Université de Bretagne Occidentale

Évaluation des algorithmes de traitement d'images et de vidéos sous-marines

Auteurs :
Mkrtchyan Artur

Enseignant :
Franc Singhoff
Hai Nam Tran
Vincent Rodin
Barbara Dzaja

Mail : Artur.Mkrtchyan@etudiant.univ-brest.fr

9 septembre 2021

Table des matières

1 Remerciements

Je tiens à remercier tous les professeurs qui ont contribué à la réussite de ce stage.

J'aimerais en premier lieu remercier M. Singhoff Franc pour m'accepter en tant que stagiaire mais aussi d'être disponible tous les moments pour me guider dans mon parcours professionnel.

Ensuite mes grands remerciements à M.Tran Hai Nam et M.Rodin Vincent pour leur accompagnement attentif et encouragement pendant les moments durs.

Finalement j'aimerais remercier Mme.Dzaja Barbara et a l'Université de Split pour organiser ce stage enrichissant et de nous partager cette esprit croate tellement positif.

n raison du fait que c'est un rapport final avant la fin de grande épopée de 5 ans d'obtention de diplôme de master en informatique, j'aimerais remercier tous les professeurs qui m'en accompagnées pendent tout ce chemin et qui l'ont rendu facile, intéressent et lumineux.

2 Introduction

2.1 Contexte et problématique

Le domaine aquatique est encore aujourd'hui insuffisamment étudié. Plus de 80% des zones très profondes restent inexplorées, et les conditions environnementales compliquent la recherche sous-marine [16]. A une profondeur de seulement 150 mètres, 99% de la lumière solaire est absorbée [9]. Du point de vue du traitement des images sous-marines, le concept de distance est différent en comparant avec le modèle atmosphérique. Outre les réflexions lumineuses et l'absorption de certaines parties du spectre des couleurs rend les images inutilisables pour certains usages précis. De plus, la mer agitée rend généralement les images encore plus illisibles.

Aujourd'hui, de nombreux programmes permettent d'améliorer la qualité des images sous-marines. La plupart des recherches dans les profondeurs des océans sont menées avec des véhicules sous-marins. Ces véhicules utilisent des caméras pour le positionnement automatique. Cependant, pour faire le meilleur usage des images, elles doivent être traitées immédiatement. Il y a donc une contrainte de temps réel, car la vitesse de transmission des données sous-marines est limitée, et un traitement à bord est nécessaire. Pour permettre le traitement en temps réel, tous les algorithmes intégrés dans le véhicule sous-marin doivent être applicables dans ce contexte. De plus, des résultats de qualité sont essentiels pour un traitement d'images efficace, comme la reconnaissance ou la détection d'objets.

FIGURE 1 : Drone sous-marin

2.2 Objectifs

Pour répondre à ce problème, dans le cadre d'un stage de fin d'études en master d'informatique dans le domaine de systèmes embarqués, on a fixé un objectif d'améliorer la qualité des images sous-marines en appliquant les algorithmes et ensuite d'évaluer leur performance en temps réel.

Pour ce faire, l'approche utilisée a consisté à étudier cinq algorithmes différents et à les évaluer selon neuf critères. Ensuite, la possibilité d'une application en temps réel a été étudiée, et les algorithmes ont été optimisés lorsque cela était possible.

Plus précisément l'objectif final était de trouver l'algorithme suffisamment rapide et suffisamment efficace pour pouvoir l'embarquer dans la donne sous-marine construite par les soigner de l'Université de Split 1. Mais suite aux problèmes techniques et manque de temps les études de domaine de traitement d'images sous-marines pendent ce stage sont restées plus théoriques qu'applicatif.

2.3 Organisation

Le stage a débuté le 22 mars 2021 pour une durée de 4 mois, en partenariat avec le projet SEA-UE, l'Université de Bretagne Occidentale et l'Université de Split sous l'autorité de M. Franc Singhoff (maître de stage) et la supervision de M. Hai Nam Tran, M. Vincent Rodin et Mme. Barbara Dzaja.

Les étapes principales sont :

- Étude des algorithmes et de l'état de l'art.
- Test de code source des algorithmes choisit avec l'environnement de développement MatLab.
- Sélection des critères de qualité d'image.
- Comparaisons des algorithmes en plusieurs étapes. En thermes de qualité de traitement d'images et de la vitesse de traitement.
- Sélection des algorithmes les plus intéressants.
- Optimisation des algorithmes en différents niveaux.
- Conclusion.

Ce stage a été effectué en binôme avec mon collègue Alan Leboudec.Les deux premières étapes ont été effectuées ensemble, ensuite le travail a été partagé et je me suis plus concentré sur certains algorithmes et l'optimisation de ces derniers grâce à l'environnement de développement et langage MatLab.

3 L'état de l'art

Au début du stage deux algorithmes ont été proposés par nos tuteurs pour nous immerger dans le domaine traitement d'images sous-marin :

- Local color mapping and color transfer [17]
- Sea-thru [6].

Ces deux articles ont été écrits très récemment et sont très enrichissants pour débuter dans la compréhension de traitement d'images sous-marines.Après une lecture de 2 semaines et plusieurs échanges par mail avec les auteurs des articles quelques décisions importantes ont été prises :

- Premièrement, choisir l'algorithme non seulement pour ces qualités innovantes, mais aussi par disponibilité de code source. Car malgré nos attentes l'algorithme favori Sea-thru [6] n'avait toujours pas un code accessible.
- Deuxièmement, trouver de nouveaux algorithmes en préférence avec un code source disponible au langage MatLab.
- Troisièmement, diversifier la recherche en choisissant les algorithmes utilisant différents moyens pour améliorer une image sous-marine.

3.1 Algorithmes

Suite aux recherches avec les paramètres annoncés précédemment cinq algorithmes ont été choisis : Local color mapping and color transfer [17], Underwater hazelines [8], Fusion enhancing [7], Backscatter removing [23] et Au-tomatic red-channel underwater image restoration [10].

3.1.1 Local color mapping and color transfer

Protasiuk et al. [17] décrivent une approche qui utilise conjointement deux méthodes déjà existantes LOCAL COLOR MAPPING et GLOBAL COLOR TRANSFER. L'algorithme compare chaque pixel avec un ensemble de pixels dont la couleur sous-marine est déjà connue. Il calcule et met en correspondance la covariance de l'image d'entrée et d'une image de référence. Les deux critères sont trouvés dans leur fonction objective 1 :

$$\min_{\boldsymbol{A},\boldsymbol{b}} f(\boldsymbol{A},\boldsymbol{b}) \underbrace{\|\boldsymbol{AX}-\boldsymbol{Y}+\boldsymbol{b}\mathbf{1}_{\mathbf{3}}^{\boldsymbol{T}}\|_F^2}_{\text{Local Color Mapping}} + \frac{\lambda_1}{2}\underbrace{\|\boldsymbol{AC_iA^T}-\boldsymbol{C_r}\|_F^2}_{\text{Global Color Transfer}} + \frac{\lambda_2}{2}(\|\boldsymbol{A}\|_F^2+\|\boldsymbol{b}\|_2^2) \tag{1}$$

Le troisième terme $\frac{\lambda_2}{2}(\|\boldsymbol{A}\|_F^2+\|\boldsymbol{b}\|_2^2)$ a pour but de trouver les paramètres **A** et **b** pour lesquels la fonction est minimale. Ils ont utilisé un troisième terme pour mettre à l'échelle le canal de la couleur rouge, de plus pour mettre à l'échelle la fonction ils ont utilisé les paramètres λ_1, λ_2.

3.1.2 Underwater Hazelines

L'idée principale donnée par Berman et al. [8] imprégnée par cet algorithme est que dans chaque canal de couleur **c∈R,G,B** l'image à chaque pixel est composée de deux composantes, le signal atténué et la lumière voilée, ce qui peut être vu dans l'équation 2 et Figure 2 :

$$I_c(\boldsymbol{x}) = t_c(\boldsymbol{x})J_c(\boldsymbol{x}) + (1-t_c(\boldsymbol{x}))\cdot A_c \tag{2}$$

Où les caractères corresponds respectivement :

- Les caractères **gras** désignent les vecteurs.
- $\boldsymbol{x}$ - la coordonnée du pixel.
- I_c - la valeur de l'image acquise dans le canal de couleur c
- t_c - la transmission de canal de couleur c.
- A_c - est la valeur de la scène dans les zones sans objet ($t_c = 0, \forall c \in \{R, G, B\}$).

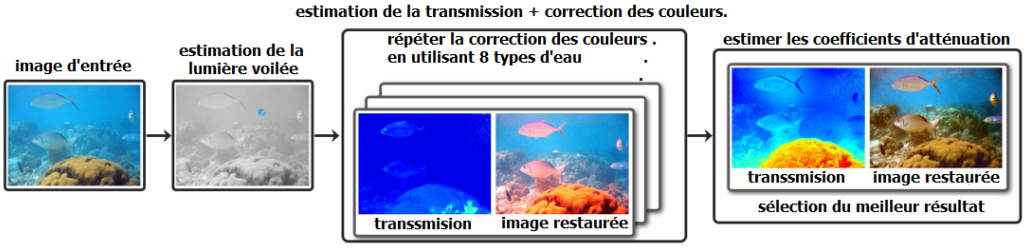

FIGURE 2 : Une description visuelle des étapes de l'algorithme UNDERWATER HAZELINES

3.1.3 Fusion enhancing

Ancuti et al. [7] proposent une solution alternative à **l'image unique**[1] basée sur les principes de la fusion multi-échelle. Elle vise une approche simple et rapide, capable d'augmenter la visibilité d'une grande variété de vidéos et d'images sous-marines. Le framework mélange des entrées spécifiques et choisit soigneusement les poids pour surmonter les limitations de tels environnements. Ceci est généralement vrai pour les scènes sous-marines correctement éclairées par la lumière naturelle. La stratégie d'amélioration comprend trois étapes principales :

- L'affectation des entrées (dérivation des entrées à partir de l'image sous-marine originale).
- La définition des mesures de poids.
- La fusion multi-échelle des entrées et des mesures de poids.

Dans l'équation 3, la version de l'image améliorée $R(x, y)$ est obtenue en fusionnant les entrées définies avec les mesures de poids à chaque emplacement de pixel (x, y) :

$$R(x,y) = \sum_{k=1}^{K} \bar{W}^k(x,y) I^k(x,y) \tag{3}$$

[1]Sans image de référence

3.1.4 Backscatter removing

Zhang and Chau [23] ont proposé d'éliminer le phénomène de baskscatter sur l'image. Pour ce faire, les auteurs utilisent la technologie de la fusion. Ils séparent l'image d'entrée en une image de réflectance et d'illuminance. Après cela, la couleur et la brume sur les deux images sont corrigées. De cette façon, ils appliquent une pyramide gaussienne et laplacienne basée sur la fusion multi-échelle. Le schéma général de la procédure de l'algorithme d'élimination de la rétrodiffusion est présenté dans la Figure 3.

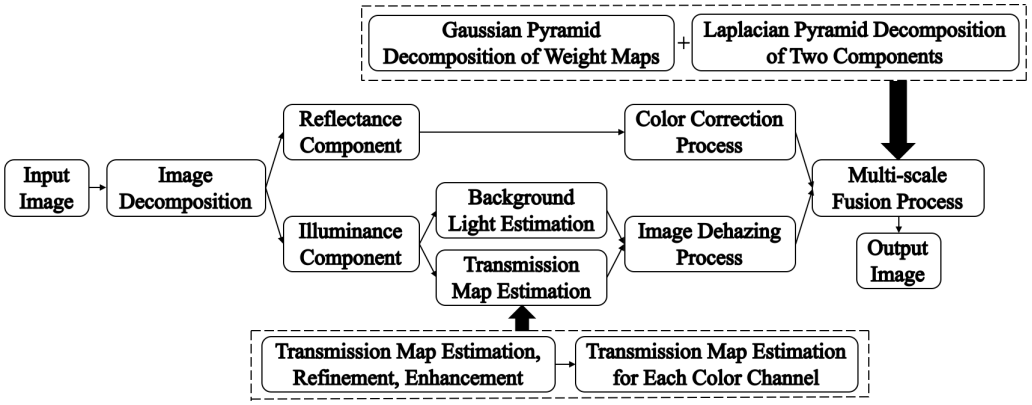

FIGURE 3 : Les procédures générales de traitement d'images avec l'algorithme BACKSCATTER REMOVING

3.1.5 Automatic red-channel underwater image restoration

Galdran et al. [10] expliquent que la mer a la tendance de déformer la lumière et d'absorber des longueurs d'onde spécifiques.Une majeur partie de la couleur rouge est absorbé. Ils ont proposé une approche pour restaurer une image sous-marine en trouvant les couleurs manquantes. La méthode du canal rouge est basée sur le travail du canal de couleur sombre qui l'utilise pour dégrader une image avec de la brume provenant d'un modèle atmosphérique.

4 MatLab

MatLab[2](« matrix laboratory ») est un langage de script émulé par un environnement de développement du même nom ; il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran.

Une partie importante de ce stage était l'apprentissage de langage Matlab avec son environnement de développement. Les divers outils que cet environnement nous offre permettent d'accélérer l'étude approfondie des algorithmes de traitement d'images sous-marines.

Les principaux outils utilisés :

- *Run and time* - permets de lancer l'algorithme et mesurer le temps d'exécution de chaque fonction et sous-fonction(Figure 4).
- *Application Compiler* - permets de créer une application autonome (stade alone) et dans la suite le lance indépendamment de l'environnement(Figure 5). MatLb
- *MATLAB Coder* - permets de générer du code *C* a partir de code. Cet outil était récemment intégré dans l'ensemble des outils MatLab. MatLab(Figure 5).

[2]https://fr.wikipedia.org/wiki/MATLAB#cite_note-2

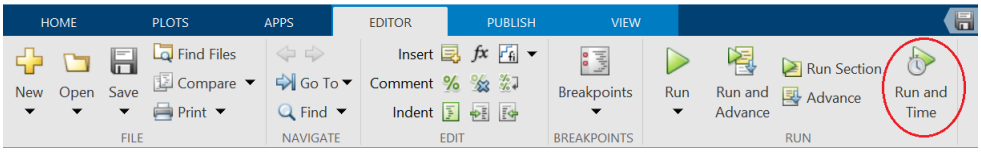

FIGURE 4 : MATLAB EDITOR

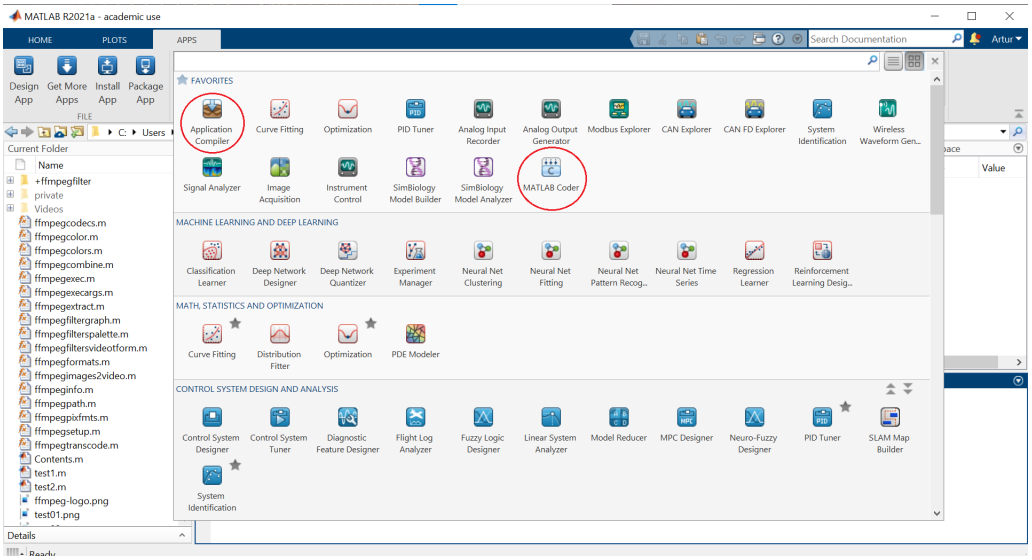

FIGURE 5 : MATLAB APPS

Mise à part de l'environnement de développement MATLAB propose une très grande communauté MATHWORKS[3] avec plusieurs forums et les librairies spécifiques pour les usages particuliers.

5 Analyse et adaptation de code

Les cinq algorithmes n'ont pas forcément été conçus pour l'usage particulier envisagé pour notre stage. C'est pour cela une fois le code source testé dans l'environnement MATLAB on passe aux étapes de l'analyse puis de l'adaptation de code. Dans cette partie nos efforts ont été plus concentrés sur deux algorithmes :

[3]https://fr.mathworks.com/

- Local color mapping and color transfer, étudié plus en détaille par mon collègue Alan.
- Underwater Hazelines, étudié de ma part.

C'est pourquoi dans la suite de ce rapport une grande partie des exemples vont être concentrés autour de ces deux algorithmes. Bien évidemment les solutions et les astuces trouvées pour les deux premiers algorithmes vont être appliquées dans l'ensemble de 5 algorithmes.

5.1 Suppressions des fonctions et des fonctionnalités inutiles

Plusieurs fonctionnalités des algorithmes ont été supprimées pour les rendre moins incombant et plus compréhensible. Pour un futur embarquement sur un drone sous-marin.

Exemples :

1. L'interface utilisateur dans l'algorithme Local color mapping and color transfer (Figure 17) a été supprimée pour assurer la possibilité de traitement d'images sous-marin indépendant de choix utilisateurs. Pour création d'un programme indépendante (stande alone) à l'aide de l'outil *Application Compiler* de MatLab.

2. La mode "Verbose" de l'algorithme Underwater Hazelines était inactif. Pour l'activer il fallait changer la valeur de variable globale *verbose = 1*. C'est une fonctionnalité bien utile en stade de test de l'algorithme mais complètement inutile pour une application embarquée (sauf pour création des files de l'historique de l'exécution dont on n'avait pas besoin). Ça nous a permis de supprimer plusieurs lignes de code inactive dans différentes fonctions de l'algorithme.(Figure 6)

```
42      % Preparations for saving results.
43 -    if ~exist(result_dir, 'dir'), mkdir(result_dir); end
44 -    jetmap = jet(256);  % Colormap for transmission.
45 -    verbose = false;    % Whether to print and save verbose details.
46 -    max_width = 2010;   % Maximum image width - larger images will be resized.
```

Figure 6 : Mode Verbose

5.2 Création d'un programme autonome

Après avoir suffisamment adapté le code d'algorithmes on a passé à l'étape de conditionnement de programmes autonomes à l'aide de l'outil *Application Compiler* de MatLab. Pour éviter de montrer toute la procédure pas à pas j'indiquerais simplement que c'est un outil assez intuitive et ne demande pas beaucoup d'efforts pour le maîtriser correctement en quelques clics. Seule difficulté rencontrée c'est l'ajout manuel des fichiers et des dossiers dont votre application aura besoin pour s'exécuter correctement en mode autonome.(Figure 7)Et donc les fonctions comme *adpath()* qui permettent de cibler les fichiers ou dossiers, ne seront plus utiles dans le code.

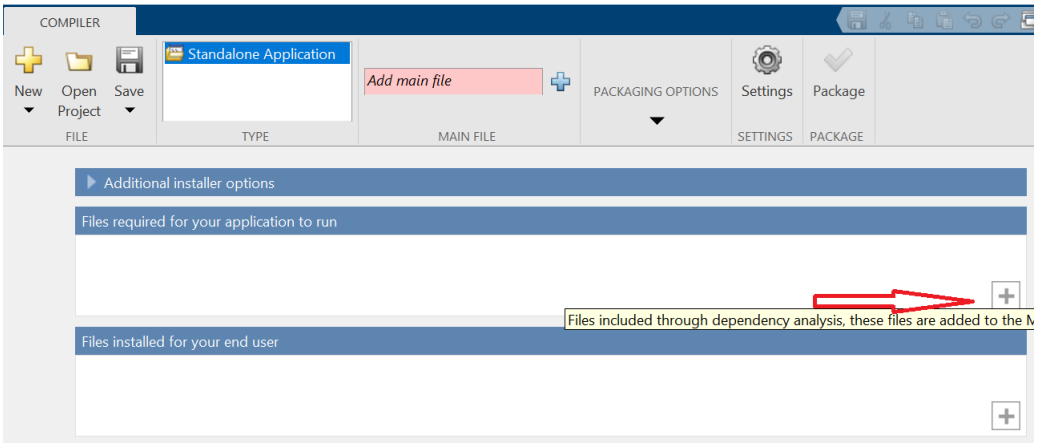

Figure 7 : *Application Compiler*

La tentative de créer des programmes autonomes a été un succès. Les algorithmes ont fonctionné correctement. Et sur les conseils de nos professeurs, il a été décidé d'entamer les phases d'optimisation et de comparaison des algorithmes. La partie de comparaison des algorithmes a été plus destinée à mon collègue Alan et donc je me suis occupé de la partie optimisation.

6 Optimisation

Plusieurs approches ont été testées pour optimiser des algorithmes, mais dans ce rapport je tiens à vous présenter ceux qui ont été couronnés de succès.

Dans cette partie je vais vous présenter les optimisations des l'algorithmes Underwater Hazelines et Fusion. Dans la suite du projet Underwater

Hazelines n'a pas été retenu en raison de la lenteur de ce dernier, mais les solutions trouvées pendent l'optimisation ont été réutilisées sur l'ensemble des algorithmes.

6.1 Correction de gestion des fichiers

Après avoir testé le code source de l'algorithme Underwater Hazelines j'ai constaté que la gestion des fichiers n'était pas au point car tout était sauvegardé dans le même répertoire ***images***. Donc avant de commencer l'optimisation principale je me suis soucié de mettre à jour la gestion des fichiers. Six répertoires ont été crée(Figure 8).

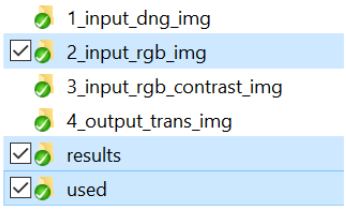

Figure 8 : ***2__input__rgb__img*** - *images d'entrée,* ***results*** - *images résultantes,* ***used*** - *images déjà traitées, les 3 autres dossiers sont pour les résultats intermédiaire.*

Ensuite j'ai mis en boucle infinie la fonction principale *uw_restoration* de l'algorithme pour traiter toutes les images dans le dossier ***2__input__rgb__img*** et avec une attente de 20 secondes avant arrêter la boucle si le dossier est vide(Figure 18).Cette approche de boucle infinie était une préparation de terrain pour adapter l'algorithme au traitement vidéo.

6.2 Graphe de flot

Un graphe de flot de l'algorithme Underwater Hazelines était indispensable pour voir l'ensemble de chemin qu'une image passe à travers de l'algorithme avant d'être améliorée. Sur ce graphe les temps d'exécution ont été calculés en majorité avec l'outil *Run and time* de MatLab et partiellement avec la fonction *tic toc*. La version complète [3] est très volumineuse donc je vais présenter uniquement les morceaux intéressants.

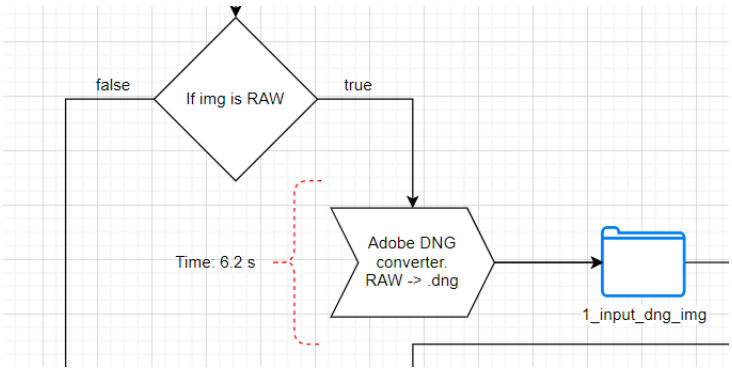

FIGURE 9 : *Adobe DNG converter*

En regardant le graphe de flot on peut constater en plusieurs endroits un bloc de ***IF image is RAW*** qui indique la possibilité de traitement d'images de format RAW [5] par l'algorithme UNDERWATER HAZELINES(Figure 9). Mais pour faire ça un programme externe *"Adobe DNG converter"* doit être installé. Ce programme permet la conversion de format RAW vers le format DNG [2]. L'inconvénient de cette application est d'être disponible uniquement pour le système d'exploitation *Windows*. Mais très rapidement nos professeurs nous ont demandés de nous concentrer plutôt sur l'amélioration de traitement d'images au format *JPEG*.

6.3 Optimisation de la vitesse de traitement d'images

Plusieurs approches ont été utilisées pour améliorer la vitesse de traitement des algorithmes mais souvent on constatait que les fonctions prédéfinies de MATLAB font appel aux fonctions écrit avec les langages FORTRAN, C ou $C++$. Donc souvent ces fonctions sont bien optimisées et donc impossibles de les améliorer. Malgré les difficultés on a trouvé quelques moyennes d'amélioration de codes statiques.

Quasiment tous les algorithmes avant de travailler avec l'image en format RGB font une conversion vers un autre format LAB spécifique au MATLAB et ensuite reconversion de format LAB vers RGB. Pour ce faire les fonctions prédéfinies *makecform()* et *applycform()* sont utilisées (Figure 10 11).

```
function lab = rgb_to_lab(rgb)
cform = makecform('srgb2lab');
lab = applycform(rgb,cform);
```

FIGURE 10 : *Conversion de format RGB au format LAB*

```
function rgb = lab_to_rgb(lab)
cform = makecform('lab2srgb');
rgb = applycform(lab,cform);
```

FIGURE 11 : *Conversion de format LAB au format RGB*

Mais une autre librairie de conversion **Colorspace Transformations** [1] permet de faire plus rapidement la conversion de format LAB au format RGB.

L'amélioration moyenne de traitement d'images pour l'algorithme FUSION après avoir intégré la librairie **Colorspace Transformations** :

- Conversion de format RGB au format LAB avec librairie Colorspace - 47% moins rapide.
- Conversion de format LAB au format RGB avec librairie Colorspace - 85% plus rapide.

Donc suite aux résultats on a décidé d'intégrer la librairie **Colorspace Transformations** uniquement pour conversion de format LAB au format RGB(Figure 12).

```
%{
   Lab to RGB  MatLab version
%}
% tic;disp('Lab to RGB  MatLab version');
% img2 = lab_to_rgb(lab2);
% toc;disp('--------------------------');


%{
   Lab to RGB  by Colorspace
%}
res2 = im2double(lab2);
img2 = colorspace('RGB<-Lab',res2);
img2 = im2uint8(img2);
```

FIGURE 12 : *Conversion de format LAB au format RGB avec la librairie* ***Colorspace Transformations***

Après la légère réussite d'amélioration de code statique je me suis concentré sur le traitement vidéo.

6.4 Adaptation de l'algorithme de traitement d'images au traitement vidéo

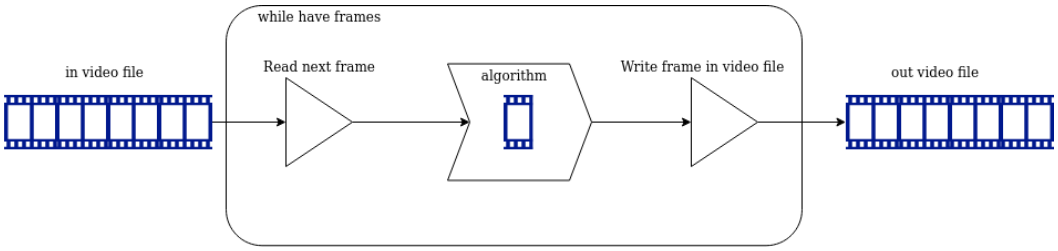

FIGURE 13 : *Boucle de traitement vidéo*

Pour débuter on a commencé avec un semple vidéo sous-marin de 12 secondes. Je me suis plus concentré sur l'adaptation de l'algorithme de FUSION. L'idée principale c'est de voir un fichier vidéo comme une suite des images

en moyenne 30 images par seconde. Par conseil de nos professeurs on a choisi le format H264 [4].

Langage MatLab nous offre les fonctions prédéfinies pour lire(fonction **VideoReader()**), écrire(fonction **VideoWriter()**) et lister(fonction **read()**) cadre après cadre d'un fichier vidéo. Donc une simple boucle *For* ayant un nombre d'itérations égale au nombre de cadres dans le fichier vidéo suffit pour appliquer l'algorithme de traitement à chaque cadre et donc à l'ensemble de fichier vidéo(Figure 13). En même temps on constatait bien que la vitesse de traitement et loin des contraintes de traitement en temps réel. En moyenne pour l'algorithme Fusion traitement d'une cadre demandait 2.3 secondes et donc pour traitement de 12 secondes de vidéo il fallait 12,3 minutes de temps de traitement.

C'est pour ça une première réflexion était de vouloir diminuer le nombre de cadres par seconde, réalisé dans le code sur la Figure 19. Chaque $5^{ième}$ cadre sera supprimé dans l'ensemble des cadres de fichier vidéo sortant.

Deuxième réflexion c'est d'exploiter mieux notre matérielle CPU ou/et GPU pour paralléliser le traitement de fichier vidéo.

6.5 Traitement de vidéo en parallèle

Un nouveau défi était la parallélisation de traitement vidéo avec langage MatLab avec un bloc spmd.

En premier lieu je me suis intéressé à paralléliser les calculs internes de l'algorithme et un diagramme de flot était nécessaire pour le faire(Figure 20). Mais vis-à-vis de la grosse différence entre les temps d'exécution de différentes fonctions interne de l'algorithme le parallélisme interne n'a rien amélioré.

Deuxièmement un traitement parallèle de cadres de fichier vidéo était visé. Pour éviter le problème de concurrence le programme partage d'abord le fichier vidéo en nombre de coeurs et chaque coeur va traiter sa propre partie(Figure 21). Cette approche a été beaucoup plus efficace. Avec une configuration *OS Ubuntu 18.04,Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz,16 GB of RAM*, sans utilisation de GPU, une amélioration de 47% de temps de traitement a été obtenue. Mais malgré cette amélioration de temps de traitement on reste loin de contraintes de système temps réel.

Finalement une fonction générale(**video_filter_parallel_function()**) a été créée qui prend en paramètre quiconque algorithme de traitement d'images l'emplacement de fichier vidéo, le chemin de vidéo de sortie et le format de vidéo de sortie voulut(Figure 22 23). Cette fonction s'adapte au-

tomatiquement au nombre N de coeurs de CPU et partage la vidéo en N partie avant de commencer de traiter en parallèle.

7 Comparaison des algorithmes

Dans cette partie je vais vous parler de la comparaison de 5 algorithmes effectué en grande majorité par mon collègue Alan.

7.1 Critères

Tout d'abord pendant les tests des algorithmes on a compris que choisir le meilleur résultat a l'oeil nu et quasiment impossible, car ça dépend de plusieurs facteurs comme le moniteur sur laquelle vous faites la comparaison ou encore notre goût esthétique qui peut varier. Donc certains critères plus scientifiques ont été choisis, recueilli dans l'article[11].

Il s'agit du BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) [13], du NIQE (Naturalness Image Quality Evaluator) [14], de l'UIQM (Underwater Image Quality Measures) [15], du PSNR (Peak Signal to Noise Ratio), du VIF (Visual Information Fidelity) [19], du IFC (Information Fidelity Criterion) [20] et du SSIM (Structural Similarity) [22].

Deux autres critères sont ensuite ajoutés : la distance euclidienne ou seconde norme [12] et l'erreur quadratique moyenne (EQM) [18]. Les métriques sont séparées en deux types, *avec référence complète* et *sans référence*.

7.1.1 Référence complète

Le critère de référence complet compare deux images à l'aide de calculs mathématiques. Le premier paramètre est l'image obtenue par l'algorithme qui est évalué et le second paramètre est l'image de référence. Dans ce cas, l'image de référence est l'original.

Pour le critère de référence complet, le plus important est l'EQM, qui indique la différence moyenne entre les pixels de l'image. Une valeur plus élevée représente une différence significative entre les images, c'est-à-dire que l'image obtenue avec l'algorithme est loin de l'image originale (de référence). Le PSNR est une mesure de la distorsion de l'image, il mesure la capacité à supprimer le bruit d'une image. Une valeur élevée de PSNR signifie une meilleure qualité d'image. SSIM est normalement employé pour mesurer les

informations récupérées de la lumière, du contraste et de la structure. Il est lié au critère Universal Quality Image (UQI) [21], et comme le critère précédent, il est meilleur si sa valeur est élevée. VIF est un indice d'évaluation de la qualité de l'image proposé en combinant le modèle statistique de l'image naturelle, le modèle de distorsion de l'image et le mode du système visuel humain. L'IFC est utilisé pour évaluer la qualité de l'image en utilisant les statistiques de la scène naturelle et la notion d'information de l'image extraite par le système visuel humain. Des valeurs élevées de VIF et d'IFC sont interprété de bons résultats.

7.1.2 Sans référence

Le critère sans référence n'évalue la qualité d'une seule image sur la base de calculs mathématiques. Le seul paramètre est donc l'image de l'algorithme qui doit être évaluée.

Les critères sans référence ont moins de termes. BRISQUE mesure la qualité de l'image en utilisant des coefficients de lumière normalisés localement qui ont été utilisés pour calculer les caractéristiques de l'image.

NIQE est un analyseur de qualité d'image totalement aveugle qui utilise uniquement les écarts mesurables par rapport aux régularités statistiques observées dans les images naturelles. Pour NIQE et BRISQUE, une petite valeur indique un meilleur résultat.

UIQM est basé sur trois mesures d'attributs d'images sous-marines : la mesure de la coloration de l'image sous-marine (UICM), la mesure de la netteté de l'image sous-marine (UISM) et la mesure du contraste de l'image sous-marine (UIConM).Plus la valeur est élevée, meilleur est le résultat.

7.2 Les résultats

Les principaux résultats sont obtenus pendant 2 expériences effectuées sur un ensemble de 15 images :

- Expérience $N°1$: trouver le meilleur algorithme parmi les 5 en matière de qualité d'image.
- Expérience $N°2$: mesure de temps d'exécution des algorithmes et optimisation de l'algorithme lidere de l'expérience $N°1$.

7.2.1 Expérience $N°1$

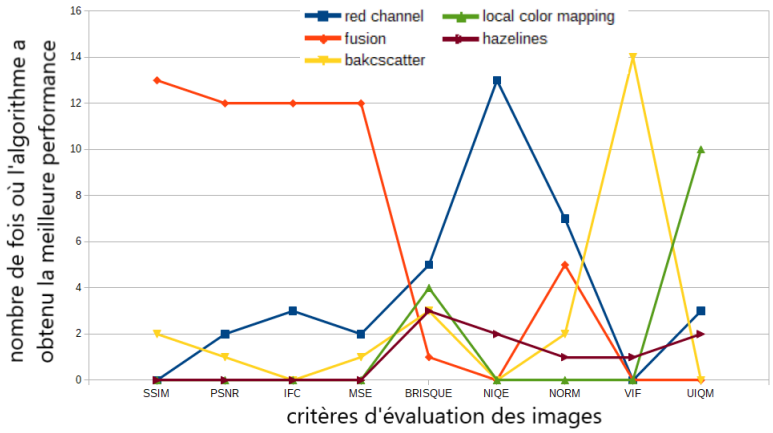

FIGURE 14 : Diagramme de l'expérience $N°1$

Les résultats de l'expérience montre le suivant(Figure 14) :

- L'algorithme de FUSION est le meilleur 4 fois avec les critères suivantes :
  - PSNR *référence complète*
  - SSIM *référence complète*
  - IFC *référence complète*
  - MSE *référence complète*
- L'algorithme AUTOMATIC RED-CHANNEL est le meilleur 3 fois avec les critères suivantes :
  - BRISQUE *sans complète*
  - NIQE *sans complète*
  - NORM *sans complète*

Les résultats montrent que l'algorithme de FUSION et le meilleur en thermes de qualité d'image.

7.2.2 Expérience $N°2$

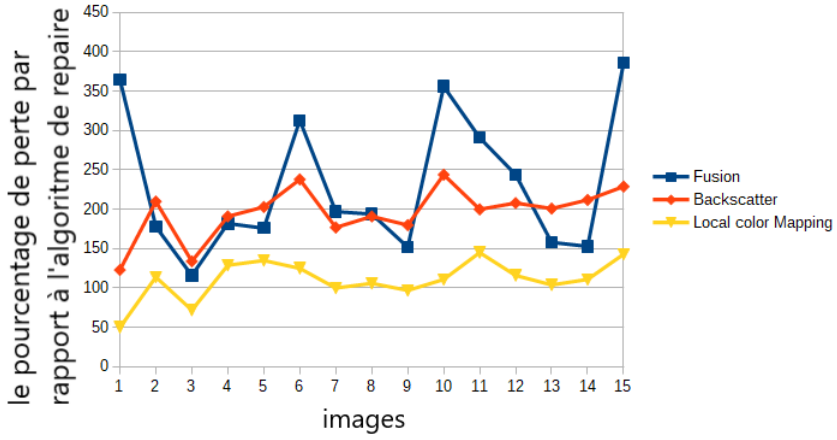

FIGURE 15 : Diagramme de l'expérience $N°2$ $1^{ière}$ partie

Les mesures de temps d'exécution des algorithmes avant optimisation (Figure 15) montre que :

- L'algorithme de AUTOMATIC RED-CHANNEL est le plus rapide et donc choisie comme repaire dans les Figures 15, 16
- l'algorithme de UNDERWATER HAZELINES a été exclu de la suite des expériences car trop lent, au moins 20 fois plus lent que les autres algorithmes.
- l'algorithme de FUSION est choisi pour l'optimisation dans la suite de l'expérience $N°2$

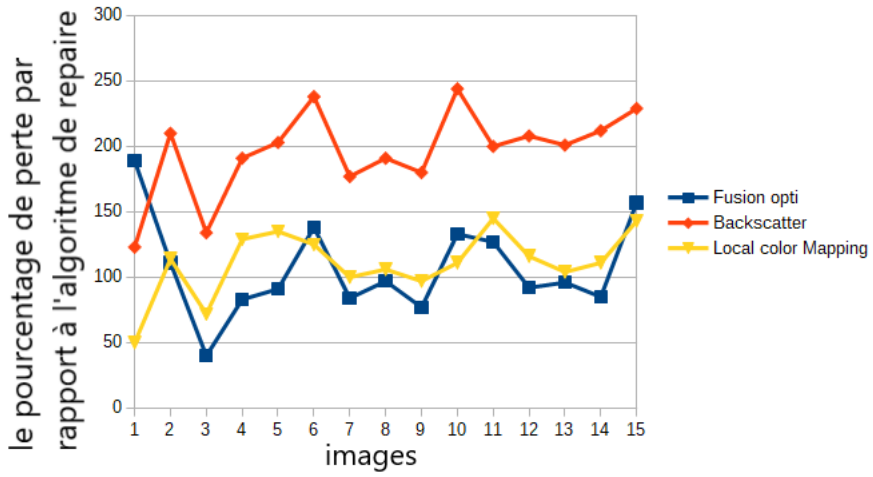

FIGURE 16 : Diagramme de l'expérience $N°2$ $2^{ième}$ partie

Les mesures de temps d'exécution des algorithmes après optimisation de l'algorithme FUSION(Figure 16) montre que :

- L'algorithme de FUSION monte un temps d'exécution qui n'est devancé que par AUTOMATIC RED-CHANNEL
- Une amélioration de 47% de temps d'exécution de l'algorithme de FUSION est obtenu

8 Conclusion

Pour conclure , pendant ce stage on a pu accomplir plusieurs objectifs :

- Faire une recherche exhaustive des algorithmes de traitement d'images sous-marin nouvellement créé.
- Optimisation et comparaison de ces derniers. Une travaille qui pourrait être utile pour les futures recherches dans le domaine.
- t par conséquent un article scientifique publiée "*Evaluation and Optimization of Underwater ImageRestoration Algorithms*"

Bien évidemment il en reste des objectifs incomplets et a réalisé dans les futures recherches :

- Conception de l'algorithme qui pourra surmonter les contraintes de système de traitement d'images/vidéo sous-marin temps réel. Tous les algorithmes étudiés ont des solutions intéressantes et des éléments à retenir pour concevoir un algorithme "parfait".
- Tester les solutions directement sur le drone sous-marin. Pendant ce stage malheureusement on n'a pas eu la chance de le faire.

9 Discussion

Ce stage de fin de Master d'informatique m'a permis de développer plusieurs compétences comme langage MatLab avec son environnement de développement, le domaine de traitement d'images en particulier dans les milieux sous-marins, la mise en pratique la langue anglaise tout au long du stage et surtout découvrir le monde professionnel en écrivant un vrai article scientifique.

10 Annexe

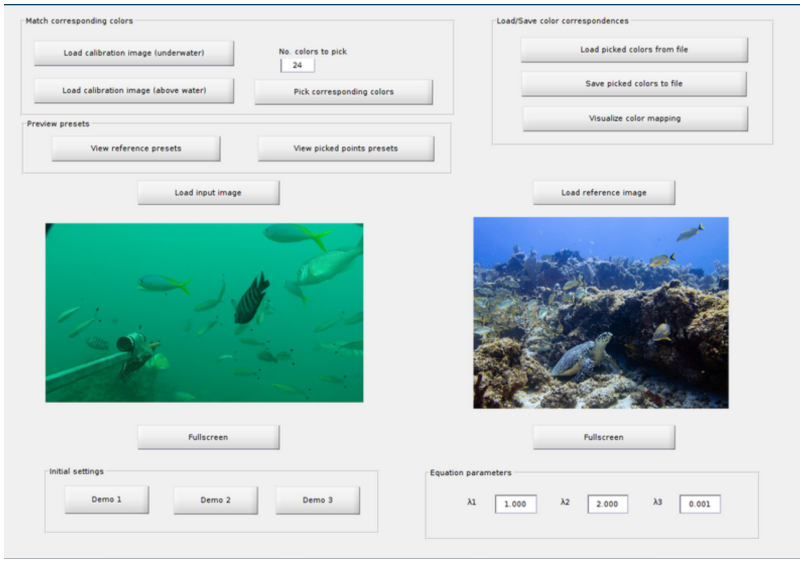

FIGURE 17 : interface utilisateur de l'algorithme LOCAL COLOR MAPPING AND COLOR TRANSFER

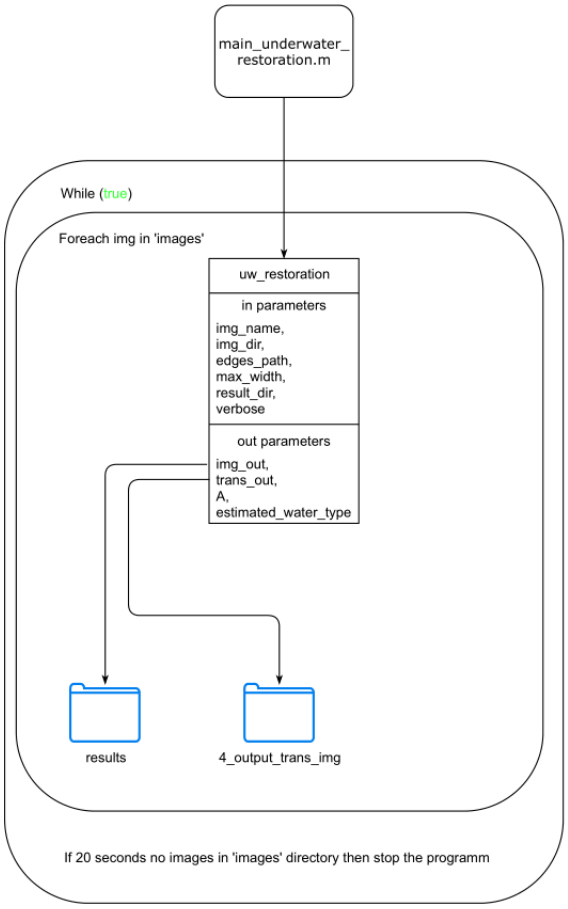

FIGURE 18 : *Boucle infinie de la fonction principale uw_restoration*

```
7   global frameRate;
8   frameRate = 5;
9
10
11  %Read video file
12  videoFile = VideoReader('Videos/video2.mp4');
13
14  %Write video
15  myVideo = VideoWriter('myNewVideoFile.avi');
16
17  %Display video
18  depVideoPlayer = vision.DeployableVideoPlayer;
19
20  %Open write video file
21  open(myVideo);
22
23  %% Filter application
24
25  % assume frameRate < vid.FrameRate
26  frameToExtract = ceil(videoFile.FrameRate/frameRate);
27
28  for frame = 1 : videoFile.NumFrames
29      % Extract the frame from the movie structure.
30      videoFrame = read(videoFile, frame);
31      if frame == 1 || mod(frame - 1, frameToExtract) == 0
32          newFrame=fusion_filter(videoFrame);
33          depVideoPlayer(newFrame);
34          writeVideo(myVideo,newFrame);
35      end
36  end
```

FIGURE 19 : *Lecture de fichier vidéo avec application de l'algorithme de* FUSION *et diminution de nombre de cadres*

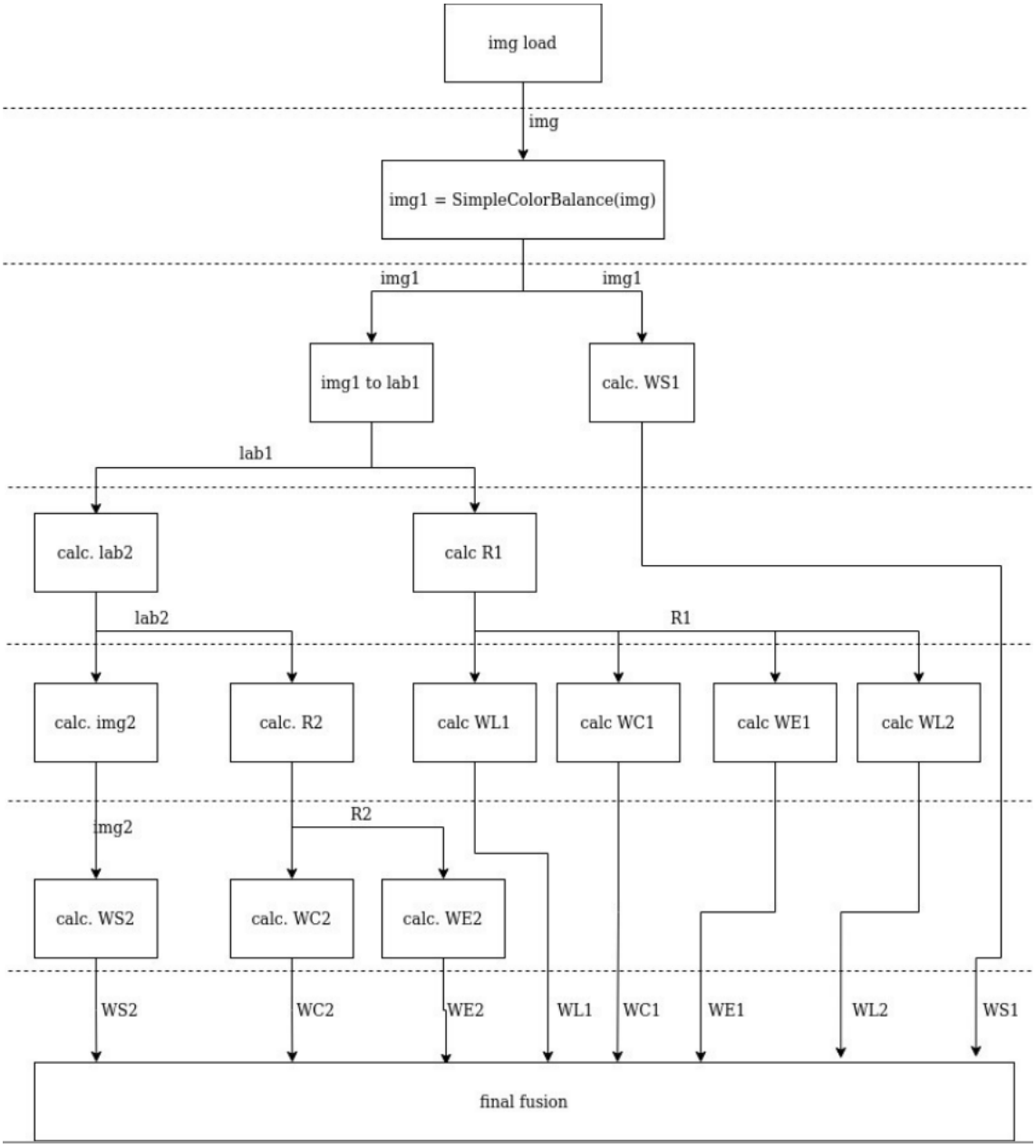

FIGURE 20 : *Diagramme de flot de l'algorithme* FUSION

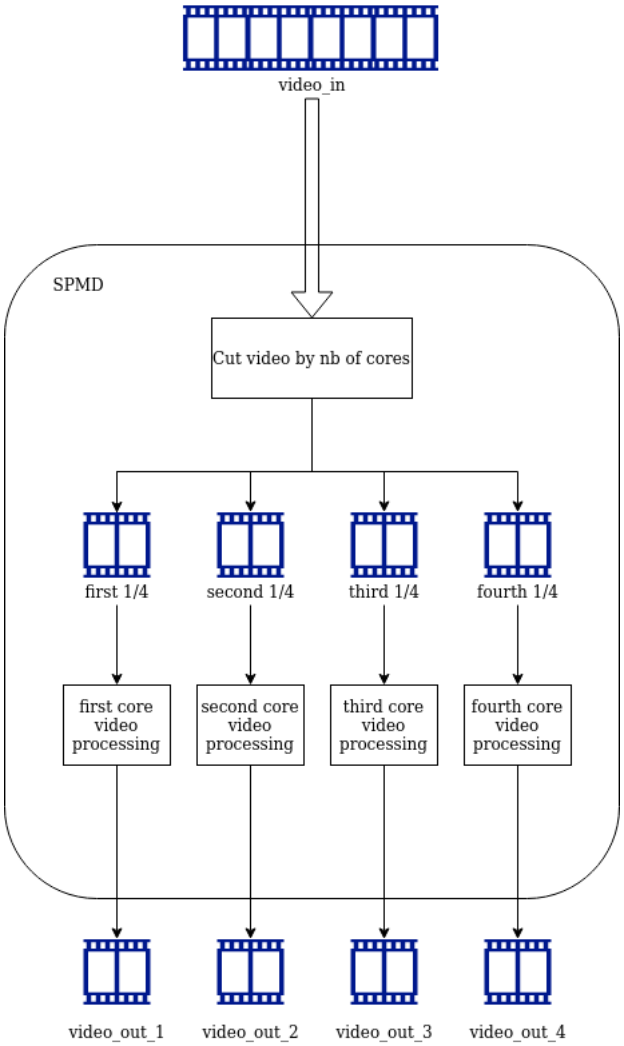

FIGURE 21 : *Diagramme de traitement vidéo en parallèle*

```
function video_filter_parallel_function(filterFunction,inVideoPath,outVideoDirectory,outVideoExt)
 % read_video_parallel_function weel read the videofile and cut it by the
 % nomber of cores and aplicate the filterFunction filter in parallel.
 % in out you get N filtered videos in outVideoDirectory with N = nomber of
 % cores.
    %{
        filterFunction     -> the function to filter every frame
        inVideoPath        -> teh full PATH of in video
        outVideoDirectory  -> the full PATH of out videos directory
        outVideoFormat     -> out videos format, exemple:'.avi','.mp4' ...
    %}



%Read video file
videoFile = VideoReader(inVideoPath);


% -----------Parallel pool creation------------%
    % test if pool is empty
    poolobj = gcp('nocreate');
    myCluster = parcluster('local');
    if isempty(poolobj)
        % pool creation with max number of workers
        parpool('local',myCluster.NumWorkers);
    end
%-----------------------------------------------%

% use the in video name to create out video name

[~,outVideosName,~] = fileparts(inVideoPath);
```

FIGURE 22 : *video_filter_parallel_function* $1^{ière}$ *partie*

```
spmd
    % divide the frames indexs in N equal parts (N is the nomber of workers)
    % numlabs - Total number of workers operating in parallel on current job
    myFrameIndexs = find(discretize(1:videoFile.NumFrames, numlabs) == labindex);


    % open VideoWriter in every worker
    outVideoFilePath = [outVideoDirectory,outVideosName,'.',int2str(labindex),outVideoExt];
    disp(outVideoFilePath);
    myFilteredVideoArray = VideoWriter(outVideoFilePath);

    %Open write video file in every worker
    open(myFilteredVideoArray);


    % Filter application in every worker
    % and writing nbCores filtered parts of main video
    for myFrameIndex = myFrameIndexs
        myFrame = read(videoFile, myFrameIndex);

        tic;
        filteredFrame = filterFunction(myFrame);
        writeVideo(myFilteredVideoArray, filteredFrame);
        toc;

    end

    %Close write video file in every worker
    close(myFilteredVideoArray);

end
```

FIGURE 23 : *video_filter_parallel_function* $2^{ième}$ *partie*

Références

[1] Colorspace Transformations. https://fr.mathworks.com/matlabcentral.

[2] DNG format information. https://fr.wikipedia.org/wiki.

[3] graphe de flot de l'algoritme Underwater Hazelines. https://drive.google.com/file/.

[4] Format H.264, ou MPEG-4 AVC. https://fr.wikipedia.org/wiki/H.264.

[5] RAW format information. https://fr.wikipedia.org/wiki/RAW.

[6] Derya Akkaynak and Tali Treibitz. Sea-thru : A method for removing water from underwater images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1682–1691, 2019.

[7] Cosmin Ancuti, Codruta Orniana Ancuti, Tom Haber, and Philippe Bekaert. Enhancing underwater images and videos by fusion. In *2012 IEEE conference on computer vision and pattern recognition*, pages 81–88. IEEE, 2012.

[8] Dana Berman, Tali Treibitz, and Shai Avidan. Diving into haze-lines : Color restoration of underwater images. In *Proc. British Machine Vision Conference (BMVC)*, volume 1, 2017.

[9] Dive & Discover. Underwater Research Vehicles,How Deep Can We Go ? https://divediscover.whoi.edu/underwater-vehicles/how-deep-can-we-go/, 2021.

[10] Adrian Galdran, David Pardo, Artzai Picón, and Aitor Alvarez-Gila. Automatic red-channel underwater image restoration. *Journal of Visual Communication and Image Representation*, 26 :132–145, 2015.

[11] Guojia Hou, Xin Zhao, Zhenkuan Pan, Huan Yang, Lu Tan, and Jingming Li. Benchmarking underwater image enhancement and restoration, and beyond. *IEEE Access*, 8 :122078–122091, 2020.

[12] MD Malkauthekar. Analysis of euclidean distance and manhattan distance measure in face recognition. In *Third International Conference on*

Computational Intelligence and Information Technology (CIIT 2013), pages 503–507. IET, 2013.

[13] Anish Mittal, Anush Krishna Moorthy, and Alan Conrad Bovik. No-reference image quality assessment in the spatial domain. *IEEE Transactions on image processing*, 21(12) :4695–4708, 2012.

[14] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. Making a "completely blind" image quality analyzer. *IEEE Signal processing letters*, 20(3) :209–212, 2012.

[15] Karen Panetta, Chen Gao, and Sos Agaian. Human-visual-system-inspired underwater image quality measures. *IEEE Journal of Oceanic Engineering*, 41(3) :541–551, 2015.

[16] EMILY PETSKO. Why does so much of the ocean remain unexplored and unprotected ? https://oceana.org/blog/why-does-so-much-ocean-remain-unexplored-and-unprotected, 2020.

[17] Rafał Protasiuk, Adel Bibi, and Bernard Ghanem. Local color mapping combined with color transfer for underwater image enhancement. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1433–1439. IEEE, 2019.

[18] Mark D Schluchter. Mean square error. *Encyclopedia of Biostatistics*, 5, 2005.

[19] Hamid R Sheikh and Alan C Bovik. Image information and visual quality. *IEEE Transactions on image processing*, 15(2) :430–444, 2006.

[20] Hamid R Sheikh, Alan C Bovik, and Gustavo De Veciana. An information fidelity criterion for image quality assessment using natural scene statistics. *IEEE Transactions on image processing*, 14(12) :2117–2128, 2005.

[21] Zhou Wang and Alan C Bovik. A universal image quality index. *IEEE signal processing letters*, 9(3) :81–84, 2002.

[22] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment : from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4) :600–612, 2004.

[23] Hao Zhang and LP Chau. *Removing Backscatter to Enhance the Visibility of Underwater Object*. PhD thesis, Master's Thesis, Nanyang Technological University, Singapore, 2016.