Rapport de stage

Etude de faisabilité d'un marégraphe vidéo

Marégraphie

Traitement d'image

Python

OpenCV

Valentin Ducret

SUPINFO International University

Du 1/07 au 30/09 2015

SONEL

cnrs

Université La Rochelle

LIENSs Littoral ENvironnement et Sociétés

Rapport de stage

Développement d'un marégraphe vidéo

Stage effectué par Valentin Ducret, étudiant entrant en 3ème année à l'école d'informatique Supinfo Toulouse au sein du service SONEL (Système d'Observation du Niveau des Eaux Littorales).

mailto:valentin.ducret@gmail.com

Lieu du stage : LIENSs, Bâtiment ILE, 2 Rue Olympe de Gouges
17 000 La Rochelle.Le laboratoire LIttoral ENvironnement et Sociétés est une Unité Mixte de Recherche (UMR7266) CNRS / Université de La Rochelle.

Crédits du stage attribués par le CNRS (Centre National de la Recherche Scientifique).

Encadrant : Etienne Poirier. Ingénieur Hydrographe. +33(0)5 46 45 83 94

LIENSs, Bâtiment ILE, 2 Rue Olympe de Gouges, 17000 La Rochelle.

mailto:etienne.poirier@univ-lr.fr

Remerciements

Je tiens tout d'abord à remercier les membres de SONEL pour leur accueil, et tout particulièrement Etienne Poirier, mon encadrant, qui a été d'une grande aide pour ce projet.

Je souhaite aussi remercier Clément Mayet qui m'a apporté des pistes de réflexion concernant le traitement d'image.

Je remercie enfin Michel Ménard, enseignant chercheur au L3I à l'université de La Rochelle pour ses informations concernant le traitement d'image.

Notation

Les citations, sources ou liens vers divers sites seront situés en fin de page. Exemple :

Exemple [i]

Les photos présentes dans ce document ont été fortement compressées pour limiter le poids final en octets du rapport. Vous pouvez me contacter à mon adresse mail valentin.ducret@gmail.com si vous les voulez en taille, résolution et compression originale.

Sommaire

Introduction

Pourquoi ce stage en particulier ?

Je fais actuellement mes études dans une école d'ingénieur en informatique : Supinfo. Cette école est particulièrement axée sur l'entreprise. Le fonctionnement même de l'école est pensé pour s'apparenter à celui d'une entreprise. Il est d'ailleurs courant de suivre la formation tout en travaillant en alternance avec des entreprises d'informatique. Alors, pourquoi choisir un stage dans un domaine différent, celui de la Recherche ?

Je pense avant tout qu'il est important pour tout ingénieur d'être ouvert d'esprit. Que ce soit au niveau de son travail, de ses loisirs ou plus généralement de sa façon d'être. Ce stage sera donc pour moi un moyen de découvrir une façon de travailler, des méthodes que je ne serai surement plus amené à côtoyer dans le futur. La rigueur scientifique, les méthodes de recherches et les expériences sur le terrain ne peuvent être que très enrichissantes pour la suite de mon parcours professionnel. Vient ensuite le thème du stage : l'Océan et la marégraphie. J'ai toujours été curieux et plus généralement intéressé par les problématiques liées à la mesure du niveau de la mer. De plus, dans le contexte actuel de changement climatique, ce sera pour moi l'occasion d'en apprendre plus, de comprendre les besoins informatiques de ce domaine d'études.

Une grande partie de ce stage sera orientée vers la photographie et le traitement d'image. Le traitement d'image est une discipline récurrente dans beaucoup de projets de solutions informatiques et il est donc intéressant de le pratiquer dans un cadre professionnel. Je suis de plus sensible à l'art de la photographie que je pratique comme loisir.

Au final, je profite donc de pouvoir encore choisir des domaines variés pour mes stages afin de découvrir des façons de penser et de travailler différentes qui me seront utile, j'en suis sûr, pour le futur.

Pourquoi un marégraphe vidéo ?

Comme l'indique l'offre de stage, le but est la mise au point d'un système de marégraphe vidéo. Vous pouvez aussi trouver dans l'offre de stage (cf. Annexe) les premières caractéristiques pensées pour le marégraphe. Notamment pour la

fréquence de capture et la précision. Nous verrons donc comment le cahier des charges a évolué au fil du temps.

Dans un premier temps nous nous intéresserons aux marégraphes d'un point de vue global. Comprendre leur fonctionnement est essentiel pour penser le développement d'un nouveau système. Nous allons ensuite nous intéresser, dans la partie « Objectifs », aux raisons qui sont à l'origine d'un tel projet : pourquoi un marégraphe vidéo ? Quels seraient ses atouts par rapport aux marégraphes déjà existants ?

Enfin les travaux concernant le développement du marégraphe vidéo à proprement parler seront abordés après une présentation de l'imagerie numérique et une étude bibliographique de quelques travaux semblables à notre projet.

I. Les marégraphes existants

Etat de l'art, les marégraphes « conventionnels »

Nous allons aborder dans cette première partie les différents types de marégraphes. J'ai pu en observer plusieurs au cours de mes sorties sur le terrain. L'observatoire de l'île d'Aix possède en effet un marégraphe à capteur de pression et un marégraphe à ondes radar. J'ai pu récupérer leurs données, utiles au développement du projet. J'ai aussi, dans le cadre du stage, participé à des travaux de maintenance de l'observatoire marégraphique.

1. Le Marégraphe Côtier Numérique ou MCN du SHOM du SHOM (Service Hydrographique et Océanographique de la Marine)

Ce type de marégraphe reprend le fonctionnement d'un radar. Le principe étant d'envoyer une onde radio en direction de la mer et de mesurer le temps que l'onde met pour revenir pour en déduire la distance entre la mer et le radar.

Le SHOM assure le fonctionnement d'un large réseau de marégraphes de ce type le long du littoral Français. Celui situé sur l'embarcadère à l'île d'Aix (lieu-dit : jetée Barbotin), sur lequel nous avons effectué les opérations de maintenance (Fig.1 et Fig.2) fonctionne avec des ondes de fréquence élevée (24 à 26 GHz). Il s'agit d'un modèle Krohne OPTIWAVE 8300 C Marine. Nous avons pu récupérer les données transmises par celui-ci (Fig.4) à l'aide d'un ordinateur branché au boîtier de pilotage

Figure 3 MCN avant entretien

Figure 2 MCN après entretien

Figure 1 Centrale Marelta

du marégraphe (une centrale Marelta développée par Elta Fig.3).

Les données sont transmises en temps réel ce qui nécessite un terminal branché en permanence afin de pouvoir les obtenir. Nous récupérons ainsi une mesure par seconde. Il faut cependant noter que les données sont archivées sur internet[1], à une fréquence et une précision réduite et gratuitement accessible par tout le monde.

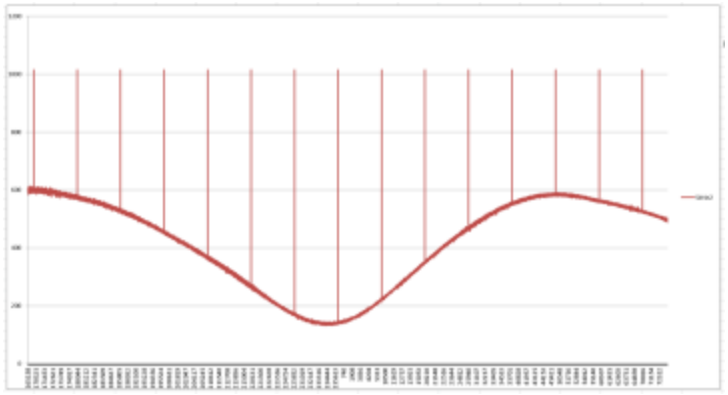

Figure 4 Exemple de donnée du MCN

Le 3 et 4 août, j'ai effectué des travaux de maintenance avec mon encadrant sur le site du marégraphe. Pour l'entretien des structures métalliques, rongées par la rouille, nous avons décapé les parties abîmées avant de les repeindre. Les conditions météorologiques et environnementales du milieu côtier nous obligent à penser à la pérennité et à l'entretien de notre futur dispositif. Si le prototype actuel est majoritairement constitué de bois et de plastique, il faudra, je pense, pour le modèle final, utiliser des matériaux résistants au milieu marin (Acier galvanisé, aluminium, acier inoxydable …)

2. Le marégraphe à capteur de pression

[1] http://data.shom.fr/#donnees/refmar/

Ce type de marégraphe utilise un capteur de pression immergé. En connaissant la pression marine « Pm », la masse volumique « p », la pression atmosphérique « Patm » et « g » l'accélération du champ de pesanteur terrestre, il est alors possible d'obtenir les variations du niveau de la mer.

Figure 5 Boitier du RBR

A noter que la masse volumique moyenne de l'eau de mer est fonction de la température de la salinité et de la pression. Pour de plus amples informations sur ce type de marégraphe, le site du SHOM, REFMAR[2] est tout indiqué.

Sur l'île d'Aix, le marégraphe à pression est situé en bout de jetée Barbotin. C'est un appareil de la marque RBR qui est alimenté pour le moment avec une série de piles mais des panneaux solaires sont prévus afin d'effectuer des mesures en continu et de limiter ses besoins de maintenance (Fig.5). Nous avons aussi pu récupérer les données lors de la sortie sur le terrain. Le marégraphe disposant d'une mémoire interne, les données sont stockées et nous avons ainsi récupéré des données commençant au 8 Avril. (Fig.6). Il faut aussi noter que le marégraphe enregistre une valeur toutes les minutes. En réalité, il moyenne des mesures faites chaque seconde. Il moyenne les valeurs mesurées de 20 secondes avant à 20 secondes après chaque minute précise pour délivrer une seule valeur moyennée.

[2] http://refmar.shom.fr/fr/documentation/instrumentation/les-maregraphes-a-capteur-de-pression

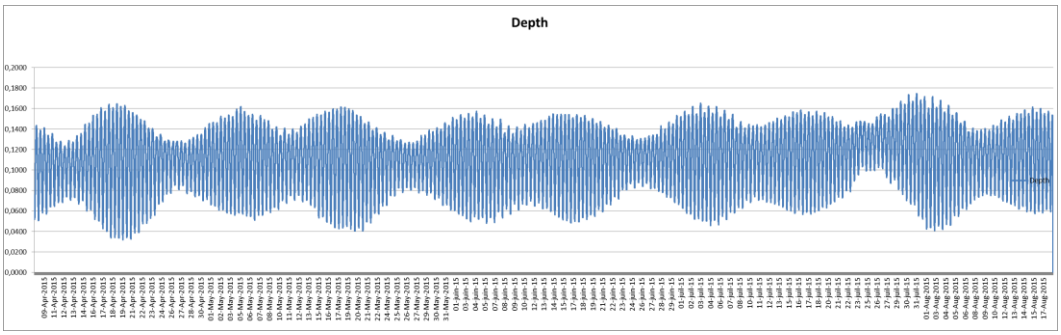

Figure 6 Exemple de donnée du RBR

3. Le test de Van de Casteele

On peut aussi mesurer le niveau de la mer par lecture directe sur une échelle à marée (Fig.8). Le test de Van de Casteele utilise ce principe. Il est utile pour calibrer et plus généralement connaitre les imprécisions et les erreurs des marégraphes utilisés. Pour faire court, l'utilisateur doit à la fois relever les données d'un marégraphe et lire l'échelle de référence. Il peut alors dessiner un diagramme en reportant le niveau de la mer sur l'axe Y et la différence entre les deux sources de données sur l'axe X. La forme de ce diagramme renseigne immédiatement sur le type d'erreur que le marégraphe peut avoir. Là où une ligne droite verticale centrée en zéro indiquera une absence d'erreur, une tache oblique indiquera une erreur d'échelle (Fig.7). Le site internet du SHOM propose un article détaillé sur cette méthode[3]. Ce test nécessite cependant une lecture à l'échelle durant un cycle de marée complet ce qui s'avère être une tache longue et pénible.

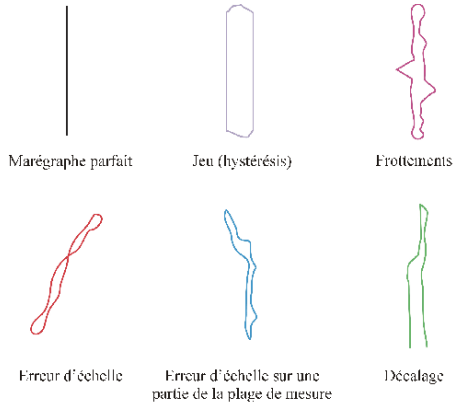

Figure 7 Diagrammes résultant du test de Van de Casteele indiquant le type d'erreur d'un marégraphe.

[3] http://www.shom.fr/les-activites/activites-scientifiques/maree-et-courants/marees/test-de-van-de-casteele/

Nous avons effectué une lecture à l'échelle au cours d'une sortie sur le terrain de façon à comprendre le travail que devra faire l'algorithme par analogie avec l'œil et le cerveau humain.

Nous avons donc effectué trois lectures sur l'échelle à marée de 5 minutes chacune. La lecture a été effectuée à deux personnes. L'une, crayon et montre en main indiquait à la seconde le « Top » de prise de mesure. L'autre, les yeux rivés à l'échelle annonçait à voix haute la valeur lue à l'échelle. L'intervalle de temps entre chaque mesure était de 10 secondes. Nous nous sommes calés sur l'heure UTC exacte à l'aide de nos portables connectés à internet. Nous étions physiquement situés à une dizaine de mètres de l'échelle à marée, dans l'axe de celle-ci ce qui nous a permis de faciliter la lecture.

Figure 8 Echelle à marée de l'île d'Aix

Ces paramètres de mesure (intervalle de temps, durée) ne sont pas les plus optimisés ni les plus efficaces mais ils m'ont permis personnellement d'avoir une idée de ce qu'est un test de Van de Casteele et du travail que le marégraphe vidéo devra effectuer. Au final, on remplace juste l'opérateur par une caméra et une unité de calcul.

Pour de plus amples informations et des exemples de test Van de Casteele complets, je vous invite à lire la présentation de Belén Martín Míguez[4]

[4] http://refmar.shom.fr/documents/10227/146468/Martin-Miguez_Journees-REFMAR-2013.pdf

II. Objectifs

Pourquoi un marégraphe vidéo ?

1. Les atouts d'un marégraphe vidéo

Les marégraphes déjà existants possèdent plusieurs atouts. Ils sont précis (au mm pour le MCN, en théorie au dixième de mm pour le RBR), possèdent une fréquence de mesure élevée et le matériel utilisé a fait ses preuves et les chercheurs le connaissent bien. Ils possèdent cependant de nombreux inconvénients. Ils sont chers à produire et nécessitent un entretien venant de personnes compétentes et spécialisées dans le domaine[5]. De plus, les sites propices à leur installation sont rares. Le prix et l'entretien sont des facteurs essentiels car ils sont souvent un frein à l'installation de nouvelles stations marégraphiques notamment dans les pays en voie de développement. Ainsi un marégraphe peu coûteux, facile à installer et à entretenir permettrait de multiplier les stations de mesures autour du globe.

Il faudra donc prendre en compte le coût du système durant sa conception. Peut-on utiliser des caméras grand public ? Quelle est la puissance nécessaire de l'unité de calcul intégrée ? Le but final étant de développer un système peu onéreux et simple permettant à un maximum de personnes de l'utiliser. Les usages d'un tel système sont alors nombreux. Chacun selon son besoin, marin pêcheur, ostréiculteur, agriculteur, océanographe ou simple citoyen pourra avoir des informations de hauteur d'eau sur le site qui l'intéresse de façon simple et robuste. On peut aussi penser à des programmes scolaires ayant pour but l'entretien et le traitement des données d'un marégraphe vidéo. On peut aussi penser à l'implémentation de marégraphes vidéo dans des pays défavorisés pour surveiller les risques d'inondations ou de tsunamis.

Enfin, là où un marégraphe à pression peut être sujet à des dérives de mesures, le marégraphe vidéo quant à lui est « calé » sur une échelle à marée qui ne bouge pas. Lui ne doit pas bouger non plus. Il n'y pas de dérive du zéro instrumental donc notre dispositif ne nécessitera pas de test de Van de Casteele pour être calibré. Il pourra même remplacer l'œil humain lors de ces tests.

[5] http://www.hydrology.bee.cornell.edu/Papers/RoyemASABE12.pdf

2. Un algorithme multi-usage

Le marégraphe vidéo repose sur le principe de traduire une image en une donnée de hauteur d'eau à l'aide d'un traitement d'image. Cet algorithme de détection de l'interface air-met est le point essentiel du projet car il est au cœur du fonctionnement du marégraphe.

Cependant, cet algorithme n'est pas lié théoriquement au type de matériel photo utilisé. Ainsi une nouvelle utilité au projet apparaît : on pourrait utiliser notre algorithme sur toute source photo ou vidéo afin d'en ressortir le niveau d'eau. Cela permettrait à n'importe quel utilisateur de prendre seulement une caméra, de filmer une échelle à marée, et une fois rentré chez lui, de traiter les images prises pour en retirer des données.

Cela implique plusieurs choses : un algorithme de traitement universel (on ne pourra pas se baser sur la forme et les motifs de l'échelle à marée par exemple), un calibrage manuel, une fonction pour unifier le format des fichiers (extension et nom d'un fichier). Il faudra aussi que le logiciel s'adapte à toutes les conditions de prises de vues différentes. L'eau n'a pas toujours la même couleur, la houle est différente, les conditions de luminosité aussi, etc. ... La non utilisation de matériel spécifique tel une bande réfléchissante, un flotteur, un panneau de calibration ou bien un type de caméra spécifique représente à la fois le grand intérêt et la difficulté du projet. Cette idée de « boite noire » de traitement s'est donc imposée au fur et à mesure du stage comme la tâche prioritaire.

Je m'oriente donc vers le codage d'un genre de « boîte noire », logiciel auquel on donnerait une vidéo et un fichier de calibration et qui serait capable de sortir des hauteurs d'eau.

Un tel choix rend donc le projet unique, différent de ce qui a été fait jusqu'à maintenant (voir partie sur les projets semblables) et souligne l'importance du logiciel. Cela a aussi soulevé des problèmes auxquels nous n'avions pas pensé au début. Nous verrons cela dans les parties suivantes.

III. Le traitement d'image

Manipuler les pixels pour transformer une image

Nous aborderons dans cette partie mes recherches concernant le traitement d'image, l'algorithme permettant de traduire une photo en une hauteur d'eau. Mais dans un premier temps, il convient de présenter dans un premier temps les bases de l'imagerie numérique.

1. Les bases

Une image numérique est une image sous forme binaire. Elle est donc composée de 0 et de 1 formant des octets. C'est donc en pratique une suite de valeurs. Cette suite de valeurs est formatée en une matrice à plusieurs dimensions. Dans le cas d'une image conventionnelle 2D, il s'agit d'une matrice à deux dimensions, un tableau où chaque case est un pixel. On peut ajouter une 3ème dimension pour représenter le temps (on parle alors de vidéo) ou une dimension spatiale (photos 3D).

Les images que nous allons traiter sont donc des matrices où chaque pixel contient des données traduisant la couleur de ce pixel. Nous allons donc nous intéresser aux différents modes de représentation des couleurs :

- L'espace colorimétrique RGB (Red Green Blue) : c'est un mélange de trois composantes, semblable à la synthèse additive des couleurs. Ainsi, les trois composantes à leur maximum donnent du blanc. Les combinaisons des 3 composantes ou canaux permettent de retranscrire une grande partie du spectre visible. C'est le mode de codage utilisé pour ce projet.
- CMJN (Cyan Magenta Jaune Noir) : se compare à la synthèse soustractive. Utilisé dans l'impression.

- TSL (Teinte Saturation Luminance) : la couleur est codée suivant le cercle des couleurs. Un canal pour l'angle sur le cercle, les deux autres valeurs pour la saturation et la luminance[6] (Fig.9).

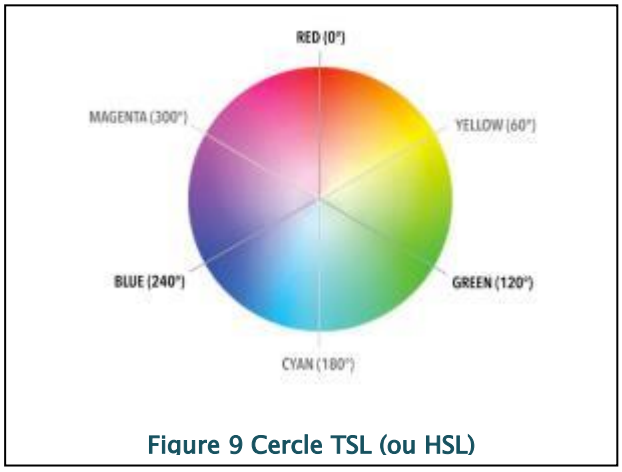

Figure 9 Cercle TSL (ou HSL)

- N&B : Les images noires et blanches ne possèdent qu'un seul canal. Une seule valeur de 0 à 255 par pixel permet de définir son niveau de gris. Nous les utiliserons beaucoup dans ce projet pour leur simplicité.

On peut ensuite choisir le nombre de bit (0 ou 1) attribué pour la valeur de couleur de chaque pixel. Il existe plusieurs modes de codage des couleurs, faisant varier le nombre de couleurs disponibles et le poids de l'image :

- Images 24 bit : Chaque pixel est composé de 3 octets, c'est à dire 3 fois 8 bits donc 24 bit. Chaque octet comprend la valeur de 0 à 255 d'une composante de couleur. Le nombre de couleurs pouvant être représenté est donc de 256x256x256 c'est-à-dire environ 16,7 millions de couleurs. C'est le mode de codage avec lequel nous allons travailler par la suite.
- Images à palette (8 bit) : Nous n'allons pas nous étendre ici sur cette méthode de codage des couleurs. Il faut juste connaître son principe. On « attache » à l'image une table de 256 couleurs tirées des images 24 bit. Cela forme une palette de couleur à laquelle on fait appel pour chaque pixel. Chaque pixel ne « pèse » alors qu'un seul octet. Le poids de l'image se trouve donc grandement diminué.

[6] http://la-cascade.io/utiliser-hsl-pour-vos-couleurs/

- Images avec canal Alpha : On peut rajouter aux pixels une gestion de la transparence avec un canal Alpha. Une valeur de 255 indiquera un pixel totalement transparent et une valeur de 0 indiquera que le pixel est opaque

Vient enfin le format de l'image à proprement parler, c'est à dire son extension. Ce principe de format permet « d'encapsuler » une image. Il va contenir des informations comme les métadonnées ou bien les informations concernant le codage, la manipulation et le décodage de l'image. Cela va s'avérer utile pour le reste du projet mais nous le verrons plus tard. Chaque logiciel va donc manipuler les images différemment en fonction du format de l'image. Certains logiciels ne vont pas pouvoir ouvrir un format particulier tandis que d'autres utiliseront un format propriétaire. Voici une liste des formats principaux et de leurs caractéristiques (tab.1) :

Tableau 1 Les différents formats d'image

	Type (matriciel/ vectoriel)	Compression des données	Nombre de couleurs supportées	Affichage progressif	Animation	Transparence
JPEG	matriciel	Oui, réglable (avec perte)	16 millions	Oui	Non	Non
JPEG2000	matriciel	Oui, avec ou sans perte	4 milliards	Oui	Oui	Oui
GIF	matriciel	Oui, Sans perte	256 maxi (palette)	Oui	Oui	Oui
PNG	matriciel	Oui, sans perte	Palettisé (256 couleurs ou moins) ou 16 millions	Oui	Non	Oui (couche Alpha)
TIFF	matriciel	Compression ou pas avec ou sans pertes	de monochrome à 16 millions	Non	Non	Oui (couche Alpha)
SVG	vectoriel	compression possible	16 millions	* ne s'applique pas *	Oui	Oui (par nature)

Vous disposez maintenant des bases de l'imagerie numérique. Nous verrons par la suite que nous travaillerons avec des images RGB 24 bits possédant le format *.png ou *.jpeg Pour information, le format de la photo n'a pas d'incidence avec les logiciels et bibliothèques utilisées.

2. Les logiciels utilisés

Nous allons aborder dans cette partie l'aspect technique du stage : les outils utilisés pour modifier les images, les logiciels d'édition et les langages de programmation.

a. Python

Langage créé en 1989 par le programmeur Guido van Rossum.

La première étape du projet a été le choix d'un langage de programmation. Et cela s'est fait assez naturellement. Il fallait choisir un langage que je connaissais déjà, qui soit suffisamment puissant pour le traitement vidéo (c'est le cas de la quasi-totalité des langages) et qui possède une librairie de traitement d'image. Nous avons donc choisi Python. Voici quelques points justifiant le choix de ce langage :

- Langage très utilisé dans les laboratoires de recherche et spécialement au LIENSs ce qui permet une relecture facile du code source afin de poursuivre le travail commencé.
- Langage haut niveau, proche de l'utilisateur pour faciliter la compréhension de personnes néophytes. Il possède une gestion de la mémoire automatique. La syntaxe est simple d'utilisation et le typage est dynamique et fort (pas besoin d'attribuer un type à une variable).
- Présence de la bibliothèque OpenCV dédiée au traitement d'image que nous détaillerons plus tard.

Mais ce langage possède aussi des inconvénients :

- Langage non compilé : les machines qui exécuteront le logiciel devront avoir un compilateur Python d'installé ce qui n'est pas le plus simple pour les personnes novices en informatique. Cela rend aussi le programme légèrement moins performant et moins rapide à exécuter.

Pour les détails techniques, j'ai choisi d'utiliser Python 2.7 (Python 3.0 est sorti et fonctionnel) afin de m'assurer de la compatibilité avec de futurs Frameworks ou bibliothèques. Enfin, il sera possible dans un futur plus ou moins lointain de créer un service Web de notre algorithme (un site internet avec le logiciel intégré, le serveur Web faisant les calculs) grâce à des Frameworks comme Django.

b. Open CV

OpenCV *(pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel.*

Les langages de programmation ne disposent pas à la base de fonctions de traitement photo ni vidéo. Pour cela, il faut utiliser une bibliothèque qui ajoute des fonctions à notre programme. Il suffit alors d'importer Open CV dans son programme (cf. Partie sur le logiciel) et installer OpenCV sur l'ordinateur faisant fonctionner le logiciel pour avoir accès à l'intégralité de ses fonctions. OpenCV est un outil très puissant et lourd utilisé dans bon nombre de projets de programmation utilisant des photos ou des vidéos (logiciels de tracking avec une webcam, de surveillance vidéo, …) aussi bien par des professionnels que par des particuliers pour leurs projets personnels.

OpenCV propose plusieurs filtres de traitement et des outils pour afficher des éléments à l'écran. OpenCV manie les images en les transformant en matrice Numpy. Numpy est une autre bibliothèque d'outils mathématiques, utilisée pour la gestion des matrices. Il est donc aussi conseillé d'utiliser Numpy pour manier directement ces matrices.

J'ai utilisé tout au long de mon stage la documentation en ligne disponible sur plusieurs sites ou blogs proposant des tutoriels et des exemples de travaux. Je conseille tout particulièrement la documentation officielle fournie avec OpenCV qui est disponible en ligne[7]. Il est aussi très intéressant de regarder et comprendre les exemples fournis avec le package[8]. Nous expliquerons dans la partie sur notre méthode de traitement d'image, les filtres et outils d'OpenCV que nous avons utilisés.

3. Les travaux existants

Nous présenterons dans cette partie, un petit état de l'art des travaux effectués par d'autres personnes, sociétés ou laboratoires qui se rapprochent de notre projet.

---

[7] https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html

[8] On les trouve dans le dossier opencv : opencv\sources\doc\py_tutorials

Même si nous n'avons rien lu sur un marégraphe vidéo aussi « universel » que le nôtre, s'inspirer de ce qui a déjà été fait est très important et instructif. Nous expliquerons par ce biais des méthodes de traitement d'image qui nous seront utiles.

a. GaugeCam : la détection des contours

GaugeCam est une société américaine formée par François Birgand, professeur à l'université de Caroline du Nord. Cette société est composée de quatre membres, dont deux ingénieurs informaticiens et un ingénieur spécialisé dans l'agriculture et la biologie. Ils proposent depuis 2012 un produit comportant des similarités avec notre projet.

Le principe de GaugeCam est identique au nôtre : monitorer et mesurer un niveau d'eau à partir d'une caméra. La spécificité de GaugeCam est le panneau qui est mis dans l'eau pour permettre la détection de la surface de l'eau et la calibration automatique des données par rapport au monde réel. L'ensemble du dispositif est expliqué très clairement sur le site internet de GaugeCam[9]. En voici une explication plus succincte.

- La caméra repère la ligne formée par l'interface air/eau sur un fond plat (le panneau avec les doubles triangles). On utilise pour cela une détection de contours verticale.

- La donnée enregistrée est alors convertie en valeur dans le monde réel grâce aux motifs disposés sur le panneau. (Fig.10 Panneau GaugeCam)

Figure 10 Panneau GaugeCam

[9] http://www.gaugecam.com/product/overview/

-Un module GPRS envoie les données sur le serveur GaugeCam où elles sont ensuite accessibles en ligne par le biais d'une application ou d'un navigateur internet.

La détection de contours

Le but de la détection de contour est, comme son nom l'indique, de repérer les contours de l'image : c'est-à-dire les changements importants et brutaux de luminosité ou de couleur entre les points de l'image. On crée ainsi une nouvelle image où seuls les contours apparaissent (Fig.11). Ils traduisent d'un changement important du monde réel : contour d'un objet, changement de perspective, d'éclairage, etc… Il existe en pratique plusieurs filtres donnant des résultats différents mais le principe reste identique. Pour chaque pixel de l'image, on regarde ses voisins et lorsque la différence de valeur dépasse une valeur choisie, on considère que c'est un contour. La relation entre les valeurs de pixels d'entrée et de sortie est donnée par la matrice du filtre. Les filtres varient donc en fonction de la matrice appliquée. Voici quelques filtres connus :

- Prewitt : Utile pour la détection de contours verticaux. Pour la matrice : h_x = [-1 0 1] et h_y est sa transposée.
- Sobel : Similaire à Prewitt mais utilise un filtre triangulaire.
- Canny : Le plus poussé. On reprend le filtre Sobel, on rajoute aussi un flou gaussien pour enlever le bruit de l'image et on prend compte en plus de la direction des contours.

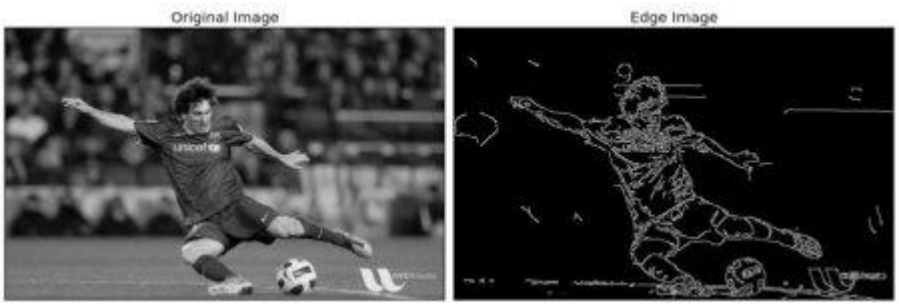

Figure 11 Filtre Canny

Ce que l'on retient pour notre projet :

GaugeCam se rapproche de ce que l'on veut accomplir avec notre projet. C'est un produit stable, précis (+ ou – 3 mm) et autonome qui a été testé et éprouvé sur de nombreuses rivières américaines pour le compte du USGS (United States Geological Survey) et même au Bangladesh. Comme nous le verrons plus tard notre prototype ressemble beaucoup à la caméra de GaugeCam. Cependant GaugeCam utilise du matériel supplémentaire intrusif (le panneau manque de discrétion) et ne marche que dans certaines conditions (marécages, mer calme, marnage faible). Il représente donc une bonne source d'inspiration pour notre projet mais ne correspond pas à notre besoin d'où la nécessité d'inventer autre chose.

b. Mashiro IWAHASHI et Sakol UDOMSIRI : Addition et différence.

Les travaux de ces deux chercheurs japonais portent aussi sur la détection de la surface de l'eau, dans le but de surveiller la hauteur d'eau des rivières. Il ne s'agit pas ici de détecter directement l'interface air/mer traduisant directement du niveau d'eau mais plutôt de pouvoir séparer les deux milieux. Cette nuance est importante car elle implique une vision différente en termes de traitement d'image. On va se concentrer ici sur une différence de couleurs, de texture plutôt que sur une limite pareille à un trait sur l'image. Dans leur publication[10], ils présentent deux méthodes :

- Détection de contours puis différence de deux images
- Addition des images, détection de contour puis calcul de la variance

C'est l'occasion pour nous d'introduire deux techniques liées pour faire du traitement d'image :

L'addition d'images

Nous avons vu dans les bases de l'imagerie numérique que les images sont constituées de pixels. L'addition des images va consister très simplement à additionner les valeurs des pixels de deux images pour chaque canal R, G puis B ou

10 http://www.researchgate.net/publication/4276164_Water_Level_Detection_from_Video_with_Fir_Filtering

seulement pour un canal si on est en niveaux de gris. Cela va donc produire une autre image, somme des deux premières, où pour chaque pixel :

$Px_{final} = Px_{image1} + Px_{image2}$

Cette technique n'apporte pas grand-chose en l'état. Mais elle devient intéressante Quand on applique un coefficient à chaque membre de l'addition :

$Px_{final} = \alpha * Px_{image1} + \beta * Px_{image2}$ où $(\alpha + \beta) = 1$

Cela permet tout simplement de fusionner les deux images. Ainsi l'image finale va être une composition des deux premières images (Fig.12)

L'intérêt principal d'une telle méthode dans le cadre de notre projet est la suppression des éléments perturbateurs se situant potentiellement sur le trajet optique de la caméra. Si l'on prend une vidéo à 30 images par secondes et que l'on additionne ces 30 images ou « frames », les éventuels oiseaux, la pluie ou la neige seront nettement moins visibles car les images dans lesquelles ils apparaissent seront « fondues » avec les 29 autres.

Figure 12 Addition d'images

La différence d'images

Comme son nom l'indique, cette différence est le contraire de l'addition que nous avons vu précédemment :

$Px_{final} = Px_{image1} - Px_{image2}$

Là où l'addition fait disparaître les éléments mouvants d'une suite de photo, la différence les fait ressortir. Ainsi, les pixels identiques sur les images soustraites seront noirs et les pixels dont la valeur a changé apparaîtront gris ou blanc suivant leur degré de changement. (Fig.13). Cela permet donc de faire disparaître le « background » c'est-à-dire le fond des images.

Figure 13 Différence

Ce que l'on retient pour notre projet :

Les méthodes présentées par ces chercheurs sont extrêmement intéressantes. L'addition et la soustraction peuvent être utilisées en plus d'autres méthodes et n'apportent que des avantages. Même si l'on ne peut pas utiliser les deux méthodes à la fois, la soustraction de deux images successives à l'avantage d'être indépendante des conditions de luminosité et de météo. De plus les tests effectués donnaient des résultats très différents suivant le matériel utilisé.

c. A.A Royem et al : ROI et couleurs

A. Royem, C. K. Mui, D. R. Fuka et M. T. Walter proposent une autre approche du problème dans leur publication de 2012. Le but est ici de détacher l'échelle à marée du reste de l'image. Ils utilisent donc une règle d'une couleur qui tranche avec le reste de l'image positionnée à côté d'une échelle à marée classique. L'utilisateur, sélectionne une partie de l'image correspondant à la règle et sauvegarde la couleur de la zone afin de pouvoir séparer la règle du reste de l'image : c'est une ROI (« Region Of Interest »).

ROI et Couleurs

Une ROI ou « Region Of Interest » est tout simplement une partie de l'image. Elle est définie par deux points, le coin supérieur gauche et le coin inférieur droit. On peut cependant grâce à des opérations de translation et de rotation lui faire prendre la forme que l'on veut. On l'utilise dans ce cas pour sélectionner une partie caractéristique de la photo. Cela nous permet de connaitre la couleur moyenne de cette ROI. Une plage de valeurs autour de cette couleur est alors définie. On va ensuite, à l'aide d'une double boucle (cf. partie sur l'algorithme), parcourir l'image pixel par pixel et voir si chaque pixel appartient à la plage de couleur définie. Cela permet donc de faire ressortir ou disparaitre les parties de la photo ayant une couleur semblable à celle de la ROI. Dans le cas de A.Royem et al. , la ROI est définie sur la règle (Fig.14). On peut voir sur la photo (a) la ROI définie par les losanges bleus en haut de l'échelle. La différence entre les pixels de la photo et ceux de la ROI est alors affichée sur la photo de droite (b).

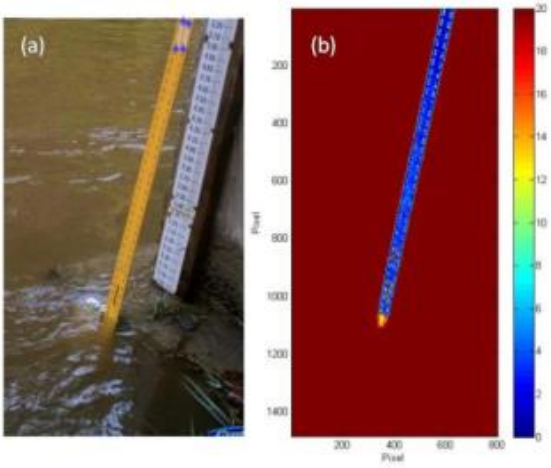

Figure 14 Exemple de ROI

Ce que l'on retient pour notre projet :

Une méthode basée sur les couleurs possède ses avantages et ses

Figure 15 Test d'une ROI sur l'eau

inconvénients. Elle est tout d'abord très sensible aux conditions de luminosité. Si un rayon de soleil apparaît sur l'échelle, la détection est fortement perturbée. De plus cette technique requiert une règle de couleur spéciale, se détachant du reste, ce qui implique du matériel supplémentaire peu discret, contraire aux spécificités de notre projet. Sur le site de l'île d'Aix, nous ne pouvons pas exploiter cette méthode car l'échelle à marée possède des couleurs multiples. Cependant en observant ci-dessous la photo de droite, on remarque que la mer est d'une couleur relativement unie. Partant de cette analyse, nous avons voulu tester la méthode de Royem et al. Mais en faisant une recherche de ROI sur la mer plutôt que sur une règle. Les tests de cette méthode ne se sont pas montrés concluants (Fig.15).

4. Notre choix

Le choix d'une méthode pour notre projet ne s'est pas décidé en une seule fois. Il a été repensé en fonction des images ramenées du terrain et des résultats des différents tests. Il faut rappeler que la spécificité de notre projet réside dans ses conditions de fonctionnement élargies. Le système doit fonctionner dans toutes les conditions météo, et ce dans un maximum d'endroits différents. Il faut donc un traitement d'image le plus global possible.

Figure 16 Exemple de différence de couleurs sur une même scène

Ce premier élément implique déjà une conséquence : Les méthodes se basant sur la couleur seront très difficiles à implémenter. En effet la couleur de l'eau est changeante en fonction de la météo, de la position géographique, du type de site et

de la caméra utilisée (Fig.16). Les échelles à marée ne sont pas toutes identiques. Il faudra plutôt s'orienter vers une méthode se basant sur le mouvement ou la texture.

Un deuxième élément est à prendre en compte. Nous travaillons dans des conditions de capture non contrôlées car la mer peut être tout à fait calme comme très agitée. Notre algorithme de traitement doit marcher dans un maximum de conditions possibles. Cela rend la simple détection de contour plus qu'approximative : un contour non prévu pourrait facilement apparaître sur notre image (Fig.17).

Au final, il est apparu qu'il n'existait pas de solution miracle, marchant en permanence tout en restant précise. Il faudrait donc s'orienter vers une méthode la plus générale possible et travailler durant le traitement avec les différents paramètres des filtres pour s'ajuster aux conditions diverses. C'est là où réside la difficulté du projet : la gestion des erreurs et des paramètres de traitement. Il faudra donc effectuer plusieurs tests et prétraitements afin de définir une valeur pour le seuil, un taux de flou à appliquer, des plages de mesures, … Il faudra ensuite vérifier les valeurs par post traitement pour voir si elles sont représentatives et sinon, relancer l'algorithme en changeant les paramètres.

Nous allons désormais expliquer les différentes étapes de notre algorithme de traitement.

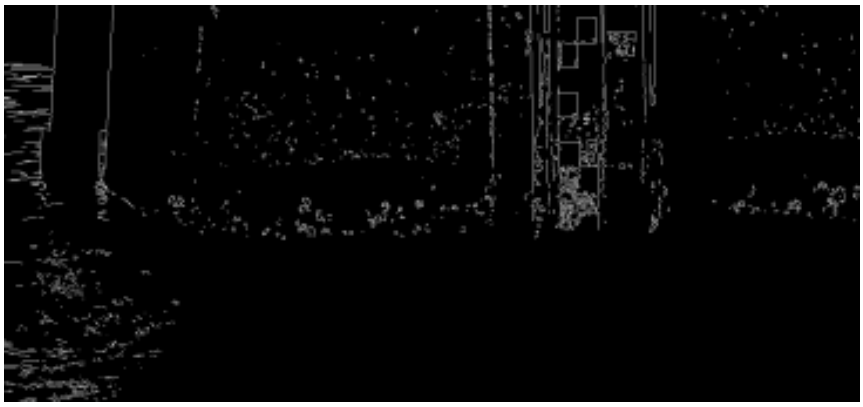

Figure 17 Exemple illustrant la difficulté de la détection de contour dans des conditions changeantes.

a. Le prétraitement.

Avant d'appliquer des filtres et des algorithmes sur nos images, il est essentiel de les formater afin d'optimiser le traitement et de s'assurer que les images à traiter seront les bonnes. Pour cela deux fonctions sont utilisées :

- Le recadrage. Une fonction pour recadrer les photos a été créée (Fig.18). L'utilisateur clique dans le coin en haut à gauche et glisse le pointeur jusqu'au point en bas à droite du nouveau cadre. Le programme garde alors

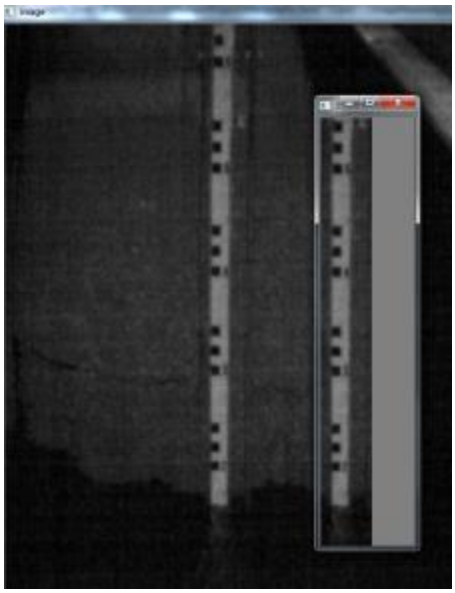

Figure 18 Fonction de recadrage

les coordonnées de ces deux points et recadrera automatiquement les photos à traiter (cf. Chapitre IV sur le code). Cela permet à la fois d'alléger le traitement (moins de pixels à traiter) et d'éviter la détection de pixels se situant hors du cadre de l'échelle à marée

- La rotation. On utilise ici une fonction déjà présente dans Open CV. L'utilisateur choisit un angle de rotation pour que l'échelle à marée soit la plus verticale possible. Encore une fois, on sauvegarde cet angle et on applique la rotation à toutes les photos.

- Le flou. On applique ensuite un filtre de flou aux photos. Cela a pour but de réduire le bruit et les artefacts inhérents à la photo numérique. Le principe

étant d'appliquer à chaque pixel de l'image une nouvelle valeur en fonction des pixels voisins. Pour un « averaging » ou filtre de moyennage, on utilise la matrice suivante :

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

La taille de la matrice est un paramètre d'entrée du filtre. Ici la matrice est de taille 3 mais on peut utiliser toute matrice de taille impaire. K est la nouvelle valeur du pixel. Dans notre cas spécifique, de limiter la perte de précision, nous utilisons un flou gaussien. La formule change mais le principe reste identique. Pour de plus amples informations je vous invite à lire la page Wikipédia expliquant plus précisément le filtre[11].

b. La différence

Une fois ce prétraitement effectué, on va différencier un couple d'images successives. Comme expliqué précédemment, cela va nous permettre de faire ressortir les éléments différents entre les deux images et la différence de niveau d'eau qui nous intéresse (Fig.19).

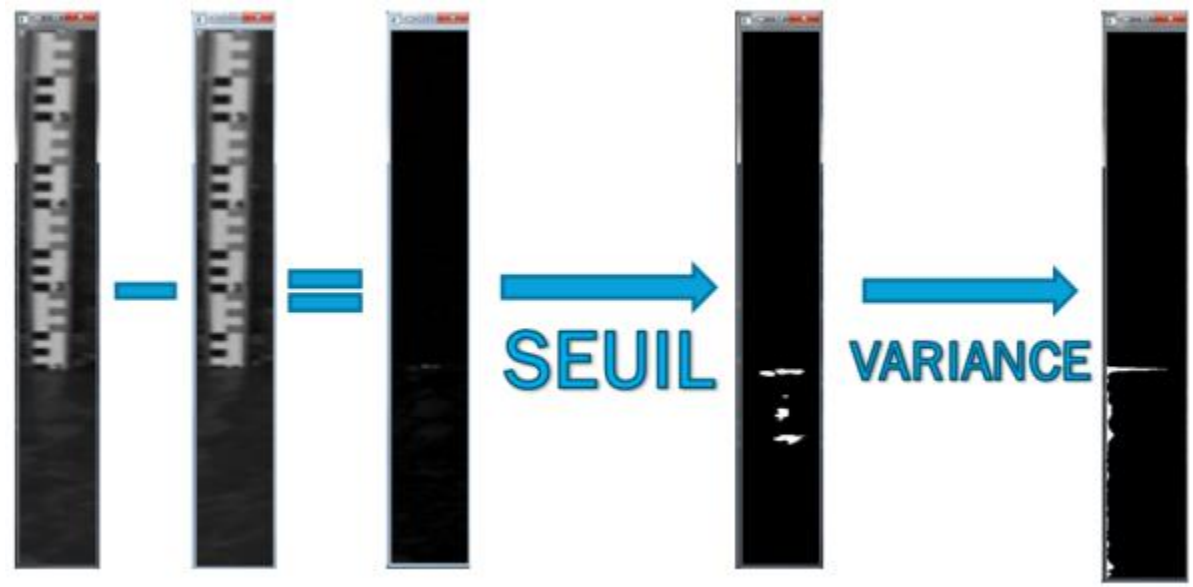

Figure 19 Différence,seuil et variance

[11] https://en.wikipedia.org/wiki/Gaussian_blur

Cette opération de différence ne prend pas de paramètre. Elle consiste seulement en une opération arithmétique. L'image résultante doit être traitée afin de récupérer une mesure. Pour cela on applique un seuil afin de faire ressortir plus clairement les différences. Le seuil est un filtre très simple, binaire ; on définit une limite : le seuil, si un pixel est au-dessus de cette limite, il devient blanc (valeur 255), si il est en-dessous il devient noir (valeur 0). Comme nous le verrons dans la partie sur le code, le seuil est déterminé à partir de la luminosité de l'image.

c. Histogramme et Variance

Le but du traitement d'image est de modifier une image par des filtres afin de faciliter la détection du niveau d'eau par l'intermédiaire d'un algorithme. Nous avons réussi à obtenir une image noire et blanche où des tâches blanches traduisent des mouvements de l'eau devant l'échelle à marée (Fig.19). Il faut maintenant à l'aide d'un algorithme (cf. partie suivante) détecter le niveau d'eau à partir de cette image. Afin de faciliter l'opération et de réduire les risques de fausses détections, nous faisons une sorte d'histogramme de l'image seuil. Cela consiste à compter le nombre de pixels blancs par ligne et afficher cela sous forme d'un diagramme. Suite à différents tests, afficher la « variance » du nombre de pixels blancs par ligne par rapport à la moyenne de pixels blancs par ligne, s'est avéré être plus efficace.

$$V(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2.$$

Dans cette équation, x_i est le nombre de pixels blancs pour chaque ligne, m la moyenne, et V(X) la variance de la suite statistique. Nous ne cherchons donc pas vraiment la variance « globale » mais bien $(x_i - m)^2$ pour chaque ligne de notre image.

d. Le cas nocturne.

Comme vous pouvez le voir sur la figure 21, les prises de vues nocturnes réussies grâce à un éclairage Infrarouge sont extrêmement différentes. On s'aperçoit rapidement que l'échelle se détache plus fortement du reste de par sa

couleur blanche. Il y a un bien meilleur contraste que le jour. Cela simplifie l'opération de détection d'autant plus que la mer arbore une teinte très sombre. Il suffit donc d'appliquer un simple seuil afin de rendre notre image noire ou blanche : blanche sur la partie échelle et noire dans la partie eau. (Fig.20)

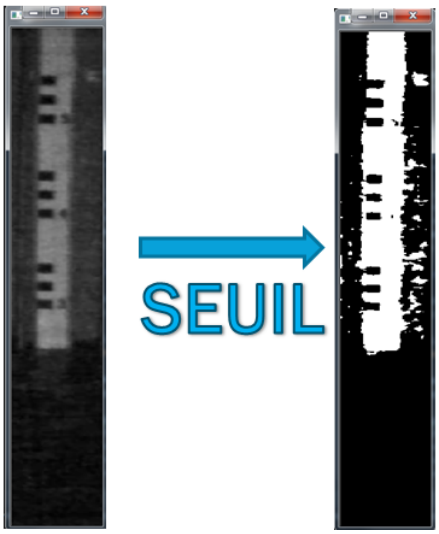

Figure 21 Seuil sur image nocturne

Figure 20 Prise de vue nocturne

IV. Le code, l'algorithme de détection

Quelques portions de code importantes.

Nous présenterons dans cette partie l'aspect purement logiciel du marégraphe vidéo. On rappellera donc que le code source du logiciel faisant fonctionner le marégraphe (prises d'images, traitement d'image, algorithme) est écrit en Python version 2.7. Python est un langage haut niveau et de par ce fait, est facile à appréhender. Si vous souhaitez en apprendre plus sur ce langage, je vous conseille de suivre le cours du site OpenClassrooms qui est parfaitement adapté aux néophytes[12]. Il convient aussi de préciser que les images utilisées pour illustrer mes propos présentent des parties de codes tirées de l'état actuel du projet, dans une phase d'expérimentation et de tests. La syntaxe est donc non optimisée et le code peut sembler « brouillon »

1. Architecture du logiciel

Séparer toutes les fonctions du logiciel dans des fichiers spécifiques permet d'organiser le programme. Ainsi, on va créer un fichier contenant les fonctions de traitement d'image, un permettant de gérer l'écriture et la sauvegarde des données dans un fichier *.txt, etc. …

Cela représente plusieurs avantages. Dans un premier temps le programme sera plus ordonné, ce qui va permettre à quelqu'un qui lit le code source du programme de mieux s'y retrouver. Cela permet aussi de faciliter la maintenance du programme car on sait à quoi chaque fonction correspond.

Voici l'architecture du programme imaginée jusqu'à présent. Elle se compose de 9 fichiers, dont 3 fichiers texte :

- *Main.py* : Ce fichier est le « corps » du programme. C'est le fichier que l'on va exécuter au lancement du programme. Il contient toutes les instructions

[12] https://openclassrooms.com/courses/apprenez-a-programmer-en-python

nécessaires à la bonne exécution du logiciel. Il contiendra majoritairement des appels vers les fonctions des autres fichiers *.py (que l'on pourra appeler par analogie les « organes » du programme). Le fichier « Main » doit donc d'être le plus court possible pour faciliter la compréhension du code source du programme (Fig.22) .

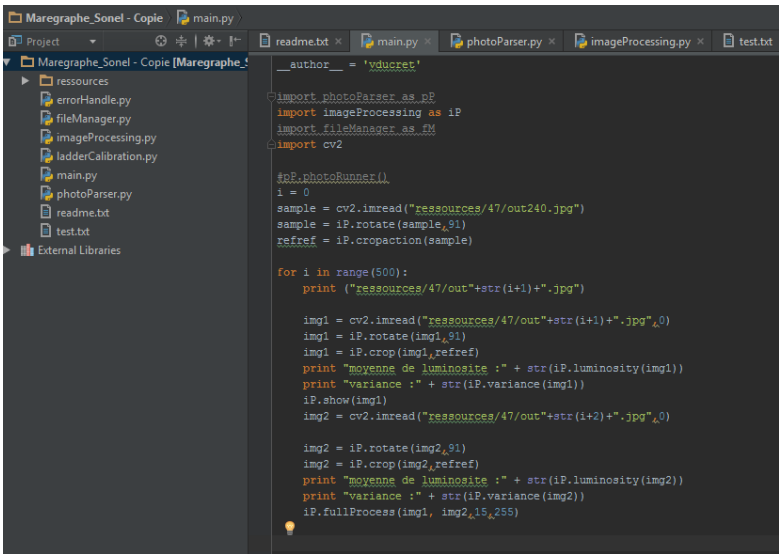

Figure 22 Le fichier Main.py, le corps du programme

- *imageProcessing.py* : Ce fichier contient toutes les fonctions nécessaires au traitement d'image. Cela va de l'affichage d'une image au calcul de l'histogramme de celle-ci. Il va cependant être le fichier le plus long et peut-être allons-nous le séparer en deux parties (une partie pour le traitement, une autre partie pour la détection de la limite à proprement parler).

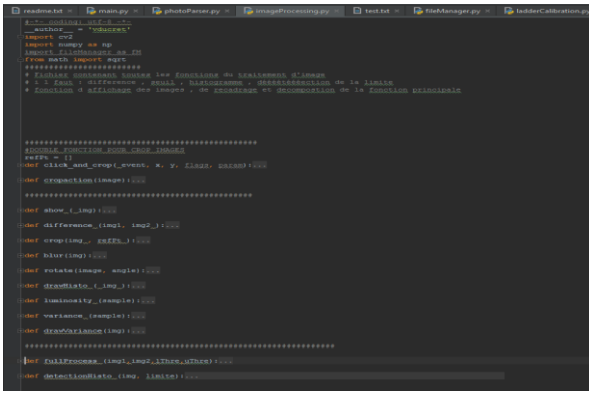

Figure 23 imageProcessing.py

- *photoParser.py* : On stocke ici nos fonctions permettant de parcourir une liste de photos dans l'ordre. On pourra aussi renommer les photos et découper les vidéos.
- *ladderCalibration.py* : Ce fichier contiendra les fonctions nécessaires à la traduction pixel/centimètre. Il travaillera avec le fichier CALI.txt pour stocker les valeurs de calibration entrées manuellement par l'utilisateur.
- *errorHandling.py* : Ici, on va gérer les erreurs et exceptions du programme. Par exemple, dans le cas où l'utilisateur entre une valeur incohérente lors de la calibration manuelle (une chaîne de caractères à la place d'un entier par exemple). On gère aussi ce qui concerne les valeurs de mesure incohérentes qui devront être corrigées.
- *fileWriter.py* : Enfin, ce fichier comprend les fonctions qui permettront de lire et d'écrire dans des fichiers texte.

Viens ensuite les 3 fichiers texte :

- *Save.txt* : Permet de sauvegarder les données. Le choix d'un fichier texte s'est imposé dans un premier temps de par sa facilité et son intégration plutôt simple dans d'autres logiciels (Microsoft Excel par exemple).
- *Cali.txt* : Ce fichier n'a pas vocation à être lu directement par l'utilisateur. Il s'agit plutôt d'un fichier « buffer » stockant les valeurs rentrées manuellement par l'utilisateur. Le choix d'un fichier texte pour stocker ces valeurs présente plusieurs avantages : on peut garder une trace des variables de calibrations. On peut aussi penser à un futur système pour sauvegarder et charger différents fichiers de calibration en fonction des différents marégraphes.
- *Log.txt* : on écrira dans ce fichier les évènements important qui se sont déroulés durant l'exécution du programme. C'est un fichier log classique permettant à l'utilisateur de pouvoir garder un œil sur le déroulement du programme et voir où se situent les éventuels problèmes.

Nous intégrerons aussi un fichier « readme.txt » standard comprenant la version du logiciel, ses « patch notes », ses crédits, etc…

2. Quelques portions de code

a. Les fonctions

Les fonctions permettent de structurer notre programme. Au lieu de placer tout le code au même endroit ce qui serait très indigeste pour le codeur et aussi les personnes lisant le code source, on sépare notre programme par fonctionnalités et chaque fonctionnalité se verra attribuer son propre code. On peut ensuite appeler ces fonctions. Cela représente l'avantage de pouvoir réutiliser les fonctions sans recopier 50 lignes, de faciliter l'entretien du logiciel car si cela a bien été pensé, on peut changer une fonction en gardant ses spécifications sans toucher au reste du programme et ainsi faciliter la compréhension du code source.

Nous pouvons voir plusieurs fonctions dans la figure 23. Prenons l'exemple de la fonction « show » (Fig.24).

```
def show(img):
    cv2.namedWindow("preview", cv2.WINDOW_FREERATIO)
    cv2.imshow("preview", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figure 24 La fonction show

Commençons par la définition d'une fonction. En Python on définit une fonction à l'aide de la commande :

Def « nomDeLaFonction »(Paramètres) :

Instruction1

Instruction 2 …

« Def » permet donc de définir une fonction ou méthode mais nous reviendrons sur la différence plus tard. On renseigne le nom de la fonction et optionnellement des paramètres. Les paramètres permettent de passer à la fonction des variables lors de l'appel de celle-ci. La fonction « show » ci-dessus permet d'afficher une image. On lui donne donc en paramètre l'image à afficher. Vous remarquerez que la fonction « show » est elle-même constituée d'appels à des fonctions d'OpenCV.

Donc, lorsque dans notre programme principal nous écrirons :

Show(mon_image)

Le programme comprendra que nous faisons appel à la fonction « show » et remplacera cette ligne par :

Cv2.namedWindow(« preview »,cv2.WINDOW_FREERATIO)

Cv2.imshow("preview", mon_image)

Cv2.waitKey(0)

Cv2.destroyAllWindows()

La différence entre une fonction et une méthode est simple : une fonction retourne une variable tandis qu'une méthode n'en retourne pas et se contente d'appliquer des actions : la fonction « show » est en fait une méthode car elle permet seulement d'afficher une image. La fonction « luminosity » que nous verrons plus tard (Fig.25) est bel et bien une fonction car elle retourne une valeur « luminosity ». Ainsi la ligne *moyenne = luminosity (mon_image)* va permettre d'attribuer à la variable *moyenne* la luminosité de « mon_image » tandis que *moyenne = show(mon_image)* n'a aucun sens et provoquera une erreur.

b. Les « import »

Comme vu précédemment, nous découperons les fonctionnalités du programme en différentes fonctions que l'on placera dans des fichiers regroupant plusieurs catégories. Une notion importante entre alors en jeu. Il faut en effet indiquer au programme que l'on souhaite importer les fonctions d'un fichier. Cela ce fait grâce à la commande « import » (cf fig.3). Cela nous permettra donc d'importer nos différentes fonctions dans notre fichier « Main ». Il faut aussi noter que cela permet d'importer des bibliothèques téléchargées précédemment comme OpenCV ou Numpy par exemple. Afin d'utiliser les fonctions importées, il faut indiquer le nom du fichier depuis lesquelles elles sont importés. Pour le côté pratique on peut aussi rajouter « as {nom} ». Cela nous permet de ne pas taper le nom du fichier en entier mais juste {nom} :

Import imageProcessing as iP

iP.show(mon_image)

est beaucoup plus rapide, surtout si on utilise beaucoup les fonctions de imageProcessing que:

import imageProcessing

imageProcessing.show(mon_image)

c. Parcourir une image

Comme expliqué précédemment, une image est une matrice de pixels. Un pixel est le plus petit point de définition d'une image. Il possède 1 ou 3 valeurs dans notre cas. Pour une image noire et blanche, une seule valeur de 0 à 255 définit le pixel (du blanc au noir). Pour les images en couleurs, 3 valeurs, RGB, définissent un pixel comme nous l'avons vu précédemment.

Dans notre cas, il est utile afin de manipuler les images, de parcourir pixel par pixel la matrice afin d'en retirer des informations ou bien d'appliquer un traitement. Même si la bibliothèque OpenCV ne nécessite pas forcément de parcourir les images, cette opération nous est utile dans le cadre des calculs de luminosité, de variance, de l'histogramme.

Ci-dessous vous pouvez trouver une capture d'écran qui explique l'opération (Fig.25).

```
def luminosity(sample):
    somme = 0
    # On récupère la hauteur de pixel d'une photo
    hauteur = sample.shape[0]

    #On récupère la largeur de la photo
    largeur = sample.shape[1]

    #On parcourt maintenant la photo ligne par ligne
    for h in range(hauteur):
        #Pour chaque ligne au parcours les colonnes unes à unes
        for l in range(largeur):
            #Ici on applique l'opération sur le pixel spécifique de coordonnée (h, l)
            somme = somme + sample.item(h,l)

        #En se placant ici on peut appliquer une opération sur la ligne entière

    # En se placant ici on applique une opération sur l'ensemble de l'image
    luminosity = (somme / (hauteur*largeur))

    return luminosity
```

Figure 25 Parcours d'une image

Nous allons maintenant expliquer cette fonction plus précisément.

La boucle FOR

Une boucle FOR permet de répéter une opération un nombre de fois précis. Regardons plus précisément la formulation : « for h in range(hauteur) ».

La fonction « range » permet de convertir une valeur en une liste. Ainsi si hauteur = 8, range(hauteur) = 1, 2, 3, 4, 5, 6, 7, 8.

On demande donc au programme de parcourir cette liste en assignant à la variable « h » l'itération de la boucle.

Dans notre cas, h prendra donc la valeur 1 puis la valeur 2 puis la valeur 3 et ainsi de suite jusqu'à la valeur 8.

L'imbrication de deux boucles FOR

Nous pouvons donc maintenant parcourir ligne par ligne une image. Notre but est cependant de parcourir l'image pixel par pixel. Nous allons donc écrire une deuxième boucle FOR à l'intérieur de la première. Ainsi pour chaque ligne de pixel, nous parcourrons la ligne colonne par colonne. Ici la variable correspondant au numéro de colonne actuel est « l ».

Ainsi le couple de valeur « h » et « l » correspond aux coordonnées du pixel. On peut donc désormais appliquer une opération sur chaque pixel individuellement. On peut aussi se placer à des niveaux différents si l'on veut effectuer une opération ligne par ligne ou sur l'image entière (Fig.25)

Dans le cas de l'exemple, on récupère la valeur de chaque pixel de l'image (avec sample.item(h,l) que l'on ajoute à la variable « somme ». Une fois le parcours de l'image terminé, on divise « somme» par le nombre de pixels de l'image et on retourne cette valeur.

3. L'algorithme de détection

Suite au traitement d'image expliqué à la partie III, on obtient une image en noir ou blanc où l'interface air-mer ressort fortement. On doit maintenant extraire de cette image la hauteur d'eau. On va pour cela utiliser deux algorithmes dans notre logiciel qui prendront en entrée une image et donneront en sortie une valeur en pixels. Un algorithme travaillera les images de jour, l'autre celles de nuit.

a. La zone de validité

La première étape consiste en la définition d'une zone de validité. C'est une plage de mesure créée pour éliminer les erreurs. Si le résultat d'une mesure est hors de cette zone de validité, on considère que la mesure n'est pas significative. Cette zone est déterminée à partir de la mesure précédente ; si la mesure précédente était à 2,00 m, il est très peu probable que la mesure suivante soit à 3,00 m.

b. Le jour

L'algorithme de jour va détecter les pics que l'on peut apercevoir sur l'image finale du traitement de jour. Pour cela il va définir une limite très haute qu'il va ensuite faire baisser afin de détecter le premier pic (Fig.26). Il vérifie alors que l'emplacement du pic appartient bien à la zone de validité. Si ce n'est pas le cas, la limite descend jusqu'au prochain pic et ainsi de suite. Si c'est le cas le résultat est gardé en mémoire.

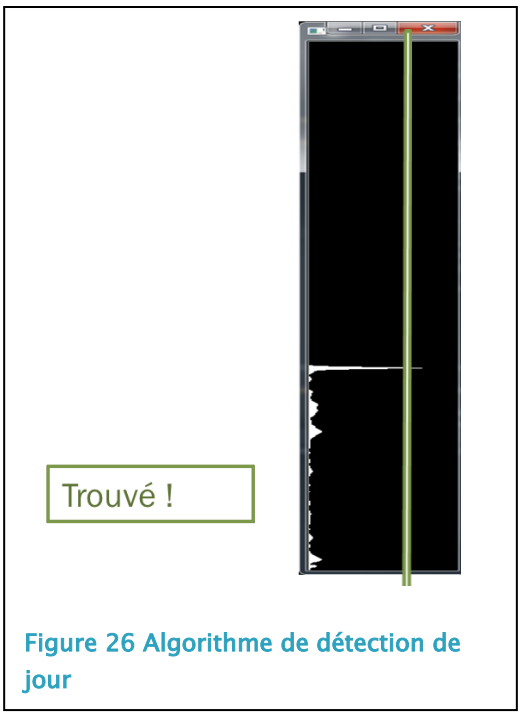

Figure 26 Algorithme de détection de jour

c. La nuit

L'image finale du traitement de nuit, beaucoup plus nette, ne requiert pas autant de travail de la part du logiciel. L'algorithme va parcourir l'image ligne par ligne pour trouver une ligne entièrement noire (Fig.27). Une fois celle-ci trouvée, il vérifie qu'il n'y ait plus de taches blanches significatives en-dessous

Figure 27 Algorithme de détection de nuit

Il convient de rappeler que les algorithmes présentés peuvent encore être grandement améliorés. Ils ont cependant été choisis à l'issue de plusieurs tests et ce sont avérés être les plus précis.

4. Fonction de calibration

Nous disposons maintenant d'une valeur en pixel correspondant au niveau d'eau. On va maintenant traduire cette valeur en coordonnées dans le monde réel, c'est-à-dire en mètres. Il va falloir pour cela créer une fonction de traduction qui prendra en entrée une valeur de pixels et donnera en sortie une valeur en mètre.

a. La calibration manuelle

La première étape va être de calibrer notre logiciel, et par extension, notre marégraphe. Plusieurs choix se sont offerts à nous. La calibration automatique, comme GaugeCam qui possède l'avantage de ne pas nécessiter l'intervention de l'utilisateur mais qui requiert du matériel spécifique. Cela permet aussi de bouger la caméra sans gêner une mesure en cours. Nous avons cependant choisi d'effectuer une calibration manuelle, plus simple, et en accord avec notre cahier des charges. Même si celle-ci est plus lente et requiert l'attention de l'utilisateur, elle ne dépend pas de matériel spécifique et marche dans tous les cas.

Lorsque l'utilisateur initialise une calibration, le programme affiche l'image, préalablement recadrée. L'utilisateur va alors pointer une graduation de l'échelle et entrer la valeur à laquelle elle correspond en terme de mètres dans la console python (Fig.29). Le programme va sauvegarder le couple de valeurs (la valeur en pixel et la valeur réelle correspondant) dans un fichier texte (Fig.28). L'utilisateur multiplie cette opération sur le nombre de graduations disponibles.

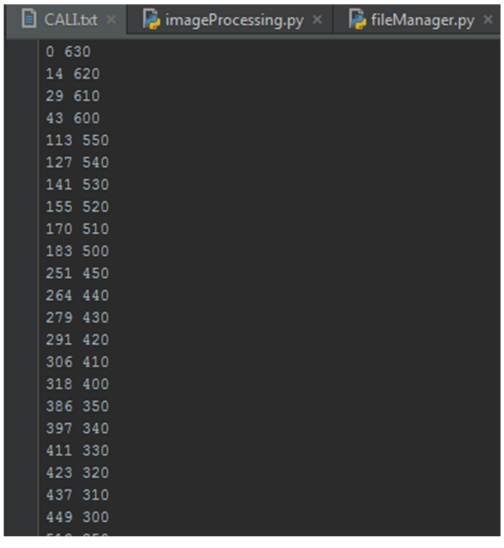

Figure 28 Le fichier de calibration

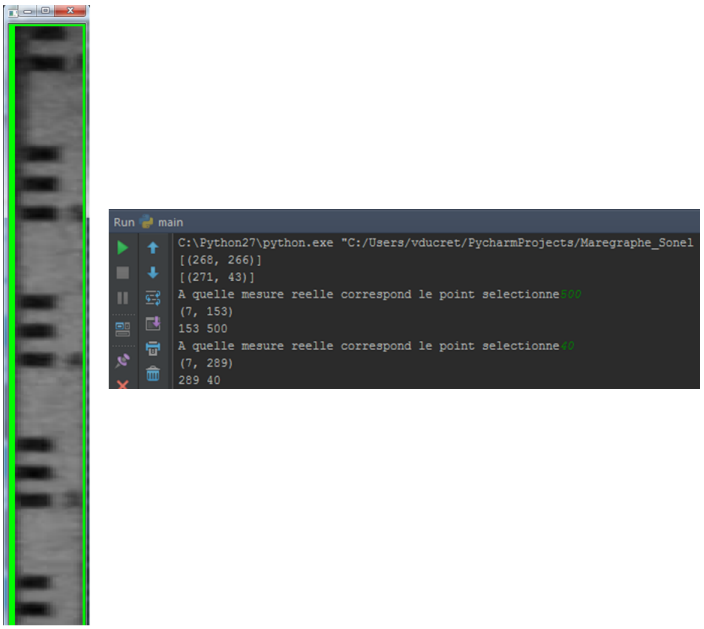

Figure 29 La fonction de calibration

b. La fonction de traduction

Vient ensuite la fonction de traduction. Pour rappel, elle prend comme paramètre d'entré une valeur en pixel et donne en sortie une valeur en mètre qui sera notre valeur finale, le résultat de notre mesure. Cette fonction nécessite aussi un fichier

de calibration. Il faut donc effectuer une calibration manuelle en amont. La fonction va comparer sa valeur d'entrée, que l'on va appeler paramètre, aux valeurs de pixels présentes dans le fichier de calibration. Elle parcourt le fichier ligne par ligne jusqu'à trouver une valeur égale ou inférieure à son paramètre.

- Si la valeur est identique au paramètre, la traduction est directe. La fonction se contente de donner la hauteur d'eau rentrée préalablement par l'utilisateur.
- Si la valeur est inférieure, cela signifie que l'on peut encadrer notre paramètre entre deux valeurs entrées par l'utilisateur. Ces deux valeurs sont en fait des couples de valeurs (en pixel et en hauteur réelle) et sont par conséquent des points. La fonction va alors calculer l'équation de la droite (Fig.29) entre ces deux points encadrants notre paramètre. Elle va ensuite pouvoir trouver, à l'aide de cette équation, la valeur réelle correspondante au paramètre.

```
#########################################################################
#Fonction de traduction
def lire_fichier(destination, valeur ):
    data = []
    fichier = open(destination)
    for ligne in fichier:

        ligne.split()
        ligne = ligne.split()
        data.append((float(ligne[0]),float(ligne[1])))

    n = 0
    while ( valeur > data[n][0] ):
        n=n+1


    coef, origine = trouver_droite(float(data[n][0]),float(data[n][1]),float(data[n-1][0]),float(data[n-1][1]))
    resultat = coef * valeur + origine
    print resultat
    return resultat
#################################################
#Fonction permettant de trouver l'equation de la droite :

def trouver_droite ( Xb, Yb , Xa ,Ya):
    coef = ((Yb-Ya)/(Xb-Xa))

    origine = Yb - (coef * Xb)
    return (coef,origine)
```

Figure 30 Fonctions de traduction et d'obtention de l'équation de la droite

La capture d’écran de la Figure 30 Fonctions de traduction et d'obtention de l'équation de la droite représente seulement le cas où la traduction n’est pas directe et où on doit trouver l’équation de la droite.

Nous disposons maintenant de notre résultat final, qui correspond à la mesure du niveau d’eau à partir d’une séquence d’images ou d’une vidéo. Il ne manque plus qu’à sauvegarder ces valeurs dans un fichier texte.

V. Le prototype

Les composants du marégraphe

Nous allons aborder dans cette partie la conception d'un premier prototype de marégraphe vidéo (Fig.32). Afin de tester le logiciel de traitement nous avons donc conçu un prototype que nous avons mis à l'essai lors de nos manipulations sur le terrain le 3, 4, 17 et 18 Août. Ce prototype représente notre vision du marégraphe vidéo : peu coûteux, simple d'utilisation tout en restant robuste et précis. Nous nous sommes inspirés du produit de GaugeCam pour la conception de notre prototype (Fig.31)

Figure 32 Le prototype

Figure 31 Le prototype de GaugeCam

1. Le Raspberry Pi 2

Notre logiciel de traitement demande une unité de calcul afin d'exécuter notre programme Python. Le Pi 2 de Raspberry représente la meilleure solution. C'est un micro-ordinateur de la taille d'une carte bancaire (Fig.33) qui coûte, totalement équipé, moins de 150 euros. La puissance délivrée est largement suffisante pour assurer le fonctionnement du logiciel en temps réel.

C'est aussi une plate-forme de développement informatique et électronique. Le système d'exploitation étant linux (le choix de la distribution est laissé à

l'utilisateur), le système est ouvert au développement et possède une grande communauté. Le Raspberry possède aussi des connecteurs GPIO pour brancher et gérer de façon logicielle des composants électroniques. Encore une fois il existe une grande communauté qui développe sans cesse de nouveaux modules.

Figure 33 Le Pi 2 de Raspberry

Enfin, le Pi possède sa propre caméra, commercialisée elle aussi par Raspberry. Ses caractéristiques sont très intéressantes. Elle prend des photos de 5 Mégapixels et peut filmer en Full HD (1920x1080). Le point le plus important est cependant la présence de pilotes en Python pour la caméra. On peut en effet changer les paramètres de capture à l'intérieur du logiciel. De plus une version Infra-Rouge de cette caméra existe, ce qui nous permettra des prises de vue la nuit.

2. L'alimentation, l'éclairage

Une caméra infra-rouge est une caméra standard à laquelle on a retiré le filtre anti infra-rouge. L'infra-rouge est une onde qui n'est pas visible par l'œil humain mais qui l'est pour un appareil photo. Associée à un éclairage en infrarouge, une caméra sans filtre infra-rouge pourra donc prendre des photos de nuits sans illuminer la scène pour un œil humain. Nous avons donc rajouté une matrice de leds infrarouges qui s'allume lorsque la luminosité devient trop basse.

Se pose alors la question de la consommation électrique de notre prototype. Pour les premiers tests nous avons construit un « bloc d'alimentation » avec une batterie 12V semblable à celle d'une voiture (Fig.34). Nous n'avons pas effectué de

tests approfondis mais le niveau de batterie après 14h de fonctionnement était resté sensiblement identique. On pourra bien sûr ajouter à notre dispositif des panneaux solaires pour le rendre totalement autonome.

Figure 34 Le bloc d'alimentation du prototype

3. Le futur

Comme expliqué précédemment, les Raspberry Pi 2 est une plateforme de développement, informatique et électronique. Il existe déjà de nombreux modules que l'on peut ajouter à notre micro-ordinateur pour lui ajouter des fonctionnalités. La première fonctionnalité intéressante serait une connexion à internet par le biais d'une clé Wifi ou d'un module GPRS/3G. Il suffirait alors d'une carte SIM et de la disponibilité d'un réseau téléphonique pour envoyer les données ou contrôler les données par internet.

On peut aussi penser à l'ajout d'un module météo pour compléter notre jeu de données en y ajoutant la pression atmosphérique ou le pourcentage d'humidité dans l'air. Si aucune connexion à internet n'est possible, il faudra aussi penser à rajouter une horloge interne. Le Raspberry Pi n'en possède pas de base et est soumis à un important décalage de son horloge au fil du temps.

Enfin, il faudra penser à améliorer la robustesse de notre prototype en remplaçant les matériaux sensibles à l'eau et au sel par des matériaux qui ne rouillent pas.

VI. Les résultats

Pourquoi avons-nous obtenu de tels résultats ? Comment les améliorer ?

Nous allons maintenant vous présenter nos résultats. Les travaux présentés dans ce document représentent l'avancement actuel du projet. Le traitement d'image ainsi que les fonctions et les algorithmes utilisés sont encore largement améliorables. Les résultats présentés ne sont donc pas définitifs ; le projet en est à ses débuts et sera soumis à de nombreuses améliorations.

Cette présentation fera la part entre les résultats obtenus avec des images de jour et les résultats obtenus de nuit. Comme vous pourrez le voir, il existe une différence importante des résultats entre les différents moments de la journée.

Il est aussi important de préciser que ce sont les images de la caméra Vivotek qui ont mené à de tels résultats. Nous nous sommes en effet heurtés à plusieurs problèmes techniques qui nous ont empêchés de récolter des données complètes avec le Raspbery Pi.

1. Le jour

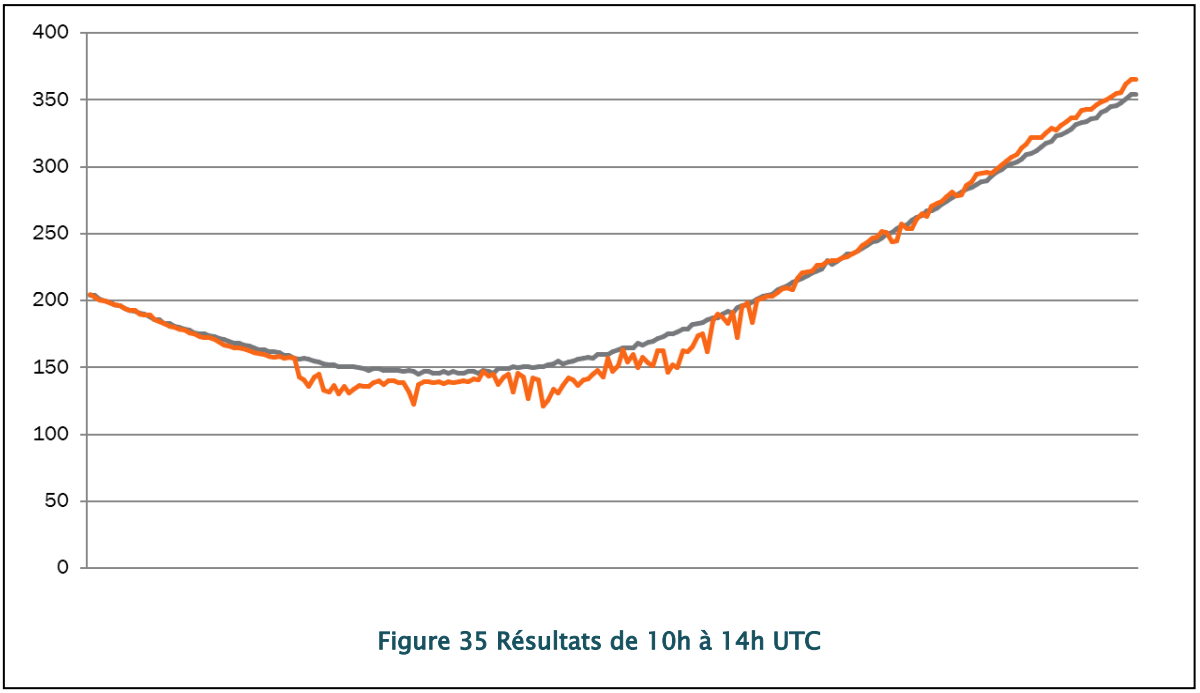

Figure 35 Résultats de 10h à 14h UTC

La courbe ci-dessus, Figure 35 Résultats de 10h à 14h UTC, permet de comparer les données du MCN et celles du marégraphe vidéo. Les données du marégraphe vidéo ont été moyennées à la minute (30 seconde avant et après la minute) pour correspondre au format des données du MCN récupérées sur le site dataSHOM. Sur cette courbe, le MCN est en gris, le marégraphe vidéo en orange. Comme le titre de la figure l'indique, la vidéo a été prise de 10h à 14h UTC ou de 12h à 16h heure locale. La moyenne quadratique de la différence entre les données du MCN et du marégraphe vidéo (ou RMS) est de 9,891.

On peut déjà séparer facilement la courbe en deux parties. Une partie où les deux courbes se confondent plus ou moins et une autre où les résultats du marégraphe vidéo s'écartent de ceux du MCN, où la courbe semble plus erratique. Pour quantifier ceci, on peut s'intéresser à la courbe des résultats non moyennés du MCN (Fig.36). Dans la partie « correcte », la différence entre deux pics consécutifs dépasse rarement les 5cm. Tandis que dans la partie où la courbe semble trembler, la différence entre deux pics consécutifs dépasse parfois les 30cm.

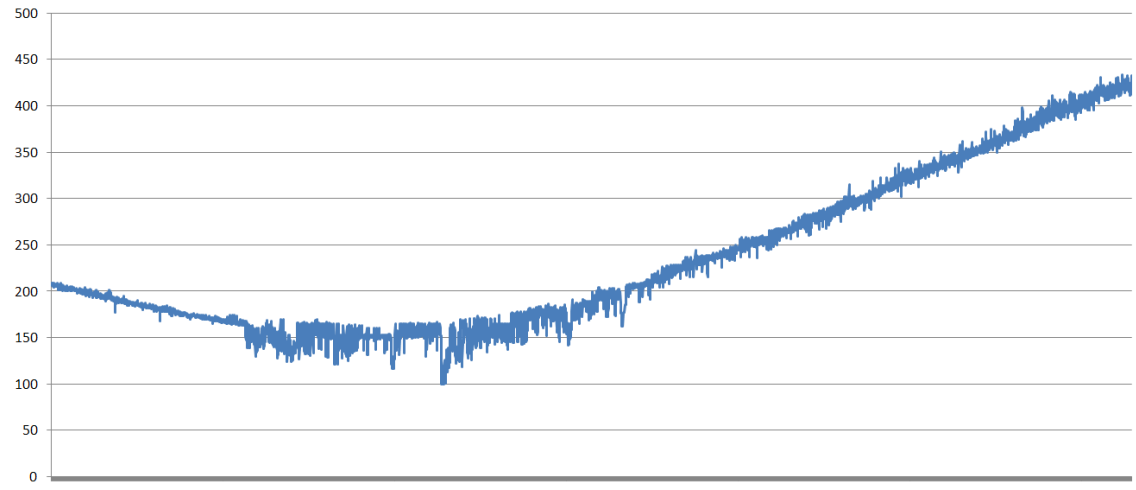

Figure 36 Courbe du marégraphe vidéo non moyennée

Cela n'est pas dû à un changement de l'état de la mer ; d'une augmentation de la houle. Ceci est dû aux conditions de luminosité qui rendent la photo surexposé (Fig.37). La zone de détection se retrouve beaucoup plus lumineuse ce qui fait apparaitre un fort reflet blanc sur l'eau. Notre traitement d'image s'en trouve fortement détérioré.

La solution pour résoudre ce problème serait d'améliorer notre image de base, la rendre moins surexposé. Et cela passe par le changement des paramètres de capture de la caméra. Comme expliqué précédemment, il existe des pilotes en Python pour la caméra du Raspberry. Il suffirait donc de changer les paramètres de la caméra à l'intérieur du logiciel pour éviter ce genre de problèmes.

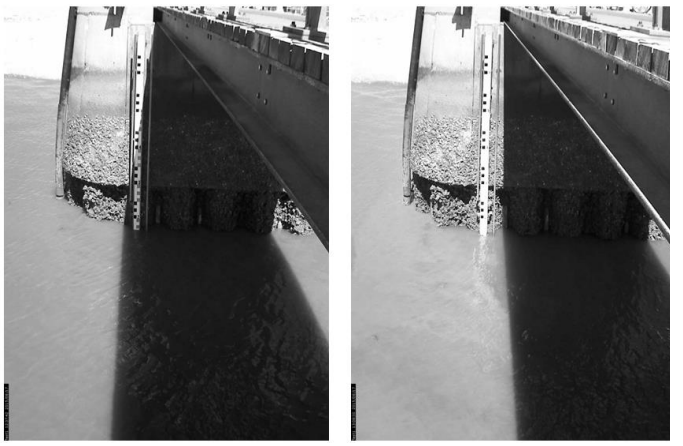

Figure 37 Conditions de capture difficiles

2. La nuit

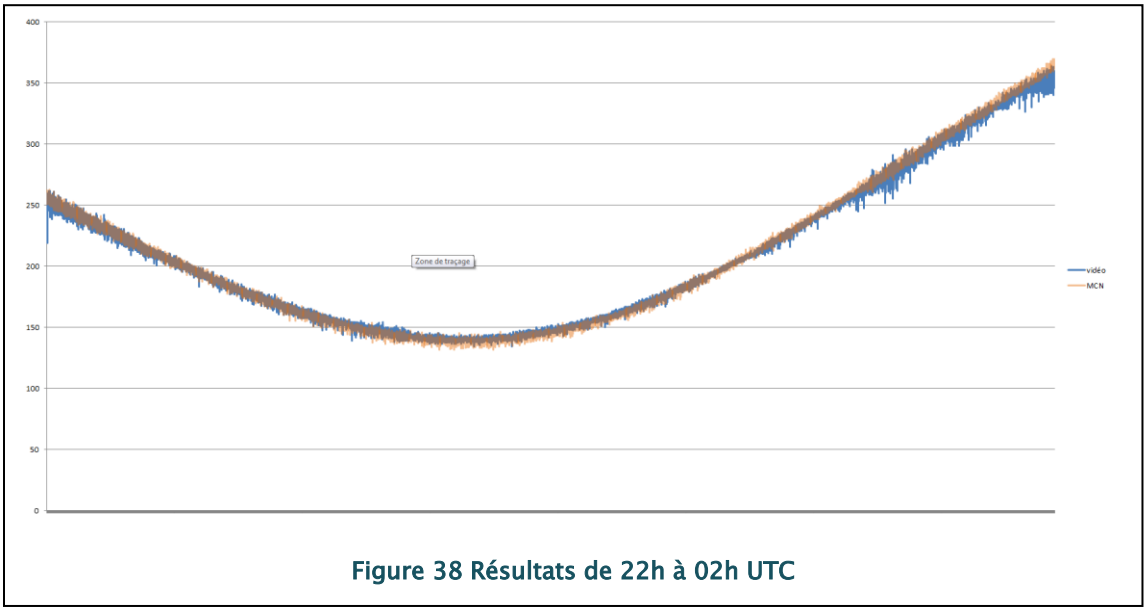

Figure 38 Résultats de 22h à 02h UTC

Pour la nuit, nous n'avons pas à nous inquiéter des conditions de luminosité car elles sont contrôlées par notre éclairage infra-rouge. De plus, l'image de base étant beaucoup plus facile à traiter les résultats sont grandement améliorés (Fig.38). On

notera bien que la courbe du marégraphe vidéo est en bleu. Les données ne sont pas moyennées et sont comparées à celles du MCN (courbe orange). Ici la différence entre deux valeurs pic à pic ne dépasse jamais les 10cm ce qui est en accord avec les conditions de capture. Les résultats de nuit sont donc les meilleurs que nous avons eu jusqu'à maintenant. La moyenne quadratique de la différence entre les données du MCN et du marégraphe vidéo (ou RMS) est de 2,836.

3. L'aube et le crépuscule

Nous disposons donc de deux méthodes de traitement, l'une pour le jour, l'autre pour la nuit. Il faut maintenant être capable d'assurer le changement entre les deux. Une fonction a donc été créée. Elle calcule la luminosité globale des images et lorsque celle-ci atteint un certain seuil, on entre dans la période de changement de méthode de traitement. On fait alors marcher les deux traitements simultanément et on garde les meilleurs résultats.

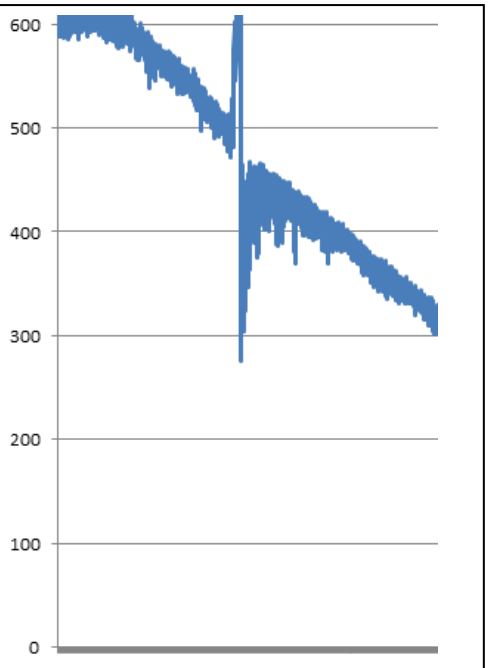

Figure 39 Passage entre les deux méthodes de traitement

Cependant, il existe deux moments de la journée où aucune des méthodes ne marche (Fig.39). Il s'agit de l'aube et du crépuscule. Cela est dû encore une fois à des conditions de captures difficiles qui entrainent une image de base plus difficile à

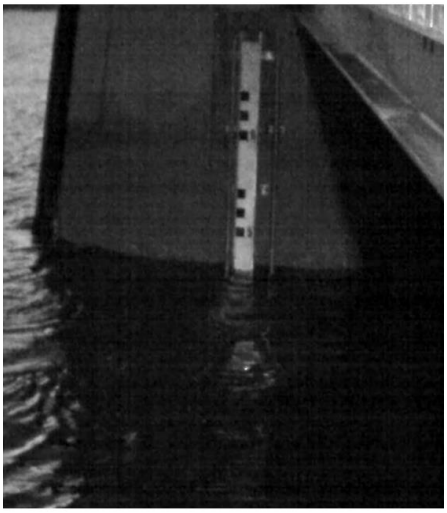

Figure 40 Photo prise au crépuscule

Figure 41 Photo prise à l'aube

traiter. Comme on peut le voir dans le cas de l'aube (Fig.41) et du crépuscule (Fig.40) de forts reflets empêchent notre traitement d'images et nos algorithmes de fonctionner normalement.

Pour régler le problème, plusieurs choix s'offrent à nous. Le premier étant l'élaboration d'une troisième méthode de traitement correspondant aux moments de l'aube et du crépuscule. Une méthode adaptée permettrait d'améliorer nos résultats mais cela alourdirait le programme ce qui le rendrait moins facile à manipuler. On peut aussi penser à changer les paramètres de capture à l'intérieur du logiciel, comme pour le jour. Cela permettrait de plus maitriser notre environnement et d'obtenir une image de base beaucoup plus facile à traiter.

Conclusion

Un projet ambitieux pour une expérience enrichissante.

La principale difficulté de ce projet réside dans son atout principal : l'universalité. Créer un marégraphe vidéo simple d'utilisation, fonctionnant dans toutes les conditions et qui ne requiert pas de matériel spécifique représente à la fois un challenge et l'attrait principal du projet. Or cette polyvalence ne peut s'obtenir sans faire de concessions dans d'autres domaines tel que la précision et la fréquence des mesures. Le choix de la méthode de traitement des images a été le fruit d'une longue réflexion. Au final, nous avons préféré nous orienter vers une méthode fonctionnant le plus souvent possible au risque de ne pas obtenir une précision suffisante dans un premier temps. Il faut alors être capable de choisir des paramètres de traitement optimaux pour affiner la méthode, que ce soit au niveau des filtres utilisés ou des algorithmes de détection. Il faut être capable à partir d'une image de retirer un maximum d'informations et de paramétrer toutes les fonctions de notre programme pour avoir le meilleur résultat possible.

Les fonctions, méthodes de traitement d'image et algorithmes présentés dans ce rapport sont le fruit de travaux préliminaires sur le sujet. Ils ne sont en aucun cas définitifs. Une grande partie du travail restant va résider dans l'affinage et le paramétrage de ces fonctions essentiels au marégraphe vidéo. Ce projet est ambitieux et ses objectifs aussi multiples qu'importants. Personnellement, ce projet m'aura apporté beaucoup, en termes d'expérience et de connaissances. J'ai pu découvrir de l'intérieur le domaine de la recherche que j'avais côtoyé durant mes études à l'INSA. J'ai pu découvrir des façons de penser et de travailler que je ne serai peut-être plus amené à côtoyer et ce sera, j'en suis persuadé, d'une grande utilité pour mon parcours professionnel.

Table des illustrations

Annexe

20/01/2014

Offre de stage : Observatoire SONEL

Développement d'un marégraphe vidéo

L'observatoire SONEL de L'Université de la Rochelle propose un stage en vidéo et traitement d'image.

Mots clés :
Programmation C++, Java, Traitement d'image, Equipement vidéo, Marégraphie

Projet qui sera confié au stagiaire :

Dans le cadre de l'observatoire SONEL (Système d'Observation du Niveau des Eaux Littorales) et de son site pilote de l'île d'Aix, nous souhaitons mettre un point un nouveau système de marégraphe par vidéo.
En effet, les échelles à marées sont un moyen stable et précis de mesurer le niveau de la mer. Cependant les lectures sont peu nombreuses et dépendent des opérateurs.
Le but du marégraphe vidéo sera lire le niveau de la mer à haute fréquence (1Hz) et à une précision millimétrique. Ceci grâce à une caméra HD fixe filmant une échelle à marée et un algorithme de traitement d'image détectant précisément l'interface air-mer.

Le travail du stagiaire sera de réaliser une première étude sur ce type de marégraphe :

- Définir le type de caméra nécessaire pour le système (Résolution, Définition,...)
- Mettre en place une méthode de géo-référencement d'image précise au millimètre
- Mettre au point un algorithme de détection du niveau de la mer dans le langage souhaité

L'étudiant sera amené à se déplacer à l'île d'Aix avec son maître de stage pour les besoins de l'étude. Selon l'avancée du travail, le matériel vidéo identifié pourra être commandé pendant la durée du stage et testé in-situ pour valider le système.

Le stagiaire devra faire preuve d'autonomie sur les activités confiées tout en ayant à faire valider et discuter ses choix. Il sera autant motivé par les aspects de programmation et de traitement d'image que par les aspects techniques (choix de matériel,..). L'intérêt pour le milieu marin et la recherche scientifique est un plus.

Durée :
1 mois minimum

Lieu :
UMR (7266) LIENSs, bâtiment ILE, Université La Rochelle

Contact :
Envoyer CV et lettre de motivation à etienne.poirier@univ-lr.fr

Rémunération :
A définir (minimum 500€/mois)

LIENSs / SONEL
Bâtiment ILE
2, rue Olympe de Gouges
17000 LA ROCHELLE
tél : 05.46.45.83.94

Annexe 1 Offre de stage