le résultat de la segmentation :

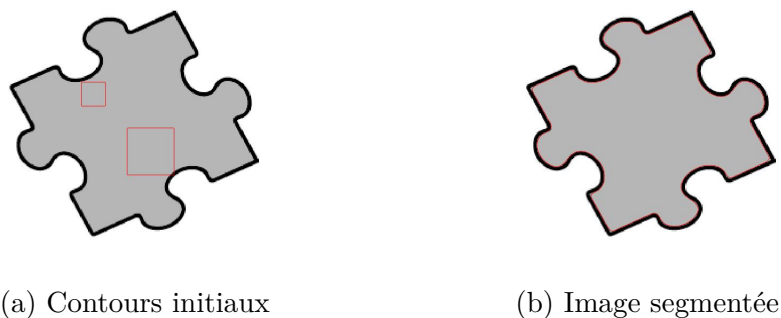


FIG. 3.23 : Résultat de la segmentation avec l'algorithme DRLSE sur l'image d'un puzzle

- **Segmentation de plusieurs formes avec un seul contour**

Dans cette étape, nous avons élargi notre analyse en testant la méthode de segmentation sur une image synthétique contenant **plusieurs formes géométriques distinctes**. L'initialisation du contour consistait en un grand contour extérieur englobant toutes les formes.

La figure 3.24 illustre le résultat de la segmentation :

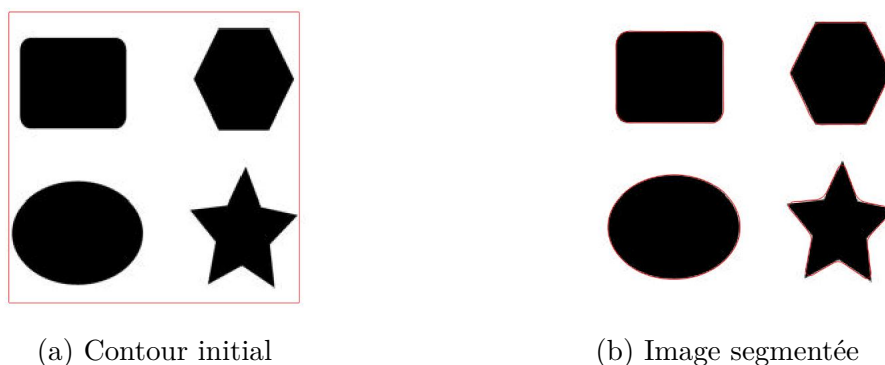


FIG. 3.24 : Résultat du modèle DRLSE sur une image synthétique contenant plusieurs formes géométriques

- Malgré cette initialisation globale, l'algorithme a convergé vers la segmentation précise de chaque forme individuelle, mettant en évidence sa robustesse dans des scénarios complexes de segmentation d'images.

Il est important de noter que bien que l'algorithme DR-LSE soit efficace pour segmenter avec précision des objets concaves et plusieurs parties distinctes dans une image, il peut rencontrer des difficultés avec la représentation précise des angles droits. En effet, le processus de régularisation tend à arrondir les angles aigus, ce qui peut entraîner une légère perte de précision dans la représentation des structures anatomiques comportant des contours nets. Cependant, malgré cette limitation, l'algorithme reste prometteur et peut être adapté avec des ajustements appropriés pour répondre à divers besoins de segmentation, en particulier dans le domaine médical où la détection précise des structures anatomiques est essentielle.

3.5.3 Résultats du modèle DRLSE sur des images médicales

Nous allons présenter les résultats de la segmentation des images médicales en utilisant l'algorithme DRLSE. Les images médicales présentent souvent des défis uniques en termes de variation de contraste, de bruit et de complexité des structures anatomiques. Nous évaluons l'efficacité de cette méthode dans la segmentation de ces images, en ajustant notamment les paramètres pour observer leurs impacts sur les résultats de segmentation.

3.5.3.1 Réglages des paramètres α , λ et μ

- **Réglage du paramètre α**

La figure 3.25 présente les résultats de l'algorithme DRLSE sur une image médicale en modifiant les valeurs de α :

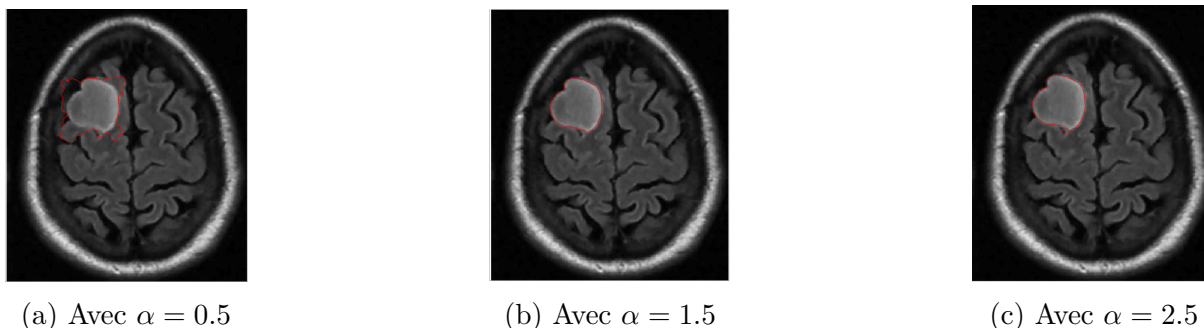


FIG. 3.25 : Résultats de l'algorithme DRLSE sur une image médicale avec différentes valeurs de α avec 100 itérations

Une valeur plus faible de α (0.5) réduit la force d'attraction ou de répulsion de la surface, ce qui signifie que la fonction de niveau évoluera plus lentement vers l'intérieur ou l'extérieur. Cela conduit à une segmentation plus conservatrice, où les régions segmentées sont plus petites et suivent de plus près les zones initialement définies. Avec une valeur intermédiaire de α (1.5), l'attraction ou la répulsion est augmentée, favorisant une expansion ou contraction plus rapide des régions segmentées, ce qui est utile pour des objets plus grands ou des cas nécessitant une segmentation plus large. Une valeur encore plus élevée de α (2.5) intensifie cet effet, permettant une segmentation très expansive ou contractive, adaptée aux objets très grands ou aux régions qui nécessitent une délimitation plus agressive.

Il est important de noter que la valeur de α doit être positive lorsque le contour est à l'extérieur de l'objet et négative lorsque le contour est à l'intérieur de l'objet. Cela garantit que la force exercée par le paramètre α pousse le contour dans la direction correcte pour atteindre une segmentation précise.

- **Réglage du paramètre λ**

La figure 3.26 présentent les résultats de l'algorithme DRLSE sur une image médicale en modifiant les valeurs de λ :

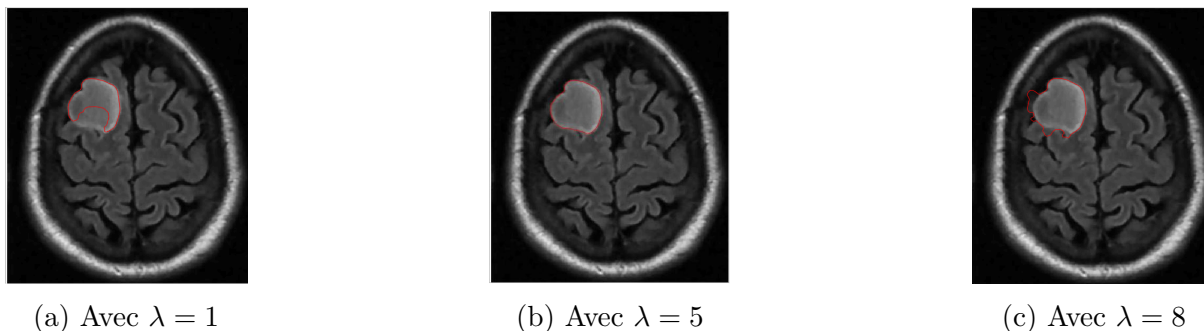


FIG. 3.26 : Résultats de l'algorithme DRLSE sur une image médicale avec différentes valeurs de λ avec 100 itérations

Une valeur faible de $\lambda(1)$ permet au contour de suivre les détails fins et le bruit de l'image, entraînant des contours plus irréguliers, adaptés aux objets aux bords complexes. Une valeur moyenne de λ (5) favorise des contours plus lisses et réguliers, adaptés aux objets aux bords clairs et bien définis. Une valeur élevée de λ (8) produit des contours très lisses et épurés, idéaux pour des segments où la précision des bords est essentielle et le bruit doit être largement ignoré.

- **Réglage du paramètre μ**

Ci dessous les résultats de l'algorithme DRLSE sur une image médicale en modifiant les valeurs de μ :

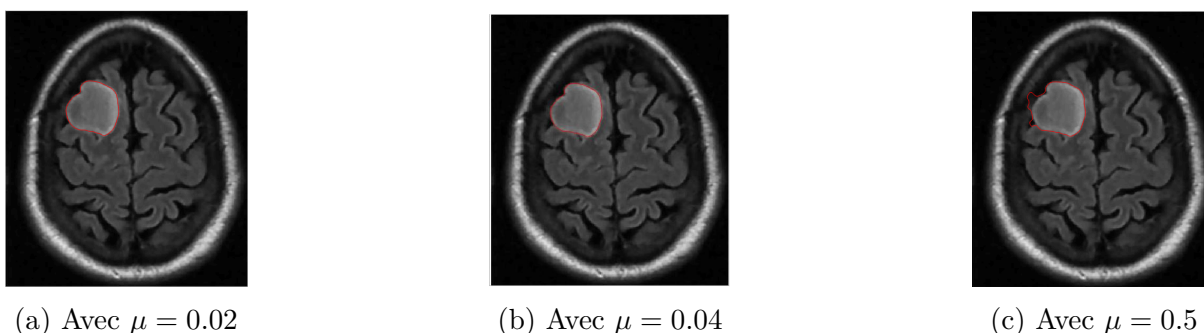


FIG. 3.27 : Résultats de l'algorithme DRLSE sur une image médicale avec différentes valeurs de μ avec 100 itérations

Avec une valeur faible de μ (0.02), la régularisation de distance est minimale, permettant des variations plus abruptes dans la fonction de niveau. Cela peut accélérer l'adaptation aux variations locales de l'image mais risque d'introduire de l'instabilité et des artefacts. Une valeur intermédiaire de μ (0.04) offre un compromis équilibré entre stabilité et adaptabilité, assurant une fonction de niveau lisse tout en permettant une certaine flexibilité pour suivre les variations locales. Une valeur élevée de μ (0.5) implique une forte régularisation de distance, maintenant la fonction de niveau très régulière et évitant les changements abrupts. Cependant, cela conduit à une évolution plus lente de l'algorithme, limitant son adaptabilité locale mais assurant une convergence robuste.

3.5.3.2 Sur l'IRM d'un cerveau

Pour segmenter ces images, nous avons ajusté de manière empirique les paramètres α , λ et μ afin de trouver la combinaison optimale qui permet de segmenter efficacement les images. Après plusieurs essais et ajustements, la figure 3.28 présente les résultats de la segmentation :

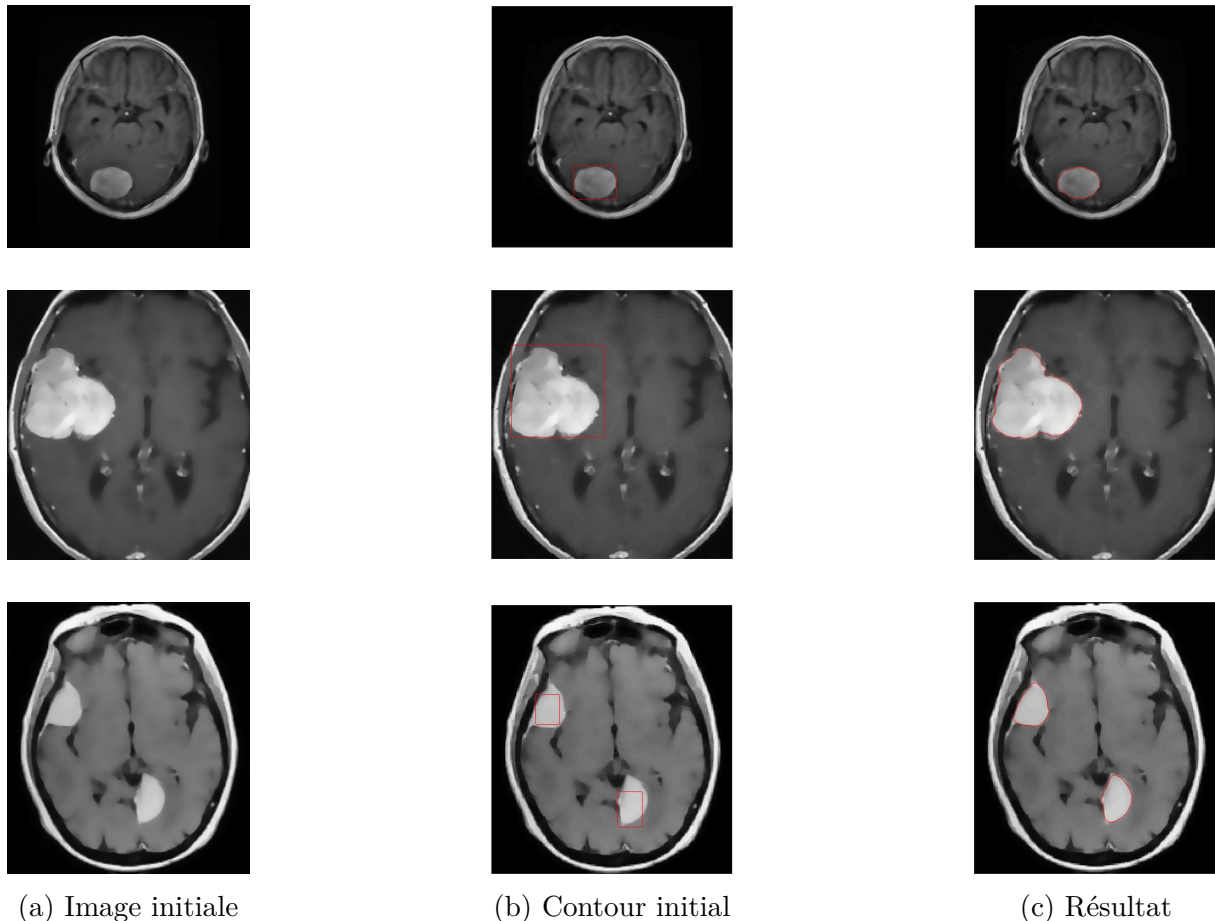


FIG. 3.28 : Résultats de l'algorithme DRLSE sur des IRM du cerveau $\alpha = 0.5$, $\lambda = 6$ et $\mu = 0.02$

3.5.3.3 Sur l'IRM d'un sein

Pour segmenter cette image, nous avons appliqué la même méthode. La figure 3.29 montre le résultat de la segmentation.

Le résultat obtenu avec l'algorithme DRLSE sur la nouvelle image médicale n'a pas été satisfaisant. Malgré plusieurs ajustements des paramètres, la segmentation n'a pas réussi à délimiter clairement les structures d'intérêt dans l'image. Ce résultat nous a conduit à explorer d'autres approches pour améliorer la qualité de la segmentation. Nous avons donc décidé d'utiliser un autre algorithme, que nous présentons dans la section suivante

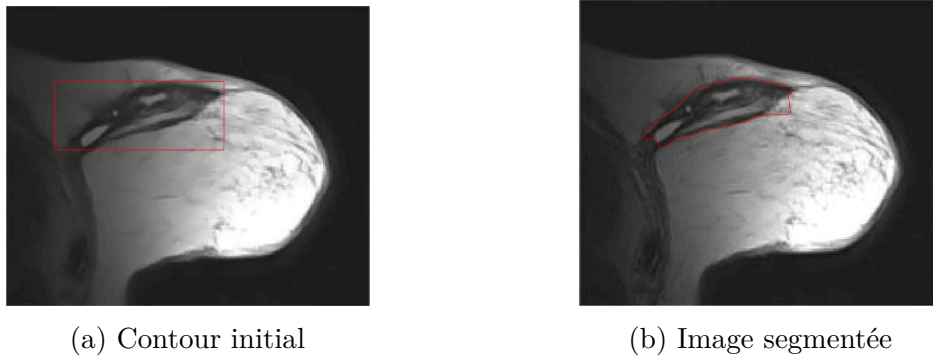


FIG. 3.29 : Résultat de segmentation avec l'algorithme DRLSE sur l'IRM du sein avec $\alpha = 1.5$, $\lambda = 5$ et $\mu = 0.02$

3.5.4 Description de l'algorithme du modèle LSE-BFE

Pour obtenir une vue d'ensemble des étapes du modèle LSE-BFE, voici un organigramme (figure 3.30) détaillant le processus, depuis le traitement initial de l'image jusqu'à l'acquisition des images segmentées.

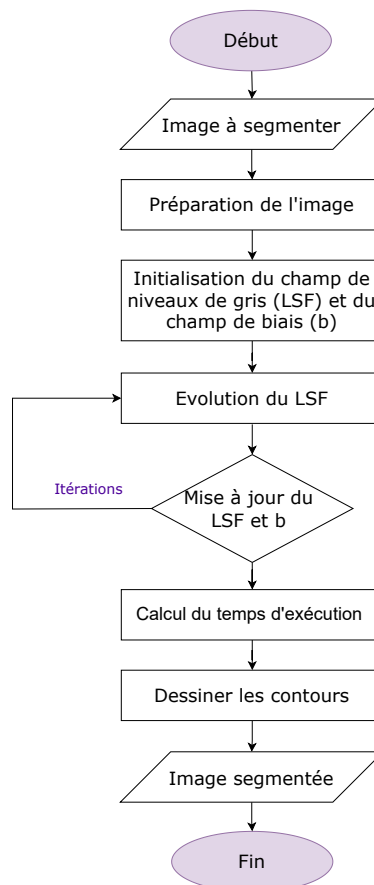


FIG. 3.30 : Organigramme du modèle LSE-BFE

Après avoir présenté un organigramme détaillé du flux de travail de l'algorithme, nous passerons en revue les étapes importantes, notamment l'initialisation du champ de niveau de gris (LSF), la boucle principale d'itération externe et la segmentation de l'image en régions distinctes. Chaque étape sera expliquée en mettant en évidence les fonctions utilisées et leur rôle dans le processus de segmentation.

3.5.4.1 Initialisation du champ de niveau de gris (LSF) et du biais (b)

- Le champ de niveau de gris (LSF) est une représentation de la frontière d'intérêt dans l'image. Dans ce code, le LSF est initialisé de manière aléatoire pour chaque pixel de l'image. Cela crée une carte de contours initiale à partir de laquelle la segmentation commencera à évoluer.
- De plus, le masque d'entrée est utilisé pour initialiser le LSF. Le masque définit les régions d'intérêt dans l'image. À l'intérieur de ces régions, les valeurs du LSF sont définies à 1, tandis qu'à l'extérieur, les valeurs sont définies à -1. Cela permet d'orienter les contours vers les régions d'intérêt dès le début du processus.
- Le champ de biais est initialisé à une matrice remplie de 1.
- L'initialisation est effectuée de la manière suivante :

```
% Initialisation du LSF
initialLSF (:, :, 1) = randn(size(Img));
initialLSF (:, :, 2) = randn(size(Img));
initialLSF (:, :, 1) = Mask;
%Initialisation du b
b=ones(size(Img));
```

3.5.4.2 Boucle principale des itérations externes

- Une boucle itérative est utilisée pour faire évoluer les contours de manière itérative.
- À chaque itération, les champs de niveau de gris et de biais sont mis à jour en utilisant la fonction `lse_bfe_3Phase`. Cette fonction effectue une estimation du champ de biais et une évolution du contour en considérant trois phases distinctes dans l'image.
- La troisième phase est ajoutée pour traiter les régions extérieures à l'objet d'intérêt. L'utilisation de trois phases permet une segmentation plus précise en considérant les pixels qui ne sont ni à l'intérieur ni à l'extérieur de l'objet.
- Voici le script de la boucle principale :

```
for n = 1:Iter_outer
    % Mise à jour des champs de niveau de gris et de biais
    [u, b, C] = lse_bfe_3Phase(u, Img, b, Ksigma, KONE, nu,
        timestep, mu, epsilon, Iter_inner);
end
```

- Dans cette boucle, **Iter_outer** contrôle le nombre total d'itérations effectuées pour faire évoluer les contours. La variable **Iter_inner** spécifie le nombre d'itérations internes effectuées à chaque itération externe de la boucle. Ces deux paramètres sont essentiels pour déterminer la précision et la convergence de la segmentation.

3.5.4.3 Boucle interne d'itération

- Dans la fonction **lse_bfe_3Phase**, une boucle interne est utilisée pour itérer plusieurs fois afin de raffiner l'évolution du contour dans chaque phase. Cela permet de mieux capturer les détails et les contours complexes de l'objet dans l'image.

3.5.4.4 Segmentation de l'image en régions distinctes

- Une fois les itérations terminées, les résultats obtenus à partir des champs de niveaux de gris et de biais sont utilisés pour segmenter l'image en régions distinctes. Les différentes régions de l'image sont déterminées en fonction des valeurs de niveaux de gris obtenues à partir des champs de niveaux de gris finaux.
- Voici le script pour la segmentation de l'image en régions distinctes :

```
% Segmentation de l'image en régions distinctes  
Img_seg = C(1)*M1 + C(2)*M2 + C(3)*M3;
```

- Dans cette équation, les coefficients $C(1)$, $C(2)$ et $C(3)$ représentent les pondérations pour chaque région segmentée, tandis que M_1 , M_2 et M_3 représentent les fonctions d'appartenance correspondantes. Ces fonctions sont obtenues à partir des champs de niveaux de gris finaux en utilisant la fonction de **Heaviside** pour quantifier la proximité des pixels par rapport au contour final. En multipliant chaque fonction par le coefficient de pondération correspondant, nous obtenons une image segmentée où chaque région est pondérée selon son coefficient respectif.
- Le processus complet de segmentation par les courbes de niveaux implique l'utilisation de différentes fonctions pour régulariser l'évolution du contour et quantifier la proximité des pixels par rapport au contour :
 - La fonction **Heaviside** régularise l'évolution du contour en fournissant une transition douce entre les régions de l'image. Cette fonction quantifie la proximité des pixels par rapport au contour et permet de déterminer dans quelle mesure chaque pixel appartient à chaque région respective (son expression est donnée dans l'équation 2.86).
 - Les fonctions **Dirac** et **NeumannBoundCond** sont les mêmes que celles de l'algorithme DRLSE.

3.5.5 Résultats du modèle LSE-BFE sur des images synthétiques

- Nous avons également évalué les performances de l'algorithme LSE-BFE, un autre algorithme de courbes de niveaux.
- Nos résultats ont révélé que l'algorithme LSE-BFE a également bien segmenté les images synthétiques, même avec une initialisation aléatoire des contours.

La figure 3.31 illustre les résultats de la segmentation :

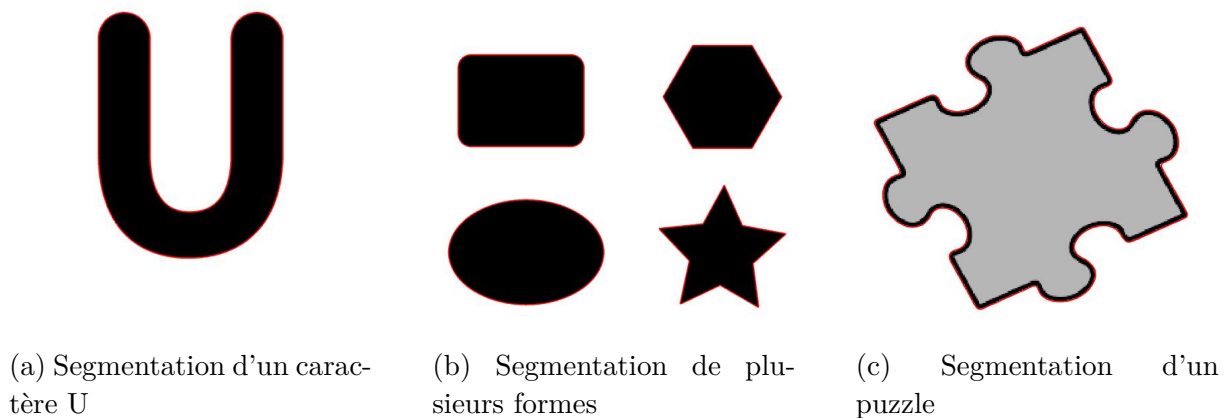


FIG. 3.31 : Résultat de la segmentation avec le modèle LSE-BFE sur des images synthétiques

- Dans la section suivante, nous présentons les résultats de l'application de l'algorithme LSE-BFE aux mêmes images utilisées pour évaluer l'algorithme DRLSE. Les résultats obtenus mettent en évidence la robustesse et l'efficacité de l'algorithme LSE-BFE, même dans des conditions d'initialisation aléatoire des contours.

3.5.6 Résultats du modèle LSE-BFE sur des images médicales

Maintenant, nous allons appliquer l'algorithme LSE-BFE sur les images médicales. Les figures 3.32 et 3.33 montrent les résultats de la segmentation :

3.5.6.1 Sur l'IRM d'un cerveau

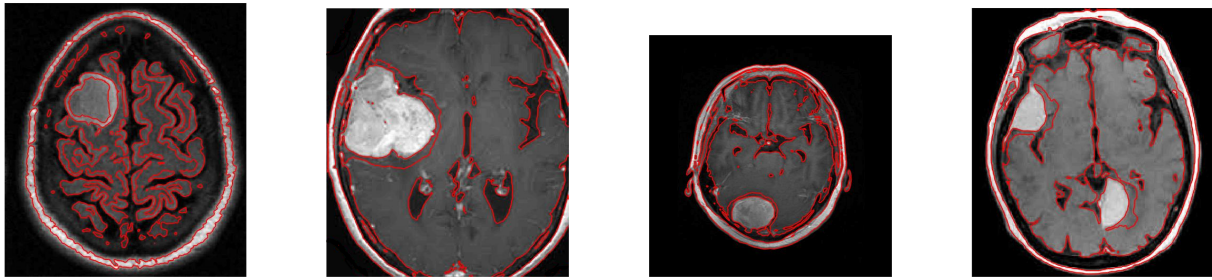


FIG. 3.32 : Résultat de la segmentation avec le modèle LSE-BFE sur des IRM du cerveau

3.5.6.2 Sur l'IRM d'un sein

le champ de biais représente une fonction spatiale utilisée pour modéliser les variations locales d'intensité au sein d'une image. Ce champ est essentiel pour corriger les inhomogénéités d'intensité, telles que celles dues à un éclairage non uniforme ou à des artefacts d'acquisition, qui peuvent compromettre la précision de la segmentation. En estimant et en appliquant ce champ de biais, l'algorithme vise à normaliser l'intensité des pixels à travers l'image, facilitant ainsi une segmentation plus précise et cohérente des objets d'intérêt. Cette correction aide à définir plus clairement les contours des structures anatomiques ou des objets présents dans l'image, améliorant ainsi la qualité globale de la segmentation obtenue par la méthode LSE BFE.

La Figure 3.33 illustre le champ de biais final estimé par l'algorithme LSE BFE.

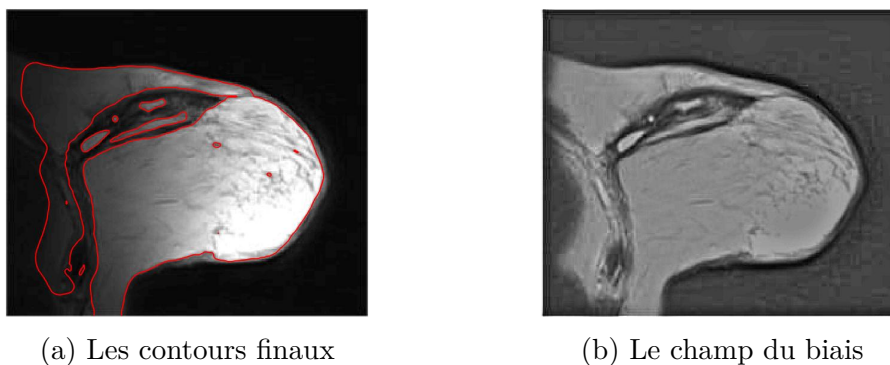


FIG. 3.33 : Résultat de la segmentation avec le modèle LSE-BFE sur des IRM du sein

L'algorithme LSE-BFE s'est révélé être le plus robuste parmi tous les algorithmes utilisés. Sa principale force réside dans son indépendance par rapport à l'initialisation, ce qui signifie qu'il segmente efficacement sans nécessiter des conditions initiales spécifiques. De plus, il est capable de segmenter plusieurs parties simultanément, offrant une solution polyvalente pour des images médicales complexes. Enfin, il demande peu de réglages, ce qui simplifie son utilisation et améliore sa fiabilité dans différentes conditions.

3.6 Conclusion

En conclusion, nos résultats confirment une fois de plus la performance supérieure des algorithmes de courbes de niveaux, en particulier l'algorithme basé sur les régions "LSF-BFE", pour la segmentation d'images complexes, notamment dans le contexte des images médicales. Nous soulignons spécifiquement que l'algorithme "LSF-BFE" a démontré une précision accrue dans la délimitation des contours, offrant ainsi des résultats plus fiables et plus cohérents.

Par ailleurs, notre étude approfondie des calculs nécessaires pour ces algorithmes a révélé leur exigence en termes de calcul, ce qui a suscité un intérêt croissant pour des recherches et des travaux contributifs visant à leur implémentation sur des dispositifs FPGA (Field-Programmable Gate Array). Nous observons ainsi une ouverture vers de nouvelles perspectives dans le domaine de l'accélération matérielle pour la segmentation d'images, avec un intérêt particulier pour l'utilisation d'un SoC-FPGA (System On Chip).

Dans le prochain chapitre, nous présenterons l'implémentation concrète d'un algorithme de contour actif "GVF" et des algorithmes de courbe de niveau "DRLSE" et "LSF-BFE" sur un SoC-FPGA. Cette démarche vise à explorer les avantages potentiels de l'accélération matérielle pour ces algorithmes et à étudier leur performance dans un environnement matériel.

Chapitre 4

Implémentation sur FPGA

4.1 Introduction

Dans ce chapitre, nous détaillons les étapes de l'implémentation des algorithmes vus au chapitre 3 à savoir : les contours actifs par la méthode GVF, les courbes de niveaux par les approches contours DRLSE et région LSE BFE, depuis le codage des algorithmes en langage C jusqu'à la récupération du résultat final, qui est l'image segmentée. Nous présentons également les outils et matériels utilisés au cours de ce processus.

Pour chacun de ces algorithmes, nous avons adopté deux approches d'implémentation : par parties et complète. Nous comparons ensuite les ressources matérielles utilisées pour chaque approche ainsi que le temps d'exécution. De plus, les résultats de la segmentation obtenus sont comparés aux résultats obtenus sur MATLAB, permettant une évaluation complète de l'efficacité et des performances de nos implémentations.

Ce chapitre est structuré de manière à fournir une compréhension claire et détaillée du processus d'implémentation, en soulignant les défis rencontrés et les solutions adoptées pour optimiser les performances et l'utilisation des ressources.

4.2 Logiciels et matériels

Il est essentiel de définir les outils logiciels et matériels utilisés pour la conception et l'implémentation des systèmes. Ces outils seront présentés **Annexe 2**.

4.3 Étapes de l'implémentation

L'approche que nous envisageons d'utiliser est l'approche de Synthèse de Haut Niveau, qui est utilisée lorsque la description de l'architecture matérielle est assez difficile à réaliser directement en VHDL (Very High Speed Integrated Circuit Hardware Description Language)[56].

Voici les étapes que nous avons suivies lors de l'implémentation :

4.3.1 Codage en C et simulation

- Programmer la fonction ou le sous-système à implémenter sur FPGA en utilisant le langage C, sur **Visual Studio Code (VS code)**.
- Exécuter la simulation pour s'assurer que le programme fonctionne correctement, en comparant les résultats du code avec ceux obtenus sous MATLAB.

4.3.2 Création des blocs IP (Intellectual Property)

- Ajouter les fichiers sources C dans notre projet sur **Vitis HLS (High Level Synthesis)**, et configurer les interfaces pour les ports de notre fonction (par exemple, AXI, BRAM, etc.).

- Synthétiser notre code en **RTL (Register Transfer Level)** et vérifier les rapports générés pour s'assurer que la synthèse s'est déroulée correctement.
- Utiliser des directives (**pragma HLS**) pour optimiser la performance, l'utilisation des ressources et la latence.
- Exporter le fichier **RTL** pour générer le **bloc IP** que nous utilisons par la suite sur **Vivado**.
- Refaire ces étapes pour chaque **bloc IP** que nous souhaitons créer.

4.3.3 Implémentation et Déploiement

- En utilisant **Vivado**, créer un **Block Design** pour le système principal qui sera déployé sur le FPGA, incluant les blocs IP générés précédemment.
- Valider le Block Design et créer un **HDL Wrapper** qui convertit le schéma de l'architecture en programme **HDL (Hardware Description Language)**.
- Exécuter la simulation pour s'assurer que l'architecture fonctionne correctement.
- Synthétiser le projet, puis réaliser l'implémentation et générer le fichier binaire "**Bitstream**".
- Programmer la carte FPGA à l'aide du fichier **Bitstream** généré et vérifier le bon fonctionnement du système.

La Figure 4.1 représente un schéma synoptique simplifié de l'approche.

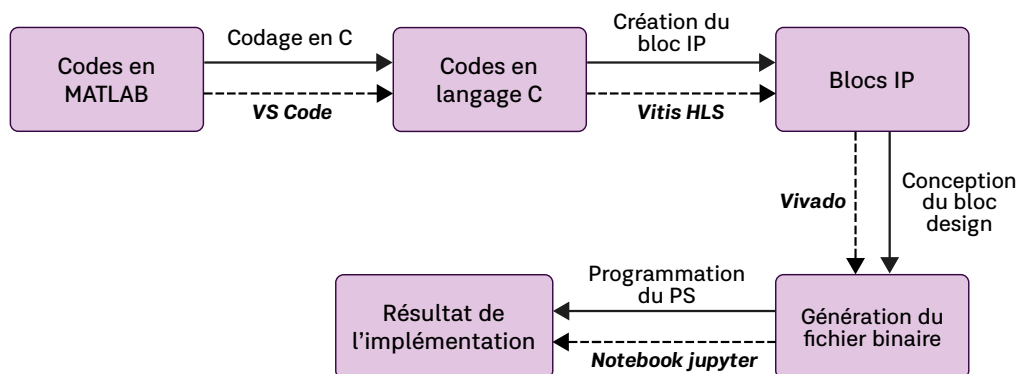


FIG. 4.1 : Schéma synoptique simplifié de l'approche HLS

4.4 Implémentation de l'algorithme GVF

Dans cette section, nous allons expliquer les étapes de l'implémentation de l'algorithme GVF. Étant donné les ressources nécessaires pour la méthode GVF et les calculs

onéreux qu'elle exige, nous avons adopté deux stratégies. Nous avons d'abord réalisé une implémentation par parties, qui a ensuite conduit à une implémentation complète de l'algorithme.

Nous allons maintenant détailler les étapes mentionnées dans le schéma synoptique ci-dessus pour l'algorithme GVF.

4.4.1 Implémentation par parties

Nous débuterons en mettant en œuvre l'approche par parties.

4.4.1.1 Codage en C

Nous avons développé les codes en C en nous basant sur les fonctions MATLAB.

Voici un exemple de traduction en langage C de la fonction MATLAB qui calcule la carte de contours f .

Code Matlab :

```
f = imfilter(I, fspecial('log', [11, 11], 0.2)) .^ 2;
```

Code en langage C (figure 4.2) :

```
// Fonction pour calculer la carte de contours, f = (conv(Image, Laplacien(Gaussienne))).^2
void EdgeMap(int image[L], float resultat[L], int largeur_image, int hauteur_image) {
    int i, j, m, n;
    float sum;
    //Filtre Laplacien de la gaussienne avec sigma = 0.2
    float filtreLapGauss[11][11] = {{ 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4153, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4153, -49.5861, 0.4153, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4153, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131},
    { 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131, 0.4131}};

    // Parcourir l'image
    for (i = 0; i < hauteur_image; ++i) {
        for (j = 0; j < largeur_image; ++j) {
            sum = 0;
            // Parcourir le filtre
            for (m = 0; m < 11; ++m) {
                for (n = 0; n < 11; ++n) {
                    int ni = i + m - (11 / 2);
                    int nj = j + n - (11 / 2);
                    // Générer les bords de l'image (padding)
                    if (ni >= 0 && ni < hauteur_image && nj >= 0 && nj < largeur_image) {
                        sum += image[ni*N + nj] * filtreLapGauss[m][n];
                    }
                }
            }
            // Affecter le résultat à l'image convoluée
            resultat[i*N + j] = sum;
        }
    }
    //Parcourir Conv et Calculer f
    for (i = 0; i < hauteur_image; i++)
    {
        for (j = 0; j < largeur_image; j++)
        {
            resultat[i*N + j] = resultat[i*N + j]*resultat[i*N + j];
        }
    }
}
```

FIG. 4.2 : Code en C de la fonction carte de contours

- **Partie 01 : Carte de contours**

La figure 4.3 illustre un schéma de la partie 01 :

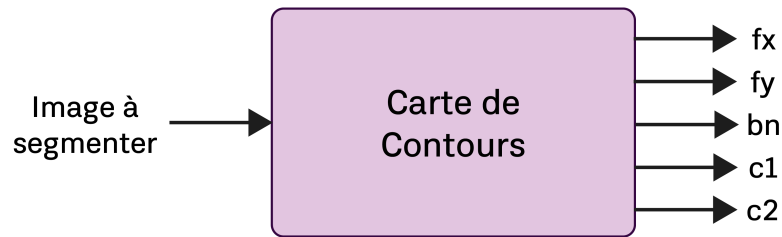


FIG. 4.3 : Schéma de la partie 01 de l'algorithme GVF

La fonction "carte de contours" prend une image filtrée en entrée et produit les valeurs de fx et fy (gradients horizontal et vertical), bn (amplitude du gradient), ainsi que $c1$ et $c2$ (caractéristiques de courbure) pour détecter les contours.

- **Partie 02 : Champs de forces externes**

La figure 4.4 illustre un schéma de la partie 02 :

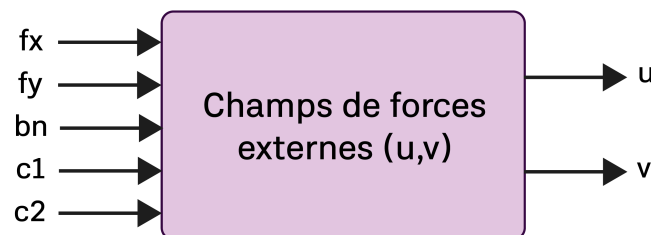


FIG. 4.4 : Schéma de la partie 02 de l'algorithme GVF

La fonction "champs de forces externes" reçoit en entrée les valeurs de fx , fy , bn , $c1$ et $c2$, puis produit en sortie les champs de forces externes u et v .

- **Partie 03 : Déformation du contour**

La figure 4.5 illustre un schéma de la partie 03 :

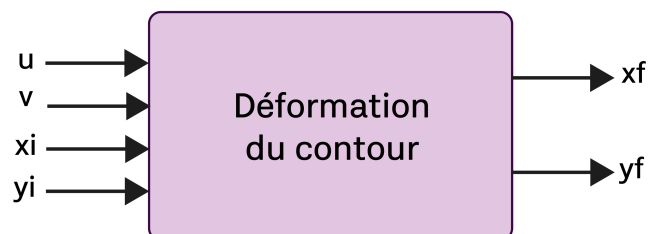


FIG. 4.5 : Schéma de la partie 03 de l'algorithme GVF

La fonction "déformation du contour" prend en entrée les champs externes u et v , ainsi que les coordonnées du contour initial représentées par les vecteurs x_i et y_i . En sortie, elle produit les coordonnées du contour final représentées par les vecteurs x_f et y_f .

4.4.1.2 Création des blocs IP

Dans ce qui suit, nous présentons les étapes sur Vitis HLS pour la partie 01. Ces étapes restent les mêmes pour les autres parties et également pour l'implémentation complète.

- Nous créons un nouveau projet.
- Ensuite, nous choisissons la carte.
- Après, nous identifions la fonction top de notre projet, c'est-à-dire la fonction principale que nous souhaitons synthétiser et implémenter sur le matériel cible.
- Maintenant, nous pouvons passer à la présentation de l'interface du logiciel avec l'exemple de la fonction "carte de contours" (Figure 4.6).

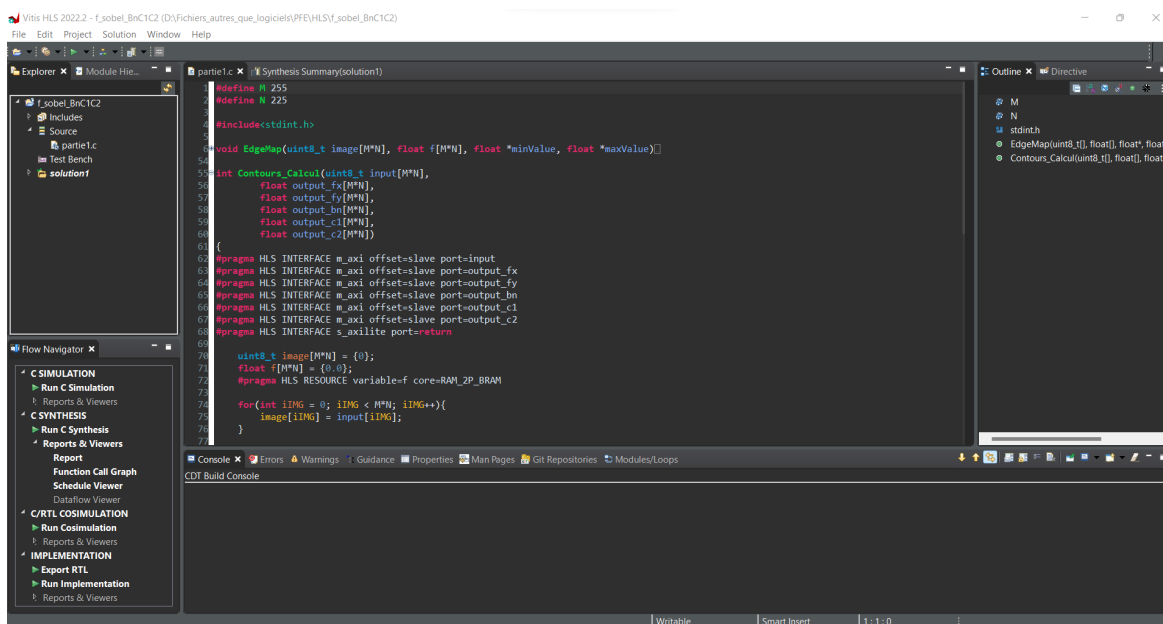


FIG. 4.6 : Interface de Vitis HLS

- Les processeurs Zynq utilisent tous des interfaces AXI. Ainsi, comprendre les bases de cette interface peut être utile pour concevoir et déboguer des designs sur PYNQ FPGA.

Protocole AXI [57]

Le protocole établit un ensemble de règles régissant la communication entre différents modules au sein d'une puce. Il impose une procédure de type **handshake** avant toute transmission, garantissant un échange de données coordonné. Cela facilite la création d'un "système" complet plutôt qu'une simple collection de modules. En effet, il fournit l'interface entre le système de traitement et les sections de logique programmable de la PYNQ. Les spécifications du protocole peuvent être résumées comme suit :

- Avant de transmettre tout signal de contrôle, adresse ou donnée, les modules maître et esclave s'engagent dans un handshake en utilisant des signaux *READY* et *VALID*.

- La transmission des signaux de contrôle et des adresses se fait en phases distinctes, chacune avec son propre canal dédié.
- La transmission des données utilise également un canal distinct des signaux de contrôle et des adresses.
- Le protocole prend en charge une communication de type rafale, permettant un transfert continu de données.

La Figure 4.7 [57] représente les canaux du protocole AXI :

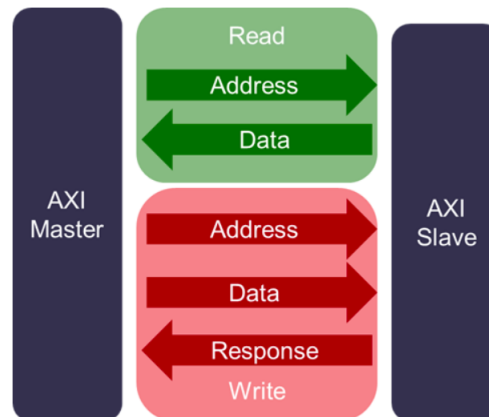


FIG. 4.7 : Canaux de lecture et d'écriture AXI

Le protocole AXI comprend trois types principaux : AXI Memory Mapped, AXI Stream et AXI Lite. Chacun est conçu pour répondre à des besoins de communication spécifiques.

- **AXI Memory Mapped :**
Utilisé pour la communication entre un maître et un esclave dans un système à mémoire mappée, il repose sur l'adressage des allocations mémoire pour les lectures et écritures. Il supporte les transferts en rafale et utilise des canaux séparés pour les adresses et les données, avec des signaux VALID et READY pour synchroniser les transferts.
Utilisé pour la communication de données entre un bloc IP et le bloc mémoire DDR.
- **AXI Stream :**
Destiné aux applications de streaming de données unidirectionnelles à haute vitesse, il transfère de grandes quantités de données sans adressage explicite de la mémoire. La communication se fait via un seul canal avec des signaux VALID et READY pour le contrôle de flux, permettant des transferts continus de données.
Utilisé pour le transfert de données d'un bloc IP à un autre.
- **AXI Lite :**
Version simplifiée d'AXI Memory Mapped, conçue pour les applications à faible bande passante. Il supporte les opérations de lecture et d'écriture de base avec un seul canal d'adresse et des canaux séparés pour les données de lecture et

les réponses d'écriture, réduisant ainsi la complexité et le nombre de signaux. Utilisé dans chaque bloc IP des signaux de contrôle, entre le bloc IP et le CPU.

- **Les directives**

Les directives que nous avons utilisées sur HLS sont :

- `#pragma HLS INTERFACE m_axis port=variable`

Cette directive configure l'interface de la variable spécifiée (variable) pour qu'elle utilise le protocole AXI Memory Mapped (m_axis pour "Master AXI Stream").

- `#pragma HLS INTERFACE s_axilite port=return`

Cette directive configure la valeur de retour de la fonction pour être accessible via une interface AXI Lite.

- `#pragma HLS PIPELINE`

Cette directive permet de paralléliser une boucle ou une fonction, ce qui signifie que les itérations peuvent se chevaucher pour améliorer le débit. En utilisant cette directive, une nouvelle itération de la boucle peut commencer avant que l'itération précédente ne soit terminée.

- `#pragma HLS RESOURCE variable=nom_variable core=BRAM`

Cette directive spécifie que la variable doit être implémentée en utilisant BRAM.

- Lors de l'exécution de la synthèse C, nous obtenons un résumé de synthèse dans lequel de nombreuses informations sont fournies, telles que les estimations de latence, de performances et de ressources.

Dans le cadre de la synthèse de haut niveau, plusieurs termes clés sont utilisés pour décrire les caractéristiques et les ressources des circuits synthétisés.

- **Latence** : La latence fait référence au nombre de cycles d'horloge nécessaires pour exécuter une fonction ou un ensemble de fonctions.
- **Digital Signal Processors (DSP)** : Ces blocs sont des ressources matérielles spécialisées utilisées pour accélérer les opérations de traitement de signal numérique telles que les multiplications et les accumulations et ils sont configurables en fonction de la précision requise.
- **Flip-Flops (FF)** : Utilisés pour stocker des données dans les circuits numériques, ces éléments de mémoire sont souvent employés dans Vitis HLS pour implémenter des registres et des états dans les conceptions matérielles.
- **Block Read Access Memory (BRAM)** : Ces ressources de mémoire embarquée dans les FPGA sont utilisées pour stocker des données de manière temporaire. Leur utilisation peut être spécifiée dans Vitis HLS pour optimiser l'accès aux données et améliorer les performances des algorithmes.
- **Look Up Tables (LUT)** : Ces éléments de logique programmable sont utilisés pour implémenter des fonctions logiques dans les FPGA. Dans Vivado HLS, ils représentent la logique combinatoire des fonctions synthétisées à partir du code source C/C++.

La figure 4.8 présente le résumé de synthèse pour chaque partie.

partie1.c | Synthesis Summary(solution1) | x

General Information

Date: Sun Jun 2 18:11:31 2024
 Version: 2022.2 (Build 3670227 on Oct 13 2022)
 Project: f_sobel_BnC1C2

Solution: solution1 (Vivado IP Flow Target)
 Product family: zynq
 Target device: xc7z020-clg400-1

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	8.424 ns	2.70 ns

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
Contours_Calcul			43719876	4.370E8	-	43719877	-	no	57	12	14	20	0
Contours_Calcul_Pipeline_1			57377	5.740E5	-	57377	-	no	0	0	-0	-0	0
Contours_Calcul_Pipeline_2			57377	5.740E5	-	57377	-	no	0	0	-0	-0	0
Contours_Calcul_Pipeline_VITIS_LOOP_74_1			57386	5.740E5	-	57386	-	no	0	0	-0	-0	0
Contours_Calcul_Pipeline_VITIS_LOOP_91_2_VITIS_LOOP_92_3			286978	2.870E6	-	286978	-	no	0	7	10	10	0
VITIS_LOOP_23_1_VITIS_LOOP_24_2			43318125	4.330E8	755	-	57375	no	-	-	-	-	-

(a) Résumé de synthèse pour la fonction "Carte de contours"

Synthesis Summary(solution1) | x | gfv.c

General Information

Date: Sun Jun 2 15:41:33 2024
 Version: 2022.2 (Build 3670227 on Oct 13 2022)
 Project: GVF

Solution: solution1 (Vivado IP Flow Target)
 Product family: zynq
 Target device: xc7z020-clg400-1

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	8.352 ns	2.70 ns

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
GVF			-	-	-	-	-	no	92	5	8	21	0
GVF_Pipeline_1			677	6.770E3	-	677	-	no	0	0	-0	-0	0
GVF_Pipeline_2			677	6.770E3	-	677	-	no	0	0	-0	-0	0
GVF_Pipeline_VITIS_LOOP_29_1			57378	5.740E5	-	57378	-	no	0	0	-0	-0	0
GVF_Pipeline_VITIS_LOOP_137_7			57514	5.750E5	-	57514	-	no	0	0	-0	-0	0
GVF_Pipeline_VITIS_LOOP_143_8			57514	5.750E5	-	57514	-	no	0	0	-0	-0	0
GVF_Pipeline_VITIS_LOOP_151_9			57378	5.740E5	-	57378	-	no	0	0	-0	-0	0
VITIS_LOOP_48_2			-	-	4417888	-	-	no	-	-	-	-	-

(b) Résumé de synthèse pour la fonction "Champs de forces externes"

Synthesis Summary(solution1) | x | deform.c

General Information

Date: Sun Jun 2 15:01:24 2024
 Version: 2022.2 (Build 3670227 on Oct 13 2022)
 Project: Deform

Solution: solution1 (Vivado IP Flow Target)
 Product family: zynq
 Target device: xc7z020-clg400-1

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	8.495 ns	2.70 ns

Performance & Resource Estimates

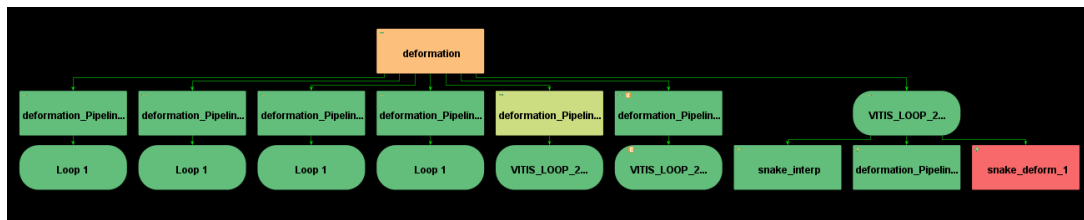
Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
deformation			-	-	-	-	-	no	7	26	13	47	0
deformation_Pipeline_1			102	1.020E3	-	102	-	no	0	0	-0	-0	0
deformation_Pipeline_2			102	1.020E3	-	102	-	no	0	0	-0	-0	0
deformation_Pipeline_3			102	1.020E3	-	102	-	no	0	0	-0	-0	0
deformation_Pipeline_4			102	1.020E3	-	102	-	no	0	0	-0	-0	0
deformation_Pipeline_VITIS_LOOP_249_1			82	820.000	-	82	-	no	~0	0	-0	-0	0
deformation_Pipeline_VITIS_LOOP_276_4			208	2.080E3	-	208	-	no	0	0	-0	-0	0
VITIS_LOOP_256_2			-	-	-	-	-	no	-	-	-	-	-

(c) Résumé de synthèse pour la fonction "Déformation de contour"

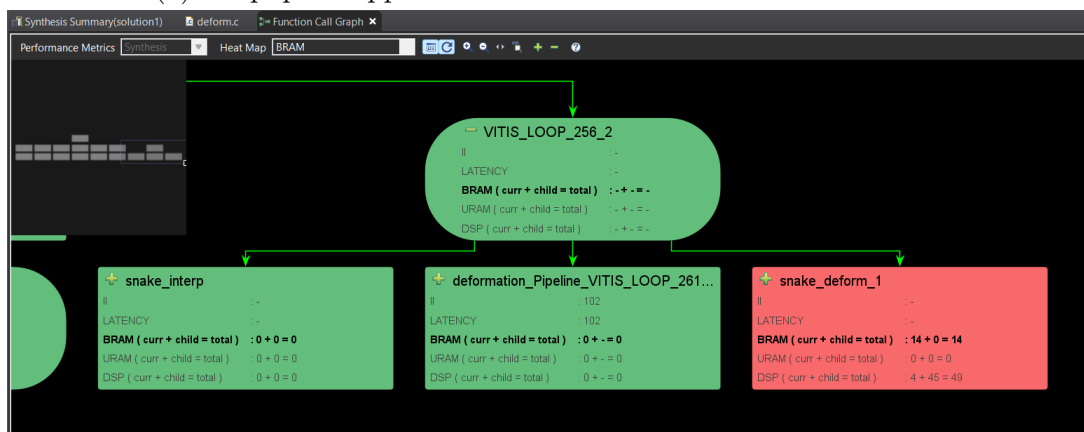
FIG. 4.8 : Résumé de synthèse pour chaque partie

- Nous pouvons également visualiser le graphique d'appel (Call graph) de fonctions avec l'option d'une carte thermique (Heat map) d'une métrique spécifique telle que les blocs de mémoire RAM (BRAM) et la latence. Cela est utile pour visualiser les différentes sous-routines du programme et leur pipeline.

Voici un exemple du graphique d'appel de la fonction déformation du contour (Figure 4.9) :



(a) Graphique d'appel de la fonction "Déformation du contour"



(b) Zoom sur une partie de la fonction "Déformation du contour"

FIG. 4.9 : Graphique d'appel de la fonction principale "Déformation du contour" avec l'option de la carte thermique de BRAM

- Une fois la synthèse terminée, nous lançons l'exportation au niveau de transfert de registre (RTL). Le code RTL spécifie les portes logiques, les registres et les interconnexions qui composent le circuit numérique.

4.4.1.3 Création du design par blocs

Une fois que nous avons généré les blocs IP pour chaque partie sur Vitis HLS, nous passons à Vivado pour la création du design par blocs pour notre système.

Dans la conception FPGA, un design par blocs utilise des modules pré-définis appelés blocs de propriété intellectuelle (IP) et peuvent être fournis par le fabricant de FPGA ou créés par le concepteur.

Chaque bloc représente une fonctionnalité spécifique, comme des processeurs ou des contrôleurs de mémoire, et les blocs sont interconnectés via des interfaces standardisées pour permettre la communication entre eux.

Cette approche modulaire facilite la conception, la réutilisabilité et l'utilisation efficace des ressources FPGA, simplifiant ainsi le processus global de conception FPGA.

Encore une fois, nous présentons ci-dessous les étapes pour la partie 1 - Carte de contours. Ces étapes sont les mêmes pour les autres parties ainsi que pour l'implémentation complète.

- Tout d'abord, nous créons un nouveau projet dans Vivado Design Suit, en spécifiant la carte que nous allons utiliser. Une fois le projet créé, nous verrons cette interface (Figure 4.10).

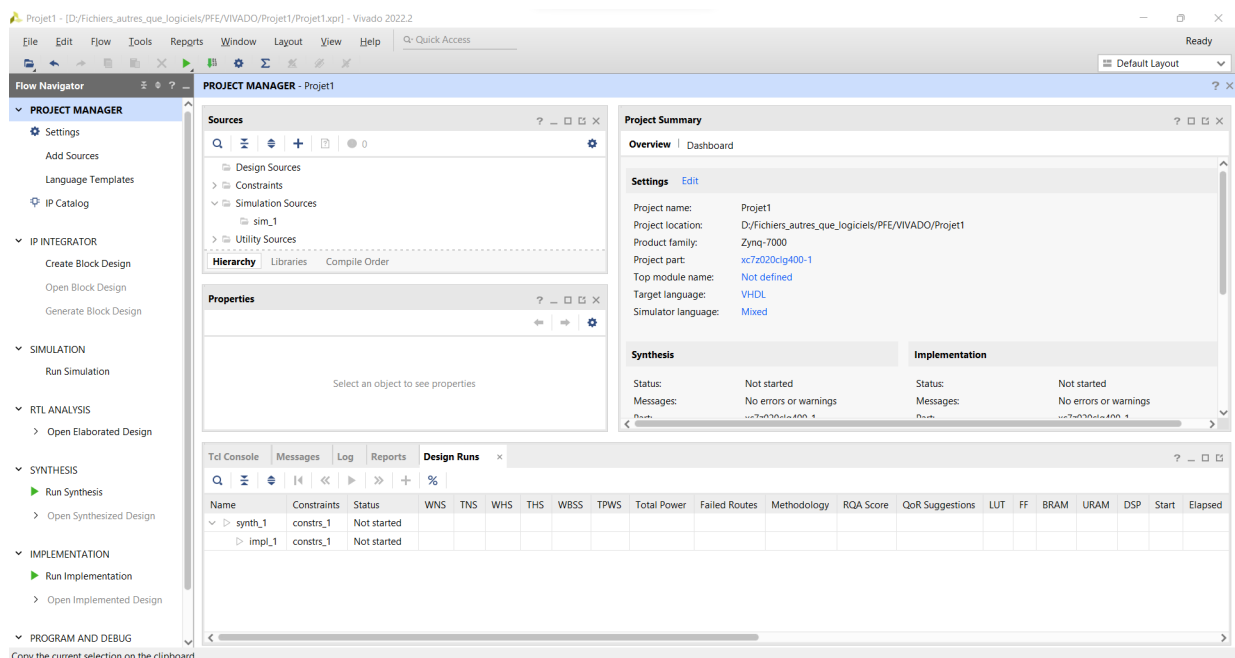


FIG. 4.10 : Interface Vivado

- Ensuite, nous ajoutons le bloc IP, précédemment créé avec Vitis HLS, de la partie qu'on souhaite implémenter (par exemple : partie 1 - Carte de contours) au "Répertoire IP" dans les "Paramètres du gestionnaire de projet".
- Par la suite, nous créons un nouveau design par blocs pour notre projet et nous ajoutons le bloc IP de la fonction "Carte de contours", comme illustré dans la Figure 4.11.

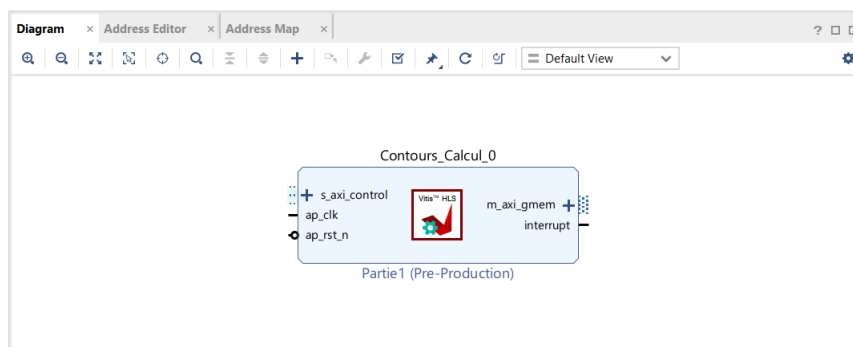


FIG. 4.11 : Bloc IP de la fonction "Carte de contours"

Ce bloc possède cinq ports :

- **s_axi_control** : utilisé pour les signaux de contrôle ou les variables déclarées avec le protocole s_axilite.
 - **m_axi_gmem** : attribué aux variables nécessitant un accès mémoire (celles déclarées avec le protocole m_axi, qui dans ce cas sont : input_image et les output).
 - **ap_clk** : pour le signal d'horloge.
 - **ap_rst_n** : pour le signal de réinitialisation.
 - **interrupt** : utilisé pour l'interruption générée par le bloc IP, en utilisant le port de retour qui est déclenché lorsque le calcul effectué par le bloc IP spécifique est terminé.
- Ensuite, nous ajoutons les blocs nécessaires, y compris le processeur Zynq, qui contrôlera le bloc mémoire et contiendra les autres sous-étapes du système. Nous activons également les interfaces esclaves AXI haute performance du processeur.

Nous pouvons remarquer sur la figure 4.12 qu'il y a des sorties **DDR** et **FIXED_IO** du processeur, générées automatiquement par Vivado lors de l'exécution de l'automatisation de bloc du processeur Zynq.

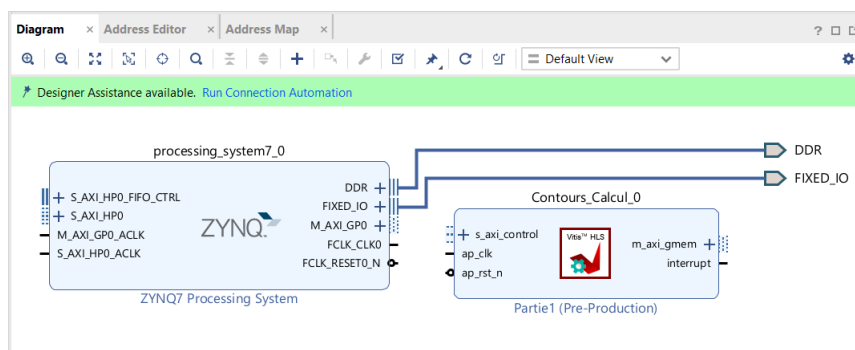


FIG. 4.12 : Ajout des différents blocs IP

- Après avoir intégré tous les blocs essentiels et configuré le processeur pour établir la communication avec le FPGA, l'étape suivante consiste à exécuter le processus d'**Automatisation de Connexion** pour faciliter l'interconnexion de tous les blocs.

Nous constatons que trois blocs sont ajoutés au design, qui représentent :

- Processor System Reset : ce bloc permet d'initier des réinitialisations à l'échelle du système qui impliquent spécifiquement le processeur.
- 2 blocs AXI Interconnect : ces blocs sont utilisés pour adapter l'horloge, la largeur ou le protocole utilisés par les différents ports du bloc IP et ceux utilisés par le processeur. Il y a deux blocs parce que nous utilisons deux protocoles différents : m_axi et s_axilite.

Remarque :

En exécutant l'automatisation de connexion, il peut y avoir des erreurs de connexion

entre les différents ports ou des connexions manquantes. Ces erreurs devront être corrigées manuellement.

Le design final est illustré dans la figure 4.13.

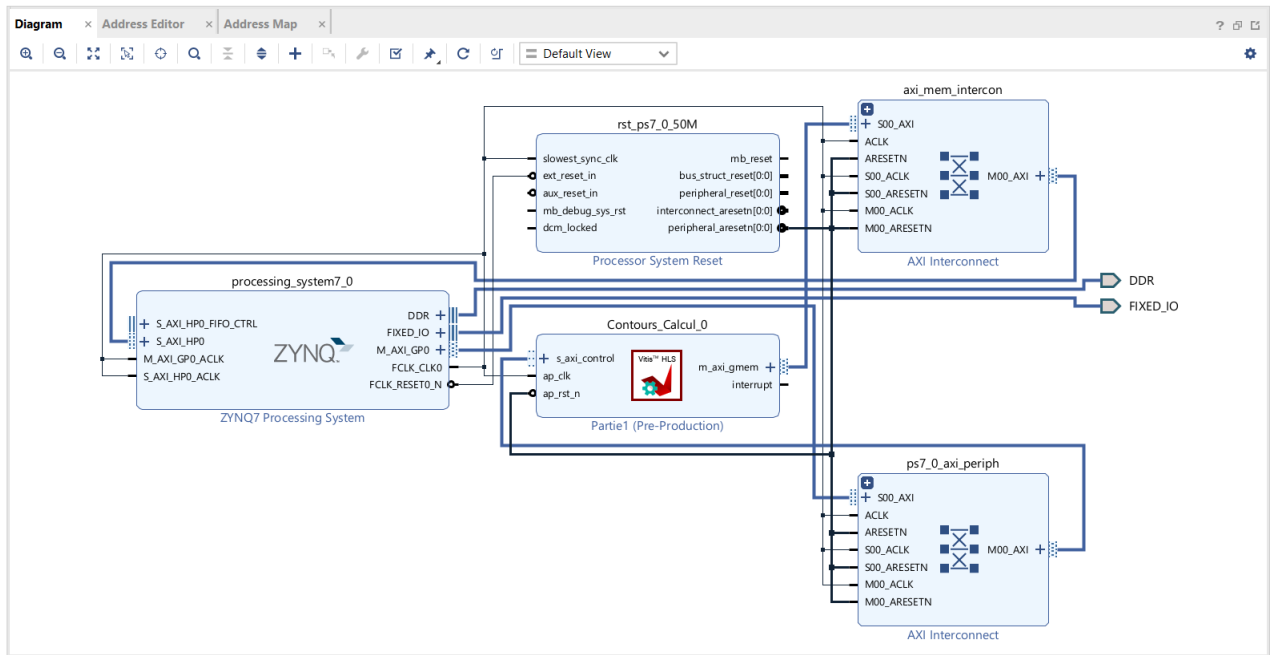


FIG. 4.13 : Bloc design de la partie 01 - Carte de contours

- Après avoir établi toutes les connexions nécessaires, l'étape suivante consiste à vérifier notre conception. Une fois la conception validée, nous pouvons vérifier et modifier les adresses attribuées à chaque esclave et maître dans l'architecture via la fenêtre "Address Editor".
- Suite à cela, nous générons l'**enveloppe HDL (HDL wrapper)**. L'objectif principal de cette enveloppe est de convertir le design en un format de description matérielle, prêt pour la synthèse et l'implémentation ultérieure dans le FPGA cible.
- Une fois l'enveloppe HDL générée, nous pouvons **synthétiser** notre système. La synthèse convertit le design HDL en une représentation composée de portes logiques, de bascules et d'autres composants numériques interconnectés, prête à être implémentée physiquement sur le FPGA.

Pendant la synthèse, Vivado analyse la description RTL du design et effectue différentes transformations afin de l'optimiser. Ces transformations incluent :

- *La correspondance technologique* : Le design RTL est associé à des éléments logiques spécifiques disponibles sur FPGA, en tenant compte de l'architecture et des ressources du dispositif.
- *L'optimisation de la logique combinatoire* : Vivado optimise la logique combinatoire en minimisant le nombre de portes logiques et en réduisant les opérations logiques inutiles. Cela aide à améliorer les performances globales et à minimiser l'utilisation des ressources.

- *L'optimisation de la logique séquentielle* : Les éléments logiques séquentiels, tels que les bascules et les registres, sont optimisés pour réduire le nombre de cycles d'horloge nécessaires au traitement des données et améliorer les caractéristiques de synchronisation du design.
 - *L'allocation des ressources* : Vivado attribue les ressources physiques du FPGA, telles que les tables de recherche (LUT), les bascules, la mémoire RAM bloc et les tranches DSP, à différentes parties du design pour garantir une utilisation efficace des ressources et respecter les contraintes de conception.
- Une fois la synthèse terminée, nous pouvons analyser les différents **rapports** que nous obtenons, tels que le rapport d'utilisation de ressources résumé (Figure 4.14), détaillé (Figure 4.15) et le rapport de consommation d'énergie (Figure 4.16).

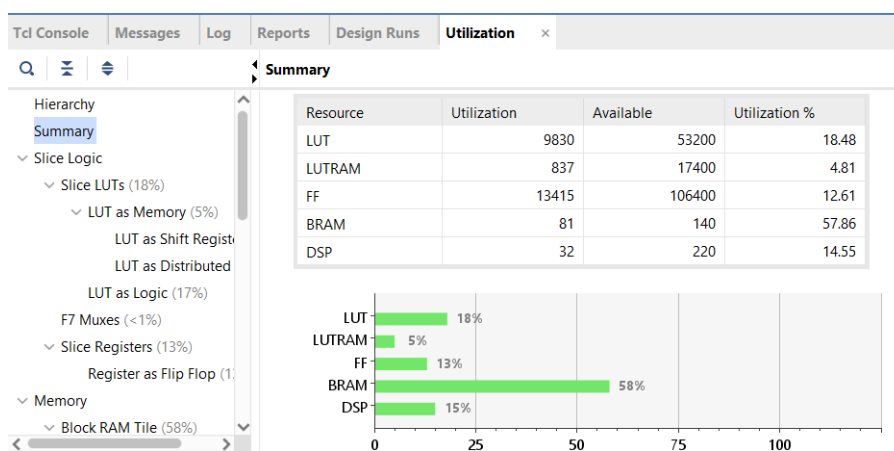


FIG. 4.14 : Rapport d'utilisation de ressources résumé

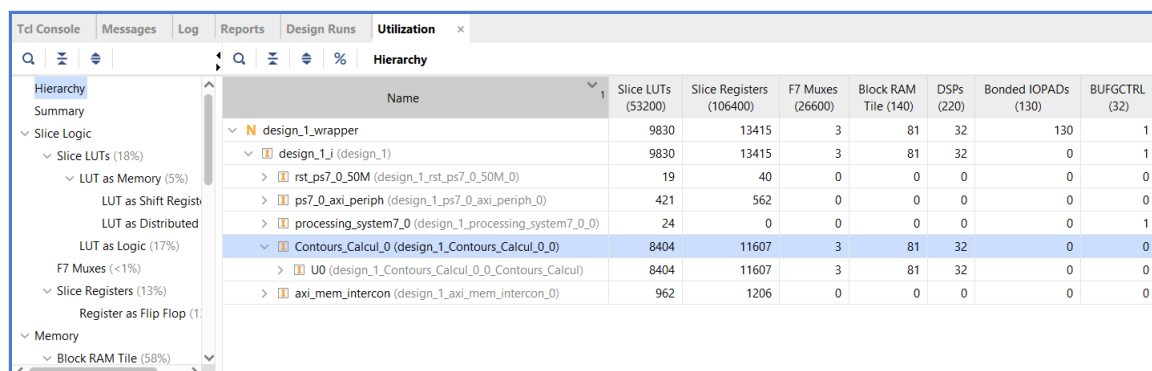


FIG. 4.15 : Rapport d'utilisation de ressources détaillé

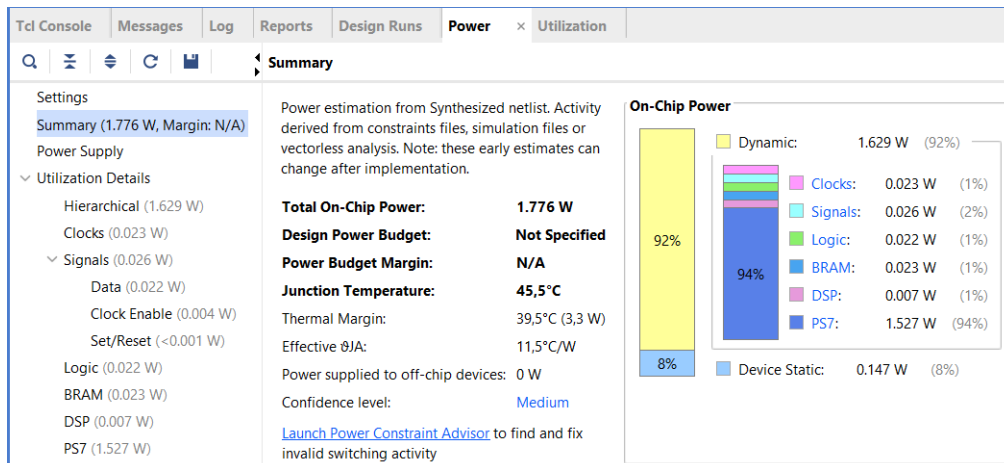


FIG. 4.16 : Rapport de consommation d'énergie

- La dernière étape sur Vivado est l'**implémentation** du système et la génération du fichier binaire "**Bitstream**".

Dans Vivado, l'implémentation fait référence au processus de mappage et de placement physique du design synthétisé sur FPGA. Cela implique plusieurs tâches pour transformer la netlist au niveau des portes générée lors de la synthèse en un bitstream de configuration qui peut être chargé sur FPGA.

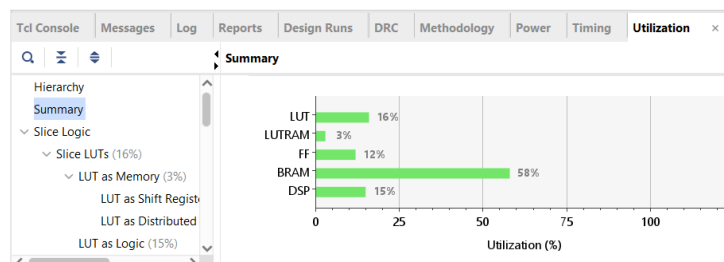
Les principales étapes du processus d'implémentation dans Vivado sont :

- **Placement** : Détermine l'emplacement physique de chaque élément logique dans la matrice programmable du FPGA. L'objectif est de minimiser les délais, d'optimiser les performances et de respecter les contraintes.
- **Routing** : Établit les connexions physiques entre les éléments logiques en fonction des interconnexions spécifiées dans la netlist.
- **Planification des horloges** : Identifie les domaines d'horloge, applique les contraintes d'horloge et optimise le routage des horloges pour garantir une synchronisation et un timing corrects.
- **Planification des entrées/sorties (I/O)** : Mappe les signaux d'entrée/sortie du design aux broches physiques appropriées du dispositif et applique les contraintes d'I/O pour des niveaux de signalisation et de tension corrects.
- **Génération du bitstream** : Le fichier bitstream contient les instructions et les données de configuration nécessaires pour programmer le FPGA, lui permettant de fonctionner selon les fonctionnalités spécifiées dans le design.
- **Analyse et fermeture du timing** : S'assure que le design respecte les contraintes de timing spécifiées.
- **Vérification des règles de conception (DRC)** : Garantit que le design adhère aux règles de fabrication et aux directives spécifiées par la carte FPGA cible. Il vérifie les violations telles que les courts-circuits, la congestion excessive ou les composants superposés.

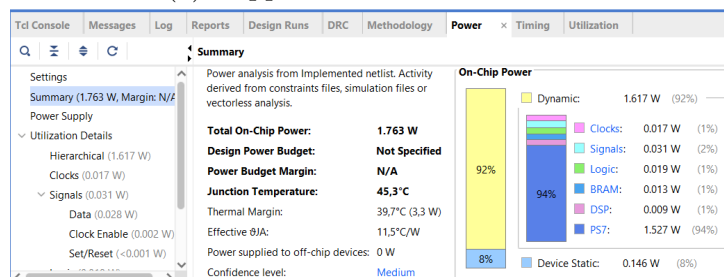
- Une fois le processus d'implémentation terminé, le fichier **bitstream** résultant peut être chargé sur FPGA, la configurant pour fonctionner selon la fonctionnalité définie dans le design original.
- Nous exportons également le design par blocs en un fichier TCL (Tool Command Language), et le fichier des métadonnées .HWH, que nous utilisons dans la dernière phase de notre implémentation.

Le fichier TCL est un script permettant de contrôler et d'étendre les applications logicielles, aidant ainsi à construire le système (design ou architecture) à partir des sources. D'autre part, le fichier HWH contient un ensemble complet de valeurs de registre, d'interfaces et de connexions du design. Ces deux fichiers aident le CPU à reconnaître l'architecture implémentée sur FPGA et à comprendre son fonctionnement.

- Tout comme lors de la synthèse, le processus d'implémentation dans Vivado génère divers rapports offrant des informations précieuses sur l'utilisation des ressources de notre système sur le FPGA. Ces rapports sont essentiels pour analyser et comprendre comment les ressources du FPGA sont utilisées par notre design. Voici les rapports d'utilisation résumés ainsi que les rapports de puissance de chaque partie (Figures 4.17, 4.18 et 4.19).

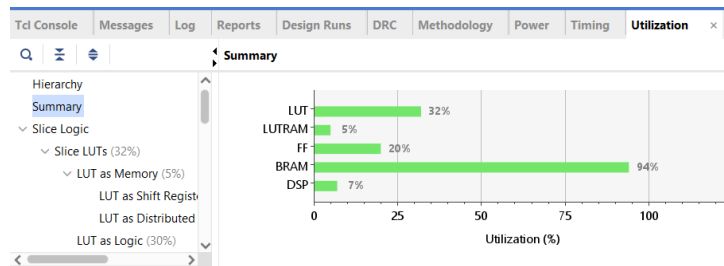


(a) Rapport d'utilisation résumé

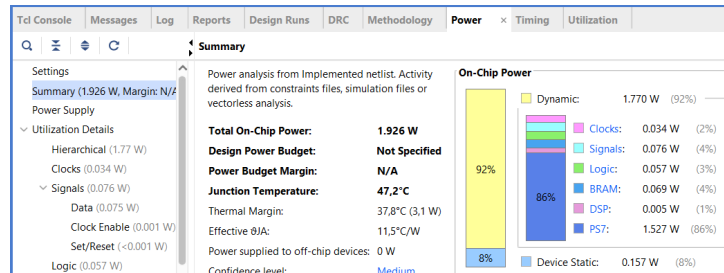


(b) Rapport de puissance

FIG. 4.17 : Résumé d'implémentation pour la fonction "Carte de contours"

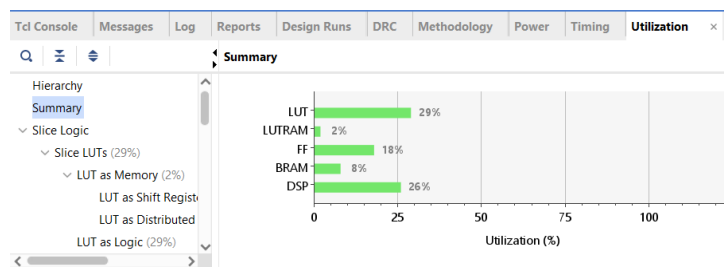


(a) Rapport d'utilisation résumé

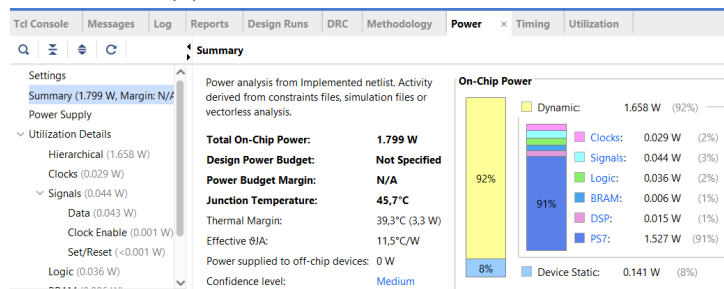


(b) Rapport de puissance

FIG. 4.18 : Résumé d'implémentation pour la fonction "Champs de forces externes (u, v)"



(a) Rapport d'utilisation résumé



(b) Rapport de puissance

FIG. 4.19 : Résumé d'implémentation pour la fonction "Déformation du contour"

4.4.1.4 Programmation du CPU

Nous expliquons maintenant la dernière phase de l'implémentation qui se fait sur Jupyter Notebook en utilisant Python.

Voici les étapes de la programmation du PS de la carte FPGA présentées pour la partie 1 - Carte de contours, mais qui sont les mêmes pour les autres parties :

- Nous commençons par configurer une carte SD en y important l'image PYNQ-Z

[58]. Ensuite, nous établissons la connexion entre le PC et la carte en utilisant une connexion Ethernet et alimentons cette dernière avec un câble USB.

- Ensuite, nous accédons au CPU via un Jupyter Notebook en utilisant l'adresse IP de la carte FPGA [59], où nous importons les fichiers : Bitstream, TCL et HWH, ainsi que l'image filtrée en fichier .csv.
- Les circuits logiques programmables implémentés dans le FPGA sont importés sous forme de bibliothèques matérielles appelées "Overlays" et contrôlés via leurs API.

Les Overlays sont des conceptions FPGA programmables/configurables qui étendent l'application du système de traitement Zynq à la logique programmable. Ces overlays peuvent être utilisés pour accélérer une application logicielle ou personnaliser la plateforme matérielle pour une application spécifique.

L'overlay permet d'implémenter et de contrôler les circuits logiques programmables (PL) sur la partie FPGA, mais son utilisation et son contrôle sont effectués à partir de la partie PS (Processing System) du FPGA via une interface Python.

La figure 4.20 montre la création d'un overlay pour notre architecture en utilisant le module `pynq.Overlay` :

```
[1]: from pynq import Overlay
    ol = Overlay("./design_1.bit")

[2]: ol

[2]: <pynq.overlay.Overlay at 0xb37e05e0>

[4]: ol?
Type:          Overlay
String form:   <pynq.overlay.Overlay object at 0xb37e05e0>
File:         /usr/local/share/pynq-venv/lib/python3.10/site-packages/pynq/overlay.py
Docstring:
Default documentation for overlay ./design_1.bit. The following
attributes are available on this overlay:

IP Blocks
-----
Contours_Calcul_0 : pynq.overlay.DefaultIP
processing_system7_0 : pynq.overlay.DefaultIP
```

FIG. 4.20 : Création d'un overlay pour l'implémentation de la partie 1 - Carte de contours

- Après cela, nous devons mapper chaque bloc IP à des adresses en utilisant la classe MMIO. Cette classe permet aux objets Python (comme les blocs IP d'un Overlay) d'accéder à la mémoire système mappée (DDR).

La figure 4.21 présente le mappage du bloc `Contours_Calcul` de la partie 1 :

```
[6]: ip_input = ol.Contours_Calcul_0
      mmio_in = ip_input.mmio
      register_map_in = ip_input.register_map
      registers_in = register_map_in._register_classes

[7]: for name, reg in registers_in.items():
      print(name, reg)

CTRL (<class 'pynq.registers.RegisterCTRL'>, 0, 32, None, None, 'read-write')
GIER (<class 'pynq.registers.RegisterGIER'>, 4, 32, None, None, 'read-write')
IP_IER (<class 'pynq.registers.RegisterIP_IER'>, 8, 32, None, None, 'read-write')
IP_ISR (<class 'pynq.registers.RegisterIP_ISR'>, 12, 32, None, None, 'read-write')
ap_return (<class 'pynq.registers.Registerap_return'>, 16, 32, None, None, 'read-only')
input_r_1 (<class 'pynq.registers.Registerinput_r_1'>, 24, 32, None, None, 'write-only')
input_r_2 (<class 'pynq.registers.Registerinput_r_2'>, 28, 32, None, None, 'write-only')
output_fx_1 (<class 'pynq.registers.Registeroutput_fx_1'>, 36, 32, None, None, 'write-only')
output_fx_2 (<class 'pynq.registers.Registeroutput_fx_2'>, 40, 32, None, None, 'write-only')
output_fy_1 (<class 'pynq.registers.Registeroutput_fy_1'>, 48, 32, None, None, 'write-only')
output_fy_2 (<class 'pynq.registers.Registeroutput_fy_2'>, 52, 32, None, None, 'write-only')
output_bn_1 (<class 'pynq.registers.Registeroutput_bn_1'>, 60, 32, None, None, 'write-only')
output_bn_2 (<class 'pynq.registers.Registeroutput_bn_2'>, 64, 32, None, None, 'write-only')
output_c1_1 (<class 'pynq.registers.Registeroutput_c1_1'>, 72, 32, None, None, 'write-only')
output_c1_2 (<class 'pynq.registers.Registeroutput_c1_2'>, 76, 32, None, None, 'write-only')
output_c2_1 (<class 'pynq.registers.Registeroutput_c2_1'>, 84, 32, None, None, 'write-only')
output_c2_2 (<class 'pynq.registers.Registeroutput_c2_2'>, 88, 32, None, None, 'write-only')
```

FIG. 4.21 : Mappage du bloc Contours_Calcul pour l'implémentation de la partie 1 - Carte de contours

- L'étape suivante est l'allocation mémoire pour chaque entrée et sortie de chaque bloc, comme l'illustre la figure 4.22 pour le bloc Contours_Calcul :

```
[9]: import numpy as np
      from pynq import allocate

[10]: #allocate buffer for input
      input_buffer_size = 255*225
      input_buffer1 = allocate(shape = (input_buffer_size), dtype=np.uint8, cacheable = False)
      register_map_in.input_r_1.input_r = input_buffer1.device_address

      #allocate buffer for output_fx
      output_buffer1 = allocate(shape = (input_buffer_size), dtype=np.float32, cacheable = False)
      register_map_in.output_fx_1.output_fx = output_buffer1.device_address

      #allocate buffer for output_fy
      output_buffer2 = allocate(shape = (input_buffer_size), dtype=np.float32, cacheable = False)
      register_map_in.output_fy_1.output_fy = output_buffer2.device_address

      #allocate buffer for output_bn
      output_buffer3 = allocate(shape = (input_buffer_size), dtype=np.float32, cacheable = False)
      register_map_in.output_bn_1.output_bn = output_buffer3.device_address

      #allocate buffer for output_c1
      output_buffer4 = allocate(shape = (input_buffer_size), dtype=np.float32, cacheable = False)
      register_map_in.output_c1_1.output_c1 = output_buffer4.device_address

      #allocate buffer for output_c2
      output_buffer5 = allocate(shape = (input_buffer_size), dtype=np.float32, cacheable = False)
      register_map_in.output_c2_1.output_c2 = output_buffer5.device_address
```

FIG. 4.22 : Allocation de mémoire pour le bloc Contours_Calcul pour l'implémentation de la partie 1 - Carte de contours

- Une fois que les buffers sont alloués, nous procédons à l'écriture dans le buffer d'entrée. Cela comprend l'image filtrée récupérée de MATLAB sous forme d'un fichier .csv.

Il faut s'assurer que l'écriture dans le buffer d'entrée est effectivement envoyée à la mémoire partagée en utilisant la fonction **flush**. En d'autres termes, cela garantit que les données écrites dans `input_buffer1` sont visibles par le matériel (PL sur PYNQ) avant de démarrer le traitement.

L'instruction :

```
register_map_in.CTRL.AP_START = 1
```

Envoie un signal au bloc matériel IP pour commencer l'exécution. Et enfin, l'instruction :

```
while (register_map_in.CTRL.AP_DONE == 0) : pass
```

attend que le bloc matériel termine l'exécution de l'algorithme.

Voici plus d'explications sur les instructions citées ci-dessus :

- *register_map_in* est un objet mappant les registres de contrôle du bloc matériel.
- *CTRL* est un champ dans *register_map_in* qui contient divers registres de contrôle, dont *AP_START* et *AP_DONE*.
- *AP_START = 1* démarre l'exécution de l'algorithme ou du traitement dans le bloc.
- Tant que *AP_DONE* vaut 0, cela signifie que le traitement n'est pas terminé, et la boucle continue d'attendre.
- Une fois que *AP_DONE* passe à 1, cela signifie que le traitement est terminé, et la boucle se termine.

La figure 4.23 illustre ce que nous venons de détailler :

```
[11]: # Charger le fichier .csv dans un tableau NumPy
image_array = np.loadtxt('./image_vector.csv', delimiter=',')

# Convertir les valeurs en entiers non signés de 8 bits
image_array = image_array.astype(np.uint8)

# Dimensions de l'image
size = image_array.shape

[15]: import time

input_buffer1[:len(image_array)] = image_array

input_buffer1.flush()

t0 = time.time()
register_map_in.CTRL.AP_START = 1

while (register_map_in.CTRL.AP_DONE == 0): pass
t1 = time.time()

print(t1-t0)

1.4379892349243164
```

FIG. 4.23 : Exécution du code sur FPGA

La librairie **time** a été utilisé pour calculer le temps d'exécution de cette première partie, qui est de **1.43 s**.

- Nous récupérons les sorties dans des fichiers .csv comme le montre la figure 4.24, pour les utiliser comme entrées pour la partie 2.

```
[21]: import pandas as pd

[23]: df = pd.DataFrame(output_buffer1, columns=['fx'])
df.to_csv('fx.csv', index=False)

print("Le vecteur a été enregistré dans le fichier buffer.csv.")

Le vecteur a été enregistré dans le fichier buffer.csv.
```

FIG. 4.24 : Création du fichier .csv pour l'output_fx

- Nous pouvons nous assurer que les calculs sont corrects en comparant les valeurs obtenues pour chaque sortie avec les valeurs obtenues sur MATLAB.

La figure 4.25 donne un exemple d'affichage de quelques valeurs de output_fx.

```
>> format long;
disp(fx(1,1:20))
Columns 1 through 9
    0.001175936816498    -0.000293895280999    -0.000616498210708    -0.000543985207444    -0.000380440411239    -0.000295157638938    0.000037493140382    0.000124187208616    -0.000129073425267
Columns 10 through 18
-0.000315550881080    -0.000024879297426    0.000180739007607    0.000149839525793    -0.000018773223439    -0.000046739382186    -0.000114262710646    -0.000116143896246    -0.000005462929734
Columns 19 through 20
    0.000127499332992    0.000105033675303
```

(a) Output_fx sur Matlab

```
[16]: print(output_buffer1[0:20])
[ 1.1758413e-03 -2.9382901e-04 -6.1640667e-04 -5.4391380e-04
 -3.8040636e-04 -2.9513889e-04  3.7489783e-05  1.2402778e-04
 -1.2915183e-04 -3.1542167e-04 -2.4811776e-05  1.8075065e-04
 1.4982863e-04 -1.8777571e-05 -4.6738489e-05 -1.1425605e-04
 -1.1613729e-04 -5.4626080e-06  1.2749828e-04  1.0503613e-04]
```

(b) Output_buffer1 (Output_fx) sur jupyter notebook

FIG. 4.25 : Valeurs de fx sur MATLAB et sur notebook

Les résultats de l'implémentation correspondent aux résultats obtenus sur MATLAB avec des différences de l'ordre de 10^{-7} .

Explications :

La différence que nous observons entre les valeurs obtenues avec MATLAB et celles obtenues après implémentation sur FPGA peut être expliquée par la différence de précision numérique entre les représentations des nombres flottants sur ces deux plateformes. En effet, MATLAB utilise des nombres flottants en double précision, ce qui signifie qu'ils sont représentés sur 64 bits, alors que sur FPGA, nous utilisons des flottants en simple précision (32 bits).

Les différences de précision entre les représentations sur 64 bits et 32 bits causent des écarts dans les valeurs calculées, surtout lorsqu'un grand nombre d'opérations mathématiques sont effectuées en séquence, ce qui est notre cas pour le calcul de f , fx , et fy . Les opérations matricielles et les calculs de gradients impliquent de nombreuses multiplications et additions, où les erreurs d'arrondi peuvent se propager et s'amplifier.

De plus, les outils de synthèse comme Vitis HLS pour FPGA peuvent appliquer différentes optimisations qui peuvent aussi contribuer à de légères variations dans les résultats.

- Nous procédons de la même manière, pour l'implémentation des 2 autres parties et les résultats de chaque partie sont validés en les comparant à ceux obtenus sur MATLAB.

Sachant que pour l'implémentation, nous avons gardé le même jeu de paramètres avec lequel nous avons obtenu le meilleur résultat de la segmentation de l'IRM du cerveau au chapitre 3, la figure 4.26 représente le résultat de l'implémentation par parties de la segmentation par la méthode GVF.

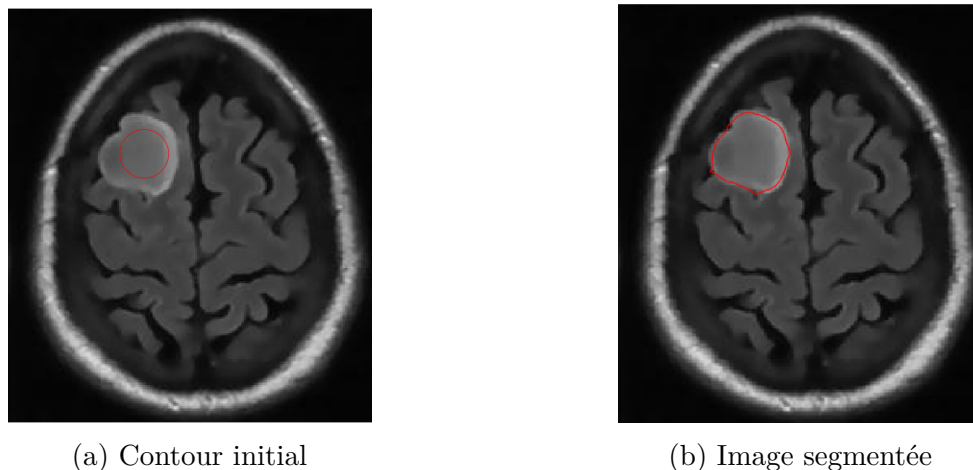


FIG. 4.26 : Résultat de l'implémentation par parties de la méthode GVF

Nous remarquons que le résultat de segmentation obtenu lors de cette implémentation est fidèle à celui obtenu par la simulation MATLAB.

Malgré les petites différences dans les valeurs, le résultat final de la segmentation n'a pas été affecté, nous expliquerons un peu plus en détails cela pour le résultat de l'implémentation complète.

- Le tableau 4.1 résume les temps d'exécution de chaque partie :

Partie	1	2	3
Temps de calculs (s)	1.4380	8.4127	0.1746

TAB. 4.1 : Temps d'exécution de chaque partie de l'algorithme GVF

4.4.2 Implémentation complète

Après avoir expliqué toutes les étapes de l'implémentation, nous pouvons à présent passer à l'approche complète.

4.4.2.1 Codage en C

Étant donné que la somme des consommations de ressources pour les trois différentes parties dépasse les 100% en BRAM, pour l'approche complète, nous avons divisé la première partie en deux blocs (bloc pour le calcul de f et bloc pour le filtre Sobel et calcul de bn , $c1$ et $c2$). Ainsi, les sorties du premier bloc seront stockées dans la mémoire DDR et seront dupliquées, puis le deuxième bloc les récupérera de la mémoire DDR. Ce processus sera répété pour le reste des variables intermédiaires des 2 autres blocs (champs de forces externes et déformation du contour).

La figure 4.27 illustre un schéma des différents blocs de cette approche.

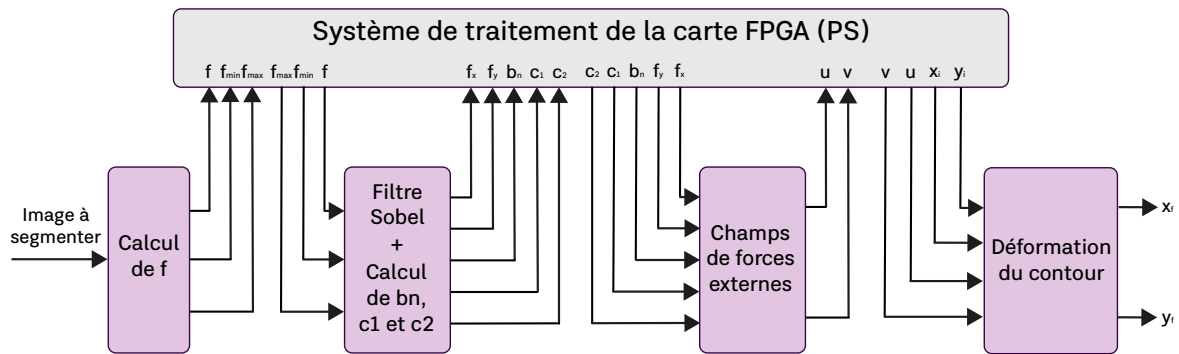


FIG. 4.27 : Schéma des différents blocs de l'implémentation complète de l'algorithme GVF

Chaque bloc correspond à une fonction top que nous développons en langage C et que nous simulons sur VS Code, et donc à un bloc IP que nous créons sur Vitis HLS.

4.4.2.2 Création des blocs IP

En suivant les mêmes étapes présentées dans la section 4.4.1.2, nous allons créer les blocs IP correspondant aux différentes fonctions de l'approche complète.

Les figures 4.28 et 4.29 présente le résumé de synthèse pour chaque bloc.

▼ Performance & Resource Estimates ⓘ

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
Gradient	-	-	-	36605260	3,660E8	-	36605261	-	no	0	2	2	7	0
VITIS_LOOP_35_1_VITIS_LOOP_36_2	-	-	-	36605250	3,660E8	638	-	57375	no	-	-	-	-	-
Gradient_Pipeline_VITIS_LOOP_38_3_VITIS_LOOP_39_4	-	-	-	629	6,290E3	-	629	-	no	0	1	-0	2	0
VITIS_LOOP_38_3_VITIS_LOOP_39_4	-	-	-	627	6,270E3	28	5	121	yes	-	-	-	-	-

(a) Résumé de synthèse pour la fonction "Calcul de f"

▼ Performance & Resource Estimates ⓘ

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
Sobel_BnC1C2	-	-	-	516496	5,165E6	-	516497	-	no	0	7	10	16	0
Sobel_BnC1C2_Pipeline_VITIS_LOOP_37_1_VITIS_LOOP_38_2	-	-	-	516480	5,165E6	-	516480	-	no	0	6	8	10	0
VITIS_LOOP_37_1_VITIS_LOOP_38_2	Resource Limitation	516478	-	516478	5,165E6	113	9	57375	yes	-	-	-	-	-

(b) Résumé de synthèse pour la fonction "Filtre Sobel + Calcul de b_n , c_1 et c_2 "

▼ Performance & Resource Estimates ⓘ

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
GVF	-	-	-	-	-	-	-	-	no	91	5	7	18	0
GVF_Pipeline_1	-	-	-	677	6,770E3	-	677	-	no	0	0	-0	-0	0
Loop 1	-	-	-	675	6,750E3	1	1	675	yes	-	-	-	-	-
GVF_Pipeline_2	-	-	-	677	6,770E3	-	677	-	no	0	0	-0	-0	0
Loop 1	-	-	-	675	6,750E3	1	1	675	yes	-	-	-	-	-
GVF_Pipeline_VITIS_LOOP_31_1	Resource Limitation	114760	-	114760	1,148E6	-	114760	-	no	0	0	-0	-0	0
VITIS_LOOP_31_1	Resource Limitation	114758	-	114758	1,148E6	11	2	57375	yes	-	-	-	-	-
GVF_Pipeline_VITIS_LOOP_131_7_VITIS_LOOP_133_8	-	-	-	57476	5,750E5	-	57476	-	no	0	3	1	3	0
VITIS_LOOP_131_7_VITIS_LOOP_133_8	-	-	-	57474	5,750E5	101	1	57375	yes	-	-	-	-	-
GVF_Pipeline_VITIS_LOOP_142_9	Resource Limitation	114758	-	114758	1,148E6	-	114758	-	no	0	0	-0	-0	0
VITIS_LOOP_142_9	Resource Limitation	114756	-	114756	1,148E6	9	2	57375	yes	-	-	-	-	-
VITIS_LOOP_50_2	-	-	-	-	-	4475264	-	-	no	-	-	-	-	-
GVF_Pipeline_VITIS_LOOP_51_3_VITIS_LOOP_53_4	-	-	-	4475261	4,475E7	-	4475261	-	no	0	0	2	5	0
VITIS_LOOP_51_3_VITIS_LOOP_53_4	-	-	-	4475259	4,475E7	88	78	57375	yes	-	-	-	-	-

(c) Résumé de synthèse pour la fonction "Champs de forces externes"

FIG. 4.28 : Résumé de synthèse pour chaque partie

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
deformation			102	1,020E3	-	-	-	no	4	15	10	39	0
deformation_Pipeline_1			102	1,020E3	-	-	-	no	0	0	-0	-0	0
Loop 1			100	1,000E3	1	1	100	yes	-	-	-	-	-
deformation_Pipeline_2			102	1,020E3	-	-	-	no	0	0	-0	-0	0
Loop 1			100	1,000E3	1	1	100	yes	-	-	-	-	-
deformation_Pipeline_3			102	1,020E3	-	-	-	no	0	0	-0	-0	0
Loop 1			100	1,000E3	1	1	100	yes	-	-	-	-	-
deformation_Pipeline_4			102	1,020E3	-	-	-	no	0	0	-0	-0	0
Loop 1			100	1,000E3	1	1	100	yes	-	-	-	-	-
deformation_Pipeline_VITIS_LOOP_231_1			170	1,700E3	-	-	-	no	0	0	-0	-0	0
VITIS_LOOP_231_1	Resource Limitation		168	1,680E3	11	2	80	yes	-	-	-	-	-
deformation_Pipeline_VITIS_LOOP_257_4			208	2,080E3	-	-	-	no	0	0	-0	-0	0
VITIS_LOOP_257_4	Resource Limitation		206	2,060E3	9	2	100	yes	-	-	-	-	-
VITIS_LOOP_238_2			-	-	-	-	-	no	-	-	-	-	-
snake_deform_1			-	-	-	-	-	no	2	11	5	17	0
snake_interp			-	-	-	-	-	no	0	2	2	13	0
deformation_Pipeline_VITIS_LOOP_243_3			102	1,020E3	-	-	-	no	0	0	-0	-0	0

(d)Résumé de synthèse pour la fonction ”Déformation de contour”

FIG. 4.29 : Résumé de synthèse pour chaque partie

4.4.2.3 Création du design par blocs

En suivant les mêmes étapes présentées dans le section 4.4.1.3, nous allons créer le design par blocs correspondant à notre architecture (voir figure 4.27) et générer les fichiers nécessaires pour la suite.

La figure 4.30 représente le design par blocs final de notre architecture pour l’implémentation complète. (Figure 4.30 plus claire dans **Annexe 3**)

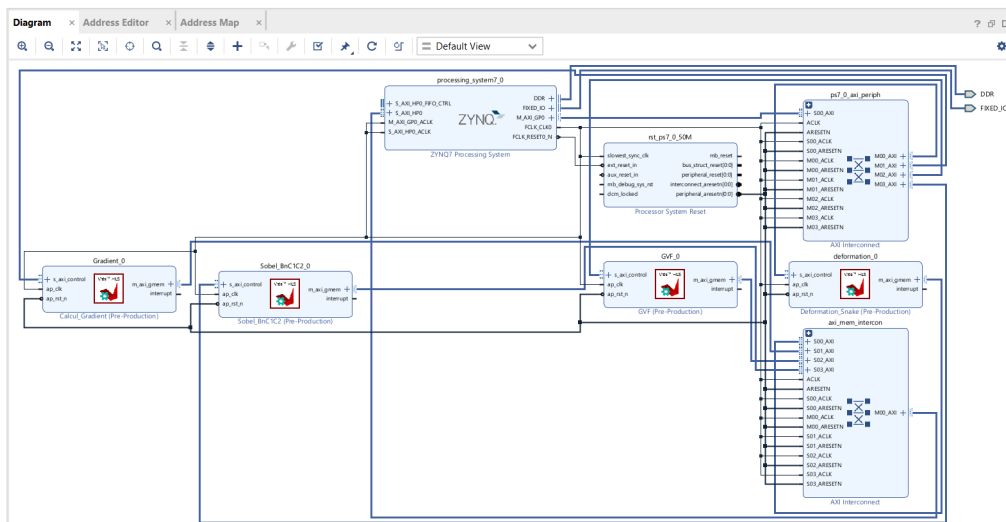
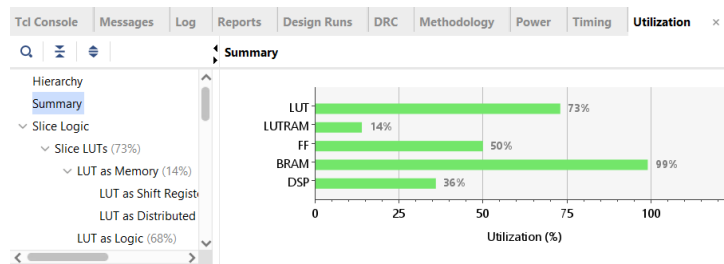
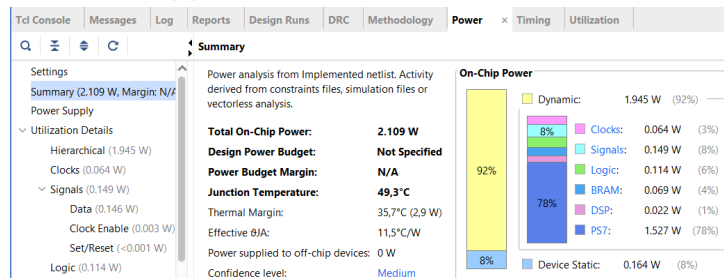


FIG. 4.30 : Design par blocs pour l’implémentation complète de l’algorithme des contours actifs par la méthode GVF

La figure 4.31 présente les rapports d’utilisation et de puissance de l’architecture complète.



(a) Rapport d'utilisation résumé



(b) Rapport de puissance

FIG. 4.31 : Résumé d'implémentation complète de l'algorithme des contours actifs par la méthode GVF

4.4.2.4 Programmation du CPU

Nous suivons les mêmes étapes présentées dans le section 4.4.1.4. En gardant encore une fois la même configuration de paramètres et le même contour initial, voici le résultat final des 2 sorties x_f et y_f obtenus sur MATLAB et obtenus par implémentation (figures 4.32 et 4.33).

Les différences numériques que nous pouvons observer entre les résultats MATLAB et FPGA sont minimales voire imperceptibles en termes de l'impact visuel sur l'image segmentée.

Explication :

Des différences de l'ordre de quelques unités dans les valeurs de contour ne modifient pas de manière significative la forme globale du contour.

Lorsque les résultats sont visualisés, l'échelle et la résolution de l'image peuvent également masquer de petites différences numériques. Les pixels d'une image segmentée ont une taille fixe, et des écarts minimes dans les valeurs numériques des contours peuvent ne pas être visibles à cette échelle.

Chapitre 4. Implémentation sur FPGA

```
>> x(1:80)
ans =
Columns 1 through 18
 98.6145  99.3946 100.4902 101.3922 102.1074 102.5701 103.3281 103.8197 104.2151 104.5697 105.4202 106.0273 105.6541 104.7087 104.1401 103.5716 102.9503 102.3467
Columns 19 through 36
101.4848 100.3197 99.2238 98.4319 96.0904 94.6350 92.7129 90.9329 89.3906 88.0050 85.4036 83.9031 82.4025 80.4494 78.6494 76.3490 74.2668 72.8909
Columns 37 through 54
 71.8986  70.8359  69.5955  68.3551  67.7052  67.4201  66.2917  65.1633  64.1165  62.8520  61.5261  60.2220  59.5333  59.0948  58.7342  58.7236  58.8764  59.0395
Columns 55 through 72
 57.7505  57.3873  59.1519  60.9709  62.6264  63.8916  65.1567  66.5571  67.9575  69.7884  71.9120  73.7297  75.3810  76.8174  78.2538  80.3401  82.2290  83.6844
```

(a) Output_xf sur Matlab

```
[23]: output_buffer7[0:80]
[23]: PynqBuffer([101.892746, 102.336586, 103.124084, 103.73271 , 104.14214 ,
 104.47731 , 105.15401 , 105.96393 , 105.98734 , 105.44485 ,
 104.55871 , 103.995895, 103.433075, 102.74962 , 102.14796 ,
 101.11641 , 99.76937 , 98.70423 , 96.79897 , 94.64484 ,
 93.11989 , 91.59495 , 89.58453 , 88.051384, 85.74751 ,
 83.18086 , 81.8929 , 80.27954 , 78.305435, 75.72746 ,
 73.50327 , 72.49473 , 71.346436, 70.19814 , 68.020775,
 67.67206 , 67.41038 , 66.6719 , 65.022285, 64.02381 ,
 62.66941 , 61.409462, 60.376484, 59.926613, 59.47674 ,
 59.117577, 58.758408, 58.722652, 58.88156 , 58.8567 ,
 57.2992 , 57.629208, 59.66721 , 61.5639 , 63.23698 ,
 65.39476 , 67.80245 , 69.27254 , 71.304726, 73.18732 ,
```

(b) Output_buffer7 (Output_xf) sur jupyter notebook

FIG. 4.32 : Valeurs de xf sur MATLAB et sur notebook

```
>> y(1:80)
ans =
Columns 1 through 18
109.7021 108.2210 106.6622 104.9483 103.2591 101.7880 100.3363 98.5793 96.5386 94.8902 93.3333 91.3109 89.1426 86.9006 85.3781 83.8555 81.6731 79.5711
Columns 19 through 36
 77.4389  75.6958  74.5206  73.7561  72.1527  71.2784  70.3245  69.3035  68.6299  68.1836  67.4247  67.0408  66.6569  66.5091  66.5895  67.1342  68.2534  69.6220
Columns 37 through 54
 71.1383  72.2718  73.3403  74.4087  75.4050  78.0824  79.2107  80.3391  81.3544  82.7206  84.6999  86.8922  88.8335  91.0107  93.2491  95.3236  97.5086  99.6821
Columns 55 through 72
101.2437 102.5547 103.3357 104.2539 105.4292 106.2944 107.1597 107.8306 108.5015 109.0256 109.4924 110.2813 111.0970 111.6688 112.2406 112.9795 113.8018 114.8835
```

(a) Output_yf sur Matlab

```
[24]: output_buffer8[0:80]
[24]: PynqBuffer([103.60872 , 102.687164, 100.859184, 98.88594 , 96.99187 ,
 95.265976, 93.866875, 92.15306 , 90.22622 , 88.56539 ,
 86.27157 , 84.70569 , 83.1398 , 81.01372 , 78.90132 ,
 76.71142 , 75.02352 , 74.0988 , 72.697556, 71.432106,
 70.46445 , 69.49678 , 68.53635 , 68.363716, 67.53088 ,
 66.71518 , 66.5528 , 66.5162 , 66.71126 , 67.676384,
 69.153534, 70.25518 , 71.39131 , 72.527435, 74.423416,
 75.46318 , 78.082436, 78.89605 , 80.407074, 81.33248 ,
 82.85907 , 84.71172 , 86.64365 , 88.18964 , 89.73563 ,
 91.267746, 92.79987 , 94.99673 , 97.413155, 99.98518 ,
 101.95102 , 102.94115 , 103.50231 , 104.46217 , 105.69542 ,
 107.19181 , 108.4564 , 108.68986 , 109.21243 , 109.718094,
```

(b) Output_buffer8 (Output_yf) sur jupyter notebook

FIG. 4.33 : Valeurs de yf sur MATLAB et sur notebook

Nous pouvons visualiser le résultat de la segmentation avec la méthode GVF en utilisant l'approche "implémentation complète" dans la figure 4.34. Nous remarquons que la

segmentation est bien correcte est semblable visuellement à celle obtenue avec MATLAB.

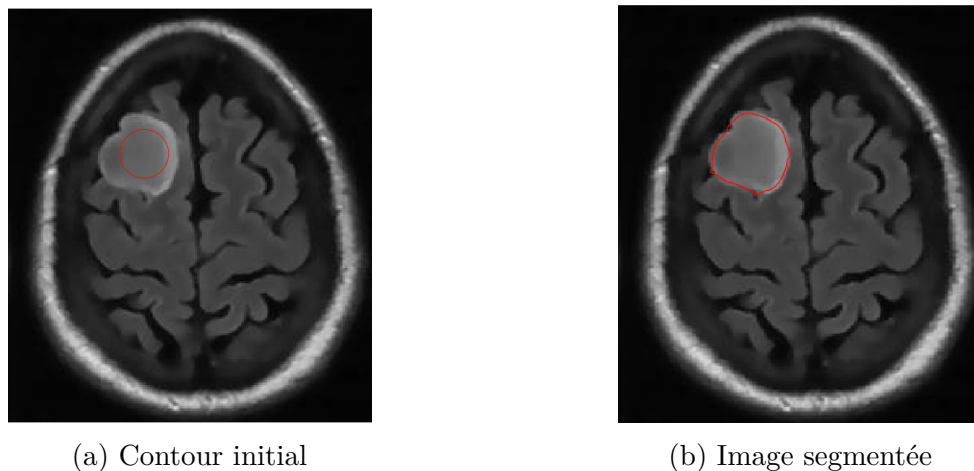


FIG. 4.34 : Résultat de l'implémentation complète de la méthode GVF

Le temps d'exécution de l'architecture complète de la méthode GVF est de **12.786s**.

4.5 Implémentation de l'algorithme DRLSE

Nous adoptons une approche similaire à celle de l'algorithme GVF, en décomposant le processus en différentes étapes et en fournissant les résumés de synthèse et d'implémentation ainsi que les résultats obtenus.

4.5.1 Implémentation par parties

Nous débutons aussi par une approche par parties pour cet algorithme.

4.5.1.1 Codage en C

- **Partie 01 : Convolution + Calcul de g + Gradient (g)**

La figure 4.35 montre un schéma de la partie 01 :

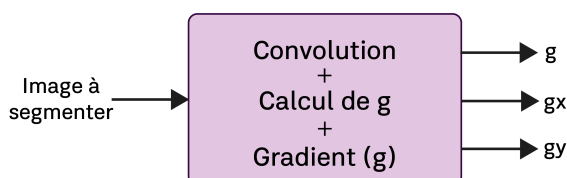


FIG. 4.35 : Schéma de la partie 01 de l'algorithme DRLSE

La fonction "Convolution + Calcul de g + Gradient (g)" prend une image filtrée en entrée et produit les valeurs de g (les contours de l'image) ainsi que g_x et g_y (gradients horizontal et vertical).

- **Partie 02 : Evolution de la LSF**

La figure 4.36 montre un schéma de la partie 02:



FIG. 4.36 : Schéma de la partie 02 de l’algorithme DRLSE

La fonction "Evolution de la LSF" prend la fonction de distance initiale ϕ_i , g , gx et gy et produit ϕ_f en sortie.

Lors de chaque itération, ϕ_f est calculée en utilisant une équation d’évolution qui combine ces données. Les contours de l’image (g) agissent comme une force externe qui attire la frontière de ϕ vers les contours détectés dans l’image, permettant ainsi à la LSF de suivre les contours précisément. Les gradients (gx) et (gy) sont utilisés pour ajuster la frontière de manière à suivre les changements d’intensité et de texture de l’image, assurant une segmentation précise et détaillée.

4.5.1.2 Création des blocs IP

La figure 4.37 présente le résumé de synthèse pour chaque partie.

Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
Conv_Grad			37806955	3.780695	-	37806956	-	no	91	4	4	13	0
Conv_Grad_Pipeline_1			57377	5.740E5	-	57377	-	no	0	0	-0	-0	0
Loop 1			57375	5.740E5	1	1	57375	yes	-	-	-	-	-
Conv_Grad_Pipeline_2			57377	5.740E5	-	57377	-	no	0	0	-0	-0	0
Loop 1			57375	5.740E5	1	1	57375	yes	-	-	-	-	-
Conv_Grad_Pipeline_VITIS_LOOP_84_5_VITIS_LOOP_86_6	Resource Limitation		459048	4.590E6	-	459048	-	no	0	0	-0	1	0
VITIS_LOOP_84_5_VITIS_LOOP_86_6	Resource Limitation		459046	4.590E6	48	8	57375	yes	-	-	-	-	-
Conv_Grad_Pipeline_VITIS_LOOP_88_1_VITIS_LOOP_100_8	Resource Limitation		172143	1.721E6	-	172143	-	no	0	-0	1	3	0
VITIS_LOOP_88_1_VITIS_LOOP_100_8	Resource Limitation		172141	1.721E6	20	3	57375	yes	-	-	-	-	-
VITIS_LOOP_64_1_VITIS_LOOP_65_2			36318375	3.630E8	633	-	57375	no	-	-	-	-	-
Conv_Grad_Pipeline_VITIS_LOOP_68_3_VITIS_LOOP_69_4			629	6.290E3	-	629	-	no	0	-0	-0	1	0

(a) Résumé de synthèse pour la fonction "Convolution + Calcul de g + Gradient (g)"

Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
DRLSE			2811537574	28115000000.000	-	2811537575	-	no	94	39	19	45	0
DRLSE_Pipeline_VITIS_LOOP_76_1			57378	5.740E5	-	57378	-	no	0	0	-0	-0	0
VITIS_LOOP_76_1			57376	5.740E5	3	1	57375	yes	-	-	-	-	-
DRLSE_Pipeline_2			5	50.000	-	5	-	no	0	0	-0	-0	0
DRLSE_Pipeline_2			5	50.000	-	5	-	no	0	0	-0	-0	0
DRLSE_Pipeline_4			5	50.000	-	5	-	no	0	0	-0	-0	0
DRLSE_Pipeline_4			5	50.000	-	5	-	no	0	0	-0	-0	0
DRLSE_Pipeline_VITIS_LOOP_264_10			57378	5.740E5	-	57378	-	no	0	0	-0	-0	0
VITIS_LOOP_264_10			57376	5.740E5	3	1	57375	yes	-	-	-	-	-
VITIS_LOOP_92_2			2811422800	28114000000.000	14057114	-	200	no	-	-	-	-	-
DRLSE_Pipeline_VITIS_LOOP_94_3_VITIS_LOOP_96_4			13999732	1.400E8	-	13999732	-	no	2	39	17	38	0
VITIS_LOOP_94_3_VITIS_LOOP_96_4			13999730	1.400E8	475	244	57375	yes	-	-	-	-	-
gradientx			11	110.000	-	1	-	yes	0	3	1	4	0
gradienty			12	120.000	-	1	-	yes	0	3	1	3	0
sin_of_cos_double_s			32	320.000	-	1	-	yes	2	24	6	12	0
DRLSE_Pipeline_VITIS_LOOP_257_9			57377	5.740E5	-	57377	-	no	0	0	-0	-0	0

(b) Résumé de synthèse pour la fonction "Evolution de la LSF"

FIG. 4.37 : Résumé de synthèse pour chaque partie

4.5.1.3 Création du design par blocs

La figure 4.38 illustre le design par blocs de la partie 2 qui représente l'évolution de la LSF de l'algorithme DRLSE.

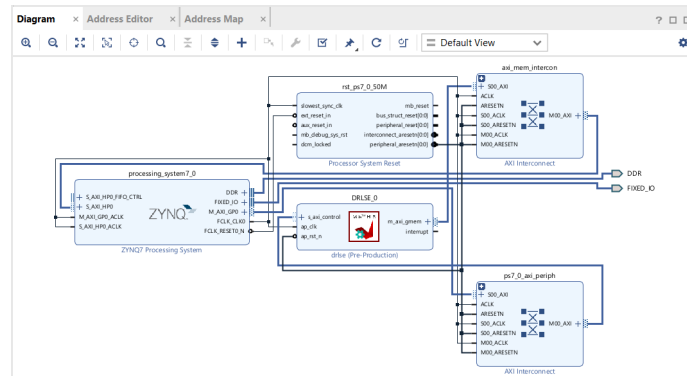
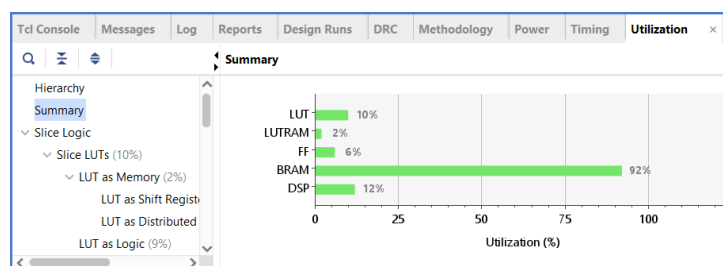
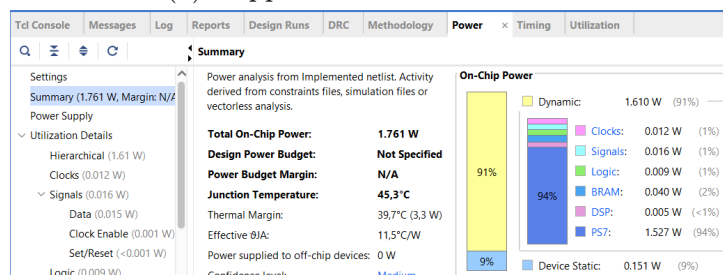


FIG. 4.38 : Schéma de la partie 02 de l'algorithme DRLSE

Les figures 4.39 et 4.40 présentent les rapports d'utilisation et de puissance pour chacune des 2 parties de l'algorithme.

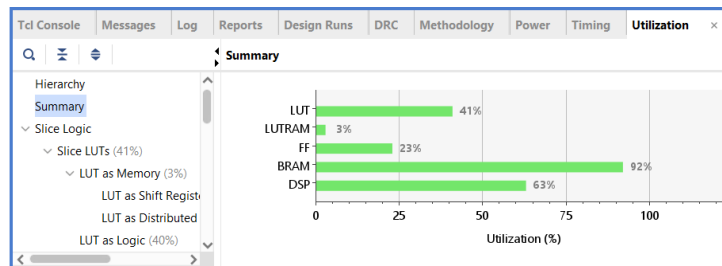


(a) Rapport d'utilisation résumé

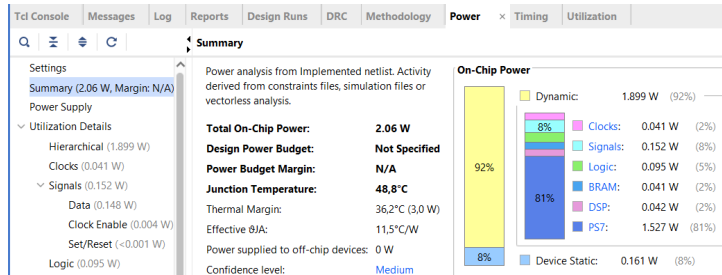


(b) Rapport de puissance

FIG. 4.39 : Résumé d'implémentation de la partie 1 de l'algorithme DRLSE



(a) Rapport d'utilisation résumé



(b) Rapport de puissance

FIG. 4.40 : Résumé d'implémentation de la partie 2 de l'algorithme DRLSE

4.5.1.4 Programmation du CPU

Comme dans l'approche de l'implémentation par parties de la méthode GVF. Nous récupérons les sorties du bloc de la partie 1 dans des fichiers .csv, pour qu'ils soient les entrées du bloc de la partie 2.

La figure 4.41 montrent quelques valeurs des différentes sorties de chaque partie. Nous remarquons que ces valeurs sont proches que celles obtenues sur MATLAB (figure 4.42).

```
[13]: print(output_buffer1[0:20])
[0.52358794 0.48237947 0.47867724 0.47160792 0.46916226 0.47004473
0.47361377 0.48174134 0.49708956 0.51853144 0.5432761 0.57233727
0.6093398 0.6517438 0.67321414 0.64749235 0.6101828 0.5917913
0.5840599 0.58047885]

[14]: print(output_buffer2[0:20])
[-0.04120848 -0.02245535 -0.00538577 -0.00475749 -0.0007816 0.00222576
0.0058483 0.0117379 0.01839505 0.02309325 0.02690291 0.03303185
0.03970328 0.03193718 -0.00212574 -0.03151566 -0.02785054 -0.01306146
-0.00565621 -0.00213638]

[15]: print(output_buffer3[0:20])
[-0.04120848 0.03568813 0.11387053 0.13563585 0.13735452 0.13483799
0.12667355 0.11131915 0.0942831 0.08110738 0.07021785 0.05934066
0.05176014 0.05321497 0.06009722 0.06208968 0.07424206 0.10107553
0.12025708 0.12609577]
```

(a) Output_buffer1 (Output_g), Output_buffer12 (Output_gx), Output_buffer3 (Output_gy)

```
[19]: print(output_buffer1[0:20])
[1.1017432 2.7950919 1.2136785 1.4258692 1.2040855 3.3212986 1.2828941
1.7033999 1.1227753 4.510457 1.7712553 2.0196269 1.2711915 2.3620524
1.4687375 1.7736763 1.1793413 2.894847 1.2871398 1.7111108]
```

(b) Output_buffer1 (Output_phi)

FIG. 4.41 : Sorties des 2 parties de l'algorithme DRLSE sur le notebook

```

>> g(1,1:20)
ans =
Columns 1 through 9
0.523832530414463 0.482602340625202 0.478908939078949 0.471919405962340 0.469444762059259 0.470231729557766 0.473817448351139 0.481988217358305 0.497276913290887
Columns 10 through 18
0.518719921040214 0.543528738220746 0.572531600352863 0.609422456818807 0.651866368638467 0.673419284181158 0.647594145213559 0.610325207283058 0.592037274267874
Columns 19 through 20
0.584290252493051 0.580730114042178
>> vx(1,1:20)
ans =
Columns 1 through 9
-0.041230189789261 -0.022461795667757 -0.005341467331431 -0.004732088509845 -0.000843838202287 0.002186343145940 0.005878243900270 0.011729732469874 0.018365851840954
Columns 10 through 18
0.023125912464930 0.026905839656324 0.032946859299031 0.039667384142802 0.031998413681176 -0.002136111712454 -0.031547038449050 -0.027778435472842 -0.013017477395003
Columns 19 through 20
-0.005653580112848 -0.002129864948527
>> vy(1,1:20)
ans =
Columns 1 through 9
-0.041230189789261 0.035602183924424 0.113814500180008 0.135486621296669 0.137185986623032 0.134723429331974 0.126514555795222 0.111146104745426 0.094201626499412
Columns 10 through 18
0.080984843184918 0.070067615432753 0.059267788780099 0.051718779795498 0.053014272621534 0.059933743659038 0.062044598795001 0.074191260025400 0.100912725826634
Columns 19 through 20
0.120139052062012 0.125922264500620

```

(a) Output_g, Output_gx, Output_gy

```

>> phi(1,1:20)
ans =
Columns 1 through 9
0.798850268740311 1.186468556390159 1.244966416207252 1.544574766166462 1.036688122472965 3.438513258854408 1.701483967673916 2.018159762697187 1.207912861560892
Columns 10 through 18
3.257268203983677 1.856417490089076 2.200400207503354 1.498331031871051 1.684018383607468 1.267793305526204 1.671987267032303 1.126322552893067 3.431546038264419
Columns 19 through 20
1.716123525838333 1.98023358537994

```

(b) Output_phi

FIG. 4.42 : Sorties des 2 parties de l'algorithme DRLSE sur Matlab

Les différences d'ordre de 10^{-5} que nous pouvons remarquer dans les valeurs de g , gx et gy peut être expliqué de la même manière que pour les valeurs de fx dans l'implémentation de l'algorithme GVF.

Les méthodes basées sur les level sets cherchent à minimiser une énergie définie, et les variations mineures dans ces valeurs intermédiaires ne changent pas le minimum local vers lequel l'algorithme converge. Ceci signifie que la forme globale du contour ne change pas significativement.

Après 300 itérations, nous récupérons output_phi dans un fichier .csv et nous obtenons le résultat de la segmentation illustré dans la figure 4.43

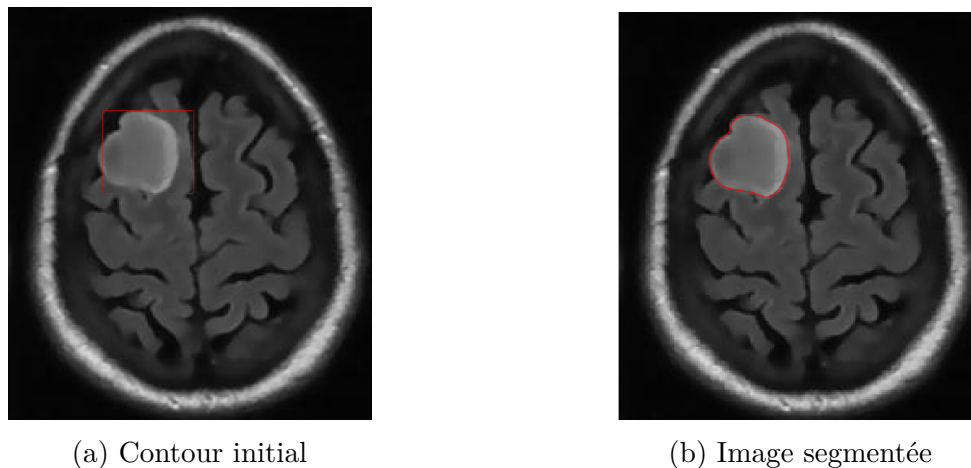


FIG. 4.43 : Résultat de l'implémentation par parties de la méthode DRLSE

Le résultat de la segmentation obtenu par l'implémentation par parties de l'algorithme DRLSE est correct et en le comparant au résultat obtenu par la simulation sur MATLAB, nous ne remarquons aucune différence ou erreur.

En comparant les valeurs de ϕ des figures 4.41(b) et 4.42(b), nous pouvons remarquer quelques différences, mais les valeurs obtenues avec l'implémentation ne divergent pas de celles obtenues sur MATLAB.

Explication :

Les variations de quelques dixièmes dans les valeurs de ϕ (comme 0.79 vs 1.10) sont imperceptibles lors de la visualisation de l'image segmentée car les contours générés à partir de ϕ sont tracés en utilisant un seuil qui lisse les petites différences.

Typiquement, ϕ prend des valeurs positives à l'extérieur du contour et des valeurs négatives à l'intérieur, et le contour lui-même est représenté par le lieu des points où $\phi = 0$.

Donc, nous n'avons pas besoin que les valeurs de ϕ soient exactement les mêmes, mais juste qu'elles ne divergent pas beaucoup pour ne pas fausser le contour final lors de l'évolution de ϕ . La magnitude de ϕ affecte la stabilité numérique de l'algorithme. Si les valeurs de ϕ deviennent trop grandes ou trop petites, cela peut conduire à des problèmes de stabilité ou à une convergence lente.

Le tableau 4.2 résume le temps de calculs de chaque partie de l'algorithme DRLSE.

Partie	1	2
Temps de calculs (s)	3.0803	97.3604

TAB. 4.2 : Temps d'exécution de chaque partie de l'algorithme DRLSE

4.5.2 Implémentation complète

Nous présentons dans ce qui suit l'implémentation complète de l'algorithme DRLSE en suivant les étapes mentionnées précédemment.

4.5.2.1 Codage en C

Face à la surconsommation de ressources pour les différentes parties de l'algorithme DRLSE, une approche similaire à celle utilisée pour l'algorithme GVF a été adoptée. Dans cette optique, la première partie a été subdivisée en trois blocs distincts : convolution, calcul de g , et calcul du gradient (g). Cette subdivision permet une gestion plus efficace des ressources matérielles en stockant les sorties de chaque bloc dans la mémoire DDR, puis en les récupérant par les blocs suivants.

La figure 4.44 illustre un schéma des différents blocs de cette approche

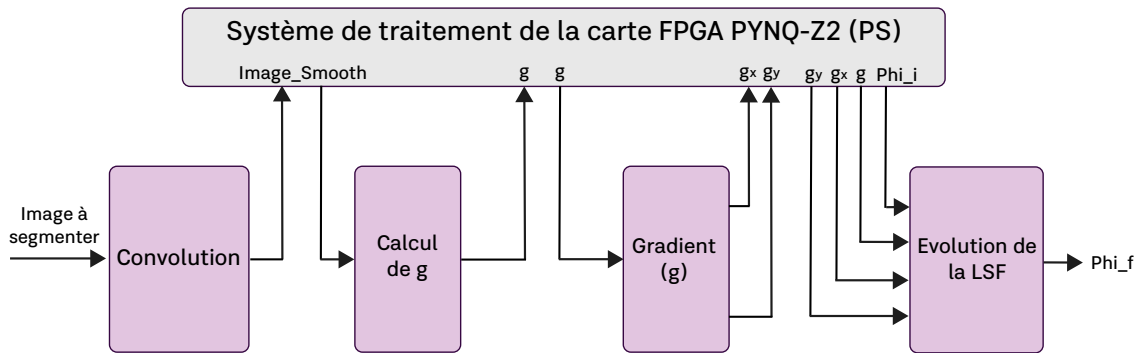


FIG. 4.44 : Schéma des différents blocs de l'implémentation complète de l'algorithme DRLSE

4.5.2.2 Création des blocs IP

La figure 4.45 présente le résumé de synthèse pour chaque bloc du schéma présenté en figure 4.44.

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)	
FilterGauss				36318382	3.630E8		36318383		no	-0	2	3	1	6	0
VITIS_LOOP_26_1_VITIS_LOOP_27_2				36318375	3.630E8	633	57375		no	-	-	-	-	-	-
FilterGauss_Pipeline_VITIS_LOOP_29_3_VITIS_LOOP_30_4				629	6.290E3		629		no	-0	2	-0	2	0	0
VITIS_LOOP_29_3_VITIS_LOOP_30_4				627	6.270E3		28	5	121	yes	-	-	-	-	-

(a) Résumé de synthèse pour la fonction "Convolution"

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
EdgeDetect				688572	6.886E6		688573		no	0	2	3	6	0
EdgeDetect_Pipeline_VITIS_LOOP_41_1_VITIS_LOOP_43_2				688564	6.886E6		688564		no	0	2	2	5	0
VITIS_LOOP_41_1_VITIS_LOOP_43_2	Resource Limitation			688562	6.886E6	64	12	57375	yes	-	-	-	-	-

(b) Résumé de synthèse pour la fonction "Calcul de g"

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
Gradient				688551	6.886E6		688552		no	0	2	4	9	0
VITIS_LOOP_41_1_VITIS_LOOP_43_2	Resource Limitation			688549	6.885E6	62	12	57375	yes	-	-	-	-	-

(c) Résumé de synthèse pour la fonction "Gradient (g)"

Performance & Resource Estimates

Modules & Loops	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
DRLSE				2811537674	28115000000.000		2811537675		no	94	39	19	45	0
DRLSE_Pipeline_LOOP_76_1				57378	5.740E5		57378		no	0	0	-0	-0	0
VITIS_LOOP_76_1				57376	5.740E5	3	1	57375	yes	-	-	-	-	-
DRLSE_Pipeline_2				5	50.000		5		no	0	0	-0	-0	0
DRLSE_Pipeline_3				5	50.000		5		no	0	0	-0	-0	0
DRLSE_Pipeline_4				5	50.000		5		no	0	0	-0	-0	0
DRLSE_Pipeline_5				5	50.000		5		no	0	0	-0	-0	0
DRLSE_Pipeline_VITIS_LOOP_264_10				57378	5.740E5		57378		no	0	0	-0	-0	0
VITIS_LOOP_264_10				57376	5.740E5	3	1	57375	yes	-	-	-	-	-
VITIS_LOOP_92_2				28114232800	28114000000.000	14057114		200	no	-	-	-	-	-
DRLSE_Pipeline_VITIS_LOOP_94_3_VITIS_LOOP_96_4				13999732	1.400E8		13999733		no	2	39	17	38	0
VITIS_LOOP_94_3_VITIS_LOOP_96_4				13999730	1.400E8	475	244	57375	yes	-	-	-	-	-
gradientx				11	110.000		1		yes	0	3	1	4	0
gradienty				12	120.000		1		yes	0	3	1	3	0
sin_or_cos_double_s				32	320.000		1		yes	2	24	6	12	0
DRLSE_Pipeline_VITIS_LOOP_257_9				57377	5.740E5		57377		no	0	0	-0	-0	0

(d) Résumé de synthèse pour la fonction "Évolution de la LSF"

FIG. 4.45 : Résumé de synthèse pour chaque partie

4.5.2.3 Création du design par blocs

La figure 4.46 représente le design par blocs final de notre architecture pour l'implémentation complète. (figure 4.46 plus claire dans **Annexe 3**)

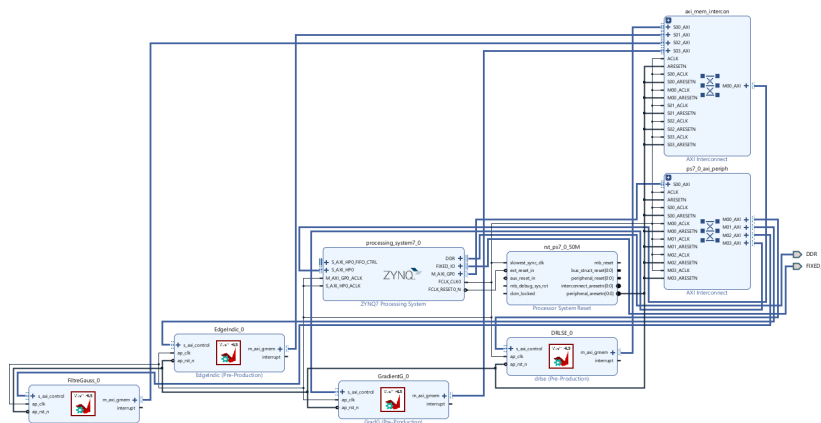
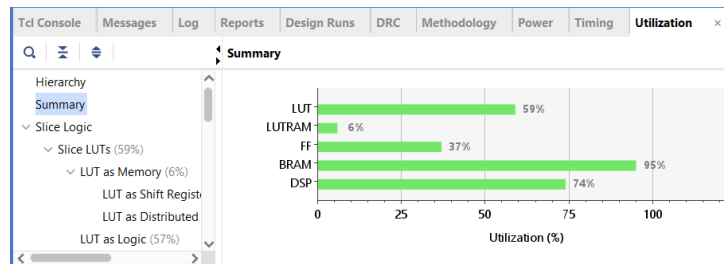
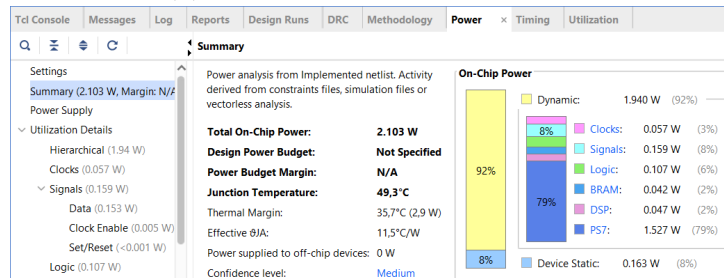


FIG. 4.46 : Design par blocs pour l'implémentation complète de l'algorithme des courbes de niveau par l'approche contours DRLSE

La figure 4.47 présente les rapports d'utilisation et de puissance de l'implémentation complète du DRLSE.



(a) Rapport d'utilisation résumé



(b) Rapport de puissance

FIG. 4.47 : Résumé d'implémentation complète de l'algorithme des courbes de niveau par l'approche contours DRLSE

4.5.2.4 Programmation du CPU

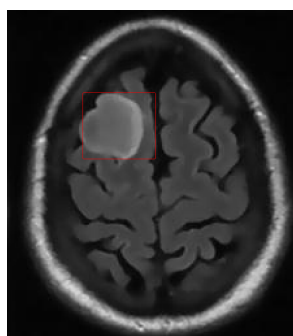
La figure 4.48 montre le résultat final de la sortie phi_f (image segmentée).

```
output_buffer5[0:80]
PynqBuffer([1.6495429, 2.5488818, 1.2177163, 1.5830121, 1.1438079,
3.5258772, 1.3948499, 1.7473769, 1.1474257, 2.5384138,
1.5785121, 2.1917121, 1.0950056, 1.8367847, 1.330897,
2.3490212, 1.2625023, 2.0102606, 1.8074617, 3.5280738,
1.1275003, 1.9318712, 1.7607563, 4.382491, 1.1735752,
```

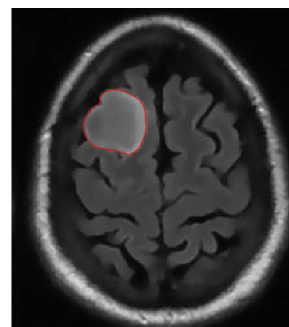
FIG. 4.48 : Output_buffer5 (Output_phi)

Avec 300 itérations, les résultats sont les mêmes que ceux obtenus lors de l'implémentation par parties.

Nous pouvons visualiser le résultat de la segmentation avec la méthode DRLSE en utilisant l'approche "implémentation complète" (figure 4.49).



(a) Contour initial



(b) Image segmentée

FIG. 4.49 : Résultat de l'implémentation complète de la méthode DRLSE

Le résultat de la segmentation est satisfaisant.

Le temps d'exécution de l'architecture complète de la méthode DRLSE est de **105.8523s**.

4.6 Implémentation de l'algorithme LSE-BFE

Nous présentons dans ce qui suit l'implémentation de l'algorithme LSE-BFE.

4.6.1 Codage en C

Compte tenu des contraintes de ressources de la FPGA et des calculs intensifs requis par l'algorithme LSE-BFE, nous avons pris en considération ces problèmes en modifiant le code C par rapport à son équivalent en MATLAB. Dans cette adaptation, nous avons simplifié le processus en supprimant une phase de calcul, ce qui signifie que nous avons implémenté l'algorithme LSE-BFE avec seulement **deux phases** (utilisation de deux fonction d'appartenance $M(1)$ et $M(2)$) au lieu de trois. Cette modification permet de réduire la complexité et la charge de calcul. De plus, nous avons utilisé le principe des **buffers** pour stocker les variables intermédiaires, plutôt que de conserver toutes les matrices dans la mémoire, ce qui contribue à l'utilisation plus efficace des BRAM mais qui rajoute une charge de calculs supplémentaires.

La figure 4.50 illustre le schéma de cet algorithme.

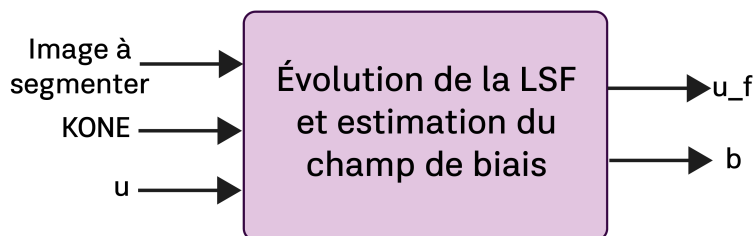


FIG. 4.50 : Schéma de l'algorithme LSE-BFE

La fonction "Evolution de la LSF et estimation du champ de biais" prend en entrées l'image à segmenter, KONE qui est le résultat de convolution d'image avec un filtre gaussien et une fonction initial u . Elle réalise ensuite l'évolution de fonction de niveau et l'estimation du champs de biais, ce qui permet de corriger les inhomogénéités d'intensité dans l'image, En sortie, elle produit la fonction de niveau final u_f , qui représente les contours segmentés de l'image et le champs de biais final.

4.6.2 Création des blocs IP

- **Défi initial :**

Lors de notre première expérience avec l'algorithme LSE-BFE sur Vitis HLS, nous avons rencontré un défi majeur :

La latence était extrêmement élevée, avec une estimation de temps d'exécution de **24 heures**, ce qui était inacceptable.

- **Solution initiale :**
 - Pour remédier à cette situation, nous avons opté pour le principe du **pipeline** : Cette approche a considérablement réduit le temps d'exécution à seulement **16 minutes**.
- **Nouveau problème identifié :**
 - Cependant, nous avons rapidement identifié un nouveau problème : une surutilisation des LUTs ; suite à l'application du pipeline.
 - Cette **surutilisation** dépassait les 100%, ce qui n'était pas envisageable pour une implémentation matérielle.
- **Analyse de la cause :**
 - Après avoir analysé le code C ainsi que le rapport de synthèse sur HLS, nous avons découvert que la fonction **arctan** était la principale source de cette surutilisation.
 - Cette fonction était utilisée dans le calcul de la fonction Heaviside, composant nécessaire dans notre algorithme.
- **Nouvelle approche adoptée :**
 - Plutôt que d'importer la fonction arctan depuis la bibliothèque *math.h*, nous avons décidé d'utiliser une **approximation** basée sur le **développement en série de Taylor**.
 - Cette approximation nous a permis de réduire considérablement la charge sur les LUTs, tout en maintenant la précision nécessaire pour notre application.

La figure 4.51 présente le dernier rapport de synthèse sur Vitis HLS auquel nous avons aboutit.

Modules & Loops	Violation Type	Distance	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)	URAM(%)
└─ LSE_BFE			111039077192	1100000000000.000	-	111039077193	-	no	84	58	26	71	0
└─└─ LSE_BFE_Pipeline_VITIS_LOOP_263_1			41587	4,160E5	-	41587	-	no	0	0	-0	-0	0
└─└─ LSE_BFE_Pipeline_VITIS_LOOP_269_2			41586	4,160E5	-	41586	-	no	0	0	-0	-0	0
└─└─ LSE_BFE_Pipeline_VITIS_LOOP_411_18			41587	4,160E5	-	41587	-	no	0	0	-0	-0	0
└─ VITIS_LOOP_274_3			111038994001	1100000000000.000	370129980	-	300	no	-	-	-	-	-
└─└─ LSE_BFE_Pipeline_VITIS_LOOP_350_9_VITIS_LOOP_352_9			2641	2,641E4	-	2641	-	no	0	-0	2	2	0
└─└─ VITIS_LOOP_276_4			68738720	6,870E8	373550	-	184	no	-	-	-	-	-
└─└─└─ VITIS_LOOP_278_5			373578	3,735E6	1653	-	226	no	-	-	-	-	-
└─└─└─└─ filtreSigma			742	7,420E3	-	742	-	no	-0	3	-0	2	0
└─└─ VITIS_LOOP_306_6			143881008	1,439E9	781962	-	184	no	-	-	-	-	-
└─└─└─ VITIS_LOOP_308_7			781960	7,820E6	3460	-	226	no	-	-	-	-	-
└─└─└─└─ calcul_curv			1661	1,661E4	-	1661	-	no	0	11	5	13	0
└─└─└─└─ VITIS_LOOP_308_7.1			9	90,000	1	-	9	no	-	-	-	-	-
└─└─└─└─ VITIS_LOOP_215_1			735	7,350E3	245	-	3	no	-	-	-	-	-
└─└─ VITIS_LOOP_361_10			1760100	1,760E7	293350	-	6	no	-	-	-	-	-
└─└─└─ VITIS_LOOP_363_11			293348	2,933E6	1298	-	226	no	-	-	-	-	-
└─└─└─└─ conv_1			639	6,390E3	-	639	-	no	-0	5	1	4	0
└─└─ VITIS_LOOP_371_12_VITIS_LOOP_373_13			154280708	1,543E9	3946	-	39098	no	-	-	-	-	-
└─└─└─ LSE_BFE_Pipeline_VITIS_LOOP_375_14_VITIS_LOOP_377			2644	2,644E4	-	2644	-	no	0	-0	3	4	0
└─└─ VITIS_LOOP_399_16			1466750	1,467E7	293350	-	5	no	-	-	-	-	-

FIG. 4.51 : Rapport de synthèse de l'algorithme LSE-BFE

4.6.3 Création du design par blocs

Nous poursuivons maintenant l'implémentation avec la création du design par bloc pour notre système (figure 4.52, plus claire dans **Annexe 3**).

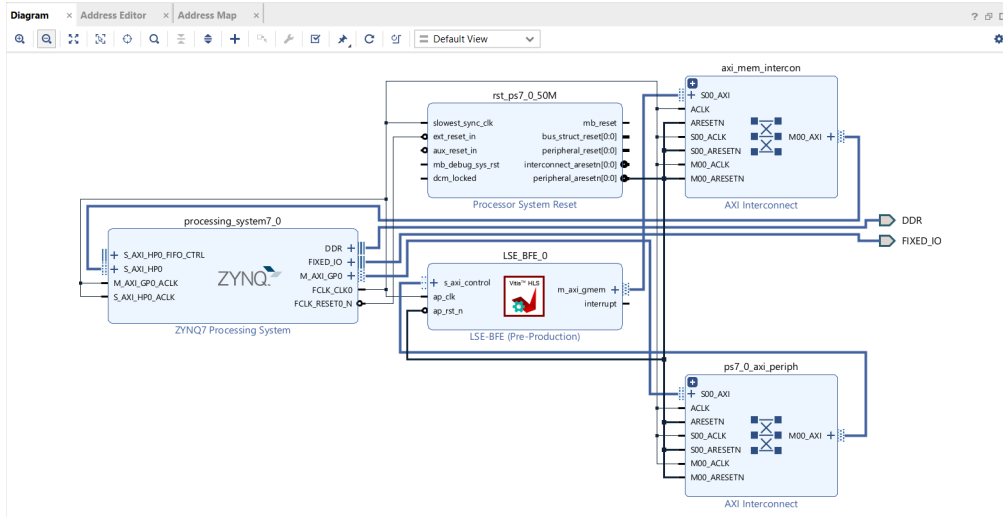
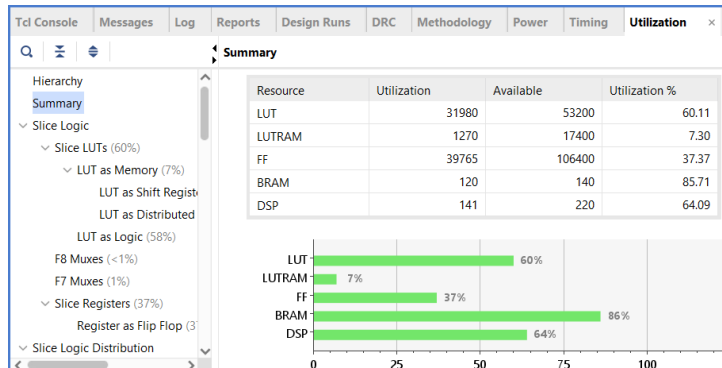
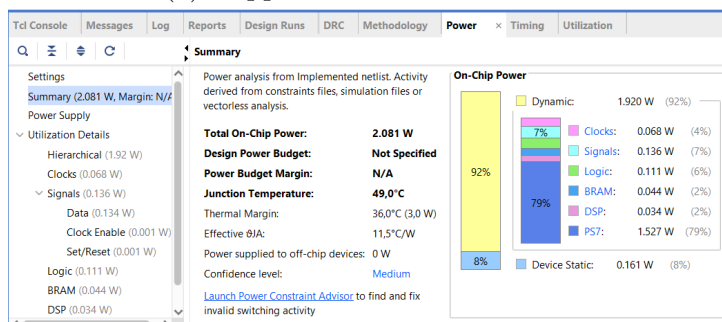


FIG. 4.52 : Design par blocs pour l'implémentation de l'algorithme LSE-BFE

Après le succès de la synthèse et de l'implémentation sur Vivado, nous pouvons visualiser les rapports d'utilisation et de puissance dans la figure 4.53.



(a) Rapport d'utilisation résumé



(b) Rapport de puissance

FIG. 4.53 : Résumé d'implémentation de l'algorithme des courbes de niveau par l'approche régions LSE-BFE

4.6.4 Programmation du CPU

Pour cet algorithme, nous avons souhaité l'implémenter pour l'IRM du sein, car LSE-BFE est la méthode qui avait donner les meilleurs résultats de segmentation pour cette image médicale en MATLAB.

Comme précédemment, nous récupérons le résultat de "Output_u" sur un fichier .csv, nous permettant de visualiser le résultat de la segmentation avec la méthode LSE-BFE (figure 4.54).

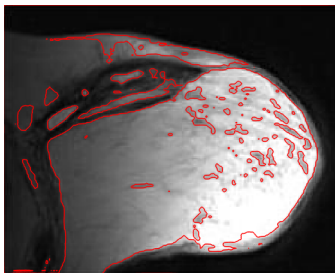


FIG. 4.54 : Résultat de l'implémentation complète de la méthode LSE-BFE

En comparant ce résultat à celui obtenu avec MATLAB, nous pouvons observer que l'image segmentée présente quelques erreurs de segmentation dues aux approximations faites, citées auparavant.

Le temps d'exécution de ce bloc LSE-BFE est de **15.33 min.**

4.7 Résultats et interprétations

D'abord, nous résumons l'utilisation des ressources matérielles et les temps d'exécution pour chaque approche adoptée (par parties et complète) des algorithmes GVF et DRLSE. Ensuite, nous comparons ces résultats. Enfin, nous présentons les résultats de l'algorithme LSE-BFE et ses défis.

4.7.1 Algorithme GVF

Pour l'algorithme GVF, le tableau 4.3 montre une répartition de l'utilisation des ressources matérielles (BRAM, LUT, FF, DSP) et des temps de calcul pour chaque partie de l'implémentation par parties ainsi que pour l'implémentation complète.

	Par parties			Complète
	Partie 01	Partie 02	Partie 03	
BRAM (%)	58	94	8	99
LUT (%)	16	32	29	73
FF (%)	12	20	18	50
DSP (%)	15	7	26	36
Temps de calcul (s)	1.438	8.4127	0.1746	12.7854
Temps de calcul total (s)	10.0253			12.7854

TAB. 4.3 : Utilisation des ressources matérielles et temps de calcul pour l'algorithme GVF

4.7.2 Algorithme DRLSE

Pour l'algorithme DRLSE, le tableau 4.4 montre une répartition de l'utilisation des ressources matérielles (BRAM, LUT, FF, DSP) et des temps de calcul pour chaque partie de l'implémentation par parties ainsi que pour l'implémentation complète.

	Par parties		Complète
	Partie 01	Partie 02	
BRAM (%)	92	92	95
LUT (%)	10	41	59
FF (%)	6	23	37
DSP (%)	12	63	74
Temps de calcul (s)	3.0803	97.3604	105.8723
Temps de calcul total (s)	100.4407		105.8723

TAB. 4.4 : Utilisation des ressources matérielles et temps de calcul pour l'algorithme DRLSE

4.7.3 Algorithme LSE BFE

Pour l'algorithme LSE-BFE, le tableau 4.5 montre une répartition de l'utilisation des ressources matérielles (BRAM, LUT, FF, DSP) et le temps de calcul pour l'implémentation du bloc de cette méthode.

BRAM (%)	LUT (%)	FF (%)	DSP (%)	Temps de calcul (min)
86	60	37	64	15.33

TAB. 4.5 : Utilisation des ressources matérielles et temps pour l'algorithme LSE-BFE

4.7.4 Discussion

Les tableaux 4.3 et 4.4 récapitulatifs des algorithmes GVF et DRLSE (respectivement) montrent que, dans leurs implémentations par parties, une utilisation accrue des cellules

logiques de calcul permet de réduire le temps d'exécution global. En revanche, pour l'implémentation complète, les ressources disponibles sur FPGA sont utilisées en une seule fois, ce qui se traduit par un temps de calcul légèrement supérieur.

Nous expliquons cela en plusieurs points :

- Dans l'implémentation par parties, nous avons exploité pleinement les ressources disponibles sur FPGA, notamment les BRAM, LUT, DSP et FF, tout en optimisant les calculs. Cette approche nous a permis de réduire le temps total d'exécution par rapport à l'implémentation complète.
- Pour l'implémentation complète, nous avons réalisé tous les calculs en une seule exécution, ce qui a nécessité l'utilisation de systèmes de buffers pour la mise à jour de certaines matrices. Cela a conduit à une utilisation accrue des cellules de calcul, augmentant ainsi le temps de calcul global.
- **Utilisation du pipeline :** Dans l'implémentation partielle, nous avons exploité le pipeline autant que possible, tout en évitant les cas de dépendance mémoire. Cependant, dans l'implémentation complète, l'ajout du pipeline entraînait souvent une surutilisation des LUTs, ce qui nous a contraint à limiter l'utilisation du parallélisme pour éviter une consommation excessive de ressources matérielles.
- **Mémoire DDR vs BRAM :** Dans l'implémentation complète, nous avons souvent subdivisé un bloc en plusieurs sous-blocs pour utiliser la mémoire DDR plutôt que les BRAM du FPGA. Cependant, l'accès aux données stockées en mémoire DDR est plus lent comparé à l'accès aux données situées directement dans la partie logique du circuit FPGA, ce qui a contribué à l'augmentation du temps d'exécution.

En résumé, l'implémentation par parties a permis une optimisation des ressources et une réduction du temps d'exécution grâce à l'utilisation efficace du pipeline et des ressources internes du FPGA. En revanche, l'implémentation complète a nécessité des compromis en termes de ressources et de temps en raison des exigences supplémentaires en calcul et en gestion de mémoire.

Pour l'algorithme LSE-BFE, nous avons rencontré des contraintes significatives en raison des ressources limitées du FPGA et des calculs intensifs requis. Afin d'atténuer ces problèmes, nous avons utilisé quelques approximations et également appliqué quelques optimisations de calcul comme nous l'avons mentionné dans la section 4.6.2, notamment la réduction de trois phases de calcul à seulement deux phases, utilisant les fonctions d'appartenance $M(1)$ et $M(2)$, ce qui a contribué à dégrader le résultat de l'implémentation de cette segmentation.

Cette approche a significativement réduit la charge sur les LUTs tout en maintenant une précision acceptable. Bien que le résultat ne soit pas parfait et diffère de celui obtenu sur MATLAB, il présente un potentiel d'amélioration. Avec plus de ressources matérielles, notamment des BRAM, l'implémentation complète avec trois phases serait faisable, augmentant ainsi la précision et la performance de l'algorithme.

4.8 Conclusion

Nous avons réussi à implémenter chacun des algorithmes GVF et DRLSE, en adoptant deux approches : par parties et complète. Pour chaque implémentation, nous avons analysé l'utilisation des ressources, le temps d'exécution de l'algorithme ainsi que les résultats de la segmentation, en les comparant à ceux obtenus sur MATLAB. Cette comparaison a montré que les résultats de l'implémentation et de MATLAB sont les mêmes. Nos résultats démontrent également que, bien que l'implémentation par parties soit plus efficace en termes de temps d'exécution et d'utilisation des ressources, l'implémentation complète reste nécessaire pour certaines applications nécessitant une exécution unique de tous les calculs. L'équilibre entre ces deux approches dépend des contraintes spécifiques de chaque application et des ressources disponibles sur le circuit FPGA.

Quant à l'algorithme LSE-BFE, nous avons trouvé des solutions pour optimiser les calculs compte tenu des ressources matérielles disponibles sur le Pynq Z2. Grâce à cela, nous avons démontré la faisabilité de son implémentation et avons constaté que, avec plus de ressources matérielles, nous pourrions obtenir une meilleure précision pour la segmentation.

Conclusion Générale et Perspectives

Notre projet a permis de relever plusieurs défis techniques et d'atteindre des résultats significatifs dans le domaine de la segmentation d'images médicales. Les algorithmes de segmentation utilisés à savoir les contours actifs et les courbes de niveau, se sont avérés efficaces pour isoler les structures médicales pertinentes, ce qui est essentiel pour le diagnostic et le suivi médical. Nous avons utilisé dans notre travail des images de synthèse et des IRM.

L'étude des algorithmes GVF, DRLSE et LSE-BFE a démontré leur capacité à gérer les variations d'intensité et les inhomogénéités souvent présentes dans les images médicales.

En ce qui concerne l'algorithme GVF, son efficacité à capturer les contours lisses et continus dans des images bruitées a été confirmée. L'algorithme DRLSE, avec son approche de régularisation de distance, a montré des performances satisfaisantes en termes de stabilité et de précision dans la segmentation. Enfin, l'algorithme LSE-BFE, bien que complexe, a permis d'estimer simultanément le champ de biais et de segmenter les images, offrant ainsi une solution intégrée pour des images présentant des inhomogénéités d'intensité.

Un aspect essentiel de notre projet réside dans l'implémentation réussie de ces algorithmes sur FPGA. Cette réalisation marque une étape majeure dans la concrétisation de notre objectif principal qui est de fournir des solutions de segmentation d'images efficaces et en temps réel, adaptées aux environnements embarqués.

Malgré les succès obtenus, plusieurs axes d'amélioration restent à explorer pour optimiser encore davantage les performances des algorithmes de segmentation d'images médicales :

1. **Optimisation du temps d'exécution** : La réduction du temps d'exécution est cruciale, en particulier pour les applications cliniques où des résultats en temps réel sont souvent nécessaires. Des techniques de parallélisation et de traitement en pipeline pourraient être explorées pour accélérer le traitement des images.
2. **Réduction de la consommation de ressources** : La mise en œuvre des algorithmes sur FPGA a révélé des problèmes de surutilisation des ressources. Il est nécessaire de trouver un compromis optimal entre l'utilisation des ressources matérielles et les performances temporelles.
3. **Diminution de la complexité de calculs** : La complexité des algorithmes influence directement la performance et la scalabilité du système. Des recherches sur des algorithmes ou des variantes optimisées de ceux existants pourraient contribuer

à réduire la complexité tout en maintenant, voire en améliorant, la précision de segmentation.

4. **Implémentation complète de l'algorithme LSE-BFE en trois phases :** Bien que l'implémentation actuelle en deux phases ait montré des résultats prometteurs, nous n'avons pas pu réaliser la version complète à trois phases de l'algorithme LSE-BFE en raison des ressources matérielles limitées. La version à trois phases pourrait potentiellement offrir une segmentation plus détaillée et précise, mais elle nécessiterait une optimisation minutieuse pour gérer efficacement les ressources disponibles et le temps de calcul.
5. **Exploration d'approches matérielles personnalisées :** Explorer l'implémentation sur FPGA en utilisant VHDL ou Verilog pourrait optimiser les architectures matérielles pour maximiser la performance et minimiser la consommation de ressources, résolvant ainsi les défis actuels de surutilisation tout en répondant aux exigences de temps réel des applications cliniques.
6. **Intégration avec d'autres techniques de traitement d'images :** l'intégration des algorithmes de segmentation avec d'autres techniques de traitement d'images revêt une importance cruciale. En associant la segmentation à des méthodes de filtrage avancé, d'amélioration de contraste et de détection de caractéristiques, nous pouvons non seulement obtenir des segments de meilleure qualité, mais aussi fournir une solution plus holistique pour l'analyse d'images médicales. Cette intégration permettrait une meilleure visualisation des structures anatomiques, facilitant ainsi l'identification des pathologies et des anomalies. De plus, elle pourrait être utilisée pour assister les chirurgiens en leur fournissant des informations précises sur la localisation et la nature des lésions, ce qui contribuerait à des interventions plus précises et à des résultats cliniques améliorés.
7. **Validation clinique et applications pratiques :** Une étape cruciale serait la validation des algorithmes sur des ensembles de données cliniques plus larges et variés. Cette validation permettrait de confirmer leur robustesse et leur applicabilité dans des scénarios réels, ouvrant ainsi la voie à une adoption plus large dans les environnements médicaux.

En poursuivant ces axes de recherche et développement, nous pourrions améliorer les performances des modèles déformables actuels dans le domaine de la segmentation d'images médicales. Ces avancées contribueront à fournir des outils plus puissants et efficaces pour les professionnels de la santé, améliorant ainsi la qualité des soins et des diagnostics.

Bibliographie

1. COCQUEREZ, Jean-Pierre ; PHILIPP, Sylvie. *Analyse d'images : Filtrage et segmentation*. MASSON, 1995. Enseignement de la physique. ISBN 2-225-84923-4. fhal00706168f.
2. GONZALEZ, Rafael C. ; WOODS, Richard E. *Digital Image Processing Fourth Edition*. In : 2018. ISBN 10: 1-292-22304-9.
3. KHELIFI, Riad. *Segmentation des images coronarographiques par les Contours Actifs*". 2008. Thèse de magistère. Ecole Nationale Polytechnique.
4. BOUADJENEK, Nesrine. *Cours de Traitement d'images*. [s. d.].
5. SANIUEL, Fosso Vvamba. *Morphologie Mathématique Appliquée au Traitement de l'Image*. Sherbrooke, Québec, Canada, 2000. Mémoire de maîtrise. Université de Sherbrooke. Mémoire présenté au Département de mathématiques et d'informatique en vue de l'obtention du grade de maître ès sciences (M.Sc.)
6. ZHANG, Y.J. A survey on evaluation methods for image segmentation. *Computer Vision and Pattern Recognition*. 1996, t. 29, n° 8, p. 1335-1346.
7. KINDERMANN, R. ; SNELL, J.L. *Markov random fields and their applications*. T. 1. Amer. Math. Soc., 1980.
8. ROUSELLE, Jean Jacques. *"Les contours actifs une méthode de segmentation. Application à l'imagerie médicale"*. 2003. Thèse de doct.
9. BRUN, Luc. *Segmentation d'images à base Topologique*. 1996. Université Sciences et Technologies - Bordeaux I. ffnnt : ff. fftel-00329494f.
10. CHANTERMARGUE, F. ; POPOVIC, M. ; CANALS, R. ; BONTON, P. Parallelization of the merging step of the region segmentation method. In : *7th Scandinavian Conference on Image Analysis*. Aalborg, Denmark, 1991, p. 993-940.
11. CLERMONT, P. ; CARTIER, S. Implementation and evaluation of region growing algorithms on CM2. In : *Proceedings of the International Colloquium on Parallel Image Processing*. Paris, 1991, p. 91-109.
12. TUCERYAN, M. ; JAIN, A.K. Texture segmentation using Voronoi polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1990, t. 13, p. 211-216.
13. HOROWITZ, S.L. ; PAVLIDIS, T. Picture segmentation by directed split-and-merge procedure. In : *Second International Joint Conference on Pattern Recognition*. 1974.
14. JOLION, J.M. ; ROSENFELD, A. *Pyramid framework for early vision*. Kluwer, 1992.

15. KHALIFA NAWRES Malek Amel, Hamrouni Kamel. "Segmentation d'images par contours actifs : Application à la détection du ventricule gauche dans les images de scintigraphie cardiaque". *IEEE Access*. 2005, p. 11.
16. KASS.M WITKIN.A, TERZOPOULOS. Snakes : Active contour models. *International Journal of Computer Vision*. 1988, p. 321-331.
17. MAHMOUD, LOUDNI. *Segmentation d'images par les méthodes des contours actifs*. 2019. Université SAAD DAHLAB Blida.
18. HOCINE, BENABIDI. *Comparaison des algorithmes des Contours Actifs pour des images vidéos*. 2006. Mémoire de projet de fin d'études. Institut National de formation en Informatique (I.N.I).
19. COHEN, Laurent D. On Active Contour Models and Balloons. *Computer Vision, Graphics, and Image Processing : Image Understanding*. 1991, t. 53, n° 2, p. 211-218. Appeared first as RR INRIA 1075, August 1989.
20. OLIVIER, Julien. "Méthodes d'accélération et approches supervisées pour les contours actifs. Applications à la segmentation d'images 2D, 3D et texturées". 2009. Thèse de doct.
21. BERGER, M.-O. *Les contours actifs : modélisation, comportement et convergence*. 1999. Thèse de Doctorat. Institut National Polytechnique de Lorraine.
22. AMINI, A. A. ; WEYMOUTH, T. E. ; JAIN, R. C. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1990, t. 12, n° 9, p. 855-867. Disp. à l'adr. DOI : [10.1109/34.57681](https://doi.org/10.1109/34.57681).
23. NADIA, REZZOUG. *Segmentation d'images Médicales par contours actifs*. 2020. Mémoire de projet de fin d'étude. UNIVERSITE SAAD DAHLEB BLIDA.
24. WILLIAMS, D. J. ; SHAH, M. A Fast algorithm for active contours and curvature estimation. *CVGIP : Image Understanding*. 1992, t. 55, n° 1, p. 14-26. Disp. à l'adr. DOI : [10.1016/1049-9660\(92\)90003-L](https://doi.org/10.1016/1049-9660(92)90003-L).
25. LAM, K.-M. ; YAN, H. Fast greedy algorithm for active contours. *Electronics Letters*. 1994, t. 30, n° 1, p. 21-23. Disp. à l'adr. DOI : [10.1049/e1:19940040](https://doi.org/10.1049/e1:19940040).
26. XU, Chenyang ; PRINCE, J. L. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*. 1998, t. 7, n° 3, p. 359-369. Disp. à l'adr. DOI : [10.1109/83.661186](https://doi.org/10.1109/83.661186).
27. OSHER, Stanley ; SETHIAN, James A. "Fronts propagating with curvature-dependant speed : Algorithms based on Hamilton-Jacobi formulations". *Computational Physics*. 1988, t. 97, p. 12-49.
28. GOMES, J. ; FAUGERAS, O. "Reconciling Distance Functions and Level Sets". In : *Proceedings of the Second International Conference on Scale-Space Theories in Computer Vision*. 1999.
29. HAO, J. ; SHEN, Y. ; WANG, Y. "Segmentation for MRA Image : An Improved Level Set Approach". 2007, t. 56, n° 4, p. 6. Disp. à l'adr. DOI : [10.1109/tim.2007.899839](https://doi.org/10.1109/tim.2007.899839).

30. MALLADI, R. ; SETHIAN, J.A. ; VEMURI, B.C. "Shape Modeling with Front Propagation : A Level Set Approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1995, t. 17, n° 2, p. 158-175.
31. OSHER, Stanley ; RONALD, Fedkiw. "Level Set Methods and Dynamic Implicit Surfaces". *Applied Mathematical Sciences Volumes*. 1988, t. 153, p. 12-49.
32. SETHIAN, J. A. "Fast Marching Methods and Level Set Methods for Propagating Interfaces". In : *von Karman Institute Lecture Series, Computational Fluid Mechanics*. 1998.
33. SETHIAN, J. A. "*Level Set Methods and Fast Marching Methods*". Cambridge University Press, 1999.
34. LI, C. ; XU, C. ; GUI, C. ; FOX, M.D. Level set formulation without re-initialization : a new variational formulation. In : *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. San Diego, 2005, p. 430-436.
35. CASELLES, V. ; KIMMEL, R. ; SAPIRO, G. "Geodesic active contours". *International Journal of Computer Vision*. 1997, t. 22.
36. YATING CHEN Gaoxiang Chen, Yu Wang ; DEY, Nilanjan. "A Distance Regularized Level-set Evolution Model Based MRI Dataset Segmentation of Brain's Caudate Nucleus". *IEEE Access*. 2019, n° 7, p. 12. Disp. à l'adr. DOI : [10.1109/access.2019.2937964](https://doi.org/10.1109/access.2019.2937964).
37. LI, C. ; XU, C. "Distance Regularized Level Set Evolution and Its Application to Image Segmentation". *International Journal of Computer Vision*. 2010, p. 3243-3254. Disp. à l'adr. DOI : [10.1109/TIP.2010.2069690](https://doi.org/10.1109/TIP.2010.2069690).
38. KAIHUA ZHANG Huihui Song, Lei Zhang. Active contours driven by local image fitting energy. *Elsevier*. 2009, t. 43, p. 1199-1206. Disp. à l'adr. DOI : [10.1016/j.patcog.2009.10.010](https://doi.org/10.1016/j.patcog.2009.10.010).
39. LI, Chunming. "A Level Set Method for Image Segmentation in the Presence of Intensity Inhomogeneities With Application to MRI". 2011, n° 7, p. 10.
41. LARIE, S. M. ; ABUKMEIL, S. S. Brain abnormality in schizophrenia : a systematic and quantitative review of volumetric magnetic resonance imaging studies. *Journal of Psychiatry*. 1998, t. 172, p. 110-120.
42. ZIJDENBOS, A. P. ; DAWANT, B. M. Brain segmentation and white matter lesion detection in MR images. *Critical Reviews in Biomedical Engineering*. 1994, t. 22, p. 401-465.
43. DAVATZIKOS, Christos A. ; PRINCE, Jerry L. An active contour model for mapping the cortex. *IEEE Transactions on Medical Imaging*. 1995, t. 14, p. 65-80.
44. KHOO, V. S. ; DEARNALEY, D. P. ; FINNIGAN, D. J. ; PADHANI, A. ; TANNER, S. F. ; LEACH, M. O. Magnetic resonance imaging (MRI) : considerations and applications in radiotherapy treatment planning. *Radiotherapy and Oncology*. 1997, t. 42, p. 1-15.

45. GRIMSON, W. E. L.; ETTINGER, G. J.; KAPUR, T.; LEVENTON, M. E.; WELLS, W. M.; AL., et. Utilizing segmented MRI data in image-guided surgery. *International Journal of Pattern Recognition and Artificial Intelligence*. 1997, t. 11, p. 1367-1397.
46. XU, Chenyang; PHAM, Dzung L.; PRINCE, Jerry L. Image Segmentation Using Deformable Models. In : *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2000*. Springer, 2000, p. 210-218.
50. DARGENT, Régis. *FILTRAGE ADAPTATIF ET DIFFUSION ANISOTROPE POUR L'AIDE A L'INTERPRETATION DES DONNEES SISMiques*. 2006. Thèse de doctorat. Université Sciences et Technologies - Bordeaux I. Traitement du signal et de l'image [eess.SP].
51. HE, L.; PENG, Z. G.; EVERDING, B. et al. A Comparative Study of Deformable Contour Methods on Medical Image Segmentation. *Image and Vision Computing*. 2008, t. 26, n° 2, p. 141-163. Disp. à l'adr. DOI : [10.1016/j.imavis.2007.07.010](https://doi.org/10.1016/j.imavis.2007.07.010).
52. XU, Chenyang; PRINCE, Jerry L. Active Contours, Deformable Models, and Gradient Vector Flow. *Image and Vision Computing*. 2002, t. 20, n° 7, p. 537-538.
53. XU, C.; PRINCE, J. L. Gradient Vector Flow : A New External Force for Snakes. In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos : IEEE Computer Society Press, 1997, p. 66-71.
56. REKROUK, Maroua; NBRI, Ryhame; ALLAM, Fatima Zohra; HAMAMI, Latifa. Study and Implementation on FPGA of Human Recognition System via Iris Based on Deep Learning. *IEEE*. 2023, p. 5. Ecole Nationale Polytechnique, Algiers, Algeria.
61. PEYRÉ, Gabriel. Le traitement numérique des images. 2011. Chercheur au CNRS, Paris, Le 28 novembre 2011.
66. XILINX. *Vitis High-Level Synthesis User Guide* [<https://shorturl.at/J3uGM>]. v2021.20^e éd. 2021. UG1399 (v2021.2).
67. XILINX. *Vivado Design Suite User Guide* [<https://shorturl.at/J3uGM>]. v2021.10^e éd. 2021. UG973 (v2021.1).
68. PYNQ. *PYNQ-Z2 Reference Manual* [<https://shorturl.at/MfMtm>]. v1.00^e éd. 2018.

Webographie

40. *Imagerie médicale* [<https://shorturl.at/vc1VG>]. [s. d.].
47. *Imagerie par résonance magnétique nucléaire (IRM)* [<https://shorturl.at/lc13i>]. [s. d.].
48. *BRATS MICCAI brain tumor dataset* [<https://shorturl.at/P8sIi>]. [s. d.].
49. *Qu'est-ce que le filtrage anisotrope ?* [<https://shorturl.at/jUWqC>]. [s. d.].
54. *Code MATLAB de l'algorithme GVF* [https://github.com/MaxMB/GVF_Image_Segmentation]. [s. d.].
55. *Code MATLAB de l'algorithme DRLSE* [<https://shorturl.at/JB5sh>]. [s. d.].
57. *Protocole AXI* [<https://shorturl.at/tWXDF>]. [s. d.].
58. *PYNQ-Z2 IMAGE* [www.pynq.io]. [s. d.].
59. *Jupyter Notebook PYNQ-Z2* [192.168.2.99:9090]. [s. d.].
60. *What Is Image Processing ?* [<https://shorturl.at/6RdUu>]. [s. d.].
62. *Traitement d'histogramme* [<https://shorturl.at/U9XxJ>]. [s. d.].
63. *What Is Matlab ?* [<https://shorturl.at/RkKBD>]. [s. d.].
64. *Présentation de Matlab* [<https://shorturl.at/MU1rf>]. [s. d.].
65. *Visual Studio Code* [<https://shorturl.at/loS6A>]. [s. d.].

Annexe 1 : Généralités sur le traitement d'images

.1 Définition générale

Le traitement d'images est le processus de transformation d'une image en forme numérique et d'exécution de certaines opérations pour obtenir des informations utiles. Le système de traitement de l'image traite généralement toutes les images comme des signaux 2D en appliquant certaines méthodes prédéterminées de traitement du signal [60].

.2 Notions de base

Pour aborder efficacement le traitement d'images, commençons par définir quelques notions de base.

.2.1 Image

Une image est une représentation visuelle en deux dimensions d'une scène, d'un objet ou d'une information. Dans le contexte du traitement d'images, une image est représentée sous forme d'une matrice bidimensionnelle de pixels. En mathématiques, une image peut être formalisée en tant qu'application d'un sous-ensemble $M \times N$ de $R \times R$ vers l'ensemble des réels R .

Cette application est représentée par une fonction [18] :

$$\begin{aligned} f : M \times N &\longrightarrow R \\ (x, y) &\longrightarrow f(x, y) \end{aligned} \tag{1}$$

Dans cette représentation mathématique :

- f : Fonction d'intensité lumineuse définie sur un domaine borné.
- x, y : Coordonnées linéaires d'un point donné de l'image.

Cependant, cette description mathématique seule ne suffit pas dans le contexte informatique. Pour rendre l'image exploitable dans le domaine informatique, il est nécessaire de passer par le processus de numérisation. La numérisation transforme l'image continue définie par la fonction $f(x, y)$ en une représentation discrète où les valeurs de luminosité sont échantillonnées à des positions discrètes sur une grille. Cela aboutit à une représentation matricielle de l'image, généralement appelée matrice de pixels.

Pour expliquer ce processus de numérisation, on peut citer ces trois principales étapes [18] :

- **Balayage** : parcourir l'image ligne par ligne pour renvoyer une fonction continue qui est le spectre de luminosité par exemple.
- **Échantillonnage** : prendre des valeurs discrètes pour chaque intervalle, ce dernier représenté par la taille de l'ouverture du balayage. Il en résulte donc l'image originale en petits carrés, chacun caractérisé par une mesure d'intensité.

- **Quantification** : codifier les valeurs des carrés en entier puis en binaire.

.2.2 Pixel

Une image est composée d'un ensemble de points appelés pixels, un terme dérivé de la contraction de l'expression britannique "Picture Element". Chaque pixel représente le plus petit élément constitutif d'une image numérique auquel une couleur (ou un niveau de gris) et une intensité peuvent être individuellement associées. Ces pixels forment une matrice à deux dimensions, constituant l'image finale. Chaque pixel dans une image est défini par des coordonnées, et généralement, le pixel situé en haut à gauche de l'image est désigné par les coordonnées $[0, 0]$. Étant donné que l'écran effectue un balayage de gauche à droite et de haut en bas, cette convention facilite la référence aux emplacements des pixels dans l'image.

La figure 1 montre une grille de pixels :

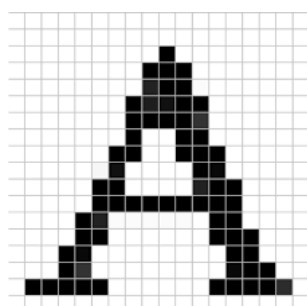


FIG. 1 : Grille de pixels [4]

Une image peut être représentée en mémoire selon différents schémas de codage, en fonction de ses caractéristiques et des informations qu'elle doit contenir. Voici quelques exemples de représentations en mémoire en fonction du type d'image [18] :

- Images monochromes (noir et blanc) :
 - Un bit par pixel : 0 pour le noir, 1 pour le blanc.
- Images à niveaux de gris :
 - Un octet par pixel : permettant 256 niveaux de gris, où 0 représente le noir et 255 le blanc.
- Images en couleur (format RVB - Rouge, Vert, Bleu) :
Trois octets par pixel :
 - 1 octet pour la composante rouge (256 nuances de rouge).
 - 1 octet pour la composante verte (256 nuances de vert).
 - 1 octet pour la composante bleue (256 nuances de bleu).

Chaque composante RVB peut prendre des valeurs de 0 à 255, permettant ainsi la création d'une vaste gamme de couleurs en combinant différentes intensités de rouge, vert et bleu.

.2.3 Définition et résolution d'une image

La définition d'une image se réfère au nombre total de pixels qui la composent, représenté par la multiplication du nombre de colonnes par le nombre de lignes de l'image. Par exemple, une image de 1920 pixels en largeur et 1080 pixels en hauteur aurait une définition de 1920x1080 pixels.

La résolution mesure le nombre de points par unité de surface et est généralement exprimée en points par pouce (PPP ou DPI en anglais, pour Dots Per Inch). Un pouce équivaut à 2,54 centimètres. La résolution permet de déterminer le rapport entre le nombre de pixels d'une image et sa taille réelle sur un support physique. Par exemple, une résolution de 300 DPI signifie qu'il y a 300 colonnes et 300 lignes de pixels sur chaque pouce carré, totalisant ainsi 90 000 pixels par pouce carré. Cela offre une indication de la finesse des détails visibles dans l'image lorsqu'elle est imprimée ou affichée sur un support physique [18].

.2.4 Image à niveaux de gris

Si l'image décomposée présente des valeurs de pixels différentes de noir et blanc, on dit que l'image est en niveaux de gris. Ainsi, dans une image en niveaux de gris, nous attribuons à un pixel sa valeur de luminosité. La valeur est comprise entre 0 et 255. Il faut cependant noter que le nombre de niveaux de gris dépend du nombre de bits alloués à son codage. En effet, plus il y a de bits, plus les niveaux de gris peuvent être représentés [18].

.2.5 Image en couleurs

Une image en couleurs est en réalité composée de trois images représentant trois canaux ; le rouge, le vert et le bleu. Cette représentation en rouge, vert et bleu imite le fonctionnement du système visuel humain. Chaque pixel de l'image couleur contient ainsi trois nombres (r,v,b), chacun étant un nombre entier entre 0 et 255 [61].

.2.6 Région

Le terme "région" désigne un groupe de pixels partageant des caractéristiques similaires et satisfaisant un critère d'homogénéité. Typiquement, une région correspond à un objet présent dans l'image, et elle est définie par la cohérence des propriétés visuelles de ses pixels [18].

.2.7 Contour

Par définition, un contour est caractérisé par une variation brusque du niveau de gris dans une image, avec une amplitude a et une pente p . On peut conceptualiser un contour de différentes manières : il peut être assimilé à une marche d'escalier lorsque le changement est abrupt, à une rampe si le contour est plus progressif et flou, ou à

un toit lorsqu'il représente une ligne distincte sur un arrière-plan uniforme. Ces analogies décrivent visuellement la nature du changement dans les niveaux de gris le long du contour, permettant de différencier les contours nets des contours plus doux dans une image [18].

Les types de contours sont illustrés dans la figure 2 :

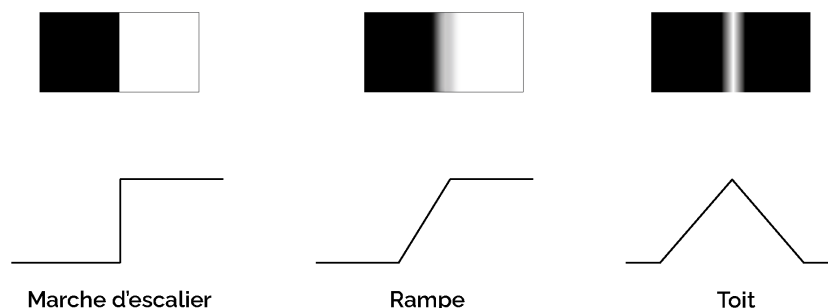


FIG. 2 : Types de contours

Un contour peut être défini comme un ensemble de pixels formant une frontière entre des régions d'une image. L'épaisseur d'un contour peut être constituée d'un ou plusieurs pixels, bien que dans un cadre idéal, elle serait d'un seul pixel. En d'autres termes, les contours servent à marquer les variations significatives dans les niveaux de gris, créant ainsi des frontières visuelles entre différentes zones de la scène capturée.

.2.8 Histogramme

Dans le contexte d'une image en niveaux de gris, l'histogramme est défini comme une fonction $H(i)$ représentant la distribution des intensités de gris. Cette fonction assigne à chaque niveau de gris i le nombre de pixels de l'image qui possèdent cette intensité.

Pour une image couleur en mode RVB, l'histogramme est étendu à trois fonctions distinctes, correspondant respectivement aux canaux de couleur rouge, vert et bleu. Ces histogrammes RVB fournissent une représentation statistique de la répartition des niveaux d'intensité pour chaque composante chromatique.

L'histogramme, en tant qu'outil d'analyse d'images, offre une synthèse concise mais souvent suffisante du contenu visuel, facilitant ainsi l'évaluation globale des caractéristiques chromatiques et de luminosité de l'image [62].

.2.9 Bruit

Le bruit, également appelé parasite, dans une image se manifeste par des variations soudaines d'intensité d'un pixel par rapport à ses voisins. Généralement, le bruit est considéré comme une anomalie non désirée, souvent causée par divers facteurs lors de l'acquisition ou du traitement d'une image. Les sources potentielles de bruit dans une image incluent [18] :

- **Événements inattendus lors de l'acquisition** : tels que le mouvement de la

scène, des changements brusques dans les conditions d'éclairage et d'autres perturbations imprévues pouvant introduire du bruit.

- **Qualité des capteurs** : des capteurs de mauvaise qualité ou mal utilisés peuvent générer du bruit. Des imperfections dans les capteurs d'image peuvent entraîner des variations non désirées dans les données capturées.
- **Échantillonnage** : lors du passage de la forme analogique à la forme numérique de l'image, des erreurs peuvent être introduites, provoquant des fluctuations d'intensité indésirables.
- **Nature de la scène** : certaines scènes, en raison de la présence de poussière, de perturbations atmosphériques ou d'autres éléments, peuvent naturellement générer du bruit.

.3 Types de traitement d'images

Il existe cinq types principaux de traitement d'images [60] :

- **Visualisation** : trouver les objets qui ne sont pas visibles dans l'image.
- **Reconnaissance** : distinguer ou détecter les objets de l'image.
- **Aperçu et restauration** : créer une image améliorée à partir de l'image originale.
- **Reconnaissance des motifs** : mesure les différents motifs autour des objets de l'image.
- **Récupération** : parcourir et rechercher des images à partir d'une grande base de données d'images numériques qui sont similaires à l'image originale.

.4 Principaux outils mathématiques

Voici quelques outils mathématiques clés utilisés dans le traitement d'images [1] :

- **Transformation d'intensité et filtrage spatial** : ajustement des niveaux d'intensité des pixels pour améliorer le contraste (transformation d'intensité) et application de filtres basés sur la disposition spatiale des pixels (filtrage spatial) pour des améliorations visuelles.
- **Filtrage fréquentiel** : modification des composantes fréquentielles d'une image pour accentuer ou atténuer des fréquences spécifiques, souvent réalisée à l'aide de techniques basées sur la transformation de Fourier.
- **Restauration d'images et reconstruction** : processus visant à récupérer des images altérées par le bruit ou la dégradation (restauration) et à créer une image à partir d'une représentation incomplète ou dégradée (reconstruction).

- **Traitement d'images en couleur** : manipulation d'images contenant des informations couleur, incluant des opérations comme la correction des couleurs et la segmentation basée sur la couleur.
- **Transformée en ondelettes** : technique de transformation d'image permettant une analyse multirésolution en représentant une image en termes de composantes d'ondelettes.
- **Compression d'images** : réduction de la taille d'une image tout en préservant une qualité acceptable, utilisant des méthodes telles que la compression avec ou sans perte.
- **Tatouage numérique d'images** : incorporation discrète d'informations spécifiques dans une image, souvent pour des objectifs d'identification ou de protection contre la contrefaçon.
- **Traitement morphologique d'images** : opérations basées sur la forme et la structure des objets dans une image, incluant l'érosion, la dilatation, l'ouverture et la fermeture.
- **Segmentation d'images** : division d'une image en régions ou objets significatifs pour faciliter l'analyse, basée sur des seuils, des contours ou d'autres caractéristiques.
- **Extraction des caractéristiques** : identification et isolation d'aspects spécifiques d'une image, tels que les contours ou les textures, pertinents pour une tâche donnée.
- **Classification de motifs d'images** : attribution de catégories prédéfinies à des images en fonction de leurs caractéristiques visuelles, souvent réalisée à l'aide d'algorithmes d'apprentissage.

Annexe 2 : Logiciels et matériels

.1 Logiciel MATLAB

Dans cette section, nous présentons le logiciel MATrix LABoratory, en fournissant une définition, en expliquant ses principales fonctionnalités et en présentant son interface.

.1.1 Définition du logiciel

MATLAB, est un environnement de programmation et de calcul numérique largement utilisé dans les domaines scientifiques et techniques. Il offre un ensemble d'outils puissants pour effectuer des calculs, analyser des données, créer des visualisations et développer des algorithmes [63].

.1.2 Principales fonctionnalités du logiciel

Les fonctionnalités principales de MATLAB sont essentielles pour un large éventail d'applications scientifiques et techniques. Voici quelques-unes des fonctionnalités qui en font un outil indispensable [64] :

- Langage de haut niveau pour le calcul scientifique et technique ;
- Environnement bureau pensé pour l'exploration itérative, la conception et la résolution de problèmes ;
- Graphiques destinés à la visualisation de données et outils conçus pour créer des tracés personnalisés ;
- Applications dédiées à l'ajustement de courbes, la classification de données, l'analyse de signaux et bien d'autres tâches spécialisées ;
- Boîtes à outils additionnelles conçues pour répondre à de nombreux besoins spécifiques aux ingénieurs et aux scientifiques ;
- Outils permettant la création d'applications avec interface utilisateur personnalisée ;
- Interfaces vers C/C++, Java, .NET, Python, SQL, Hadoop et Microsoft Excel ;
- Options de déploiement libre de droits permettant de partager des programmes MATLAB avec les utilisateurs finaux.

.1.3 Aperçu de l'Interface du logiciel

Pour fournir un aperçu visuel de l'environnement MATLAB utilisé dans notre étude, voici une figure (figure 1) présentant l'interface utilisateur, notamment la fenêtre des variables :

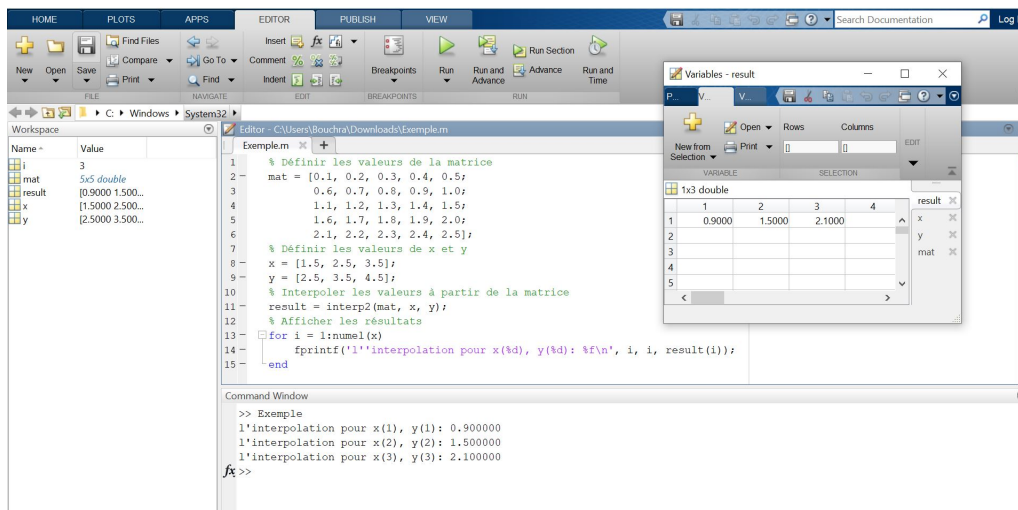


FIG. 3 : Interface MATLAB

.2 Visual Studio Code

Visual Studio Code (VS Code) est un éditeur de code puissant et léger, offrant une multitude d'extensions et d'outils de développement intégrés, facilitant ainsi l'écriture et le débogage du code en C, C++, Python, JavaScript, et bien d'autres langages [65].



.3 Vitis High-Level Synthesis

Vitis High-Level Synthesis (Vitis HLS) est un ensemble d'outils de Xilinx permettant de convertir des algorithmes décrits en langages de haut niveau comme C et C++ en code VHDL (Very High Speed Integrated Circuit Hardware Description Language) ou Verilog pour une exécution sur des FPGA. Cela permet d'accélérer la conception et l'optimisation des algorithmes pour des systèmes embarqués, offrant ainsi des performances élevées et une faible consommation d'énergie [66].



.4 Vivado

Vivado Design Suite est un logiciel de Xilinx conçu pour la conception de circuits FPGA et SoC. Il intègre des outils avancés pour la création, la simulation, la synthèse, et la mise en œuvre des conceptions matérielles, ainsi que pour l'analyse temporelle et le débogage en temps réel, optimisant ainsi la productivité des concepteurs [67].



.5 FPGA PYNQ-Z2

La carte FPGA PYNQ-Z2 est une plateforme de développement basée sur le SoC Xilinx Zynq-7000. Cette carte intègre à la fois un système de traitement (Processing System, PS) et une logique programmable (Programmable Logic, PL), offrant une solution complète pour le développement matériel et logiciel embarqué.

La figure suivante représente la carte FPGA utilisée :

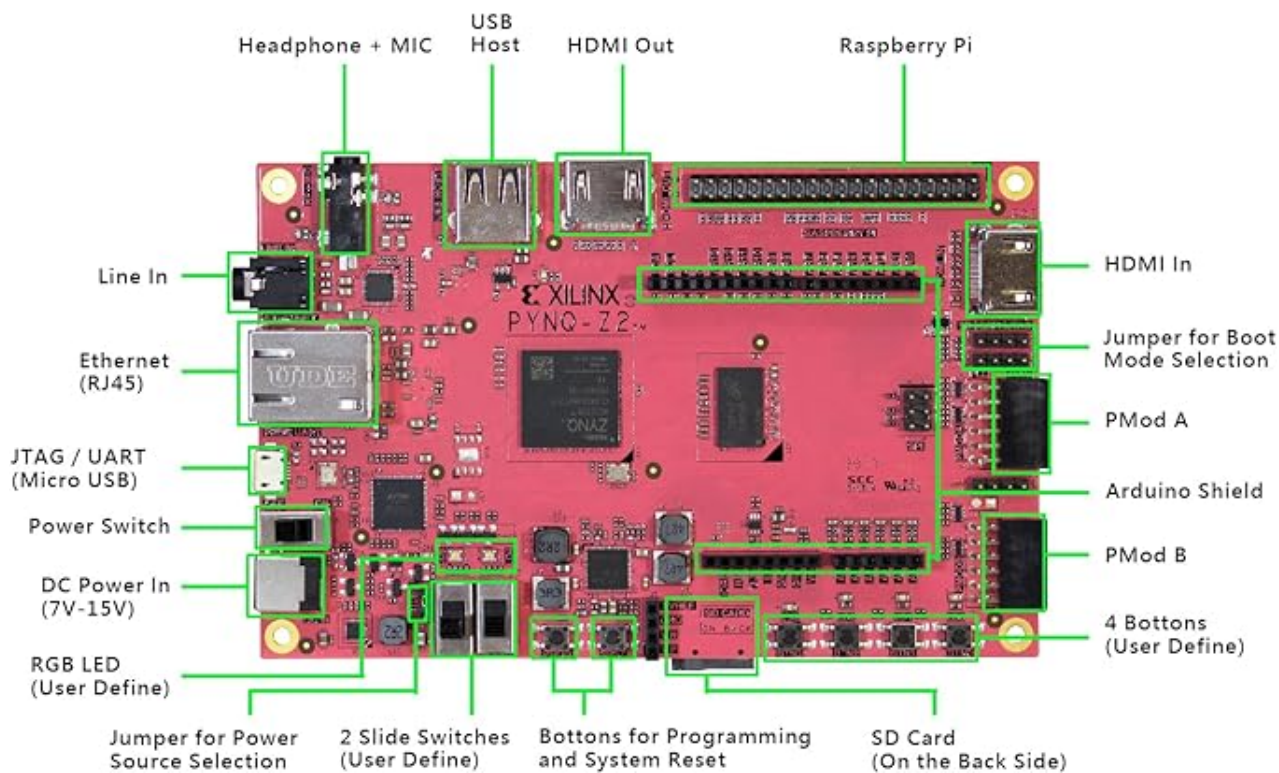


FIG. 4 : FPGA PYNQ-Z2

Considérons certaines des spécifications pertinentes de la carte Pynq-Z2 [68] :

- **Système de Traitement (Processing System, PS) :**
 - SoC Xilinx Zynq-7000 (XC7Z020-1CLG400C),
 - Processeur ARM Cortex-A9 double cœur,
 - 512 Mo de RAM DDR3,
 - Mémoire Flash Quad-SPI de 16 Mo,
 - Emplacement pour carte MicroSD pour un stockage supplémentaire.
- **Logique Programmable (Programmable Logic, PL) :**
 - FPGA de la série 7 de Xilinx (XC7Z020-1CLG400C),
 - 13 300 tranches logiques, chacune avec quatre LUTs à 6 entrées et 8 bascules,
 - 220 tranches DSP,

- 630 Ko de bloc RAM rapide.
- **Alimentation :**
 - Alimentation par USB ou alimentation externe (7V à 15V DC).
- **Support Logiciel :**
 - Framework PYNQ pour le développement basé sur Python,
 - Vivado Design Suite pour la synthèse et l'implémentation FPGA,
 - Vitis pour le développement de logiciels embarqués.

Annexe 3 : Schémas des designs par blocs des implémentations complètes des algorithmes GVF, DRLSE et LSE-BFE

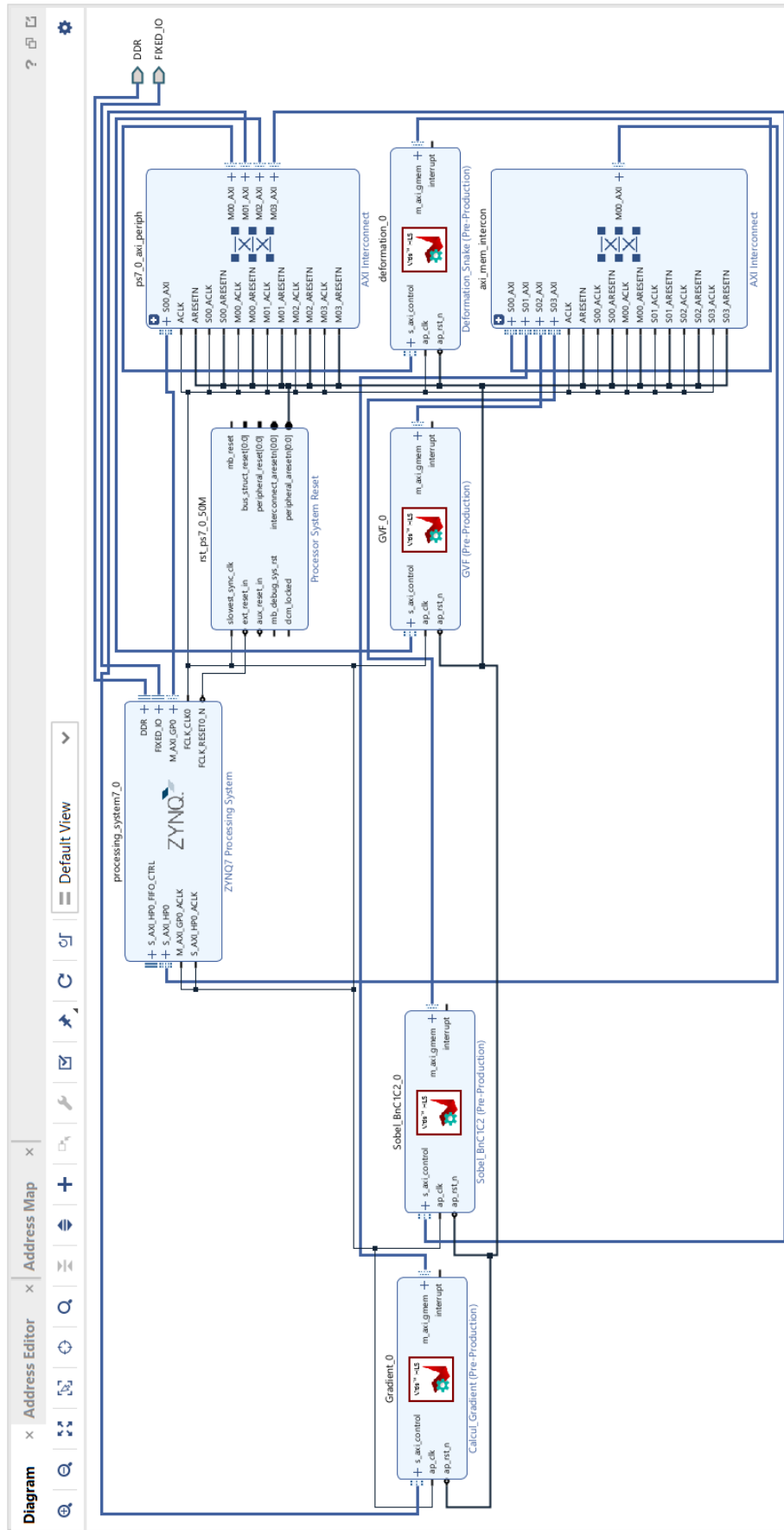


FIG. 5 : Design par blocs pour l'implémentation complète de la méthode des contours actifs par l'algorithme GVF

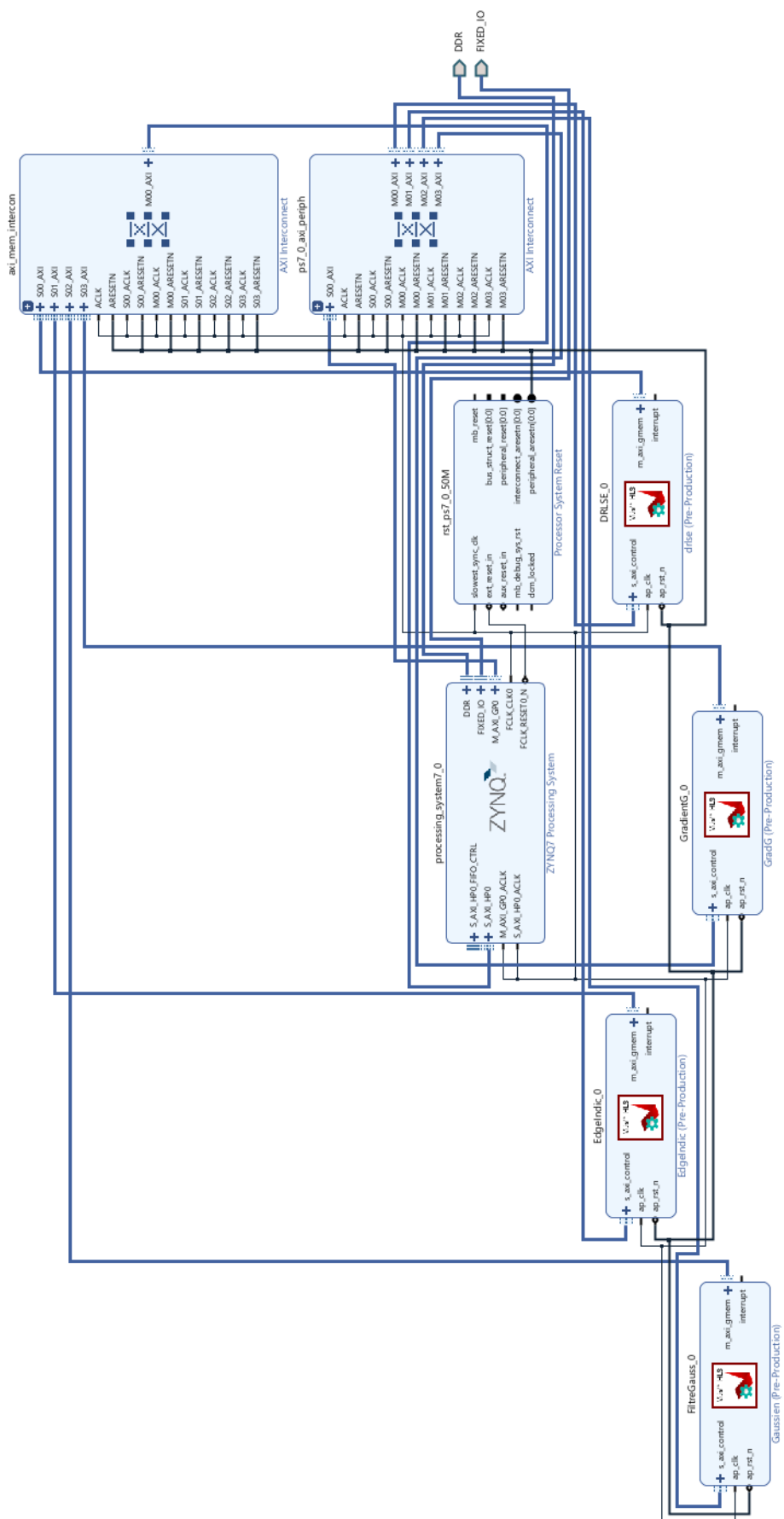


FIG. 6 : Design par blocs pour l'implémentation complète de la méthode des courbes de niveau par l'algorithme DRLSE

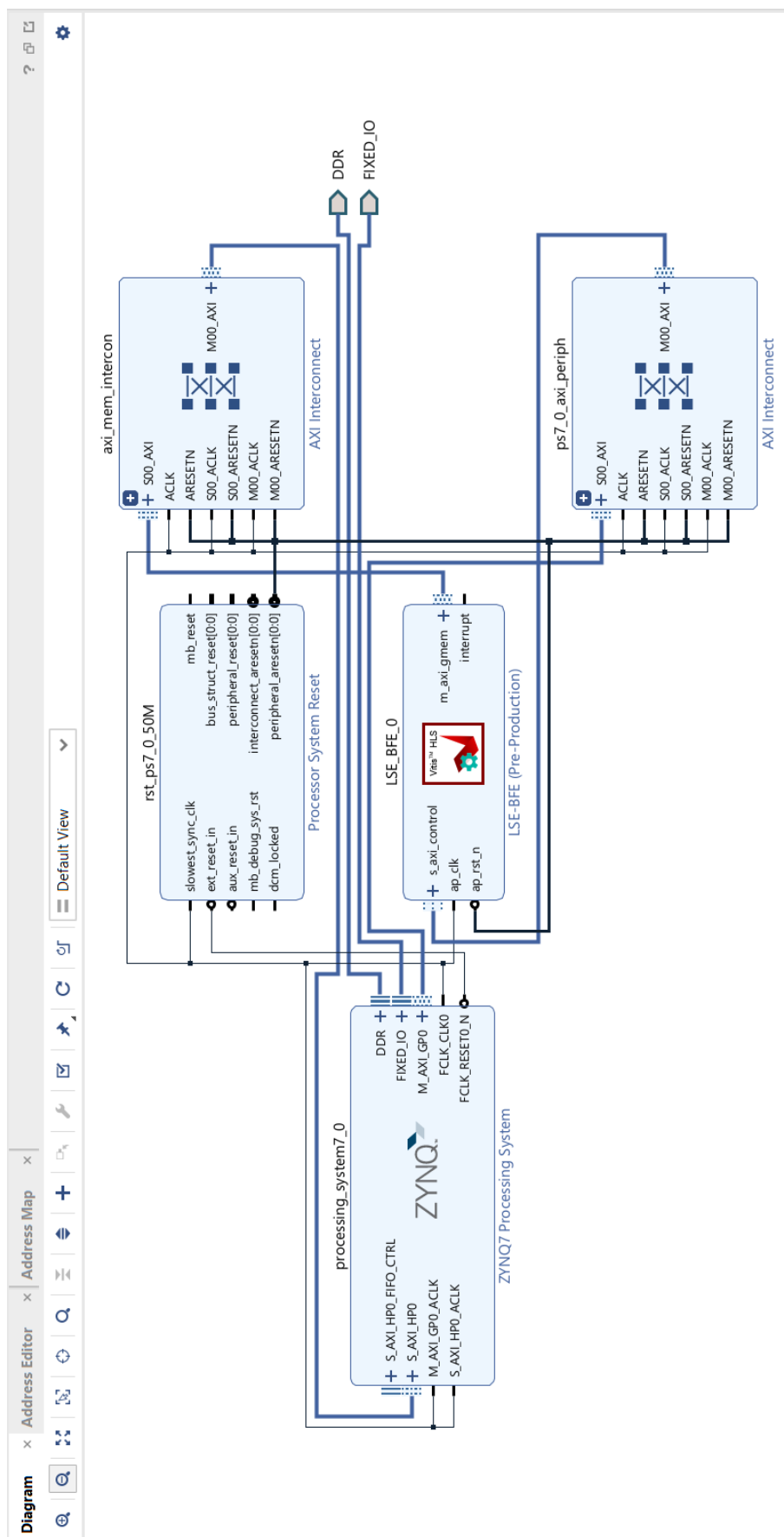


FIG. 7 : Design par blocs pour l'implémentation complète de la méthode des courbes de niveau par l'algorithme LSE-BFE