Université de Bretagne Occidentale

Rapport de Stage

Études d'algorithmes pour améliorer une image sous l'eau

écrit par

Alan Le Boudec

Master 2 Logiciels pour les systèmes embarqués

supervisé par

Hai-Nam Tran Vincent Rodin

Barbara Džaja (University of Split)

Maître de stage

Frank Singhoff

10 September, 2021

Remerciements

Je tiens tout d'abord à remercier l'Université de Brest, l'université de Split ainsi que le Lab-STICC de m'avoir permis de réaliser ce stage. J'exprime aussi ma reconnaissance envers les différents employés du Laboratoire avec lesquels j'ai pu travailler qui ont permis de faire de ce stage une expérience enrichissante.

Je remercie tout particulièrement M. Hai-nam Tran, mon tuteur de stage de m'avoir encadré et conseillé tout au long de ce stage mais aussi pour sa disponibilité et son aide.

Mes remerciements vont également à Mme. Barbara Džaja et M. Vincent Rodin pour leur accompagnement et leurs conseils, auprès desquels j'ai beaucoup appris et découvert le domaine du traitement d'images.

Sommaire

Acronymes

ARUIR Automatic red-channel underwater image restoration

BRISQUE Blind/Referenceless Image Spatial Quality Evaluator

BS Backscatter removing

Fusion Fusion enhancing

GUIDE Graphical User Interface Development Environment

HAZELINES Diving into haze-lines: Color restoration of underwater images

IFC Information Fidelity Criterion

LCMCT Local Color Mapping and Color Transfert

MSE Mean Square Error

NIQE Naturalness Image Quality Evaluator

PSNR Peak Signal to Noise Ratio

SSIM Structural Similarity

UCIQE Underwater Color Image Quality Evaluation

UICM underwater image colourfulness measure

UIConM underwater image contrast measure

UIQM Underwater Image Quality Measures

UISM underwater image sharpeness measure

UQI Universal Quality Image

VIF Visual Information Fidelity

Chapitre 1

Introduction

1.1 Contexte et problématique

Le traitement d'images est de plus en plus régulièrement utilisé, de la reconnaissance faciale à la détection d'objets les domaines sont vastes et variés. Néanmoins, l'utilisation de ces traitements se fait avec un modèle atmosphérique, dans le milieu sous-marin c'est un modèle différent qui prévaut.Les propriétés physiques de l'eau dégradent les images qui peuvent être capturé par des appareils photos ou des caméras. Aujourd'hui beaucoup de recherches traites sur la capacité à restaurer la qualité de ces images sous l'eau.

Dans le cadre de la collaboration avec Sea-ue et l'université de Split en Croatie, Ils ont à disposition un rover destiné à la recherche sous-marine. Equipé de caméras et d'un système "On Board", Il doit être capable de traiter l'information et immédiatement. Ce contexte nous confronte à 2 types de contraintes, la première temps réel, car la vitesse de transfert de données sous l'eau est très limitée et on ne peut qu'envisager un traitement "On Board". La seconde, une bonne qualité d'image, afin de pouvoir réaliser les traitements avec le plus de précision possible.

1.2 Objectifs

Les objectifs de ce stage est double. Tout d'abord nous voulons tester et appliquer des algorithmes de traitement d'images afin d'améliorer la qualité des images sous l'eau et évaluer les différentes possibilités d'optimisation des algorithmes pour une application temps réel.

1.3 Organisation

Le stage s'est effectué sur une durée de 4 mois, dans l'enceinte de l'université de Brest au laboratoire du Labsticc. Le sujet est divisé en 2 parties pour 2 étudiants. Ainsi, le travail s'est déroulé en parallèle entre moi et un autre étudiant de la même promotion. Dans l'objectif d'avoir un travail différent à présenter, nous avons travaillé chacun sur des algorithmes et sur des moyens d'optimisations différentes.

1.4 Approche

Dans ce contexte, afin de répondre à cette problématique, nous choisissons une approche en 2 étapes. En premier lieu, nous allons faire une évaluation de la qualité d'images de différents algorithmes de traitement d'images sous l'eau par le biais de critères mathématiques. Dans un second temps, nous allons tester l'algorithme avec les meilleurs résultats pour voir s'il est possible de l'appliquer sur une cible temps réel.

Chapitre 2

Algorithmes

Il existe un bon nombre d'algorithmes qui permettent d'améliorer les images sous l'eau.Nous avons été Tout d'abord invités à regarder de plus près 2 méthodes de traitement d'images Local Color Mapping and Color Transfert [8] et Diving into haze-lines: Color restoration of underwater images [3] codé sur Matlab. Ceci pour comprendre comment fonctionne un algorithme de traitement d'images mais aussi de voir comment cela est géré avec Matlab. On a fait un état de l'art des 2 articles et l'étude de leur code, moi pour LCMCT et mon collègue sur HAZELINES.

2.1 Article

Protasiuk et al. [8] décrivent une approche utilisant 2 algorithmes de traitement d'images ensemble.

- Local color mapping

  Compare chaque pixels de l'image avec un ensemble de pixel dont les couleurs sous l'eau sont connues. Ainsi lorsque regarde un pixel marron par exemple, cela vérifie avec l'ensemble et donne la couleur qui correspond dans l'atmosphère soit rouge.

- Color Transfer

  Calcule la différence entre la covariance de l'image à traiter et d'une image de référence.

Les 2 méthodes sont associés et réunis dans l'équation suivante 1

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

Cette équation est divisée en 3 termes, le premier qui correspond au *Local color mapping* est une fonction affine de la forme Ax + b. Le second correspond à *Color transfer* avec C_r et C_r la covariance des images, et enfin le dernier terme est là pour équilibrer la couleur rouge de l'image. En effet, Lors de leur test ils ont remarqué que la couleur rouge était atténuée. De plus, ils ont rajouté des lambdas devant chaque terme afin d'équilibrer la formule.

2.2 Analyse du code

Le code source est disponible sur la page github [9] de l'un des auteurs de l'article . Écrit sous Matlab, il contient plusieurs fichiers de fonctions et un programme principal que l'on peut voir sur la figure 13. Mis à part le programme principal "correction_gui.m", les autres fichiers sont des fonctions de gestion d'interface graphique et de calcul de l'algorithme. En Matlab un nom de fichier correspond au nom de la fonction, ainsi si l'on veut créer par exemple une fonction somme(A,B) il faudra que le fichier s'appelle "somme.m."

2.2.1 Fonctions

Le fichier principal "correction_gui.m" est créé avec GUIDE. Cet outil Matlab permet en effet de générer et gérer une interface utilisateur. Ainsi le programme contient une partie gestion d'interface mais aussi algorithmique. Dans le cadre de notre recherche on s'intéresse principalement à la partie calcule. La fonction qui lance le traitement de l'image est décrite dans l'équation 2 :

$$[A, B] = estimate\_ab\_matrix\_trust(picked\_i, picked\_o, im, ref, l1, l2, l3, w2) \tag{2}$$

Dans Matlab les images sont gérées grâce à des matrices, la simplicité d'utilisation permet une grande flexibilité pour le traitement des images. Ainsi A et B sont les deux matrices résultantes de la fonction. Elles sont entre crochets ce qui signifie qu'elles sont en plus fusionné en 1 matrice. Les 2 premiers paramètres sont les ensembles de pixels qui, *im* et *ref* sont les images à traiter et de référence. *l1, l2 et l3* sont les lambdas pour l'équilibre de la formule. Cependant dans l'équation 1 il y en que 2, en effet, puisque dans le code *l1* est défini sur "1.000" ce qui explique pourquoi ils ne l'ont pas affiché sur l'équation. Néanmoins, ils laissent la possibilité de le modifier. Le dernier paramètre *w2* est un poids ajouté à l'équation, toujours dans cette optique d'équilibrer la formule.

Le calcul de l'algorithme se fait en 3 étapes

- Calcul des premières valeurs de la fonction objective.

$$T = estimate\_initial\_color\_t\_matrix(psi\_o(1,:)', psi\_o(2,:)', psi\_o(3,:)', \\ psi\_i(1,:)', psi\_i(2,:)', psi\_i(3,:)', 0) \tag{3}$$

psi_o et psi_i font référence aux ensembles de correspondances de pixels. Cette manière d'écrire en Matlab, "matrice"(1,:) signifie que l'on veut la première ligne de la matrice sous forme de colonne.

- Évaluation de la covariance pour les deux images

$$Ci = get\_color\_cov\_matrix(input\_im); \\ Cr = get\_color\_cov\_matrix(ref\_im); \tag{4}$$

Cette fonction fait elle-même appelle à une fonction Matlab *cov()* qui calcule la covariance.

- Calcule le minimum de la fonction objective par technique de gradient avec les valeurs initiales trouvé dans l'étape précédente.

$$[x, fval, exitflag, output] = fminunc(fun, x0, options); \tag{5}$$

Le premier paramètre *fun* pointe sur la fonction à utiliser pour le calcul du minimum. Ajouté à cela on lui donne des paramètres écrits dans *options* qui contiennent les résultats obtenus précédemment et d'autres données utile pour la fonction *fun*.

2.2.2 Interface Utilisateur

Le programme principal généré par GUIDE affiche une interface utilisateur lors de son lancement. Son interface est décrite ci-dessous sur la figure 1 :

- 1. Création de l'ensemble de correspondances des pixels

  Permet de créer l'ensemble de pixels à partir d'une image sous et en dehors de l'eau. Associe un certain nombre de pixels entre eux, afin de faire la correspondance. De plus, ce nombre de pixels peut aussi être choisi par l'utilisateur.

- 2. Sélection d'un fichier de correspondance de pixels

  Sélectionner un fichier ".dat" qui contient déjà les données de correspondance entre les pixels.

- 3. Vision sur les fichiers

  Permet de voir les différents fichiers pour l'image de référence ou encore les fichiers ".dat" pour les ensembles de correspondances des pixels.

- 4. Choisir l'image d'entrée et de référence

  l'image d'entrée et l'image que l'on cherche à améliorer et l'image de référence servira pour le calcul de la covariance.

- 5. Paramétrage des lambdas

  Équilibrage de l'équation de l'algorithme avec les 3 lambdas.

- 6. Démonstration différentes configurations pré-enregistrées pour tester le programme .

- 7. Start Lancement du programme avec les paramètres, avec possibilités de l'appliquer sur plusieurs images en même temps avec le bouton "multiple file" ou encore de sauvegarder le résultat. C'est ce bouton qui va lancer le calcul de l'algorithme vu dans l'équation 1 avec la fonction 2.

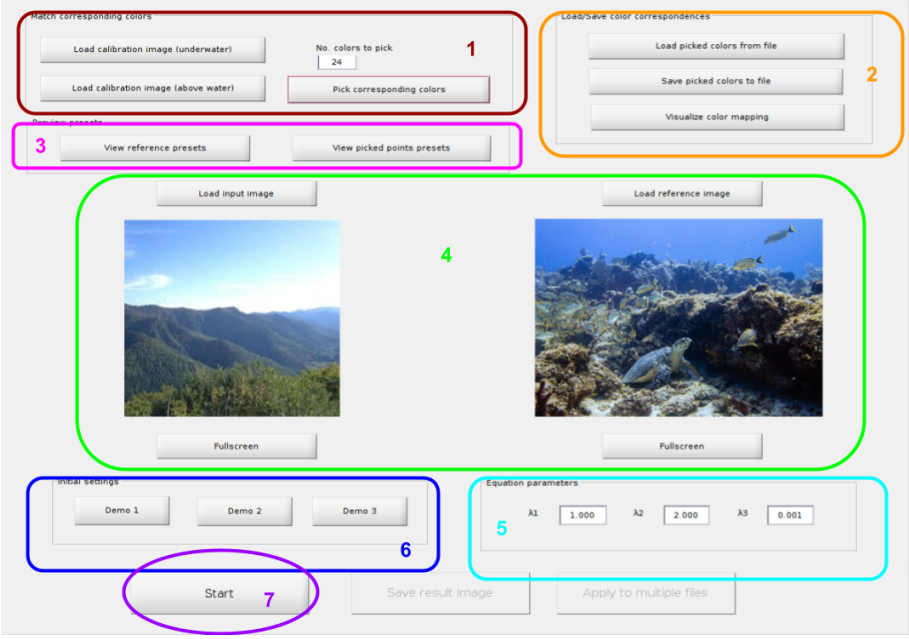

Figure 1: Interface Utilisateur LCMCT

2.3 État de l'art des algorithmes

La première étape de notre approche est d'évaluer les algorithmes de traitement d'images sous l'eau, pour cela nous allons les comparer. L'étude approfondie de LCMCT nous a permis de comprendre comment fonctionne un programme de traitement d'images, Comment appliquer l'équation de l'algorithme mais aussi de voir comment cela est codé et géré en Matlab. On a au total choisi 5 algorithmes pour étoffer notre liste. Mis à part HAZELINES, les 3 autres algorithmes sont

disponibles sur github à cette adresse "REFERENCE". Contrairement à LCMCT nous n'allons pas rentrer dans les détails du code mais juste présenter en quelques lignes leur fonctionnement.

- Underwater Hazelines

  L'idée générale qui est donnée dans l'article [3] à travers cet algorithme est que pour chaque couleur c ∈ R,G,B l'image à chaque pixel contient 2 composantes "attenuated signal" et "veiling light", qui peut être vu dans l'équation 6:

$$I_c(\boldsymbol{x}) = t_c(\boldsymbol{x})J_c(\boldsymbol{x}) + (1 - t_c(\boldsymbol{x})) \cdot A_c \tag{6}$$

  $\boldsymbol{x}$ est la coordonnée du pixel, I_c est la valeur de l'image obtenue dans ses canaux de couleurs c, t_c est la transmission de canaux de couleur, et J_c est la radiance de l'objet qui a besoin d'être restauré. Le composant global "veiling light" A_c est la valeur de la scène dans une zone de l'image où il n'y a pas d'objets ($t_c = 0, \forall c \in \{R, G, B\}$). C'est-à-dire, si l'on prend une zone dans l'image ou il n'y a pas d'objet et que la couleur générale est le rouge, alors la valeur de la composante sera ce rouge.

- Fusion Enhancing

  Cette solution décrite dans l'article Ancuti et al. [2] se base sur le principe de fusion multi-échelle. Elle vise une approche simple et rapide, capable d'améliorer la qualité d'une grande variété d'images et de vidéos sous l'eau. La stratégie consiste en 3 étapes : affectation des entrées, définition des mesures des poids et fusion multi-échelle des entrées et de la mesure calculée. Dans l'équation 7 la version de l'image améliorer $R(x, y)$ est obtenu par fusion des différentes entrées avec les mesures de poids pour chaque pixel (x, y):

$$R(x, y) = \sum_{k=1}^{K} \bar{W}^k(x, y)I^k(x, y) \tag{7}$$

- Backscatter removing

  Le principe de cette méthode décrite dans l'article Zhang and Chau [11] est simple, enlever le "backscatter" d'une image. Le "backscatter" est la rétrodiffusion de la lumière dans l'eau. Effectivement, dans l'eau les bulles d'air présentes créent de mauvaises diffusions de la lumière. Leur méthode se base sur la fusion, ils séparent l'image en 2, une pour la réflectance et la seconde pour l'illuminance. Ensuite, ils corrigent la couleur sur les deux images ainsi que l'effet de "haze". Ils appliquent la pyramide de "gaussian and laplacian" sur ces images avec la fusion "multi-échelle". la figure 2 présente le diagramme de la procédure de l'algorithme.

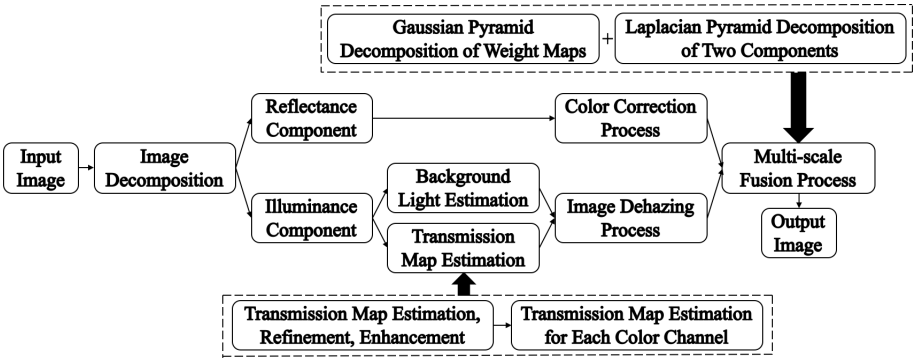

Figure 2: Diagramme de la procédure de BS

- Automatic red-channel underwater image restoration

  La première chose à savoir qui est décrite dans l'article Galdran et al. [5] est que l'eau à des propriétés qui distorde la lumière, comme lorsqu'on on plonge une paille dans l'eau elle parait coudée, et une absorption spécifique des longueurs d'onde. Ils proposent une approche qui restaure une image sous l'eau en retrouvant les couleurs manquantes. Sachant que dans l'eau c'est la couleur rouge qui est principalement absorbée, c'est surtout qui sera restauré. Cette méthode est basée sur le travail des canaux de couleurs sombres qui l'utilise pour dégrader une image avec la brume obtenue à partir du modèle atmosphérique.

Chapitre 3

Évaluation des algorithmes

L'évaluation est une comparaison d'algorithmes sur différents critères mathématiques. Cette méthode n'est pas nouvelle, beaucoup comparent leur propre solution afin de la mettre en avant, cependant ce n'est pas très objectif puisqu'ils ne travaillent par la suite que sur leur approche sans étudier plus les résultats des autres algorithmes. Notre travail consiste à évaluer les méthodes pour voir leur qualité de résultats en se basant sur des critères mathématiques choisis.

3.1 Critères

L'évaluation de la qualité des images des algorithmes va se faire par le biais de différents critères. Nous avons utilisé ceux présents dans l'article [6], 7 sont utilisés UIQM, SSIM, VIF, IFC, NIQE, BRISQUE et PSNR. Deux autres sont ajoutés à la liste : la deuxième norme ou plus couramment appelée distance euclidienne et le MSE. Les critères d'évaluation sont séparés en 2 types, les *full référence* et les *no reference*.

- *full référence*

  Les critères de ce type comparent deux images, une d'entrée et une de référence. Dans notre cas la première est l'image que l'on cherche à évaluer et la seconde est l'originale. Parmi les critères *full référence*, l'un des plus importants est le MSE, qui permet de vérifier la moyenne d'erreur entre chaque pixel des images. Une haute valeur du MSE représente un taux d'erreur important. PSNR mesure la capacité à enlever le "bruit" d'une image, Si cette valeur est élevée la qualité de l'image est correcte. SSIM est une mesure de similarité de structure entre deux images, plutôt qu'une différence pixel par pixel. Ce critère est basé sur Universal Quality Image [10]. Comme précédemment, plus le résultat a une haute valeur, meilleur il est. VIF est un index d'évaluation de qualité d'image en combinant le modèle statistique

d'image naturelle, le modèle de distorsion d'image et celui du système visuel humain. IFC est utilisé pour l'évaluation de la qualité d'image qui utilise les statistiques des scenes naturelles et de la notion d'information d'image extraite par le système visuel humain. IFC et VIF ont de bons résultats lorsque leur valeur est élevée.

- *no reference*

  Les critères *no reference* sont ceux qui évaluent juste l'image d'entrée et à l'inverse ne font pas de comparaison avec une autre image. BRISQUE mesure la qualité de l'image en utilisant des coefficients de lumière normalisés 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 3 mesures d'images sous-marines, the UICM, the UISM and the UIConM. Un bon UIQM à des résultats élevés.

3.2 Jeu de données / Benchmark

La partie de test nécessite un jeu de données assez conséquent pour avoir de bons résultats. Nous avons choisit un ensemble de 20 images, avec un représentant de chaque échelle, grand, moyen et petit format. Ces images sont libres de droits et disponibles sur internet. Notre objectif est de comparer et de voir quels algorithmes est capables de restaurer au mieux la qualité d'une image. Pour cela l'ensemble d'images choisi est un ensemble d'images de bonne qualité. Ainsi nous allons les dégrader et appliquer les algorithmes. Cette idée va permettre de savoir à l'avance quel doit être le résultat du traitement, avec le système visuel humain. On va comparer l'original et le résultat de l'algorithme de traitement d'images. La dégradation effectuée sur les images est relativement simple. Nous avons ajouté un filtre bleu pour l'aspect de l'eau et aussi du *bruit*. Le *bruit* est un ensemble d'anomalies qui peuvent apparaître aléatoirement sur une image prise par un appareil photo ou une caméra. Le résultat de cette dégradation peut se voir sur la figure 3. Après une première phase de test l'ensemble de données s'est réduit, montrant certaines limites quant à la possibilité de travailler avec n'importe quel format d'image. C'est-à-dire que pour certains algorithmes lorsque l'image était trop petite cela entraînait une boucle infinie qui n'était pas réglable sauf si l'on modifiait l'algorithme. Sachant que nos connaissances du code, de l'environnement et aussi de l'algorithme était limité on a décidé de retirer les images incriminantes. 5 images ont donc été supprimées, on a un jeu de données comportant 15 images à traiter pour les tests.

Figure 3: Dégradation d'une image (source image :[1])

3.3 Étude de cas lcmct

Notre travail approfondi sur LCMCT nous a permis de voir qu'il y avait de nombreux paramètres à utiliser. C'est dans cette optique que l'ont à crée un programme capable de gérer certains des paramètres et de les choisir en fonction du résultat. Cette fonction calcule le résultat des critères pour une image de référence de l'ensemble donné avec le code, puis refait le même travail avec les autres tout en prenant soin de sauvegarder les résultats précédents. Ainsi, le meilleur résultat sur les critères révèle la meilleure image de référence à choisir. Le programme visible sur la figure 14 nous permet d'assurer un minimum de bons résultats, cependant cela n'implique pas les autres paramètres utilisés par l'algorithme.

3.4 Résultats

Tous les tests ont été réalisés sous Matlab, de plus nous avons utilisé Excel pour le traitement des résultats obtenus. La visibilité et la compréhension étant plus compliqué avec des tableaux nous allons voir ça sous forme de graphe. Chaque graphe de cette partie utilise la légende de la figure 15.

3.4.1 *full reference*

Le critère MSE donne de mauvais résultats si sa valeur est grande. La figure 4 présente ces résultats. On peut voir qu'une courbe se dessine en dessous des autres c'est celle de Fusion qui a les plus petites valeurs et donc les meilleurs résultats. Si le PSNR est élevé cela indique que le résultat est bon. Fusion apparaît encore comme la meilleure des solutions sur la figure 4. VIF obtient de bons résultats si ceux-ci sont élevés et c'est le cas pour la courbe jaune ou autrement dit ARUIR. C'est cet algorithme qui a les meilleurs résultats. On s'aperçoit sur la figure qu'il est difficile à l'oeil humain de distinguer les courbes tellement elles sont proches. Cependant on peut voir que LCMCT possède les pires résultats. De plus si l'on regarde de plus près ces Fusion et ARUIR qui sont les meilleurs, mais avec les résultats du tableau 20 on peut lire que c'est Fusion

qui obtient les meilleurs résultats. SSIM obtient de bons résultats avec une valeur élevée. Sur la figure 5 on peut voir que Fusion est le plus efficace. Comme pour IFC les résultats sur la figure 5 sont dur à lire. Si l'on regarde les valeurs dans le tableau 21 en moyenne c'est ARUIR qui est le meilleur.

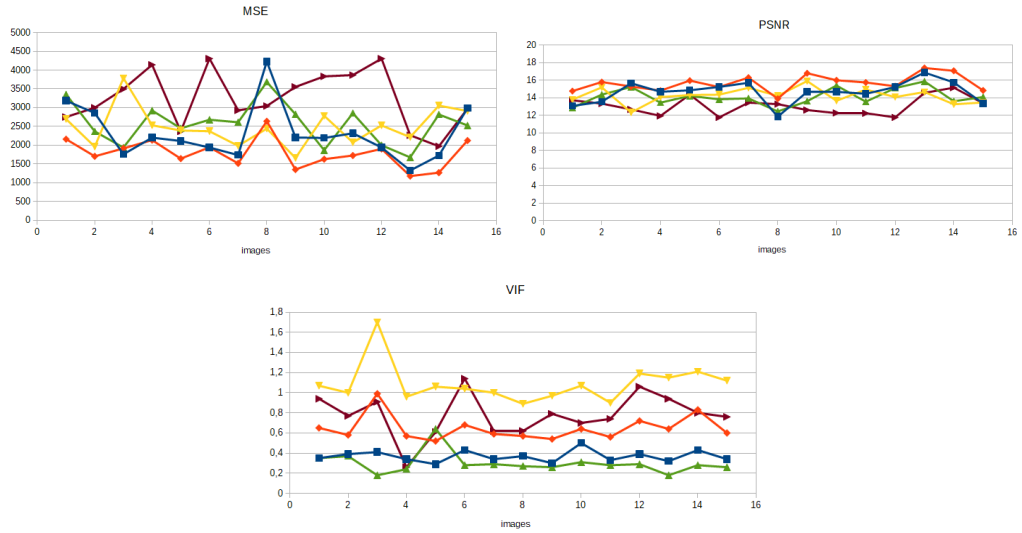

Figure 4: Résultats critères "Full reference" partie 1

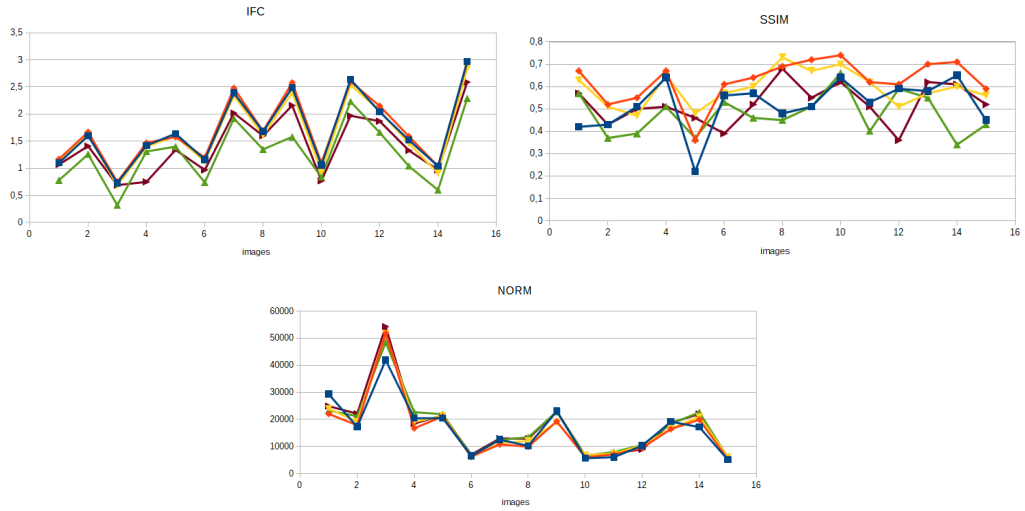

Figure 5: Résultats critères "Full reference" partie 2

3.4.2 *No reference*

Pour BRISQUE, l'algorithme qui a les valeurs les plus basses a les meilleurs résultats. En bleu Automatic red-channel underwater image restoration est la courbe la plus basse, sur ce critère c'est le meilleur. NIQE comme pour BRISQUE a de meilleurs résultats si les valeurs sont basses. On

s'aperçoit que la courbe de Automatic red-channel underwater image restoration est encore une fois la plus basse, c'est donc cet algorithme qui a le meilleur résultat. UIQM est bon lorsque sa valeur est haute, ainsi si l'on regarde la figure 6 LCMCT en vert à en moyenne la courbe la plus élevée.

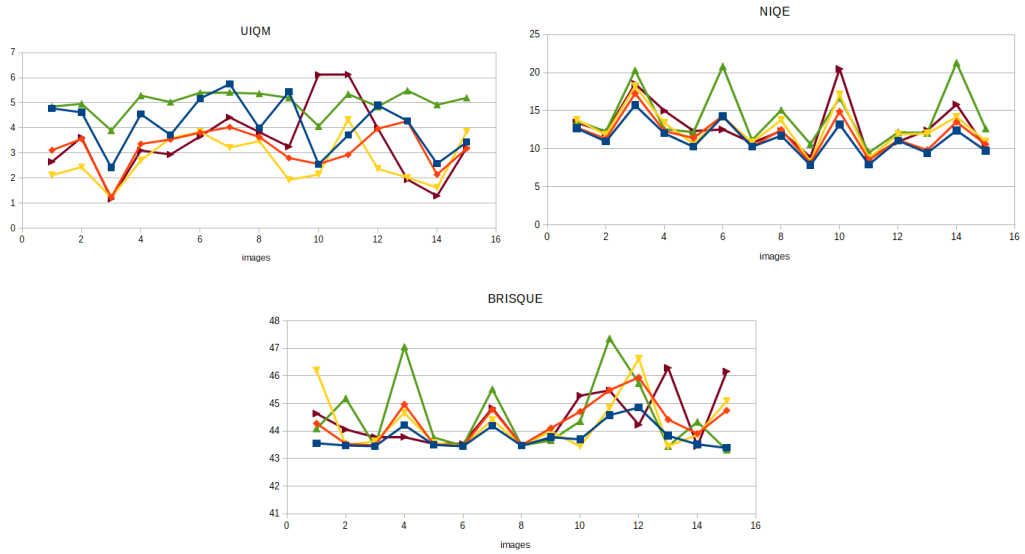

Figure 6: Résultats critères "No reference"

3.4.3 Moyenne des critères

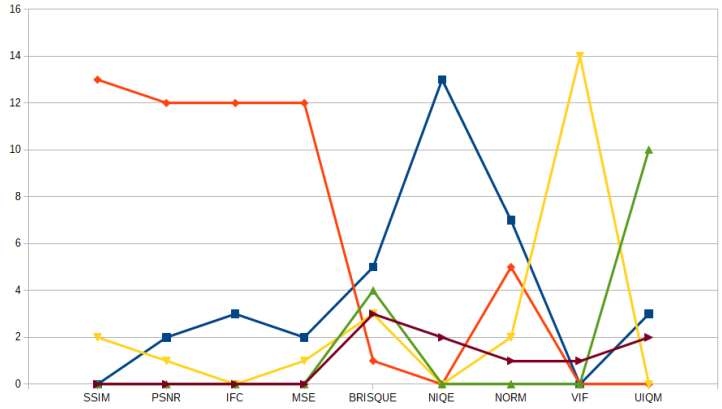

Figure 7: Résultats des critères en moyenne

Les premières mesures des critères se révèlent assez conséquentes comme le montre le tableau 18 de ARUIR. Dans un souci de simplification des données nous avons utilisé un système de score. Un algorithme qui obtient le meilleur résultat sur une image pour un critère reçoit un point, ainsi nous obtenons un nouveau tableau 19. De plus, l'évaluation des critères un par un est longue et fastidieuse. Afin de rendre tout cela plus simple et clair nous avons réuni toutes ces valeurs sur un

graphe sur la figure 7. L'ordonnée correspond au système de score et l'abscisse aux critères. On a pu déjà voir que Fusion et ARUIR ont de bons résultats sur différents critères. Maintenant la figure 7 nous présente plus largement ces résultats. Fusion est le meilleur pour SSIM, PSNR, IFC et MSE. D'autre part ARUIR obtient le meilleur score sur les critères BRISQUE, NIQE et sur la distance euclidienne. Cependant pour les deux derniers critères c'est BS qui a la meilleure qualité d'image selon le critère VIF et LCMCT possède le plus haut score pour UIQM. L'évaluation de ces algorithmes avec les critères met en lumière 2 algorithmes Fusion et ARUIR avec un léger avantage pour le premier.

Chapitre 4

Optimisation

La seconde phase de notre approche est de tester la possible application temps réel d'un algorithme. Celui-ci est choisi par sa qualité d'image donnée par l'évaluation des critères effectués dans la partie 3.4.3. Arriver à ce stade du stage, sachant le temps restant imparti, nous avons vérifié uniquement le temps d'exécution des algorithmes et travaillé sur leur optimisation. L'optimisation s'est faite différemment entre moi et mon collègue. Nous avons chacun aborder une méthode, dans ce cas présent c'est la réduction et adaptation du code. Afin de tester si un algorithme peut être appliqué dans un contexte temps réel, nous testons l'optimisation du programme LCMCT par réduction de code. Ensuite, nous allons vérifier si le programme peut être "standalone", c'est-à-dire autonome sans implication de l'utilisateur, notamment pour pouvoir le faire fonctionner sur un véhicule sous-marin. La vérification sert surtout pour voir comment rendre un programme "standalone" avec Matlab.

4.1 Réduction et adaptation du code

4.1.1 Suppression de l'interface utilisateur

Sachant que le programme doit être "standalone" nous n'avons pas besoin de l'interface utilisateur, cela signifie qu'il faut enlever toute cette partie du code sans en modifier le bon fonctionnement. La partie interface est écrite dans le programme principal "correction_gui.m" avec comme fonction principale "correction_gui_OpeningFcn()", voir la figure 17. Néanmoins, il y aussi toutes les fonctions liées aux boutons de l'interface qui y sont aussi présents. Le nettoyage de l'interface se débute en ciblant les zones et fonctions à garder. Dans notre cas il y en a 4 :

- L'image à traiter et de référence

La gestion de ces deux images par l'interface se fait comme avec la fonction décrite dans la figure 8. Dans cette fonction la chose qui nous intéresse c'est de savoir comment est enregistrée l'image, ici c'est grâce à la fonction "imread" de Matlab. On supprime ainsi toute cette fonction et on garde juste ce "imread" pour charger les 2 images.

```
function load_input_im_Callback(hObject, eventdata, handles)
    global input_im_path
    [im_path,full_path] = uigetfile('*.jpg;*.png;','Image file (*.jpg,*.png)',
                                    fullfile(handles.basedir,'input_images'));
    if (im_path==0)
        return
    end
    full_im_path=fullfile(full_path,im_path);
    input_im_path = full_path;
    input_im=imread(full_im_path);
    axes(handles.input);
    imshow(input_im);
    handles.input_im = input_im;
    guidata(hObject, handles);
```

Figure 8: Code de la fonction de chargement de l'image d'entrée de GUIDE

- Les correspondances de pixel

  Pour rappel, l'interface permet de charger les correspondances de pixels de 2 façons, soit on choisit nous même les pixels sur l'image ou on utilise un fichier de paramètre ".mat". Dans l'optique de faciliter l'utilisation de l'algorithme, nous avons décidé d'utiliser un fichier de paramètre. Dans le code le chargement du fichier est décrit dans la figure 9, c'est tout simplement un "load" du fichier ".mat" avec la déclaration des paramètres à l'intérieur. On va garder ce "load" et pouvoir supprimer cette fonction ainsi que les autres fonctions liées à la gestion des correspondances de pixels. Le chargement des données se fera ainsi :

$$load('pick\_match.mat','picked\_rbg','output\_rbg') \tag{1}$$

  Avec "picked_rbg" qui correspond aux couleurs de pixels sous l'eau et "output_rbg" en dehors.

- Les paramètres lambda

  Les lambdas sont gérés un peu différemment dans le code avec une structure. Cependant, les auteurs du code ont aussi mis en place une façon de les gérer avec un fichier ".mat". Par la suite nous gérons les lambdas comme pour la correspondance des pixels.

- Les fonctions de calcul de l'algorithme

  La fonction principal que l'on a déjà vu dans la partie 2.2.1 applique l'équation 1. On va tout simplement garder l'appelle de la fonction.

```
function load_picked_points_Callback(hObject, eventdata, handles)
    [mat_path,full_path] = uigetfile('*.mat;','Mat file (*.mat)',
                                     fullfile(handles.basedir,'picked_colors',
                                     'sample_1_picked_colors.mat'));
    if (mat_path==0)
        return
    end
    full_mat_path=fullfile(full_path,mat_path);
    load(full_mat_path);
    handles.picked_colors = struct('picked_rbg', picked_rbg,'output_rbg',output_rbg);
    guidata(hObject, handles);
```

Figure 9: Code de la fonction de chargement des correspondances de pixels GUIDE

Une fois les éléments importants gardés, nous enlevons aussi toutes les fonctions liées aux boutons et interface utilisateur. Avec cette sauvegarde des informations importantes s'ajoute un nettoyage complet du code avec la suppression des fonctions et des variables non utilisées.

4.1.2 Vectorisation

Le terme "Vectorisation" est employé dans Matlab pour désigner le faite de transformer une boucle, comme une boucle *for*, en une ligne. Effectivement, Matlab permet de faire cela grâce à sa gestion intelligente des matrices. La méthode est simple, s'il n'y a pas de dépendance dans la boucle on peut l'écrire de manière linéaire. Pour illustrer ça, on peut regarder l'exemple sur la figure 10.

```
i = 0;                  |       t = 0:10;
for t = 0:10            |       y = sin(t);
    i = i + 1;          |
    y(i) = sin(t);      |
end                     |
```

Figure 10: Exemple de "vectorisation"

Sur la gauche le code de base et sur la droite la "vectorisation", on s'aperçoit que Matlab peut gérer de manière implicite la boucle par l'absorption du i et cela à un impact conséquent sur le temps d'exécution. Pour en savoir plus sur la "vectorisation" se rendre sur le site de Matlab [7]. A cela s'ajoute une vérification des variables sur les boucles ou l'on ne peut "vectoriser". En effet, Matlab permet de gérer dynamique les variables, c'est-à-dire que la déclaration n'est pas obligatoire car il le fait implicitement. Cependant, si l'on ne déclare pas les variables utilisées dans une boucle, celleci se voit augmenter son temps d'exécution.

Maintenant On a un code réduit et compact, avec seulement les fonctions et variables nécessaires pour le fonctionnement de l'algorithme.

4.2 Application Standalone

Un traitement sur une cible temps réel et qui plus est sous l'eau implique que cela soit autonome, autrement dit "standalone". Avant de vérifier les mesures des temps d'exécution des algorithmes il va de soi qu'il faut vérifier s'ils peuvent déjà être "standalone". Matlab permet de répondre à cette problématique grâce à un outil qui s'appelle "Matlab Compiler". Il permet de directement à partir de l'application Matlab de télécharger le code en version "standalone". Cependant avant de lancer cet outil, il convient de se demander comment est-ce qu'on veut que le programme fonctionne. C'est-à-dire, est-ce qu'on donne en argument les images dont à besoin le programme ou alors c'est déjà écrit en dur dans le code l'endroit où il va les lire. Sachant que l'on va devoir utiliser un flux vidéo, il va falloir appliquer l'algorithme dans une boucle pour chaque frame de la vidéo. Il sera donc plus simple d'avoir une fonction qui prend les images en arguments.

Pour cela il faut changer notre code "correction_gui.m" pour que ça devienne une fonction avec 2 arguments d'entrées. Ce changement s'effectue juste par la modification du "load" de l'image. On enlève ce dernier et on récupère juste les 2 arguments d'entrées. Effectivement le "load" ne se fera plus dans cette fonction mais en amont dans le programme qui va appeler cette fonction. Le programme devient alors très épuré et court comme le montre la figure 16.

4.3 Temps d'exécution

L'application d'un algorithme sur une cible temps réel implique certaines contraintes. Notamment en matière de temps d'exécution. Le programme doit être capable de traiter les images assez rapidement pour que le drone sous-marin puisse utiliser les résultats et que les résultats soient précis. Dans cette optique nous avons fait 2 expériences, la première qui évalue le temps d'exécution de chaque algorithme et la seconde qui refait les mêmes mesures mais avec l'optimisation de l'algorithme qui a eu les meilleurs résultats lors de la comparaison. La méthode d'optimisation appliquée lors de cette expérience est celle faite par mon collègue. Cette approche utilise le traitement vidéo, dans on emploie le parallélisme des boucles de traitements. De plus elle vise les fonctions Matlab prenant le plus de temps d'exécution et les remplace par des appelles de fonctions C ou des librairies Matlab, toujours dans l'optique d'améliorer le temps de traitement.

4.3.1 Première expérience

Le premier test effectué est la mesure du temps d'exécution de chaque algorithme. Veuillez noter que la version de LCMCT utilisé est celle qui a été optimisée avec la méthode décrite en 4.1.Les

mesures ont été réalisées sur le matériel suivant :

OS Ubuntu 18.04, Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 16 GB of RAM

C'est le même matériel qui a été employé pour la seconde expérience. Les résultats obtenus sont sous forme de tableaux 22, déjà on peut s'apercevoir d'une chose c'est que pour HAZELINES il y a un grand fossé entre ses valeurs et les autres algorithmes. Il est environ 20 fois plus long que les autres. Même après optimisation cela ne permettrait pas de se rapprocher des autres valeurs. C'est pour cela que nous avons décidé d'enlever HAZELINES des 2 expériences. De plus on s'aperçoit aussi que ARUIR est le plus rapide des algorithmes avec en moyenne 1 seconde par image. On va utiliser cet algorithme comme base de référence, sachant que ce n'est pas l'algorithme avec les meilleurs résultats lors de l'évaluation.

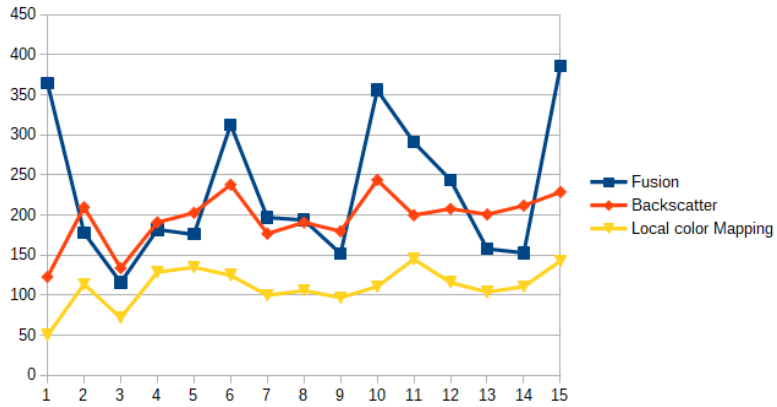

Figure 11: Graphique représentant le pourcentage de perte

Maintenant On regarde ça sous forme de graphe avec cette fois ci ARUIR comme base de référence décrite sur la figure 11. En ordonnée se trouve le pourcentage de temps en plus comparer à la base de référence ARUIR et en abscisse toujours les images. Ce graphe révèle 2 choses, LCMCT est le plus efficace avec en moyenne 50% de temps en plus que la référence, mais aussi que fusion a les pires mesures de temps d'exécution avec en moyenne 217% de perte.

4.3.2 Seconde expérience

La comparaison d'algorithmes avec les critères d'évaluation nous a montré que l'algorithme de Fusion avait les meilleurs résultats en matière de qualité d'image. Dans cette seconde expérience nous remesurons les temps d'exécution mais avec Fusion optimisé. Les nouvelles valeurs obtenues, écrites dans le tableau 23, affiche déjà de bien meilleurs résultats. L'optimisation de Fusion a permis de gagner grandement en efficacité. Avant l'optimisation, il traitait une image en 3 secondes en moyenne, maintenant il le traite en 2 secondes, avec un gain général de 50%. Si l'on regarde le graphe sur la figure 12 on peut voir que son gain se confirme, sa courbe est maintenant la plus

basse.

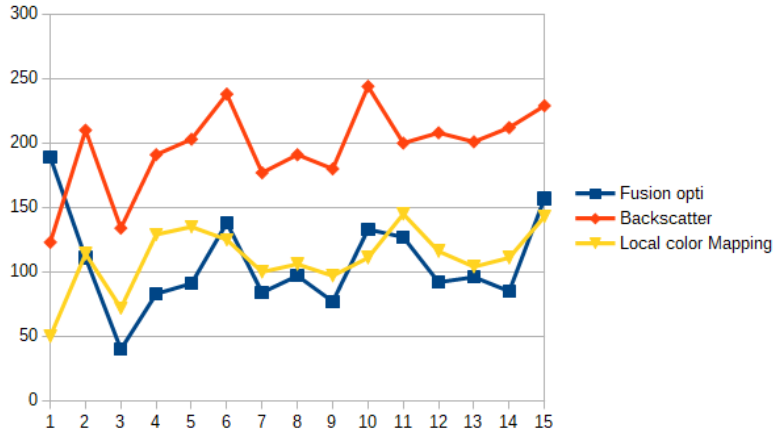

Figure 12: Graphique représentant le pourcentage de perte après optimisation

Chapitre 5

Conclusion

5.1 Analyse et interprétation des résultats

La comparaison d'algorithmes avec les critères d'évaluation a permis de mettre en lumière 2 méthodes Fusion et ARUIR, avec de meilleures performances pour le premier. La mesure des temps d'exécution a permis de voir qu'en moyenne ils étaient assez longs. Cependant, on a pu noter 3 choses, Tout d'abord que ARUIR possède les meilleurs temps d'exécution avec en moyenne 1 seconde de traitement par image. Ensuite, HAZELINES est si long que même avec une optimisation il ne serait pas envisageable de l'appliquer sur une cible temps réel et enfin que notre méthode d'optimisation sur Fusion lui a permis de gagner 50% de temps en moyenne. Malheureusement cela ne permet pas d'envisager de l'appliquer sur le drone. En effet, avec 2 secondes de traitement par image et sachant qu'une vidéo comporte 30 frames pour 1 seconde, le traitement durerait 1 minute pour 1 seconde. Même en réduisant le nombre de frame ça ne serait toujours pas envisageable.

5.2 Problèmes rencontré et Améliorations

L'une des complexités de ce stage et de cette recherche et le travail sous Matlab. Durant notre cursus scolaire nous avons été amenés à travailler avec différents environnements mais c'était la première fois avec Matlab. Il est habituel de travailler avec le langage C lorsqu'on parle de temps réel ou de systèmes embarqués et c'est notamment pour cela qu'on a essayé de convertir le code Matlab en code C. Malheureusement, après plusieurs semaines à essayer de le faire, le constat était que le stage ne durait pas assez longtemps pour aller plus loin sur cette piste, de ce fait nous l'avons laissé de coter. Mais il est important de noter que Matlab à un outil qui permet de générer le code en C.

Le choix des algorithmes et des critères n'a pas été simple, la disponibilité des codes sources n'étant pas automatique pour chaque algorithme (par exemple Sea-Thru [4]) et critères. De plus, le code fourni par certains critères comme UCIQE ne fonctionnait pas ou mal avec les algorithmes.
La suite de cette recherche serait de tester directement les algorithmes sur le rover sous-marin et de faire les mesures. De plus, on pourrait étoffer la liste de critères et d'algorithmes de traitement d'images. Avec ce plus large spectre, on pourrait évaluer de manière plus précise les méthodes et obtenir des résultats plus intéressants en matière de temps d'exécution. L'optimisation proposée ici est une parmi beaucoup de possibilités et dans un environnement spécifique. Il possible d'envisager d'autres moyens d'optimiser comme la génération du code en C avec Matlab ou la réécriture du code directement, mais cela demanderait un temps de travail bien plus conséquent.

5.3 Conclusion du stage

L'approche que l'on a mise en place, nous a permis de répondre à nos objectifs. Nous avons réussi à améliorer la qualité des images sous l'eau grâce à des algorithmes de traitement d'images et vu également la possibilité d'application temps réel par notre méthode d'optimisation. Cependant, On a rencontré différents obstacles durant ce stage, comme la complexité liée a la génération du code en C ou encore des programmes non ou peu fonctionnels.
D'un point de vue personnel, ce stage m'a beaucoup plu, il m'a permis de développer et d'acquérir de nouvelles compétences informatiques mais aussi de communiquer et d'écrire en anglais. En effet, Les réunions, notes et recherches étaient uniquement en anglais. J'ai découvert un nouvel environnement informatique avec Matlab, Le fonctionnement du traitement d'images et vidéos ainsi que l'écriture d'un article. Lors du dernier mois du stage nous avons écrit un article scientifique sur cette recherche. J'ai appris à utiliser Overleaf, un éditeur de documents basé sur Latex, avec lequel j'ai écrit ce rapport. Cet article m'a permis d'apprendre à synthétiser un travail conséquent en quelques pages.
Ce stage a été pour moi très intéressant tant sur le plan professionnel que social, j'ai eu l'occasion de partager et d'échanger avec 2 stagiaires croates qui travaillaient sur un sujet différent mais appartenant au même projet Sea-eu.

Annexe A

Figures

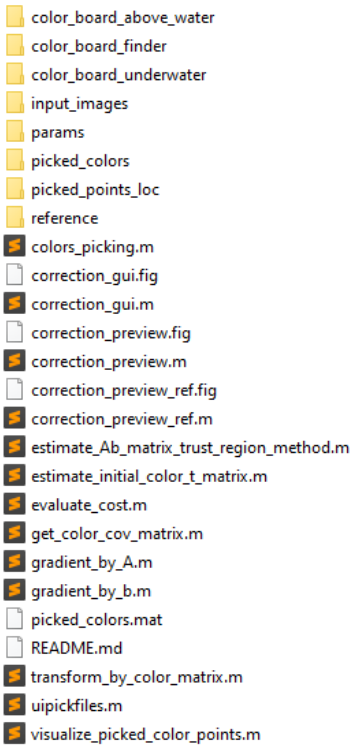

Figure 13: Arborescence du code source de LCMCT

```
%autor: Alan Le Boudec

% this programm is the same the best_ref_lcmct
% just its write like a function which return
% the best image

function out = best_ref(im,original)


    %add the lcmct function repository
    addpath('../algo/Lcmct');


    % for save result
    % psnr, norm and the id of the ref image
    psnr_best = 40;
    norm_best = 20000;
    mult_best = psnr_best * norm_best;
    id = 0;




    % compute for every reference images
    % compare their norm and their psnr
    for i=1:17
        j = num2str(i);
        ref = imread(['../algo/Lcmct/reference/',j,'.jpg']);
        replcmct = lcmct(im,ref);

        %norm
        ref1 = im2double(original);
        R1 = replcmct(:,:,1); R2 = ref1(:,:,1);
        G1 = replcmct(:,:,2); G2 = ref1(:,:,2);
        B1 = replcmct(:,:,3); B2 = ref1(:,:,3);
        s = (R1-R2).^2+(G1-G2).^2+(B1-B2).^2;
        s = s(:);
        norm_var = sqrt(sum(s));

        %psnr
        psnr_var = psnr(replcmct, ref1);

        % check if norm and psnr is the best
        % for this i multipled the norm by the psnr
        mult_var  = psnr_var * norm_var;
        if (mult_var < mult_best)
        id = i;
        bestimg = replcmct;
        psnr_best = psnr_var;
        norm_best = norm_var;
        mult_best = psnr_best * norm_best;
        end
    end

    out = bestimg;

end
```

Figure 14: Fonction de recherche de l'image de référence optimale pour LCMCT

Figure 15: légende pour les graphes

```
function out = lcmct(im,ref)
%%FUNCTION LCMCT
    %RGBtoLAB
        global LABcolorTransform;
        global RGBcolorTransform;
        LABcolorTransform = makecform('srgb2lab');
        RGBcolorTransform = makecform('lab2srgb');


    %Load images(input and ref)
        input_im = im;
        ref_im=ref; % the old version make a imread
%         ref_im_lab = applycform(im2double(ref_im), LABcolorTransform);
%         input_im_lab = applycform(im2double(input_im), LABcolorTransform);
        ref_im_lab = colorspace('Lab<-RGB', im2double(ref_im));
        input_im_lab = colorspace('Lab<-RGB', im2double(input_im));

    %load picked color data and params
        s1 = coder.load('pick_match.mat','picked_rbg','output_rbg');
        s2 = coder.load('params.mat','lambds','w2');


    %function calling
        [A,B]=estimate_Ab_matrix_trust_region_method(...);
        AB=[A B];
        result_image=transform_by_color_matrix(AB,input_im);

    %result
        out = result_image;

end
```

Figure 16: Transformation du programme LCMCT en fonction

```
% --- Executes just before correction_gui is made visible.
function correction_gui_OpeningFcn(hObject, eventdata, handles, varargin)
    % Choose default command line output for correction_gui
    handles.output = hObject;
    clc;
    warning('off','all');
    handles.basedir = pwd;
    % Update handles structure
    guidata(hObject, handles);
    global LABcolorTransform;
    global RGBcolorTransform;
    LABcolorTransform = makecform('srgb2lab');
    RGBcolorTransform = makecform('lab2srgb');
    input_im=imread(fullfile(handles.basedir,'input_images','test_alan.jpg'));
    handles.input_im = input_im;
    axes(handles.input);
    imshow(input_im);
    ref_im=imread(fullfile(handles.basedir,'reference','2.jpg'));
    handles.ref_im = ref_im;
    axes(handles.reference);
    imshow(ref_im);
    load(fullfile(handles.basedir,'picked_colors','sample_4_picked_colors.mat'))
    load(fullfile(handles.basedir,'params','sample_4_params.mat'))
    handles.picked_colors = struct('picked_rbg', picked_rbg,'output_rbg',output_rbg);
    stt1 = sprintf('%1.3f',lambds(1));
    stt2 = sprintf('%1.3f',lambds(2));
    stt3 = sprintf('%1.3f',lambds(3));
    set(handles.lambda1, 'string',stt1);
    set(handles.lambda2, 'string',stt2);
    set(handles.lambda3, 'string',stt3);
    % Update handles structure
    guidata(hObject, handles);
```

Figure 17: Code de l'interface utilisateur de LCMCT

Annexe B

tableaux

Automatic red channel										
		MSE	SSIM	NORM	PSNR	VIF	IFC	BRISQUE	NIQE	UIQM	Time
	image1	3186	0,42	29387	13,09	0,35	1,1	43,5698	12,68	4,78	2,38
	image2	2860	0,43	17335	13,56	0,39	1,6	43,4807	10,95	4,62	0,81
	image3										
	image4	1759	0,51	41961	15,67	0,41	0,73	43,4583	15,72	2,42	7,25
	image5	2202	0,64	20466	14,7	0,34	1,42	44,2233	12,03	4,55	1,01
	image6	2114	0,22	20514	14,87	0,29	1,64	43,5089	10,24	3,72	0,79
	image7	1940	0,56	6574	15,25	0,43	1,16	43,4593	14,24	5,17	0,08
	image8										
	image9										
	image10	1738	0,57	12543	15,72	0,34	2,39	44,1915	10,25	5,75	0,31
	image11	4231	0,48	10263	11,86	0,37	1,68	43,4796	11,71	3,99	0,32
	image12	2206	0,51	23111	14,69	0,3	2,49	43,786	7,84	5,44	0,96
	image13	2194	0,64	5700	14,71	0,5	1,06	43,7044	13,18	2,55	0,09
	image14										
	image15	2319	0,53	6100	14,47	0,33	2,64	44,5873	7,92	3,71	0,11
	image16	1941	0,59	10382	15,24	0,39	2,04	44,8677	11,07	4,9	0,25
	image17	1328	0,58	19188	16,89	0,32	1,52	43,8316	9,45	4,28	0,69
	image18										
	image19	1725	0,65	17204	15,76	0,43	1,041	43,5219	12,4	2,57	0,94
	image20	2990	0,45	5159	13,37	0,34	2,97	43,3989	9,72	3,43	0,07

Figure 18: Tableau des résultats de la qualité d'image pour le critère ARUIR

	MSE	SSIM	NORM	PSNR	VIF	IFC	BRISQUE	NIQE	UIQM
Automatic red channel	2	0	7	2	0	3	5	13	3
Fusion	12	13	5	12	0	12	1	0	0
Backscatter	1	2	2	1	14	0	3	0	0
LCMCT	0	0	0	0	0	0	4	0	10
Hazelines	0	0	1	0	1	0	3	2	2

Figure 19: Tableau des résultats de la qualité d'image avec le système de score

	aruir	fusion	bakcscatter	lcmct	hazelines
image1	1,1	1,17	1,12	0,78	1,07
image2	1,6	1,67	1,62	1,26	1,41
image3	0,73	0,75	0,68	0,32	0,69
image4	1,42	1,47	1,4	1,31	0,75
image5	1,64	1,58	1,59	1,4	1,34
image6	1,16	1,2	1,13	0,74	0,97
image7	2,39	2,48	2,34	1,92	2,02
image8	1,68	1,69	1,61	1,35	1,6
image9	2,49	2,58	2,4	1,58	2,16
image10	1,06	1,1	0,94	0,84	0,77
image11	2,64	2,6	2,54	2,23	1,97
image12	2,04	2,15	2,06	1,66	1,87
image13	1,52	1,59	1,46	1,04	1,33
image14	1,04	1,04	0,91	0,6	0,97
image15	2,97	2,96	2,84	2,29	2,59

Figure 20: Tableau des résultats de la qualité d'image pour le critère IFC

	aruir	fusion	bakcscatter	lcmct	hazelines
fish1	29387	22095	24007	23286	24947
fish2	17335	17881	19215	21254	22137
fish3	41961	51871	51736	48509	54317
fish4	20466	16774	19088	22719	18517
fish5	20514	21131	21389	21953	21504
fish6	6574	6282	6103	6724	7051
fish7	12543	10815	11284	12511	12998
fish8	10263	9976	12077	13442	12944
fish9	23111	19325	18992	22894	22947
fish10	5700	6064	6934	6695	6705
fish11	6100	7125	7377	8006	7574
fish12	10382	9555	10519	10612	8967
fish13	19188	16456	16559	18385	18754
fish14	17204	19974	21065	22662	22056
fish15	5159	5389	6167	5986	5875

Figure 21: Tableau des résultats de la qualité d'image pour le critère de la seconde norm

	Aruir	Fusion	Backscatter	Lcmct	Hazelines
image1	2,38	11,07	5,3	3,58	223,19
image2	0,81	2,25	2,51	1,73	125,14
image3	7,25	15,63	16,93	12,45	737,86
image4	1,01	2,85	2,94	2,31	153,89
image5	0,79	2,18	2,39	1,86	116,46
image6	0,08	0,33	0,27	0,18	12,53
image7	0,31	0,92	0,86	0,62	43,81
image8	0,32	0,94	0,93	0,66	46,92
image9	0,96	2,42	2,69	1,89	134,86
image10	0,09	0,41	0,31	0,19	13,86
image11	0,11	0,43	0,33	0,27	17,04
image12	0,25	0,86	0,77	0,54	35,19
image13	0,69	1,78	2,08	1,41	98,57
image14	0,94	2,38	2,93	1,98	138,49
image15	0,07	0,34	0,23	0,17	11,16

Figure 22: Tableau du temps d'exécution des algorithmes

Fusion opti	Fusion
6,89	11,07
1,71	2,25
10,13	15,63
1,85	2,85
1,51	2,18
0,19	0,33
0,57	0,92
0,63	0,94
1,7	2,42
0,21	0,41
0,25	0,43
0,48	0,86
1,35	1,78
1,74	2,38
0,18	0,34

Figure 23: Tableau du temps d'exécution de fusion avant et après optimisation

Bibliography

[1] Great Barrier Reef fish. https://www.newton.com.tw/wiki/.

[2] 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.

[3] 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.

[4] Akkaynak Derya. Site internet de l'algorithme "sea-thru". URL `https://www.deryaakkaynak.com/sea-thru`.

[5] 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.

[6] 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.

[7] Matlab. Informations supplémentaires sur la "vectorisation". URL `https://fr.mathworks.com/help/matlab/matlab_prog/vectorization.html`.

[8] 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.

[9] Protasiuk Rafał. code source "local color mapping combined with colortransfer for underwater image enhancem". URL `https://github.com/rprotasiuk/underwater_enhancement`.

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

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