UCLouvain

École polytechnique de Louvain

Mesure de débit en rivière : analyse et optimisation du projet riverApp

Auteurs: **Grégoire Debray, Dorian Hay**
Promoteurs: **Sandra Soares Frazao, Pierre-Yves Gousenbourger**
Lecteur: **Christophe De Vleeschouwer**
Année académique 2022–2023
Master [120] : ingénieur civil en informatique
Master [120] : ingénieur civil en mathématiques appliquées

Résumé

La mesure du débit dans une rivière permet d'estimer la quantité d'eau qui s'écoule dans cette rivière pendant un intervalle de temps. Dans le cadre de ce travail, l'application `riverApp` est utilisée comme outil pour réaliser le calcul du débit à partir d'une vidéo de l'écoulement. Cette application a été développée au sein de l'UCLouvain par des étudiants. L'objectif de ce travail est d'y apporter des modifications pour la rendre plus performante.

Ce rapport comporte tout d'abord une introduction à la mesure du débit en rivière par analyse PIV (Particle Image Velocimetry) sur une vidéo. Ensuite, une comparaison est faite entre `riverApp` et `PyOrc`, qui est une autre application ayant le même objectif. Cette comparaison permet de conclure que `riverApp` est moins performant que `PyOrc`. Une nouvelle version de `riverApp` est implémentée, basée sur le projet `PyOrc`, et est présentée avec sa nouvelle structure en pipeline.

À partir de la nouvelle implémentation de `riverApp`, une fonctionnalité supplémentaire est ajoutée. Lors du lancement de l'application, l'utilisateur doit renseigner 4 balises sur sa vidéo. Grâce à la nouvelle fonctionnalité, ces 4 balises sont détectées automatiquement. La routine est basée sur des manipulations d'images en utilisant des morphologies mathématiques telles que l'érosion et la dilation.

Ce rapport comprend également une description complète de l'état final de l'application `riverApp`, ainsi que des améliorations potentielles pour l'avenir.

Abstract

The measurement of river flow rate allows estimating the amount of water that flows through that river during a specific time interval. In this work, the riverApp application is used as a tool to calculate the flow rate from a video of the river. This application was developed by students at UCLouvain. The objective of this work is to make modifications to the application to improve its performances.

This report begins with an introduction to flow rate measurement in rivers using Particle Image Velocimetry (PIV) analysis on a video. Then, a comparison is made between riverApp and pyOrc, another application with a similar purpose. This comparison concludes that riverApp is less efficient than pyOrc. A new version of riverApp is implemented based on the pyOrc project, and it is presented with its new pipeline structure.

With the new implementation of riverApp, an additional feature is introduced. When launching the application, the user needs to mark 4 reference points on the video. With the new functionality, these 4 reference points are automatically detected. The algorithm relies on image processing techniques such as mathematical morphology, including erosion and dilation.

This report also provides a comprehensive description of the final state of the riverApp application and potential future improvements.

Remerciements

Nous tenons tout d'abord à remercier Madame Soares Frazão, notre promotrice, pour son suivi et ses conseils tout au long de cette année.

Nous souhaitons également adresser nos remerciements chaleureux à Pierre-Yves Gousenbourger, qui a apporté une contribution inestimable dans la partie informatique de notre mémoire.

Nous tenons également à exprimer notre reconnaissance envers Cédric pour son environnement de travail de qualité.

Enfin, nous tenons à remercier nos proches pour leur soutien tout au long de ce mémoire.

Table des matières

1 Introduction **1**
1.1 Méthodes de calcul de débit . . . 3
1.1.1 Définition du débit d'un cours d'eau . . . 3
1.1.2 Méthodes intrusives . . . 3
1.1.3 Méthodes non-intrusives . . . 3
1.2 Contributions et structure du mémoire . . . 4
1.3 Cadre de travail . . . 5
1.3.1 Travaux préexistants sur `riverApp` . . . 5
1.3.2 Collaboration via `Git` . . . 5

2 État de l'art **6**
2.1 Calcul de débit via une analyse PIV . . . 7
2.1.1 Calcul de débit dans une rivière . . . 7
2.1.2 L'analyse PIV single-pass . . . 8
2.1.3 Pre-processing . . . 11
2.1.4 Les données nécessaires . . . 13
2.2 `riverApp` . . . 17
2.2.1 Avantages . . . 17
2.2.2 Désavantages . . . 17
2.3 `pyOrc` . . . 19
2.3.1 Avantages . . . 19
2.3.2 Désavantages . . . 19
2.4 Fudaa-LSPIV . . . 20
2.4.1 Avantages . . . 20

2.4.2 Désavantages 20
2.5 D'autres outils de calcul de débit 22
2.5.1 FluvialGeomorph-toolbox [12] 22
2.5.2 RIVeR [39] 22
2.5.3 TENEVIA CamFlow [52] 22
2.5.4 COSH [48] 22

3 Analyse Comparative des logiciels similaires à `riverApp` 23
3.1 Mise en contexte 24
3.1.1 Scénarios envisagés 25
3.2 Protocole comparatif 26
3.2.1 Critères de comparaison 26
3.2.2 Sélection des données adéquates 26
3.2.3 Mise à niveau des projets 27
3.2.4 Rapport des résultats 27
3.3 Résultats 28
3.3.1 Débit estimé 28
3.3.2 Temps d'exécution de l'analyse PIV 29
3.3.3 Analyse des API et paramètres 31
3.4 Conclusion de l'analyse 32

4 Implémentation croisée de `pyOrc` et de `riverApp` 33
4.1 Scénarios possibles par rapport à un pull sur le projet `pyOrc` . . . 34
4.2 Fusion des deux projets 36
4.2.1 Quelles fonctionnalités de `pyOrc` sont conservées 36
4.2.2 Quelles fonctionnalités de `riverApp` sont conservées 36
4.3 Fonctionnement général et paradigme du code actuel 37
4.3.1 Avantage du paradigme de pipeline 39
4.3.2 Code du `main.py` 40

5 Détection automatique des balises 42
5.1 Raisons de l'automatisation et attentes liées à cette amélioration . . 43
5.1.1 Simplification du processus pour l'utilisateur 43
5.1.2 Détection des balises standards 43
5.2 Mise en contexte 44
5.3 État de l'art des techniques de pattern matching 46
5.3.1 Concordance de template (*template matching*) 46
5.3.2 Scale-invariant feature transform (SIFT) 47
5.3.3 Speeded up robust features (SURF) 47
5.3.4 Oriented FAST and Rotated BRIEF (ORB) 48
5.3.5 Détection des bords grâce à l'algorithme de Canny 49

5.4 Tests et choix de la routine implémentée 51
5.4.1 Template matching 51
5.4.2 SIFT 51
5.4.3 SURF 51
5.4.4 ORB 51
5.4.5 Canny 51
5.4.6 Bilan des tests 55
5.5 Implémentation finale 56
5.5.1 Constat de départ 56
5.5.2 Analyse du pattern 57
5.5.3 Threshold sur la palette de couleur 58
5.5.4 Post-traitement de l'image 60
5.5.5 Analyse de similarité des formes 64
5.5.6 Approfondissements possibles de la méthode 69
5.5.7 Tri des balises dans le sens horlogique 74
5.5.8 Résultats et précision de l'implémentation 75

6 Version finale du projet **81**
6.1 Description du projet `riverApp` final 82
6.1.1 Structure basée sur `pyOrc` 82
6.1.2 Interface graphique 83
6.2 Modifications supplémentaires apportées 86
6.2.1 Formatage des données 86
6.2.2 Création d'un *Jupyter Notebook* de première utilisation de `pyOrc` 86
6.2.3 Création d'un fichier `__init__.py` pour `pyOrc` 86
6.3 Limites et désavantages 87
6.3.1 Limites liées aux données utilisables 87
6.3.2 Limites liées à l'implémentation de `pyOrc` 88
6.3.3 Autres pistes de développement 89

A Annexes **91**
A Balise standard 92
B `data_format.md` 93
C Tests supplémentaires sur la détection de la rivière 96

Chapitre 1
Introduction

Les observations de la vitesse et du débit des rivières revêtent une importance primordiale pour appréhender les interactions de l'humanité avec la surface terrestre. Prenez le temps d'y réfléchir : chaque goutte de pluie qui ne s'évapore pas ou ne se retrouve pas stockée à long terme se retrouve inexorablement dans une rivière. Ainsi, lorsque vous contemplez un cours d'eau, vous observez en réalité une intégration des conditions météorologiques de toute la région en amont, combinées aux caractéristiques du sol et à l'influence humaine sur l'ensemble du bassin versant.

Dans un contexte marqué par des événements climatiques dévastateurs, le monde occidental est confronté à une série de catastrophes naturelles. Entre sécheresses persistantes et inondations dévastatrices, la nécessité d'étudier et de comprendre le débit des rivières s'impose comme une priorité urgente. Face à ces défis, l'avancement des connaissances et des technologies joue un rôle crucial.

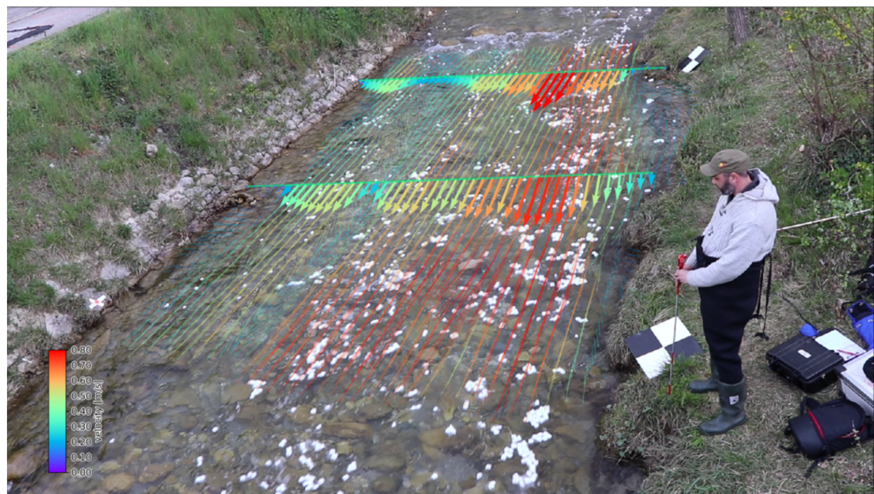
L'avènement récent d'outils novateurs tels que `riverApp` marque une étape décisive dans ce domaine. Il devient désormais possible de procéder à une observation aisée et économique du débit des rivières en utilisant des caméras abordables, que ce soit via votre smartphone, un drone ou une caméra de surveillance.

L'application `riverApp` propose une solution ergonomique pour calculer le débit d'un cours d'eau. Le développement récent des librairies open-source OpenCV[35] et OpenPIV[44] a permis la construction d'un module robuste et performant. Néanmoins, cette application n'est pas encore aboutie et le développement de celle-ci constitue la partie centrale de notre travail.

Face à la multiplication des catastrophes liées à la gestion de l'eau, une telle application s'impose désormais comme un outil indispensable. Plus particulièrement,

certaines régions du monde souffrent de stress hydriques de grande ampleur qui mettent à mal la cohabitation soutenable entre l'humain et son environnement [45]. Dans ces régions, l'observation du débit des rivières est devenue coûteuse et risquée [49]. Ce sont pourtant ces mêmes régions qui souffrent le plus de contraintes budgétaires. L'application `riverApp` associée à une caméra offre une solution gratuite et facilement utilisable sans infrastructure ou coût supplémentaire. Dans ce contexte, elle favorise l'autonomie des instances locales et préserve leur indépendance.

(a) Image originale issue d'une vidéo



(b) Champ de vitesses à la surface ainsi que l'estimation du débit sur deux sections distinctes, calculés par `riverApp` v2

FIGURE 1.1 – *Un bon croquis vaut mieux qu'un long discours - Napoléon Bonaparte*

1.1 Méthodes de calcul de débit

Nous proposons ici un bref aperçu théorique pour acquérir les notions fondamentales. Le lecteur pourra ainsi obtenir une compréhension préliminaire en vue de saisir la présentation de la structure de ce mémoire.

1.1.1 Définition du débit d'un cours d'eau

Le calcul du débit d'un cours d'eau se définit comme le volume d'eau qui traverse une section transversale de la rivière dans un certain intervalle de temps. En pratique, le calcul de ce débit dépend de la connaissance de deux principales données, d'une part la géométrie du lit de la rivière et d'autre part la vitesse d'écoulement de l'eau. Pour calculer ces données, deux grandes catégories de méthodes de mesure peuvent être appliquées : les méthodes intrusives et les méthodes non-intrusives.

1.1.2 Méthodes intrusives

Afin de calculer le débit d'eau dans une rivière, on peut utiliser des méthodes dites intrusives, c'est-à-dire des techniques qui nécessitent un contact direct avec l'eau. Ces méthodes permettent de calculer la vitesse d'écoulement de l'eau, et certaines d'entre elles sont présentées dans [16].
Ces méthodes peuvent parfois être contraignantes pour des raisons d'accès à l'eau ou de perturbation de l'écosystème marin. Elles peuvent également chronophages et coûteuses.
Néanmoins, l'utilisation de ces méthodes permet de donner des valeur précises du débit. En Wallonie, ce sont encore des techniques intrusives qui sont utilisées [22].

1.1.3 Méthodes non-intrusives

Les méthodes non-intrusives étudiées dans ce rapport sont des méthodes basées sur l'analyse de vidéos de l'écoulement de la rivière. Notons que l'analyse vidéo permet le calcul de la vitesse d'écoulement mais pas de l'aire traversée. Pour des techniques non-intrusives sur le calcul de l'aire traversée, le lecteur peut se référer à [43].
La principale méthode étudiée dans ce rapport est une analyse PIV (Particle Image Velocimetry), qui permet d'estimer un champ de vitesse sur la surface du cours d'eau filmé. Le procédé de calcul de cette technique sera détaillé dans le Chapitre 2.

1.2 Contributions et structure du mémoire

Le programme `riverApp` est développé en `Python`. Étant sous licence `GPL3`, il est qualifié de logiciel libre. Pour certains jeux de données, l'application `riverApp` correctement paramétrée fournit des résultats pertinents. Ce n'est malheureusement pas le cas pour la majeure partie des vidéos testées en entrée. Les résultats sont souvent très éloignés des valeurs attendues et présentent même parfois des incohérences. `riverApp` a donc encore du chemin à parcourir au fil des futurs mémoires qui lui seront dédiés. L'absence d'un guide développeur constitue un autre frein à son développement. Chaque année, un effort conséquent doit être fournit par le nouveau groupe afin de comprendre la structure du code de l'application ainsi que son implémentation. C'est devant ce premier constat que nous avons fixé nos objectifs de travail en début d'année.

Notre travail a été divisé en deux grandes parties distinctes. Dans un premier temps, nous nous sommes plongés dans le fonctionnement de l'application ainsi que dans le cadre théorique utilisé dans la partie scientifique du code. Ce travail est repris dans le Chapitre 2. Nous avons poursuivi avec une analyse comparative des logiciels similaires à `riverApp`. Il s'agit notamment du projet `PyOrc` qui s'avère être un concurrent très proche disposant de la même philosophie open-source. Ce travail de comparaison est présenté en détail dans le Chapitre 3. La conclusion du chapitre indique les scénarios futurs de l'application face au constat dressé. Dans le Chapitre 4, nous avons donc entrepris de fusionner le projet `PyOrc` au projet `riverApp`. C'est dans ce chapitre que nous détaillons l'implémentation du code de la nouvelle version de `riverApp`, ainsi que son paradigme de pipeline.

Nous entrons maintenant dans la deuxième grande partie de notre contribution au projet. Nous avons décidé d'implémenter une détection automatique des 4 balises présentes sur la vidéo en entrée. Le Chapitre 5 est subdivisé en plusieurs sections. Nous y décrivons successivement la recherche d'un algorithme approprié, l'explication de notre implémentation, pour finir par les performances et les limites de cette dernière. Ce mémoire aboutit à son terme avec le Chapitre 6, où nous détaillons le projet `riverApp` tel que nous le transmettons après nos contributions. De plus, nous y exposons une multitude de pistes d'amélioration qui pourront servir de point de départ pour de futures recherches prometteuses.

1.3 Cadre de travail

1.3.1 Travaux préexistants sur `riverApp`

Le projet `riverApp` a été créé au sein de l'UCLouvain notamment dans le cadre des mémoires [43], [16] et [13]. Ce projet est une application développée en `Python` qui, à l'aide d'une interface graphique, permet à un utilisateur de calculer le débit d'eau d'un cours d'eau.

Cette application nécessite d'avoir une vidéo de l'écoulement du cours d'eau ainsi que la mesure de la bathymétrie d'une section du cours d'eau. De plus, la prise de vidéo doit respecter certaines contraintes assez précises sans quoi la valeur de débit calculée risque d'être erronée. Une présentation plus complète du projet `riverApp` est faite dans le Chapitre 2.

1.3.2 Collaboration via `Git`

Le projet `riverApp` nécessite encore de nombreuses modifications avant d'être totalement opérationnel et robuste. Pour cette raison, d'autres contributions devront être apportées. Dans ce contexte, le travail collaboratif sur un même projet nécessitait l'utilisation d'un répertoire `Git`. Ce répertoire partagé nous a permis d'apporter des modifications sur le code tout en gardant une trace écrite des modifications.

En parallèle de notre travail, Arnaud Meunier, étudiant à l'EPHEC, a également contribué au projet. En effet, son rôle était de travailler sur l'interface graphique qui accompagne `riverApp`.

Chapitre 2
État de l'art

Le but de ce chapitre est, dans un premier temps, de mettre en lumière différents logiciels de calcul de débit d'eau dans une rivière. Au vu du nombre de logiciels existants, cette partie a également pour but d'appuyer les arguments utilisés dans la Section 1.1 pour démontrer l'intérêt réel du domaine de recherche. Ensuite, les avantages et inconvénients des logiciels utilisés dans le cadre de ce mémoire seront également présentés.

Ce chapitre présente quelques outils de calcul de débit dans une rivière. Certains de ces outils ont été utilisés dans le cadre de notre travail et seront présentés plus en profondeur dans les Section 2.2, Section 2.3 et Section 2.4. Ces trois outils utilisent des méthodes de calcul non-intrusives et plus précisément des analyses PIV (Particle Image Velocimetry). Cette méthode de calcul se base sur une vidéo de l'écoulement de la rivière et est présentée dans la Section 2.1.

Ensuite une liste non-exhaustive d'autres outils développés partout dans le monde sera présentée dans la Section 2.5. Ces derniers n'ont pas été utilisés et leur compréhension est donc plus limitée. Le but de leur brève introduction dans ce chapitre est simplement de montrer l'intérêt du domaine de recherche par le fait qu'il existe un grand nombre de ces softwares. Certains de ces logiciels de calculs sont gratuits et libres d'accès alors que d'autres sont payants.

2.1 Calcul de débit via une analyse PIV

2.1.1 Calcul de débit dans une rivière

Pour calculer un débit dans un cours d'eau, il faut obtenir la vitesse de l'écoulement ainsi que la section traversée. La Figure 2.1 illustre les données nécessaires au calcul du débit dans un cours d'eau.
Il est important de noter que la Figure 2.1 est une simplification d'un cas réel

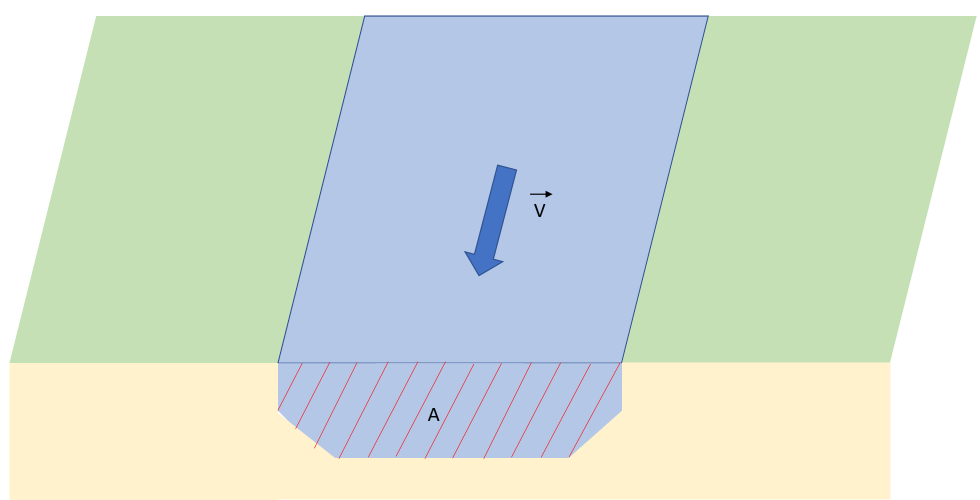


FIGURE 2.1 – Illustration des données nécessaires au calcul de débit dans une rivière. La vitesse d'écoulement est notée $\overrightarrow{\mathrm{V}}$ et l'aire traversée est notée A.

où on considère que la vitesse d'écoulement de l'eau est identique sur toute l'aire traversée.
Dans ce cas, le calcul du débit d'eau à travers cette section du cours d'eau, noté Q, se résume à la formule stipulée dans Eq. (2.1). Notons que la vitesse, V, est calculée en $[m/s]$ alors que l'aire de la section traversée, A, est calculée en $[m^2]$.

$$Q = V \cdot A \quad [m^3 s^{-1}] \tag{2.1}$$

2.1.2 L'analyse PIV single-pass

Objectif

Le but de l'analyse PIV est de calculer un champ de vitesse entre deux images. Autrement dit, l'objectif est de trouver le déplacement réalisé par certains motifs entre deux images consécutives dont on connaît l'intervalle de temps qui les sépare. L'idée est illustrée par la Figure 2.2.

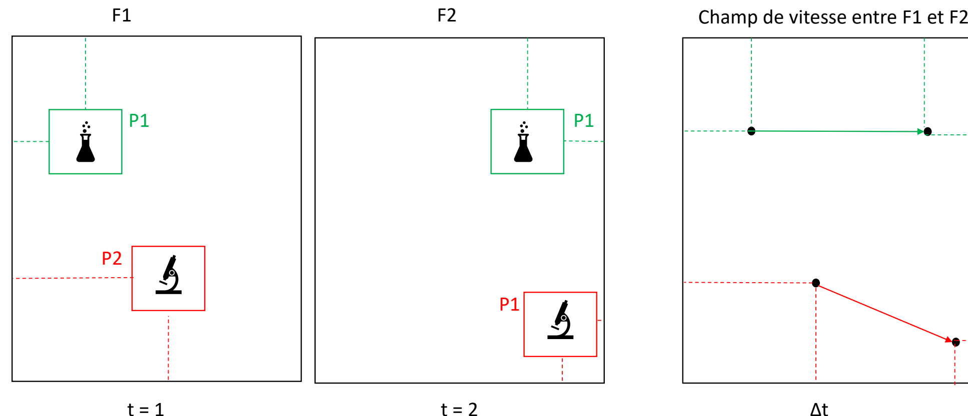


FIGURE 2.2 – Représentation schématique du processus d'analyse PIV. Les images F1 et F2 représentent la même région filmée à des temps différents. Le troisième encadré représente à nouveau la même région où les positions des motifs P1 et P2 ont été retrouvées aux différents temps et reportées, ce qui permet alors de déduire le déplacement des motifs pendant une durée Δt.

Les étapes détaillées

Le processus de calcul du champ de vitesse requiert l'utilisation de techniques particulières décomposées en sous-étapes présentées ci-dessous. Dans l'application `riverApp`, la librairie Python `openPIV` est utilisée pour calculer le champs de vitesse. Pour cette raison, les étapes ci-dessous sont celles implémentées avec cette librairie [44].

1. Sélection de deux images (disons A et B), qui correspondent à la même région enregistrée à deux instants différents.

2. Segmentation de l'image en petites sous-régions appelées fenêtres d'interrogation. Typiquement un petit carré de 16x16 pixels ou 32x32 pixels. À partir de maintenant, on va considérer une seule fenêtre d'interrogation sur l'image A (appelons-la IA).

3. Calcul du déplacement le plus adéquat entre FA et FB. Le déplacement le plus adéquat est déterminé par le maximum de la fonction de cross-corrélation. L'application de la fonction de cross-corrélation sur deux images est détaillée dans le paragraphe suivant.

4. Calcul du vecteur vitesse correspondant à IA grâce au déplacement idéal et au Δt.

5. Répétition du processus pour toutes les sous-régions de l'image. Ce qui permet de déterminer un champ de vitesse entre les images A et B.

6. Répétition du processus pour toutes les paires d'images disponibles.

7. Moyenne de tous les champs de vitesses calculés.

Fonction de cross-corrélation entre deux images

Afin de simplifier la compréhension, il est préférable de commencer par illustrer la fonction de cross-corrélation avec une fonction discrète à une dimension. La fonction est donnée par Eq. (2.2) où f et g sont les deux fonctions étudiées, n représente le décalage entre les deux fonctions et N est le nombre de points du domaine des fonctions. Notez que $\overline{f}$ réfère au complexe conjugué de f.

$$(f \star g)[n] \triangleq \sum_{m=0}^{N-1} \overline{f[m]} g(m+n) \tag{2.2}$$

La fonction de cross-corrélation mesure la similitude entre deux signaux en les comparant pour tout décalage possible.

Prenons deux signaux discrets comme illustrés sur Figure 2.3. Comme les deux signaux sont nuls partout sauf sur une seule entrée, si les signaux ne sont pas alignés [1], la cross-corrélation sera nulle.

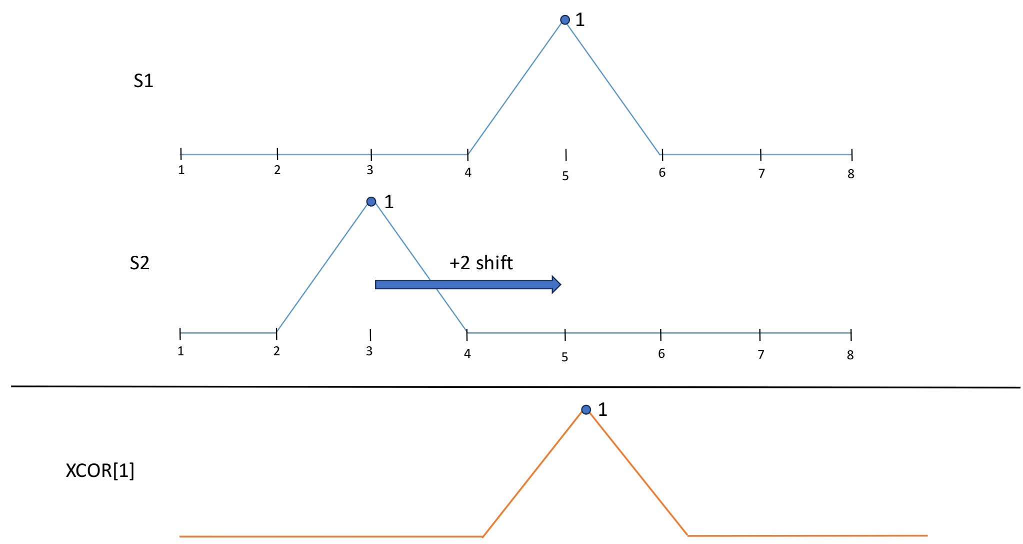


FIGURE 2.3 – Illustration du calcul de la cross-corrélation (XCOR) entre les signaux S1 et S2. La cross-corrélation évaluée pour un shift de +2 vers la droite du signal S2 mène à une cross-corrélation non-nulle.

On observe alors que pour deux signaux, identiques à un shift près, la cross-corrélation sera maximale pour la valeur de n correspondante exactement au shift entre les deux signaux.

En utilisant ce fait, on peut calculer le décalage qui existe entre deux signaux en maximisant la cross-corrélation entre ces deux signaux.

Dans le cas d'une fonction de cross-corrélation entre deux images, le procédé est le même. La fonction de cross-corrélation sera en trois dimensions comme illustrée sur Figure 2.4.

1. Dans le sens que les entrées non-nulles de S1 et S2 sont à la même abscisse

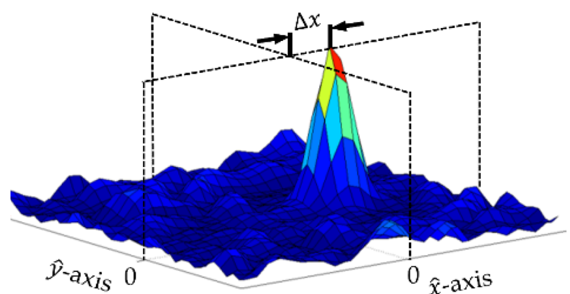


FIGURE 2.4 – Fonction de cross-corrélation entre deux images. Le maximum de corrélation est en Δx ce qui correspond au décalage entre les deux images. Figure issue de [51].

2.1.3 Pre-processing

Pour pouvoir appliquer l'analyse PIV, certaines contraintes par rapport aux images entrées dans la procédure doivent être respectées. Ces contraintes ne sont en pratique que rarement (voire jamais) respectées lors d'une prise de vidéo. Dans cette optique, il est donc nécessaire de prévoir des étapes de rectification des images données. Ces étapes sont appelées pre-processing et sont détaillées dans les paragraphe suivants. Il est important de noter que ces différentes étapes ne sont pas forcément à appliquer dans l'ordre précis dans lequel elles sont décrites et qu'elles ne sont pas toutes toujours obligatoires.

Stabilisation

La stabilisation a pour objectif de rectifier les mouvements de la caméra. En effet, lors de l'analyse PIV, afin de déterminer le déplacement des motifs, une conversion pixel-mètre est réalisée. Pour que cette conversion ait du sens, il est primordial que chaque pixel corresponde toujours au même emplacement dans l'espace. La Figure 2.5 représente cette stabilisation. Il est important de remarquer que lors de cette étape, les bordures des images sont coupées si elles n'apparaissent pas dans toutes les frames de la vidéo.

Calibration

Cette étape sert à compenser les effets de distorsion des images dus aux propriétés physiques des lentilles des caméras. Encore une fois, lors de l'analyse PIV, une

FIGURE 2.5 – Représentation du mouvement de la caméra et de la compensation qui mène à une vidéo stabilisée. Figure issue de [15]

FIGURE 2.6 – Illustration de la distorsion due à la lentille de la caméra. Sur la gauche, la version rectifiée de la même image. Figure issue de [20]

conversion pixel-mètre est réalisée. Pour que cette conversion ait du sens il faut que la distance réelle correspondante entre chaque pixel soit identique. La Figure 2.6 illustre une image originale et sa version calibrée.

Orthorectification

Cette étape sert à compenser les effets de distorsions des images dus à la perspective liée à l'angle de prise de vue. Pour les mêmes raisons que pour la stabilisation et la calibration, il faut que la conversion pixel-mètre ait du sens. La figure Figure 2.7 illustre ce phénomène.

Application de filtres

Lors de l'analyse PIV, la valeur de chaque pixel va influencer le résultat du calcul du maximum de cross-corrélation. Dans certains cas il est donc préférable

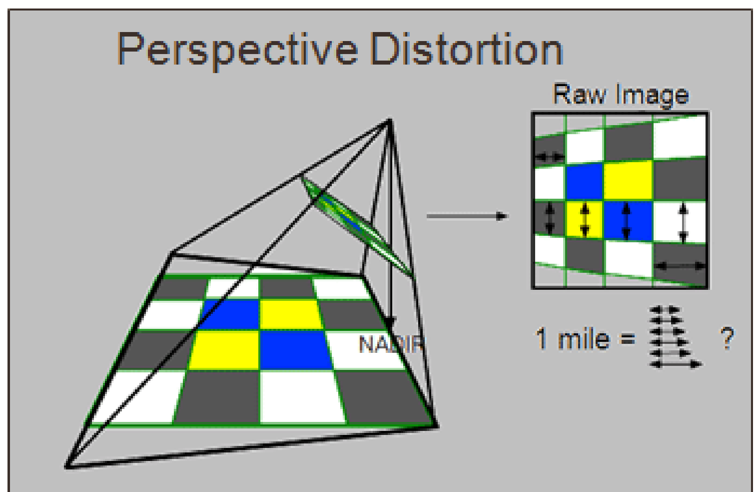


FIGURE 2.7 – Illustration de la distorsion de l'image due aux perspectives liées à l'angle de prise de vue. Figure issue de [23]

d'appliquer un filtre aux images de la vidéo de manière à amplifier les contrastes entre les pixels et rendre plus précis la détection des patterns d'une image à l'autre. Une analyse du choix optimal du filtre en fonction des conditions de prise de vidéo a été réalisée dans [16].

2.1.4 Les données nécessaires

Afin de pouvoir réaliser le calcul de débit en utilisant l'analyse PIV, plusieurs données sont nécessaires. Ces données sont présentées dans les paragraphes suivants.

La vidéo

La vidéo de l'écoulement du cours d'eau étudié est utilisée pour l'analyse PIV. Cette vidéo permet de calculer le champ de vitesse en surface. Notez que si cette vidéo doit passer par des étapes de pre-processing, d'autres paramètres seront nécessaires pour ces étapes préliminaires tels que les paramètres de calibration de la lentille, etc.

La bathymétrie

Après l'analyse PIV, le champs de vitesse est évalué sur une section de la rivière qui est intégrée sur la profondeur de l'eau. Il est donc nécessaire de connaître le

profil bathymétrique de la section évaluée.

Le niveau d'eau

Le niveau d'eau, aussi appelé hauteur d'eau permet de déterminer la profondeur maximale de l'eau au moment où la vidéo est prise.

Remarque sur la distinction entre la bathymétrie et le niveau d'eau :
La bathymétrie et le niveau d'eau ne doivent pas être confondus. En effet, la bathymétrie représente le profil du lit de la rivière alors que le niveau d'eau représente la profondeur maximum de l'eau. On remarque que la bathymétrie restera donc constante alors que le niveau d'eau peut varier lors de crues ou de sécheresses par exemple. Une illustration des deux concepts est présenté sur la Figure 2.8 et met en lumière l'importance du niveau d'eau dans le calcul de débit.

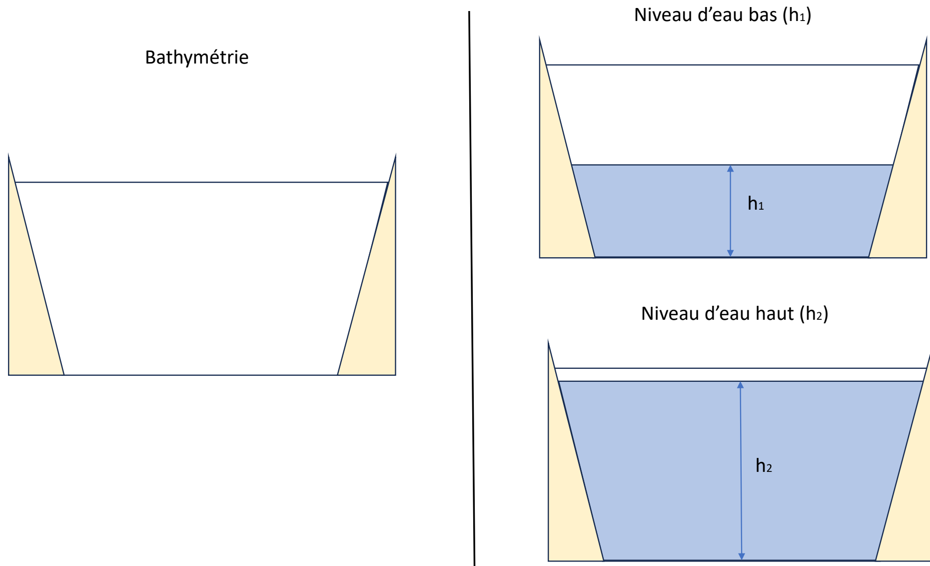


FIGURE 2.8 – Illustration de la différence entre bathymétrie et niveau d'eau. Sur la droite de la figure, deux niveau d'eau (h_1 et h_2) pour une même bathymétrie illustre que la surface mouillée (et donc le débit également) dépend bien du niveau d'eau.

La conversion pixel-mètre

Lors de l'analyse PIV, le procédé permet de déterminer un champ de vitesse exprimé en $\left[\frac{px}{s}\right]$. Pour obtenir une vitesse en $\left[\frac{m}{s}\right]$, il faut convertir les pixels en mètres. Cette conversion peut être réalisée en renseignant des distances réelles en mètres sur une image pour ensuite calculer la distance correspondante en pixel.

Un coefficient de surface

Le profil de vitesse n'est pas constant sur toute la profondeur du cours d'eau. En effet, la Figure 2.9 illustre ce phénomène. Or, lors du calcul de débit, la vitesse calculée grâce à l'analyse PIV est intégrée sur la profondeur de l'eau. Pour corriger cette surestimation de la vitesse, il faut donc déterminer un facteur de correction de la vitesse. Ce facteur de correction dépend du profil de la rivière étudiée.

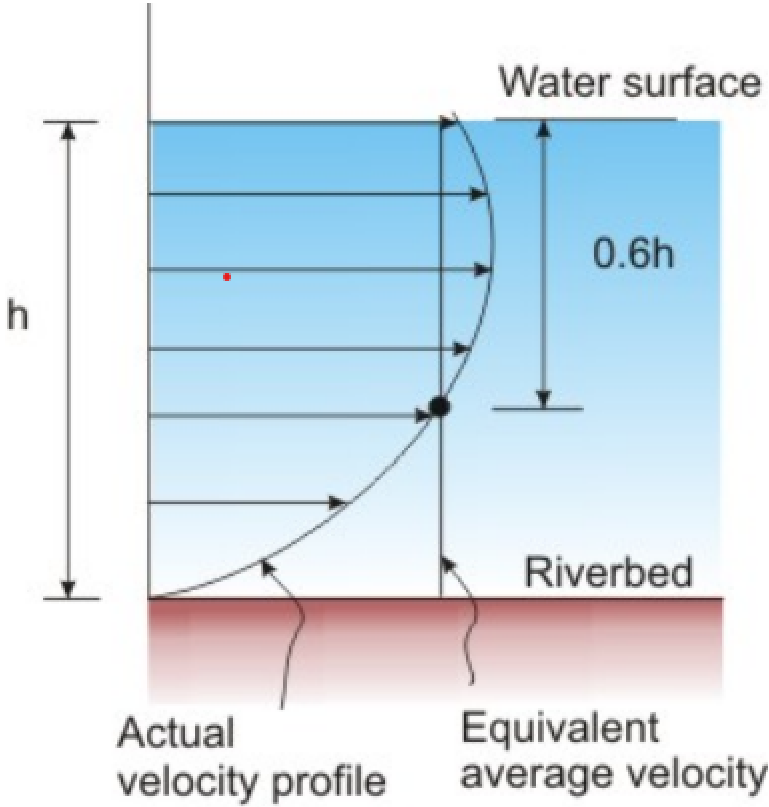


FIGURE 2.9 – Illustration du profil de vitesse en fonction de la profondeur de l'eau. La vitesse de surface est multipliée par un facteur de correction appelé coefficient de surface pour obtenir la vitesse moyenne équivalente. Figure issue de [7]

2.2 `riverApp`

Le projet `riverApp` est une application développée en `python` qui permet de calculer le débit d'eau dans une rivière à partir d'une vidéo de l'écoulement de celle-ci. Cette application prend en charge le pre-processing des images, l'analyse PIV ainsi que certaines étapes supplémentaires permettant de visualiser facilement les résultats de débit obtenus. L'architecture du projet est comme décrite sur Figure 2.10.

2.2.1 Avantages

Un des avantages de `riverApp` est son interface graphique. En effet, l'interface graphique (GUI) permet à l'utilisateur d'interagir avec le programme entre les sous-étapes de calcul. Bien que la robustesse de celle-ci ne soit pas optimale, elle permet tout de même l'utilisation du programme de la `riverApp` par des utilisateurs non-expérimentés. Une présentation complète de cette GUI et de son utilisation est présentée dans le guide d'utilisateur rédigé dans [13].

De plus, `riverApp` inclut un procédé assez complet. En effet, le projet prend en charge toutes les étapes depuis les données "brutes" jusqu'à la valeur de débit dans la rivière incluant des étapes de pre-processing ainsi que l'analyse PIV et enfin des étapes de post-processing.

En outre, `riverApp` se veut devenir un projet open-source. En effet un choix de licence a été fait dans [13] et la mise en open-source du projet y est également discutée. À l'heure actuelle, `riverApp` n'étant pas encore un projet abouti, il demeure néanmoins privé au sein de l'équipe de recherche de l'UCLouvain.

2.2.2 Désavantages

Un désavantage de `riverApp` est qu'il dépend d'autres librairies de `python` pouvant évoluer au cours du temps. En effet, il existe beaucoup de dépendances entre les packages utilisés et l'installation des versions correspondantes n'est donc pas évidente à réaliser. Ces difficultés à l'installation sont notamment dues à un souci de maintenance de la procédure d'installation.

-

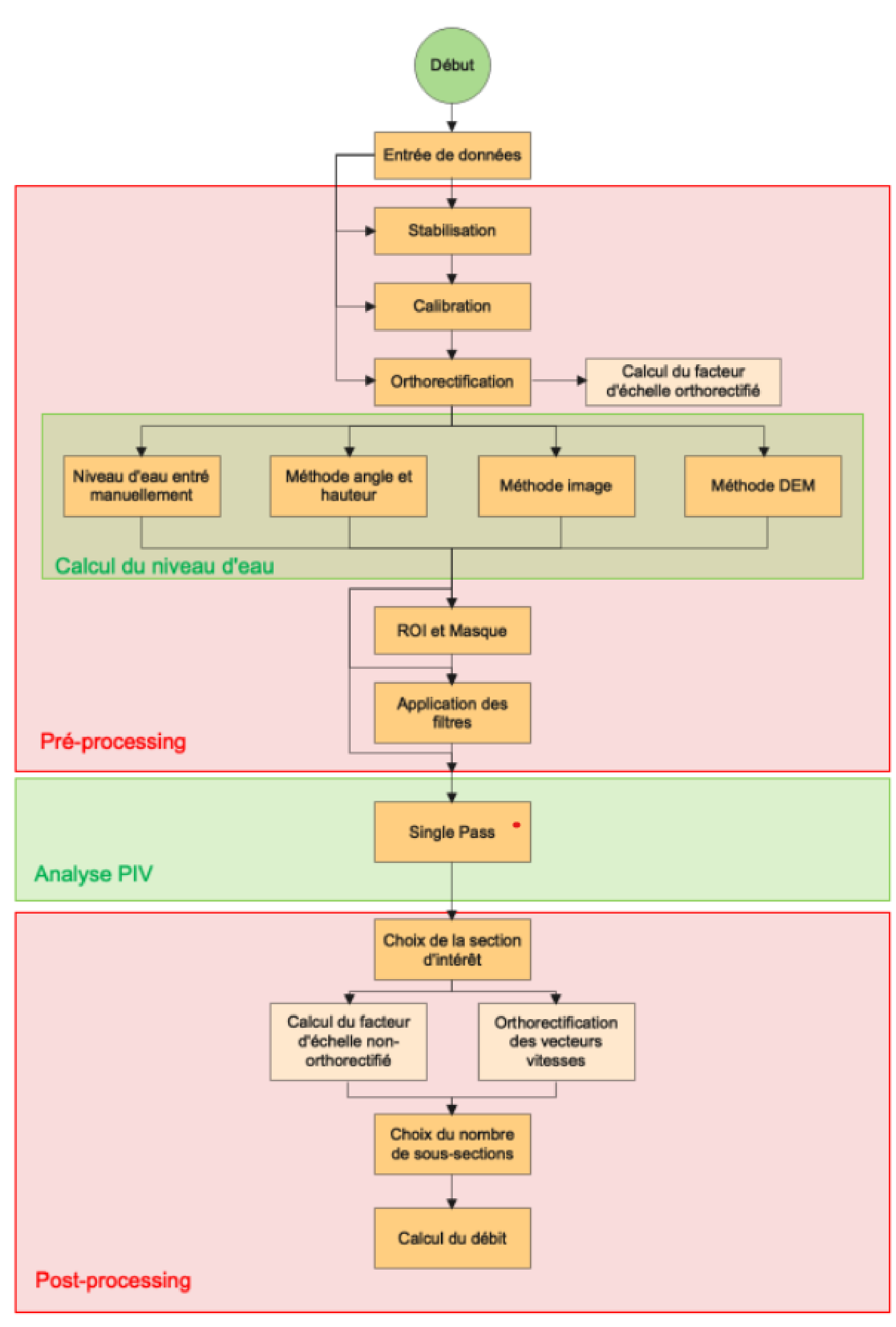


FIGURE 2.10 – Flowchart des sous-étapes de l'application `riverApp` . Figure issue de [16]

2.3 pyOrc

pyOrc (i.e. PyOpenRiverCam) est un programme développé en python qui permet de calculer le débit d'eau dans une rivière à partir d'une vidéo de l'écoulement de celle-ci. Ce programme comporte de nombreuses fonctions permettant la prise en charge des étapes de pre-processing, de l'analyse PIV ainsi que des étapes supplémentaires permettant de visualiser facilement les résultats. En outre, il est important de préciser que ce programme se présente comme une librairie de python.

2.3.1 Avantages

Tout d'abord, le programme pyOrc se présente comme une librairie python. C'est à dire qu'il est possible d'utiliser ce programme simplement en utilisant les fonctions et la documentation à propos celles-ci. Cette documentation est très complète et facile à utiliser. L'API est également documenté. Ces informations peuvent être trouvée directement sur le site internet du projet [9].

Dans la documentation, il est également détaillé comment le projet devrait être installé en fonction de l'utilisation qui va en être faite. Cette installation se fait sans encombre puisque le projet est mis à jour régulièrement. En effet, aucun problème de dépendances ne survient.

Un autre avantage de ce projet est sa complétude. En effet, le programme prend en charge toutes les étapes depuis les données "brutes" jusqu'à la valeur de débit dans la rivière incluant des étapes de pre-processing ainsi que l'analyse PIV et enfin des étapes de post-processing.

De plus, le code est totalement open-source et n'importe qui peut se procurer le programme et même le télécharger sans encombre. Les auteurs sont même ouverts à des modifications de la communauté.

2.3.2 Désavantages

Le principal désavantage de pyOrc est qu'il ne possède pas d'interface graphique ni aucun moyen de communiquer avec l'utilisateur. En effet, l'utilisation du programme doit se faire via un fichier de code python. Un utilisateur n'ayant aucune connaissance en programmation ou dans ce langage ne pourra donc pas utiliser pyOrc facilement.

2.4 Fudaa-LSPIV

Fudaa-LSPIV est une application purement GUI développée par RiverLY, l'équipe de recherche en hydraulique des rivières de l'INRAE (i.e. Institut national de recherche pour l'agriculture, l'alimentation et l'environnement) [27]. Ce logiciel permet de calculer le débit d'eau dans une rivière à partir d'une vidéo de l'écoulement de celle-ci. Ce logiciel inclut différentes étapes de pre-processing ainsi qu'une analyse PIV permettant de calculer le champ de vitesse en surface.

2.4.1 Avantages

Le logiciel `Fudaa-LSPIV` et gratuit et peut être téléchargé par n'importe quel utilisateur.

Dans certaines publications (dont notamment [25], [4] et [10]), on peut constater que le logiciel présente de bons résultats et des valeurs de débit correspondantes à la réalité mesurée.

2.4.2 Désavantages

Le programme `Fudaa-LSPIV` n'est pas encore disponible sur macOS. En effet, une distribution Linux ou Windows est requise pour l'installation.

En outre, le logiciel n'est pas open-source. Cela signifie que bien que son utilisation soit gratuite, le code source n'est pas disponible.

De plus, l'utilisation du logiciel manque de robustesse. En effet, lors de nos essais, nous n'avons pas pu l'utiliser car lors de l'analyse PIV, l'erreur présentée sur la Figure 2.11 est survenue. Cette erreur est une erreur typique de `C++`. Cette erreur ne peut donc pas être résolue par l'utilisateur qui est donc bloqué et ne peut donc plus utiliser l'application.

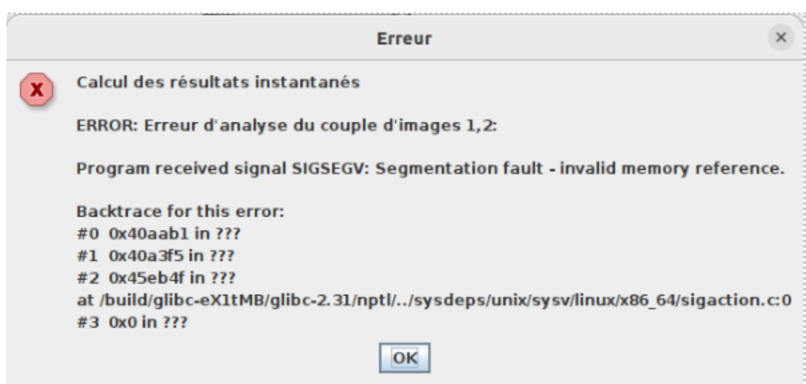


FIGURE 2.11 – Erreur survenue lors de l'analyse PIV effectuée avec le logiciel `Fudaa-LSPIV`.

2.5 D'autres outils de calcul de débit

2.5.1 FluvialGeomorph-toolbox [12]

Cette toolbox offre des outils qui permettent d'estimer l'état de la rivière. Les outils développés utilisent des données récoltées grâce à un LiDAR. Cette toolbox est elle-même inspirée des techniques utilisées dans la toolbox `River Bathymetry Toolkit, RBT` développée par `ESSA Technologies Ltd`. Cette dernière toolbox permet notamment d'estimer la bathymétrie d'une rivière en utilisant un modèle digital d'élévation (DEM).

2.5.2 RIVeR [39]

RIVeR est un logiciel de calcul de débit dans un cours d'eau à partir d'une vidéo du cours d'eau en utilisant une analyse PIV. Attention, ce logiciel est écrit en `Matlab` qui n'est lui-même pas un logiciel gratuit.

2.5.3 TENEVIA CamFlow [52]

Ce logiciel performe une analyse PIV pour estimer le débit d'eau dans une rivière. Ce logiciel est néanmoins payant.

2.5.4 COSH [48]

Ce logiciel permet de calculer le débit d'eau dans une rivière. Il a été développé avec `Matlab` qui n'est lui-même pas un logiciel gratuit. Ce logiciel a été développé pour l'analyse de la production des centrales hydroélectriques.

Chapitre 3

Analyse Comparative des logiciels similaires à `riverApp`

Ce chapitre a pour objectif d'établir une mise au point sur l'état d'avancement de `riverApp`. L'idée est de le comparer avec les alternatives open-source ou privées existantes sur le marché.

Dans un premier temps, nous dressons dans la Section 3.1 un constat après nous être familiarisés avec les différents logiciels existants, y compris `riverApp`. Ensuite, dans la Section 3.2, nous introduisons un protocole de test afin de comparer ces applications de manière objective. Nous abordons également les scénarios possibles envisagés pour le futur de `riverApp` en fonction des résultats de notre comparaison.

Les sections suivantes mettent en pratique ce qui a été présenté. Dans la Section 3.3, nous appliquons ce protocole aux différents logiciels. Pour chacun d'entre eux, nous effectuons des tests sur un jeu de données comprenant plusieurs vidéos prises dans des conditions variées (perspective, luminosité, stabilisation). Nous présentons ensuite les résultats obtenus sous deux angles différents : le temps d'exécution et la précision de l'estimation du débit.

Afin de clore ce chapitre, nous passons en revue dans la Section 3.3.3 les API proposées par les différents logiciels. Cette partie est importante car elle offre un gain de temps qu'elle procure aux futurs développeurs de l'application. Nous présentons pour finir une analyse rapide du niveau de paramétrage possible dans ces logiciels.

3.1 Mise en contexte

Cette section met en contexte la situation actuelle du développement de l'application et explique pourquoi nous remettons en question riverApp en le comparant à d'autres applications similaires. Cela permet au lecteur de comprendre la réflexion derrière notre analyse comparative ainsi que les enjeux qui en découlent.

Au début de l'année, nous avons été introduits au projet riverApp. Il nous a fallu un certain temps pour nous familiariser avec le fonctionnement complet de l'application. Nous avons également rencontré des difficultés lors de l'installation de l'environnement python associé à riverApp. Étant sur un système d'exploitation différent de Windows (Ubuntu), nous avons réalisé que le fichier d'installation n'était pas compatible avec les plates-formes croisées. Cela nous a bloqués pendant une période conséquente et a entraîné un retard dans notre planning de développement de l'application. Heureusement, le guide utilisateur développé par l'équipe de mémorants de l'année précédente nous a permis de lancer l'application de bout en bout sur l'un des jeux de données. Cependant, il n'existe actuellement aucune ressource de type "guide-développeur". La compréhension du code et du fonctionnement général du programme s'est donc avérée être un véritable défi. C'est pourquoi la découverte de l'application PyOrc a remis en question le développement de riverApp.

Notre découverte du programme PyOrc a été fortuite. Nous l'avons trouvé mentionné dans un article sur la construction d'un ensemble de données complet pour analyser des solutions d'analyse PIV par vidéos [40]. Nous sommes tombés sur un poste Linkedin présentant l'application PyOrc et renvoyant à un article démontrant son efficacité étonnante [56]. Nous avons donc entrepris d'en apprendre davantage sur son fonctionnement et avons étudié en détail leur guide pour les développeurs sur leur site officiel [9]. La découverte de PyOrc a été déconcertante pour nous en raison de sa similarité avec le projet riverApp. Les deux utilisent les bibliothèques Python d'OpenCV [35] et d'OpenPIV [44]. Leur objectif est le même : calculer un champ de vitesse à la surface de l'eau à partir d'une vidéo et en déduire le débit dans une section du cours d'eau. Leurs étapes intra_process sont également similaires, comme décrit dans le Chapitre 2.

Suite à ces démarches, nous avons souhaité améliorer l'application riverApp en réimplémentant le code source utilisé dans PyOrc. Ce mode de fonctionnement correspond à nos aspirations et ne relève en aucun cas d'un travail scientifique. C'est pourquoi une partie de notre travail de fin d'étude est consacrée à une analyse comparative entre plusieurs logiciels similaires. L'objectif est d'obtenir un point de vue objectif sur les capacités respectives de ces programmes. À la fin de cette

analyse, nous devrons avoir une vision synthétique des avantages et inconvénients des applications. Ainsi, `riverApp` sera situé par rapport à ses concurrents et replacé dans son contexte.

Nous avons concentré nos recherches sur 3 logiciels qui ont pour but commun de calculer le débit d'un cours d'eau à partir de vidéos. Les logiciels sélectionnés au départ sont : `Fudaa`, `PyOrc` et `riverApp`. Une présentation de ces logiciels peut être consultée dans le chapitre Chapitre 2. Lors de nos mesures sur le logiciel `Fudaa`, nous n'avons malheureusement pas réussi à exécuter l'application de manière consécutive. Nous avons rencontré un problème lié apparemment à une bibliothèque C++. Nous avons donc été contraints de ne pas inclure `Fudaa` dans notre analyse comparative. Notre analyse portera donc sur une comparaison entre les deux logiciels `riverApp` et `PyOrc`.

3.1.1 Scénarios envisagés

À la fin de ce chapitre, nous avons anticipé 4 scénarios, dont un seul sera sélectionné en fonction des résultats de notre analyse :

1. Supposons que `riverApp` se comporte nettement mieux que `PyOrc`. Alors, il ne sera pas nécessaire d'utiliser le code source de `PyOrc`, car cela ne serait qu'une perte de temps. Il est préférable de continuer à améliorer `riverApp` en maintenant son développement actuel.
2. `PyOrc` fournit des résultats intéressants. Nous pouvons alors l'utiliser comme une bibliothèque. Cela nous permet de bénéficier d'une version à jour. Cependant, nous perdons la possibilité de modifier les fonctions `PyOrc` que nous utilisons.
3. `PyOrc` donne des résultats très encourageants. Étant donné que `PyOrc` est un programme open-source, nous pouvons télécharger tout son code source et l'intégrer à `riverApp`. Nous aurions ainsi accès à toutes les fonctions développées pour `PyOrc` dans le projet `riverApp`. Cette option présente l'avantage de nous laisser une liberté totale pour modifier ces fonctions selon nos besoins. Cependant, nous ne pourrions pas suivre les mises à jour futures de `PyOrc` et bénéficier de ses améliorations à venir.
4. Enfin, le dernier scénario serait de collaborer avec l'équipe du projet `PyOrc`. Nous pourrions leur proposer d'ajouter une interface graphique similaire à celle développée pour `riverApp` dans leur projet.

3.2 Protocole comparatif

Dans cette section, nous introduisons un protocole qui va être utilisé afin de comparer les deux logiciels de manière objective. Nous présentons dans un premier temps les deux grands critères qui vont être utilisés pour la comparaison. Ensuite, nous détaillerons le jeu de données sur lequel nous prendrons les différentes mesures. Enfin, nous expliquerons comment les différents résultats seront reportés.

3.2.1 Critères de comparaison

La comparaison entre les performances des fonctions actuelles de `riverApp`, de `PyOrc` et de `Fudaa-LSPIV` va être comparée sur la base des critères suivants :

- précison du débit calculé,
- vitesse de calcul.

Comparaison de vitesse de calcul

Pour la comparaison de vitesse, étant donné que l'interface graphique de `riverApp` et les données entrées manuellement par l'utilisateur sont très chronophage, le temps calculé est uniquement celui pris par l'analyse PIV.

Comparaison du débit

Pour certaines vidéos, le débit exact a été calculé grâce à des méthodes intrusives dans les rivières. Ces données-là sont donc prises comme références. Il va de soi que cette analyse comparative a pour but de classer les projets les uns par rapport aux autres. En aucun cas la précision "absolue" des projets ne sera remise en question lors de cette analyse. De plus, il faut noter que la précision du débit dépend de la section d'intérêt choisie par l'utilisateur.

3.2.2 Sélection des données adéquates

Tout d'abord, il est important de sélectionner un jeu de données qui sera utilisé pour l'analyse comparative. Les données seront réparties en deux classes, une pour l'analyse de vitesse de calcul et une pour l'analyse de la précision des résultats. Certaines données peuvent se retrouver dans les deux classes. À noter que pour les données utilisées pour l'analyse de la vitesse, il n'est pas nécessaire d'avoir de validation du débit.

Dans le cas de la comparaison étudiée dans ce chapitre, toutes les données sélectionnées seront utiles pour les deux critères de comparaison.

3.2.3 Mise à niveau des projets

Pour pouvoir comparer les projets de manière objective, il faut s'assurer qu'ils calculent bien la même chose dans les mêmes conditions. Prenons par exemple le critère du temps d'exécution. Si nous comparons le temps d'exécution de l'analyse PIV de `riverApp` avec le temps d'exécution total de `PyOrc`, cela n'a aucun sens. Nous devons donc dans un premier temps nous assurer que nous comparons bien chaque critère de la même manière.

3.2.4 Rapport des résultats

Les résultats seront reportés dans un tableau mettant en comparaison chacun des projets pour chacune des données. Les deux critères seront répertoriés et placés intelligemment afin de faciliter notre post-analyse.

3.3 Résultats

Cette section est décomposée en deux parties. La première traitera du premier critère abordé plus haut : le temps d'exécution. Ensuite, nous analyserons les résultats obtenus du point de vue du temps d'exécution.

3.3.1 Débit estimé

Voici le tableau récapitulatif des résultats que nous avons menés Tableau 3.1. Dans la colonne des jeux de données, on peut retrouver entre crochets la fréquence des images utilisées. "1/3" indique que nous avons pris une image sur trois de la vidéo utilisée. Ce paramètre va jouer sur la précision du logiciel utilisé. Étant donné que la PIV procède par des analyses comparant plusieurs images, si les informations contenues dans une image sont fortement modifiées dans l'image suivante, le résultat de l'analyse PIV sera approximatif. Nous pouvons ainsi voir le comportement du logiciel face à des jeux de données peu précis. "VGC_1" est l'abréviation de VideoGlobeChallenge, jeu n°1. Il s'agit d'un des jeux de données sur lequel nous avons pu faire tourner les deux logiciels.

Jeu de données	Débit réel $[m^3/s]$	Débit `riverApp` $[m^3/s]$	Débit `PyOrc` $[m^3/s]$
Noirath [1/1]	0.203	0.2285	0.246
VGC_1 [1/5]	1.26	0.0282	0.032
VGC_1 [1/3]	1.26	0.0036	1.081
VGC_1 [1/1]	1.26	0.0031	1.46

Table 3.1 – Tableau récapitulatif des débits estimés

L'analyse comparative des résultats présentés dans le tableau récapitulatif des temps d'exécution (Tableau 3.1) permet de mettre en évidence les performances et l'efficacité des deux logiciels, `riverApp` et `PyOrc`, en termes d'estimation du débit. En observant les valeurs de débit réel et les estimations obtenues par chaque logiciel, il est clair que `PyOrc` fournit des résultats plus précis et proches des débits réels, tandis que `riverApp` présente des estimations moins précises.

Prenons l'exemple du jeu de données Noirath, où le débit réel est de 0.203 m^3/s. `riverApp` estime le débit à 0.2285 m^3/s, tandis que `PyOrc` donne une estimation de 0.246 m^3/s. On peut voir que `PyOrc` se rapproche davantage du débit réel, suggérant ainsi une meilleure précision dans son estimation.

De plus, en examinant les jeux de données VGC_1, on constate que `PyOrc` obtient également des résultats plus précis, quel que soit le paramètre de fréquence des images utilisées. Par exemple, pour une fréquence de 1/3, `PyOrc` estime le débit à 1.081 m^3/s, tandis que `riverApp` donne une estimation de seulement 0.0036 m^3/s. Ces résultats suggèrent que `PyOrc` est capable de mieux traiter les variations d'images et de fournir des estimations plus fiables, même avec des jeux de données moins précis.

En conclusion, d'après l'analyse des résultats, il est évident que `PyOrc` surpasse `riverApp` en termes d'estimation du débit. Ses estimations se rapprochent davantage des débits réels et présentent une meilleure précision, même avec des jeux de données moins précis. Par conséquent, `PyOrc` peut être considéré comme le meilleur choix dans le contexte d'une estimation plus précise du débit.

3.3.2 Temps d'exécution de l'analyse PIV

L'analyse comparative des temps d'exécution présentés dans le tableau récapitulatif (Tableau 3.2) met en évidence les différences de vitesse d'exécution entre les deux logiciels, `riverApp` et `PyOrc`, pour l'analyse PIV. Les temps d'exécution sont mesurés en secondes et ont été calculés à l'aide de la librairie "ProgressBar", fournissant une estimation du temps restant de l'analyse PIV. Un aspect saute vite aux yeux : plus nous avons d'images à analyser, plus l'analyse PIV a un grand temps d'exécution. C'est la conséquence logique du fait que l'analyse PIV compare chaque image une à une. Le temps d'exécution dépend donc directement via un certain facteur du nombre d'images à traiter. C'est pourquoi nous avons ajouté une colonne précisant le nombre d'images qui ont été utilisées en entrée pour l'analyse PIV.

Jeu de données	Images [#]	Temps exec. `riverApp` [s]	Temps exec. `PyOrc` [s]
Noirath [1/1]	125	740	76
VGC_1 [1/5]	25	158	18
VGC_1 [1/3]	42	321	31
VGC_1 [1/1]	125	950	83

TABLE 3.2 – Tableau récapitulatif des temps d'exécution

En observant les valeurs de temps d'exécution, il est évident que `PyOrc` est significativement plus rapide que `riverApp` pour l'analyse PIV. Par exemple, pour le jeu de données Noirath avec une fréquence d'image de 1/1, `riverApp` nécessite 740 secondes pour effectuer l'analyse PIV, tandis que `PyOrc` ne prend que 76 secondes. De manière similaire, pour le jeu de données VGC_1 avec une fréquence

d'image de 1/5, `riverApp` nécessite 158 secondes, tandis que `PyOrc` ne prend que 18 secondes.

Il est important de noter que cette tendance se maintient même lorsque le nombre d'images à analyser augmente. Par exemple, pour le jeu de données VGC_1 avec une fréquence d'image de 1/1, `riverApp` nécessite 950 secondes, tandis que `PyOrc` ne prend que 83 secondes. Ces résultats indiquent clairement que `PyOrc` est plus efficace et rapide que `riverApp` pour l'analyse PIV.

En conclusion, d'après l'analyse des temps d'exécution, il est évident que `PyOrc` est le choix préférable en termes de vitesse d'exécution pour l'analyse PIV. Il est nettement plus rapide que `riverApp`, offrant ainsi une optimisation significative du temps nécessaire pour effectuer l'analyse. Par conséquent, pour des contraintes de temps et une exécution plus rapide de l'analyse PIV, `PyOrc` est clairement le logiciel à privilégier.

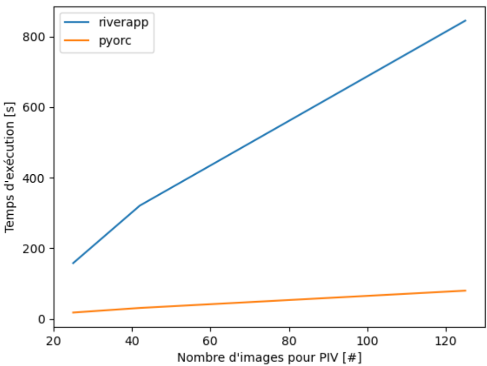


FIGURE 3.1 – Temps d'exécution par rapport au nombre d'images utilisées pour la PIV

Si l'on se base sur le nombre d'images qui ont été utilisées pour chaque jeu de données Figure 3.1, nous pouvons en déduire la complexité temporelle de chaque programmes. On peut en déduire que les deux progammes ont tous les deux une complexité linéaire par rapport au nombre d'images à analyser. Ils sont cependant tous les deux espacés d'un facteur 10. Plus précisément, avec n le nombre d'images pour la PIV,

$$\begin{aligned} \text{Complexité temporelle de la PIV } \texttt{PyOrc} &= \mathcal{O}(0.7n) \\ \text{Complexité temporelle de la PIV } \texttt{riverApp} &= \mathcal{O}(6.9n) \end{aligned} \quad (3.1)$$

3.3.3 Analyse des API et paramètres

Nous terminons ce chapitre avec un rapide aperçu de ce que les deux logiciels proposent au sein de leur écosystème. Tout d'abord, comme dit plus haut, `riverApp` possède actuellement un guide utilisateur développé dans le mémoire [13]. Celui-ci facilite grandement sa prise en main afin de tourner l'application du début à la fin. Cependant, ce n'est pas nécessaire pour des nouveaux développeurs qui ont besoin de comprendre le code de `riverApp` dans son intégralité. C'est à nos yeux l'un des gros enjeux du projet `riverApp`. Les personnes qui travaillent sur le projet changent tous les ans avec chaque année une nouvelle équipe de mémorants. Le travail de compréhension du projet et du code source doit donc être recommencé chaque année et représente sur le long terme un ralentissement au développement de `riverApp`. Côté paramètres, `riverApp` présente un niveau de paramétrage très poussé, ce qui demande de la connaissance pour l'utilisateur. Les paramètres tels que le rapport SNR ou les types de masques à appliquer ont un impact considérable sur le résultat et nécessitent une réflexion attentive lors de leur modification. Si l'utilisateur est un utilisateur avancé, cet aspect devient alors un grand avantage et les résultats obtenus seront raffinés jusqu'à satisfaction de l'utilisateur.

`PyOrc` possède quant à lui une API explicite. Chaque fonction et instance de l'API y est développée en détail. Il propose également un guide développeur comprenant des fichiers notebook expliquant précisément comment fonctionnent les 4 grandes étapes de `PyOrc`. Cela nous a permis de vite comprendre comment l'utiliser et surtout comment l'implémenter dans notre propre code. Le problème se situe plutôt du côté de l'utilisateur. Il n'existe en effet à ce jour aucun guide utilisateur pour utiliser `PyOrc` de manière autonome. Si la personne ne dispose pas de connaissances en informatique, il est impossible pour elle de lancer l'application et de l'exécuter sur un jeu de données. Le manque d'interface graphique ne pallie pas à ce problème et désavantage encore plus l'utilisateur inexpérimenté.

D'un point de vue affichage graphique des résultats, voici deux graphiques (Figure 3.2 et Figure 3.3) qui montrent les résultats de l'analyse PIV sur le jeu de données VGC_1. `PyOrc` a l'avantage de pouvoir afficher les vecteurs calculés de la PIV dans le point de vue de la caméra. `riverApp` possède uniquement le point de vue orthorectifié pour afficher les résultats.

3.4 Conclusion de l'analyse

Sur base des résultats que nous avons obtenus, nous pouvons affirmer que le projet `PyOrc` fournit des résultats plus pertinents que le projet `riverApp`. Nous pouvons le voir sur les vecteurs vitesse à la surface liés à la section d'intérêt sur les images Figure 3.2 et Figure 3.3. `PyOrc` obtient des résultats plus pertinents et a une meilleure compréhension de la vitesse générale à la surface du cours d'eau. De plus, il calcule ces résultats avec une rapidité 10 fois supérieure à celle de `riverApp`, ce qui est précieux lorsque nous souhaitons analyser de grandes quantités de données. Le chapitre suivant expliquera le choix du bon scénario sur base de cette conclusion d'analyse.

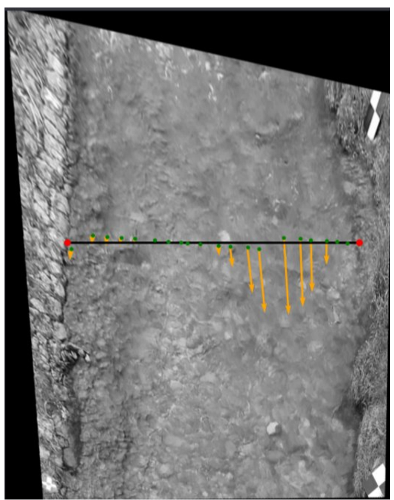


FIGURE 3.2 – plot du résultat final de `riverApp`

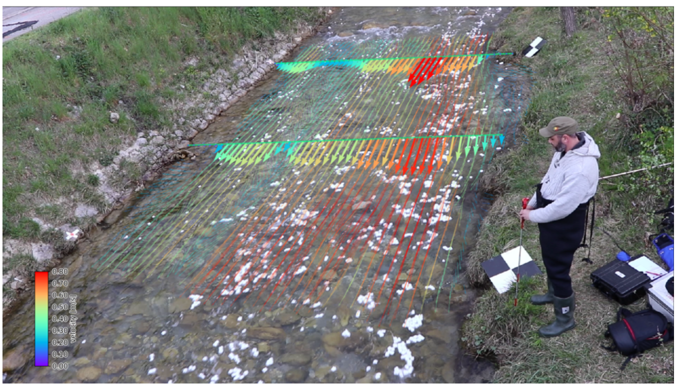


FIGURE 3.3 – plot du résultat final de `PyOrc`

Chapitre 4
Implémentation croisée de pyOrc et de riverApp

Ce chapitre expose notre contribution relative à l'intégration du développement de `PyOrc` au sein du projet `riverApp`. Notre objectif est spécifiquement d'ajouter l'interface graphique développée pour `riverApp` à `PyOrc`. Deux scénarios sont envisagés pour cette intégration. Le premier consiste à créer une branche dérivée du projet `PyOrc` en reprenant leur code source. La deuxième option consiste à collaborer avec l'équipe de `PyOrc` en leur proposant d'intégrer notre propre interface graphique à leur projet. Chacune de ces options présente des avantages et des inconvénients.

Une fois le scénario choisi, nous devons procéder à l'implémentation du code de `PyOrc` en utilisant leur API de manière rigoureuse. Il est primordial de sélectionner un paradigme approprié qui assure à la fois l'efficacité et la rapidité du code, tout en le rendant lisible pour les futurs développeurs qui y travailleront.

Dans ce chapitre, nous détaillerons tout d'abord, dans la Section 4.1, les deux scénarios possibles pour mettre en oeuvre le projet `PyOrc`. Nous analyserons ensuite les avantages et les inconvénients de chacun de ces scénarios, regroupés dans le Tableau 4.1. Par la suite, dans la Section 4.3, nous expliquerons nos choix d'implémentation actuels du code de l'application, ainsi que le paradigme de code utilisé. Enfin, une présentation détaillée du fichier Python principal sera proposée dans la Section 4.3.2.

4.1 Scénarios possibles par rapport à un pull sur le projet pyOrc

Suite à notre analyse comparative Chapitre 3, nous avons décidé d'intégrer le développement de PyOrc au projet riverApp. En réalité, nous avons plutôt procédé par l'inverse. Nous allons en effet intégrer à PyOrc l'interface graphique qui a été développée pour riverApp. Il existe deux scénarios possibles pour s'appuyer sur le développement de PyOrc. Afin de prendre la décision la plus pertinente, voici un tableau comparatif Tableau 4.1 qui détaille ces deux scénarios et explique pour chacun leurs avantages et inconvénients.

Après concertation avec nos encadrants et avec Arnaud, nous avons choisi le premier scénario : Fork du projet PyOrc. D'un point de vue développement scientifique, cette option est très intéressante. Elle nous permet de garder la main mise sur le code source et d'ajouter autant de fonctionnalités que nous le souhaitons. C'est d'ailleurs ce que nous entreprendrons dans la suite de notre travail avec l'ajout de la détection automatique des ground control points (Chapitre 5).

Toutefois, cela n'exclut pas la possibilité de rentrer en contact avec l'équipe de PyOrc. Nous pourrions leur proposer une interface graphique une fois celle-ci développée. Cela nous permettrait d'utiliser PyOrc avec une interface user-friendly tout en ayant les mises à jour futures du projet et ainsi mélanger tous les avantages cités dans la comparaison.

Table 4.1 – Tableau comparatif entre les deux scénarios possibles

Scénarios	Fork du projet `PyOrc`	Collaboration avec l'équipe de `PyOrc`
Description du scénario	Reprendre toute l'implémentation de `PyOrc`, copier l'intégralité de leur code.	Commencer un partenariat avec l'équipe de `PyOrc` pour leur proposer une interface graphique qui accompagne leur projet.
Modifications possibles	Nous permet d'avoir la main mise sur tout leur code et de le modifier comme nous le souhaitons. Possibilité d'ajouter des arguments, modifier des paramètres sur certaines fonctions.	La partie scientifique ne peut pas être modifiée. Pour ajouter une fonctionnalité, il est nécessaire que l'équipe de `PyOrc` valide au préalable le changement.
Mises à jour	Une fois le projet `PyOrc` forké, nous avons une copie du projet entre nos mains. Celle-ci ne sera pas mise à jour s'il y a des changements dans le futur. Pour obtenir le code récent, il est nécessaire de forker à nouveau leur répertoire et ensuite de le modifier pour qu'il corresponde à notre code actuel.	Chaque mise à jour sera automatiquement téléchargée et intégrée au code actuel sans qu'aucun changement ne soit nécessaire.
Risque à long terme	Imaginons que le code de `PyOrc` change de façon drastique en une version 2.0. Il est alors nécessaire de modifier toute notre implémentation afin qu'elle corresponde à cette nouvelle version.	`PyOrc` est pour l'instant en licence open-source (AGPL-3.0). S'ils décident de modifier leur utilisation actuelle afin de vendre `PyOrc` comme un produit/service, nous perdrons les futures mises à jour du projet et l'accès à leur code source.

4.2 Fusion des deux projets

4.2.1 Quelles fonctionnalités de `pyOrc` sont conservées

Le projet `PyOrc` a l'énorme avantage d'avoir été conçu par des chercheurs experts de l'université TU Delft. C'est donc un projet qui s'appuie sur une base théorique robuste et qui a été testé et validé sur une série de différents exemples. Son implémentation permet une paramétrisation poussée de chaque procédé et permet d'utiliser un grand nombre de masques à appliquer sur le résultat de l'analyse PIV. D'un autre côté, `PyOrc` ne possède à l'heure actuelle aucune interface graphique. Il est donc nécessaire de l'utiliser soit via 3 commandes génériques de pyorc dans son terminal, soit en utilisant son API. C'est le seul point noir de `PyOrc`, un utilisateur qui n'a pas de notion d'informatique ne sera pas en capacité de pouvoir l'utiliser. Nous allons donc reprendre toute la partie de l'API de `PyOrc` avec le code source de l'API. De cette manière, nous pourrons toujours modifier selon nos envies les différentes fonctions implémentées par `PyOrc`. Cette partie scientifique sera implémentée par nous ainsi que les futurs développeurs.

4.2.2 Quelles fonctionnalités de `riverApp` sont conservées

Le principal attrait de l'application `riverApp` est son interface graphique. Elle permet un environnement facilement accessible pour un utilisateur qui n'a jamais expérimenté du code informatique. Pour nous, 3 utilisations ont été suffisantes pour nous permettre d'utiliser l'application sur d'autres jeux de données. Cela nous a également permis de comprendre tous les sous-menus et les paramètres correspondants. C'est le principal avantage de `riverApp`, car sa partie scientifique est moins claire et donne de moins bons résultats. Nous garderons donc l'ergonomie de l'interface graphique de `riverApp`. Cependant, nous comptons changer quelque peu le design de l'application afin de la rendre plus agréable à utiliser. La partie graphique sera implémentée par Arnaud Meunier.

4.3 Fonctionnement général et paradigme du code actuel

Nous avons dans un premier temps utilisé l'API de `PyOrc` dans un notebook de `python` afin de nous familiariser avec celle-ci. Une fois testée sur plusieurs jeux de données, nous l'avons implémentée de manière propre dans plusieurs fichiers Python [42]. Nous pouvons voir l'application actuelle comme un pipeline comprenant 4 étapes distinctes. Ces étapes sont détaillées et expliquées dans le graphique suivant.

1. Cam Creation

Dans cette première étape, l'utilisateur crée l'instance de caméra. Il doit pour ce faire renseigner plusieurs informations. Tout d'abord, il doit préciser la frame précise sur laquelle il pointera les différents points. Ensuite, nous devons obtenir 4 paires de points pour calculer la matrice d'orthorectification. L'utilisateur renseigne d'abord les 4 points sources, donnés dans le référentiel de l'image en pixels. Ensuite, il indique les 4 points de destination donnés dans le référentiel local. Il ne reste plus qu'à lister 4 nouveaux points qui délimitent les bords de la zone d'intérêt. Finalement, l'utilisateur indique la hauteur de l'eau dans le référentiel local au moment où la vidéo a été enregistré. Toutes ces données sont enregistrées dans un fichier JSON nommé "cam_creation.json" avec le bon format pris en charge par `PyOrc`.

2. PIV Process

C'est dans cette seconde étape que l'analyse PIV est effectuée. L'utilisateur doit d'abord renseigner la frame de début (start_frame), la dernière frame (end_frame) et la fréquence (freq). La fréquence indique ici le nombre de frames que nous prenons en compte. Par exemple, une valeur de 3 signifie que nous ne prenons qu'une frame sur 3. Avec ces données, nous créons une instance de la classe vidéo de `PyOrc`. Nous pouvons ensuite parcourir toutes les frames de start_frame à end_frame en respectant la fréquence. Ensuite, nous ajoutons du contraste en normalisant les frames. Cela permet une meilleure analyse PIV et donne de meilleurs résultats. Il reste ensuite à projeter toutes les frames dans l'espace ortho-rectifié. Une fois cela fait, il est temps de lancer l'analyse PIV comme expliquée dans la Section 2.1. Ce processus est un des facteurs les plus limitants du programme d'un point de vue complexité temporelle, nous avons donc rajouté une "progress_bar" qui indique le temps restant pour finir l'analyse. Une fois les calculs terminés, nous stockons le résultat dans le fichier nommé "piv.nc".

3.Masking PIV plots

L'étape 3 permet à l'utilisateur d'appliquer une série de masques sur le résultat de l'analyse PIV. `PyOrc` propose une variété de masques qui opèrent à différentes échelles, allant de l'action la plus locale jusqu'à l'analyse spatiale et temporelle à grande échelle. Selon l'équipe de `PyOrc`, il est recommandé d'appliquer les multiples masques dans cet ordre de grandeur. Après l'application des masques sur le champ de vecteurs vitesse, nous générons une représentation graphique de celui-ci dans un espace ortho-normé, ainsi qu'une autre représentation dans l'espace de la caméra, afin d'obtenir une vision plus intuitive du champ. C'est principalement à cette étape que l'utilisateur consacrera le plus de temps pour choisir la série optimale de masques afin d'obtenir le résultat souhaité. Il est donc essentiel de sauvegarder préalablement le résultat de l'analyse PIV dans un fichier afin d'éviter de devoir le recalculer plusieurs fois.

4.Transect Creation

Lors de cette étape finale, l'utilisateur commence par renseigner les deux points situés de part et d'autre de la rive, délimitant ainsi la bathymétrie. Ensuite, il renseigne le fichier correspondant à la description de la bathymétrie. Le format de ce fichier est spécifié dans l'Annexe B et est le même que celui utilisé pour `riverApp`. Par la suite, nous générons deux représentations graphiques regroupant dans une même image le champ de vecteur PIV avec les masques appliqués, superposé à la transect contenant les vecteurs moyens calculés. L'une des représentations est située dans l'espace ortho-normé, tandis que l'autre est dans l'espace de la caméra (monde 3D). Pour finir, l'application affiche le débit estimé pour cette transect.

4.3.1 Avantage du paradigme de pipeline

Le lecteur attentif remarquera qu'à chaque étape, nous sauvegardons les données intermédiaires dans un fichier. Cela nous permet de démarrer l'application à l'étape précise souhaitée. Ce mode de fonctionnement apporte un gain de temps considérable, car l'utilisateur n'a pas besoin de relancer toutes les étapes préalables pour modifier l'un des paramètres. Cela s'applique particulièrement au changement des masques, une étape qui implique plusieurs essais-erreurs. Recalculer l'analyse PIV serait contre-productif et entraînerait une perte de temps inutile. L'utilisateur

indique l'étape à laquelle il souhaite démarrer le programme en utilisant le paramètre "step=k" dans le fichier "main.py" (où k est le numéro de l'étape). Cependant, si l'utilisateur souhaite effectuer un seul run complet sur toutes les étapes de l'application, enregistrer et charger les fichiers entre chaque étape pourrait entraîner un ralentissement. Heureusement, les divers formats de fichiers utilisés (.json pour la configuration de la caméra et .nc pour les analyses PIV) sont compacts, ce qui signifie que leur stockage ne constitue donc pas une contrainte temporelle significative par rapport au temps d'exécution des étapes.

4.3.2 Code du `main.py`

Voici le code du fichier main. On peut visualiser comment le pipeline est configuré grâce à la variable step. On peut également vérifier qu'à chaque sous-étape, le fichier intermédiaire sera sauvegardé même si l'application n'est lancée que pour un seul run depuis le début. Toutes les sous-fonctions sont implémentées dans les 4 fichiers Python différents contenus dans le dossier `dev/` (voir Section 6.1). Ces 4 fichiers correspondent respectivement aux étapes de l'application. Ces fichiers contiennent toutes les fonctionnalités essentielles qui font appel de manière transparente à l'API de `PyOrc`.

```
1
2  print("[INFO] Loading packages")
3  import dev.cam_creation as cam_creation
4  import dev.stab_and_piv as stab_and_piv
5  import dev.mask_and_plot_piv as mask_and_plot_piv
6  import dev.transect as transect
7
8
9  [...]
10
11 if __name__ == "__main__":
12
13     #todo : Add a window to enable the user to set the step
14
15     # step at which to start the process
16     # 1 = cam creation, 2 = piv process, 3 = mask
           application, 4 = transect choice
17     step = 1
18
19 [...]
20
```

```python
############
# RiverApp processes
############

    if step == 1:
        print("[INFO] Camera creation")
        cam_config = cam_creation.cam_create(video,
            directory, dimension, water_level)
    else:
        cam_config = pyorc.load_camera_config(directory + "
            cam_config.json")

    video.camera_config = cam_config

    if step <= 2:
        print("[INFO] Processing PIV")
        stab_and_piv.process_piv(directory, video)

    if step <= 3:
        print("[INFO] Applying masks and plotting results")
        ds = xr.open_dataset(directory + "piv.nc")
        mask_and_plot_piv.mask_and_plot(directory, ds,
            video)

    if step <= 4:
        print("[INFO] Compute transect")
        ds = xr.open_dataset(directory + "piv_masked.nc")
        transect.transect(ds, video, directory,
            bathy_filepath)
```

Code 4.1 – Code source du fichier `main.py`. La fonction principale du projet riverApp est structurée en pipeline composée de quatre étapes clés, après avoir chargé la vidéo : (i) la caméra doit être calibrée ; (ii) l'analyse PIV est réalisée ; (iii) le résultat est filtré ; (iv) le débit est estimé.

Chapitre 5
Détection automatique des balises

Ce chapitre présente la deuxième partie de la contribution au code réalisée dans le cadre de ce TFE. L'objectif est d'automatiser l'étape de détection des balises, appelées *ground control points* (GCP). Lors de plusieurs étapes du pre-processing, ces balises sont requises, notamment pour l'orthorectification et la définition de la région d'intérêt.

Lors des étapes de pre-processing, l'utilisateur doit renseigner les points de référence qui seront utilisés par les différents algorithmes de calcul. Dans le cas des exemples étudiés pour ce projet, ces points de référence sont les balises en damiers placées sur les rives du cours d'eau (le visuel est disponible dans l'Annexe A). L'objectif de l'automatisation présentée dans ce chapitre est donc de reconnaître automatiquement ces balises sur l'image grâce à une routine de reconnaissance de pattern. Grâce à cette reconnaissance automatique, l'utilisateur n'aura plus besoin de renseigner les points manuellement et certaines étapes pourront alors "disparaître" du pre-processing aux yeux de l'utilisateur.

Ce chapitre est structuré en cinq sections. Tout d'abord, les raisons du choix d'automatiser la détection des GCP sont présentées dans la Section 5.1. Ensuite, une brève mise en contexte de la reconnaissance de pattern sur une image est faite dans la Section 5.2. Par la suite, un état de l'art des techniques connues pour atteindre cet objectif est présenté dans la Section 5.3. Suite à cet état de l'art, des tests sont menés et les résultats sont présentés dans la Section 5.4. Finalement, une routine est choisie pour être implémentée et cette implémentation est présentée dans la Section 5.5.

5.1 Raisons de l'automatisation et attentes liées à cette amélioration

5.1.1 Simplification du processus pour l'utilisateur

L'objectif premier de cette amélioration est de simplifier l'utilisation de l'application `riverApp` par l'utilisateur. En effet, initialement, comme décrit dans le Guide d'utilisation de l'application ([13]), l'utilisateur devait passer par au moins 6 étapes, sans compter les entrées de données, avant que l'analyse PIV ne se lance. Grâce à cette automatisation du processus, l'utilisateur verra 4 de ces 6 étapes disparaître [1]. Cela simplifie d'une part la compréhension de l'application par l'utilisateur et d'autre part l'implémentation de l'interface graphique pour le développeur. De plus, cette automatisation assure une invariance des points détectés lors de plusieurs tests sur le même ensemble d'images, ce qui n'était pas le cas lorsque l'utilisateur devait cliquer sur les balises à l'aide de sa souris.

5.1.2 Détection des balises standards

Les attentes de la routine qui sera implémentée sont de repérer les 4 balises positionnées au bord du cours d'eau filmé. Comme la routine est une première version, les attentes de robustesse ne sont pas encore élevées. En effet, bien que le processus tel qu'expliqué dans la Section 5.5 soit déjà avancé, aucune analyse de sensibilité n'a été effectuée. Par exemple, une forte variation de luminosité ou de qualité d'image pourrait dégrader les performances.

1. À savoir, la détection des points de référence, l'application du masque, la détermination de la ROI et la découpe des parties superflues de la ROI

5.2 Mise en contexte

Dans un premier temps, il convient de fixer le contexte et de clarifier le vocabulaire. Les outils et méthodes utilisés dans ce chapitre appartiennent au vaste domaine de la vision par ordinateur (*computer vision* ou *CV*). La *CV* est un champ de recherche qui a pour objectif de reproduire, grâce à des algorithmes de traitement d'image, les interprétations d'images faites par le cerveau humain.

Parmi ces interprétations, celle étudiée dans ce chapitre est la reconnaissance de motifs (*pattern recognition*). Cette capacité permet de repérer une sous-région connue dans une image. Par exemple, sur la figure Figure 5.2, pour le cerveau humain, il est trivial de trouver la région de l'image où se trouve la tortue. Pour un ordinateur, c'est une tâche plus compliquée qu'il n'y paraît.

(a) issu de [18]

(b) issu de [14]

(c) issu de [17]

FIGURE 5.2 – Images de la statue de Jan Fabre vue sous différents angles. Si l'image (a) est prise comme référence, il est aisé à l'oeil nu de trouver les régions dans les images (b) et (c) où la statue apparaît. Cette tâche est néanmoins beaucoup moins facile pour un ordinateur.

Dans le cadre du projet `riverApp`, les recherches vont être concentrées sur les cas appliqués au image 2D [2]. Cette branche s'appelle la concordance d'image (*image matching*) et est un sujet de recherche très actuel [3].

L'objectif de l'*image matching* est de trouver des occurrences d'un objet (appelé *template*) dans une image. Pour trouver cet objet, il faut au préalable avoir décrit des caractéristiques (*features*) sur l'objet initial.

A cette fin, un détecteur de caractéristiques (*features detector*) doit être choisi. Ce *features detector* permet de décider si une certaine caractéristique est présente

2. Notez que l'exemple présenté sur la Figure 5.2 est un exemple en 2D mais la *pattern recognition* peut s'appliquer à un nuage de points en 3D également.

3. En effet, plus de 900 articles ont été publiés depuis 2020 sur Google Scholar avec les termes "Image Matching" dans le titre.

ou non sur le pixel analysé. Ensuite, si une caractéristique est bien présente, il faut la décrire. Cette partie est réalisée par le descripteur de caractéristique (*features descriptor*). Notons qu'en pratique, les descripteurs et les détecteurs sont souvent des routines confondues.

Dès lors, chaque pixel de l'image se verra attribuer, par le descripteur, un vecteur de caractéristiques (*features vector*) pouvant être arbitrairement grand. En effet, il existe une grande variété de différentes *features* qui peuvent être attribués à chaque pixel.

Il existe également un grand nombre de différents *features descriptors*. Certains de ces descripteurs sont présentés dans les sections suivantes. Il faut noter que le choix de *features* étudiés, et donc du descripteur dépend de l'algorithme de reconnaissance qui sera utilisé.

5.3 État de l'art des techniques de pattern matching

5.3.1 Concordance de template (*template matching*)

L'objectif du *template matching* est de pouvoir trouver un certain *template* dans différentes images plus grandes. La Figure 5.3 illustre un exemple de ce qui est attendu de l'algorithme. L'idée générale derrière la concordance d'image est de faire glisser le *template* sur l'image et de les comparer pour toutes les positions possibles. Dans ce cas, on regarde simplement la valeur de chaque pixel pour le comparer à la valeur du pixel qu'il superpose. Toutes les positions du template pour lesquelles la différence entre les pixels sera en dessous d'un certain seuil seront alors détectées comme une instance du template sur l'image [34].

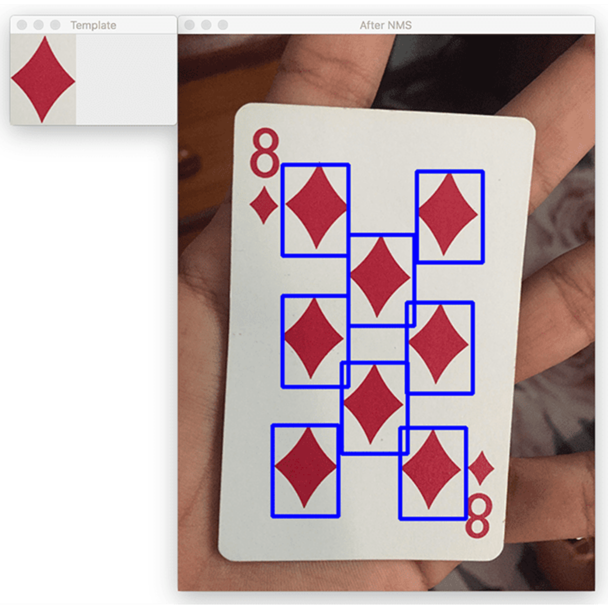


FIGURE 5.3 – Représentation d'un processus de concordance d'image. Le template du carreau est retrouvé sur l'image de la carte. Image issue de [1].

Dans ce cas de figure, les *features* utilisés sont la valeur de chaque pixel ainsi que leur position relative les uns par rapport aux autres sur le *template*. En effet, aucun traitement de l'image n'est réalisé pour extraire des *features* supplémentaires.

Il est important de noter que le *template matching* pourrait inclure une rotation du *template* via une extension de la méthode comme présentée dans [8].

5.3.2 Scale-invariant feature transform (SIFT)

Cette méthode a été introduite par David Lowe dans son article [28]. Cette routine permet d'extraire des *features* d'une image qui sont invariants aux rotations, scaling, et partiellement invariants aux changements d'illumination ainsi qu'aux mouvements de la position de la caméra.

La détection des *features* se construit en quatre étapes. Tout d'abord, la recherche des points d'intérêts (*keypoints*) potentiels en détectant les extrema d'une fonction analysant l'image sur plusieurs échelles [4]. Ensuite, la stabilité des *keypoints* est analysée pour éliminer ceux qui ne sont pas pertinents. Par la suite, une orientation sur l'image est donnée aux différents *keypoints*. Enfin, les *features* sont décrites sous forme d'un *features vector*.

Afin d'appliquer cette méthode d'extraction de *features* à la concordance d'image, un template va d'abord être analysé. Dans un premier temps, il faut extraire les *features* des *keypoints* sur le *template*. Ensuite, il faut extraire les *features* des *keypoints* sur la scène de recherche. Enfin, une routine va chercher les correspondances entre les *keypoints* trouvés pour les deux images. Ce processus se fait en analysant la distance entre chaque point d'intérêt de la scène et le point d'intérêt du template le plus proche (qu'on va appeler plus proche voisin) [5]. Une correspondance sera déterminée comme valide si la distance avec le plus proche voisin et la distance avec le deuxième plus proche voisin ne sont pas trop similaires. [6]. Ces concepts sont illustrés sur la Figure 5.4.

5.3.3 Speeded up robust features (SURF)

Cet algorithme est une version accélérée de SIFT. Il a été introduit pour la première fois dans [3]. D'une part, cet algorithme est utilisé à des fins identiques à celles de SIFT. D'autre part, l'idée derrière cet algorithme est de revisiter les étapes du processus de SIFT en appliquant des méthodes moins gourmandes d'un point de vue du calcul. Pour ces deux raisons, il a été décidé qu'il n'était pas nécessaire d'examiner cette méthode dans le cadre du projet `riverApp`. Néanmoins, il est

4. L'image est convoluée avec une gaussienne. Ensuite, la différence des convolutions entre deux zones adjacentes est prise pour trouver les localisations des *keypoints*.

5. Notez ici qu'on parle bien de distance dans l'espace des *features*

6. Lowe dans son article choisit un ratio de 0.8 entre la distance du voisin le plus proche et le deuxième voisin le plus proche comme seuil pour rejeter la correspondance.

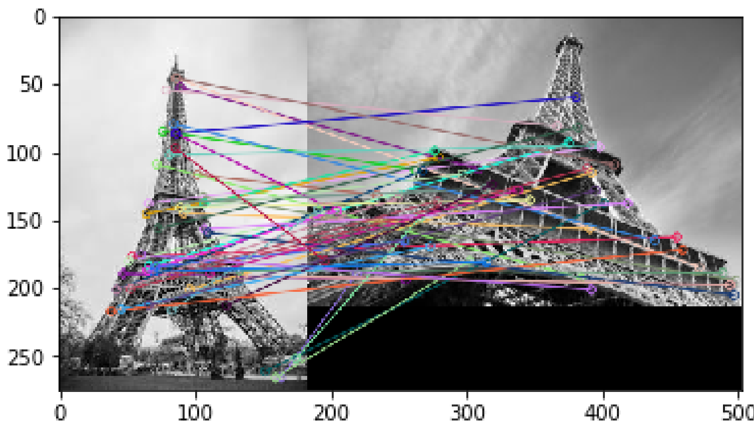


FIGURE 5.4 – L'image de gauche est prise comme template et l'image de droite comme scène de recherche. On remarque que tous les *keypoints* de la scène n'ont pas un correspondant sur le template car certains ont été considérés comme invalides par la routine. Images issues de [2]

important de la mentionner. De plus, une documentation claire de la théorie de la méthode ainsi qu'un exemple d'utilisation avec OpenCV est disponible dans [38].

5.3.4 Oriented FAST and Rotated BRIEF (ORB)

Cette méthode a été introduite comme alternative à SIFT et SURF dans [47]. En effet, elle se base également sur une recherche de *keypoints* puis sur la construction d'un *features vector* pour chacun de ces *keypoints*. Cet algorithme se base sur le détecteur FAST (*Features from Accelerated Segment Test*) introduit dans [46] et sur le descripteur BRIEF (*Binary Robust Independent Elementary Features*) introduit dans [5].

Le détecteur FAST permet de détecter les coins sur une image. Pour ce faire, l'algorithme va analyser pixel par pixel sur l'image en noir et blanc (i.e. *grayscale*). Pour chaque point d'intérêt analysé, un cercle de pixel autour est regardé et si une proportion importante de ces pixels est plus claire ou plus foncée que le point d'intérêt, alors ce dernier est considéré comme un coin. La notion de clarté du pixel vient du fait que dans une image en *grayscale*, chaque pixel se voit attribuer une valeur unique (généralement entre 0 et 255 où 0 correspond au noir et 255 au blanc). Il est important de spécifier que la comparaison des pixels est prise jusqu'à un certain seuil. Une illustration de cette procédure est montrée sur la Figure 5.5.

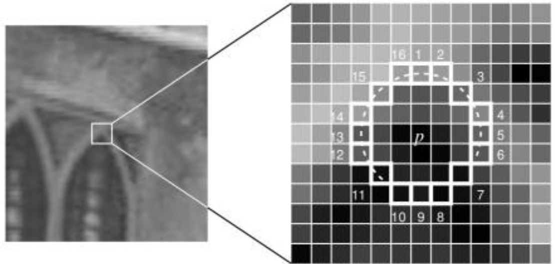


FIGURE 5.5 – Illustration de l'algorithme FAST. Sur l'image de gauche l'image originale et sur l'image de droite un zoom sur le pixel d'intérêt p. On remarque que parmi les pixels entourant p, une grande proportion d'entre eux (typiquement les pixels 1 à 6 et 12 à 16) seront considérés comme plus clairs que p. Le pixel p sera alors détecté comme étant un coin sur l'image originale. Image issue de [37].

Une fois que les *keypoints* ont été repérés, l'algorithme BRIEF va leur associer un *features vector*. L'un des principaux avantages de cet algorithme est que le *feature vector* associé à chaque *keypoint* est un vecteur binaire, c'est-à-dire contenant uniquement des 1 et des 0. Pour stocker ce vecteur, très peu de mémoire est nécessaire. Pour créer ce vecteur, l'algorithme va associer à chaque *keypoint* un *patch*, c'est-à-dire la partie rectangulaire de l'image qui entoure le *keypoint*. À l'intérieur de ce *patch*, l'algorithme va créer n paires de points de manière aléatoire, comme illustré sur la Figure 5.6. Ensuite, pour chaque paire de points (x, y), si la valeur du pixel x est plus importante que la valeur du pixel y alors l'entrée du *feature vector* correspondant à cette paire de point sera 1 et sinon cette entrée sera 0. De cette manière en comparant n paires de points, on obtient un *feature vector* de taille n.

5.3.5 Détection des bords grâce à l'algorithme de Canny

Cette méthode permet de détecter les bords des objets sur une image. Cet algorithme a été développé dans [6] et se base sur l'intensité des gradients de l'image. Pour calculer les gradients de l'image, une technique commune ([50], [36]) est de convoluer l'image avec un noyau de Sobel. Grâce à cela, il est possible d'extraire l'intensité et la direction des gradients de l'image sur chaque pixel. Ensuite, l'algorithme va supprimer les gradients de même direction mais de moins forte intensité dans chaque voisinage de pixel. Enfin, l'algorithme va utiliser deux seuils et une méthode d'hystérèse[7] pour transformer les gradients de trop faible intensité en gradients nuls ou en gradients de plus grande intensité. Un résultat d'application

7. Des explications plus détaillées de cette étape sont également expliquées dans [50]

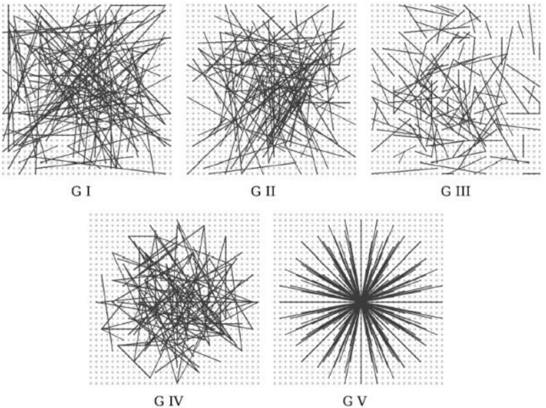


FIGURE 5.6 – Illustration de différentes méthode de sélection aléatoire de paires de points dans le *patch*. Image issue de [11]

de cette routine est montré sur la Figure 5.7.

FIGURE 5.7 – Illustration de la méthode de Canny. Les gradients de forte intensité filtrés par l'hystérèse sont présentés sur l'image de droite. On remarque que ces gradients correspondent bien aux bords présents sur l'image. Image issue de [50].

5.4 Tests et choix de la routine implémentée

5.4.1 Template matching

Cette méthode n'est malheureusement pas applicable au cas du projet `riverApp`. En effet, cette routine ne prend pas en compte la distorsion du template due à la perspective lors de la prise de vidéo.

5.4.2 SIFT

Comme l'illustre la Figure 5.8, cette méthode ne peut pas s'appliquer au projet `riverApp`. On observe que la routine ne détecte que très peu de points d'intérêt sur le template. Pour cette raison, les *keypoints* les plus proches (entre ceux du template et ceux de la scène) sont des points qui ne se correspondent pas du tout.

En effet, pour trouver des caractéristiques et des *keypoints*, SIFT a besoin d'objets à la géométrie complexe ([19]), or le damier est une géométrie simple et n'a donc quasiment pas de *features* détectés, comme le montre la Figure 5.8b.

5.4.3 SURF

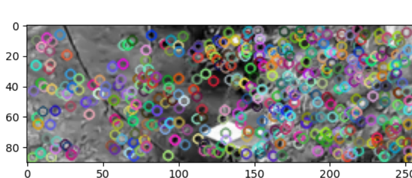
Vu que cette méthode a des résultats similaires à ceux de SIFT [26], elle n'a pas été testée dans le cadre de ce projet. En effet, elle se base également sur de la détection de *keypoints* or cette détection de *keypoints* n'est pas efficace pour la géométrie des balises.

5.4.4 ORB

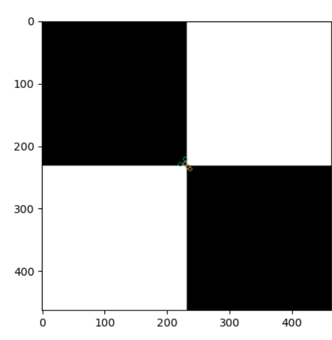
Pour des raisons similaires à la méthode de SIFT, très peu de *keypoints* sont détectés sur le damier qui a une géométrie simple. Comme l'illustre la Figure 5.9, cette méthode n'est pas applicable au cas de la `riverApp`.

5.4.5 Canny

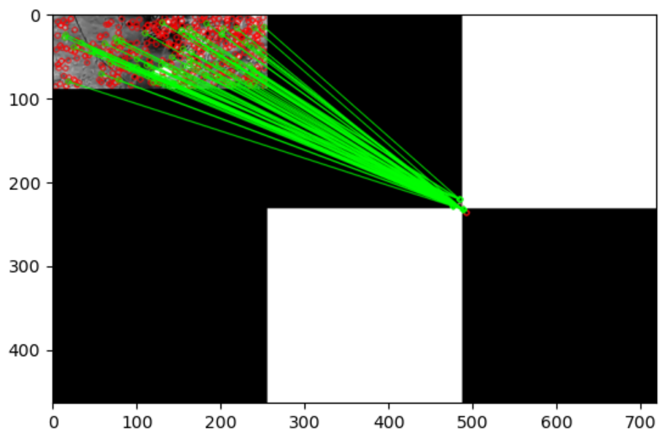
Cette méthode ne fonctionne malheureusement pas pour le cas du projet `riverApp`. En effet, la Figure 5.10 montre que pour différents thresholds, les frontières détectées sont d'abord celles créées par les feuilles sur les rives.



(a) Points d'intérêt sur la scène

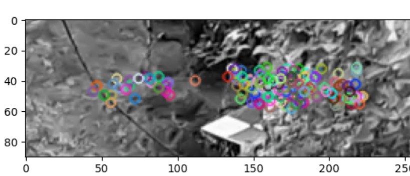


(b) *Keypoints* sur le template

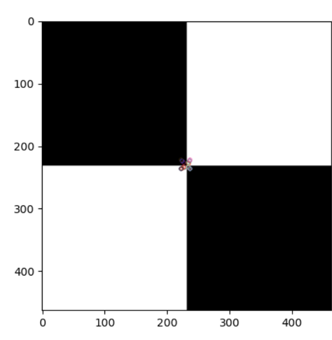


(c) Correspondance des *keypoints*

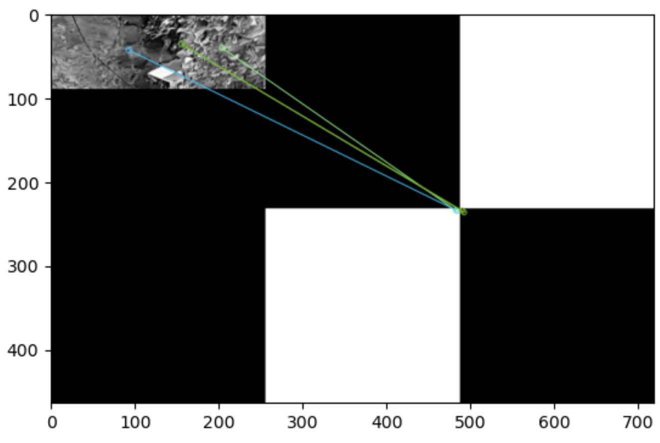
FIGURE 5.8 – Illustration du procédé de concordance utilisant les caractéristiques extraites via la méthode SIFT. Sur la figure Figure 5.8b, on peut voir les *keypoints* sur le template. Sur la figure Figure 5.8a, on peut voir les *keypoints* sur la scène de recherche. Sur la figure Figure 5.8c, on peut voir les correspondances faites entre les *keypoints* des deux images. Certains des *keypoints* de Figure 5.8a n'ont pas de correspondants car ils ont été considérés comme invalide par la routine.



(a) Points d'intérêt sur la scène

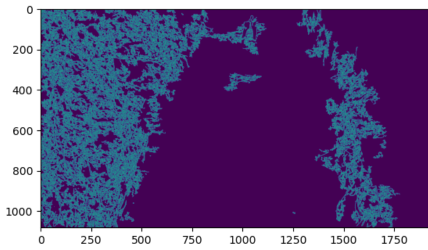


(b) *Keypoints* sur le template

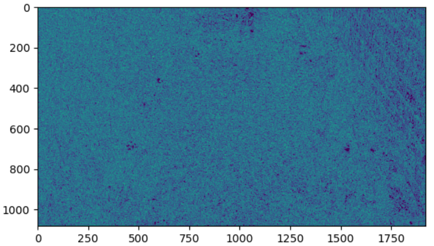


(c) Correspondance des *keypoints*

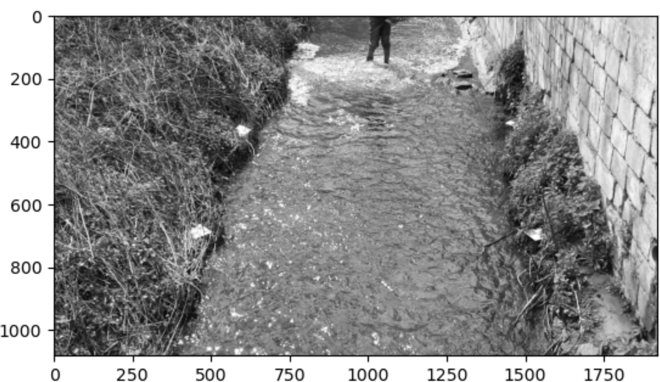
FIGURE 5.9 – Illustration du procédé de concordance utilisant les *features* trouvés grâce à la routine ORB. Sur la Figure 5.9b, on peut voir les *keypoints* sur le template. Sur la figure Figure 5.9a, on peut voir les *keypoints* sur la scène de recherche. Sur la Figure 5.9c, on peut voir les correspondances faites entre les *keypoints* des deux images. Certains des *keypoints* de Figure 5.9a n'ont pas de correspondants car ils ont été considérés comme invalides par la routine.



(a) Détection des frontières avec $thr_1 = 10$ et $thr_1 = 1000$



(b) Détection des frontières avec $thr_1 = 10$ et $thr_1 = 20$



(c) Image de référence sur laquelle l'algorithme de Canny est appliqué.

FIGURE 5.10 – Illustration de la détection de frontières sur une image utilisée par `riverApp`. Image issue du jeu de données de Noirath.

5.4.6 Bilan des tests

Comme aucune des cinq méthodes présentées dans la Section 5.3 n'est applicable au cas de la détection d'une balise en damier (2×2), il a fallu en choisir une autre. Cependant, aucun algorithme n'est spécialisé dans la détection de schémas simples comme les damiers. Pour cette raison, il a fallu construire une routine adaptée à cette recherche. Le processus itératif de construction de cette routine est expliqué dans la Section 5.5. Cette routine s'appuie sur des concepts de traitement d'images et non pas sur un algorithme complet de détection de *pattern*.

5.5 Implémentation finale

Dans cette dernière section du chapitre, nous allons présenter les processus de réflexion et d'implémentation qui nous ont amenés au résultat final. Afin de faciliter la lecture, les concepts théoriques seront introduits au fur et à mesure du développement.

5.5.1 Constat de départ

Après l'implémentation et le test des différents algorithmes mentionnés dans la Section 5.4, nous ne remarquons aucun résultat concluant. Cela s'explique en grande partie par le fait que le damier utilisé comme modèle pour la détection est un motif simple. En effet, on constate que les `keypoints` du damier utilisés pour la détection sont tous situés au centre de la cible. Leur manque de complexité pose problème car l'algorithme trouve des similarités avec ce motif à de nombreux endroits dans l'image. Il n'est donc pas possible d'extraire les 4 balises de l'image qui sont pourtant les plus similaires au motif de référence.

Il y a deux façons d'aborder ce problème. La première consiste à changer le motif utilisé pour la détection. En effet, la détection d'objets dotés de caractéristiques plus marquantes est un aspect bien maîtrisé de la science du traitement d'images. Les différents algorithmes détaillés dans les sections précédentes sont efficaces et robustes, même si la cible a subi une rotation ou un changement d'échelle. Dans ce contexte, les marqueurs ArUco [33] sont particulièrement adaptés. Ce sont des marqueurs spécialement conçus pour faciliter leur détection dans notre monde 3D. De plus, ils sont subdivisés en blocs binaires 2D, tout comme les codes QR, ce qui permet d'identifier chaque marqueur en fonction de son identité. Dans notre cas, nous avons besoin de 4 marqueurs avec leurs ID respectifs de 1 à 4. Cette méthode résout le problème de détection ainsi que l'identification des marqueurs.

(a) Image de base

(b) Marqueurs détectés et identifiés

Figure 5.11 – Exemple de détection et identification des marqueurs ArUco ([57])

Cependant, le repère en damier utilisé actuellement est un modèle standard des points de contrôle au sol (ground control point, également connu sous le nom de GCP). Il est largement utilisé dans divers domaines impliquant la capture d'images sur le terrain. Toutes les vidéos de cours d'eau que nous possédons pour nos tests utilisent d'ailleurs cette balise. La seconde approche consiste donc à remettre en question notre implémentation et à utiliser de manière appropriée les caractéristiques intrinsèques de la balise en damier. C'est sur cette partie que nous nous sommes concentrés dans la suite de notre travail.

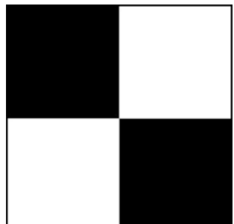
Durant la suite de ce chapitre, nous utiliserons une image comme exemple principal afin de visualiser les effets des différentes sous-étapes. Il s'agit d'une image tirée de la vidéo prise du cours d'eau de la Dyle à Limelette. La vidéo a été prise par un drone et offre d'une vue suffisamment précise des 4 marqueurs.

FIGURE 5.12 – Image utilisée comme fil rouge pour cette section

5.5.2 Analyse du pattern

Prenons le temps un moment d'analyser les aspects qui distinguent ce marqueur. Premièrement, les couleurs qui le composent incarnent les deux extrêmes de la palette chromatique. Dans l'espace de couleurs Rouge-Vert-Bleu (RGB), chaque couleur du spectre visible peut être représentée par sa décomposition en 3 couleurs primaires. Le noir incarne l'absence de lumière, l'équivalent des 3 couleurs à intensité nulle (0, 0, 0). Le blanc, quant à lui, représente l'inverse, il comporte l'ensemble des longueurs d'onde pouvant être perçues par l'oeil. Dans le système RGB, il s'agit des 3

FIGURE 5.13 – Marqueur de référence



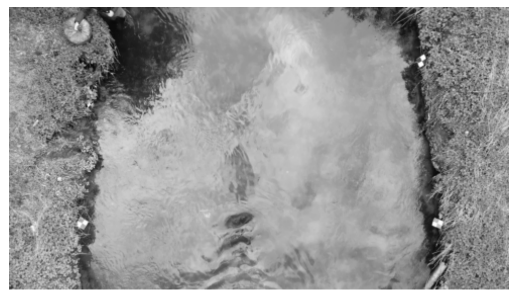
couleurs à la plus grande intensité possible (255, 255, 255). Ces couleurs sont plutôt inhabituelles dans une photo prise dans le monde réel, surtout dans la nature. En effet, les couleurs sont généralement constituées d'un mélange des intensités de ces 3 couleurs primaires. De plus, le marqueur n'est composé que de 4 carrés uniformes munis d'une même couleur. Tous les pixels qui représentent chaque carré possèdent donc une valeur RGB très similaire. Ce genre de pattern uniforme est encore moins présent dans une photo d'un espace naturel. Enfin, le dernier aspect qui caractérise le pattern en damier est sa forme. Les 4 carrés uniformes se rejoignent en un seul point au centre. Les couleurs sont alternées, ce qui rend le point central un élément très singulier du pattern. Nous avons donc identifié 3 traits différents qui distinguent le damier. Le processus que nous avons implémenté va utiliser ces 3 informations de manière cohérente.

5.5.3 Threshold sur la palette de couleur

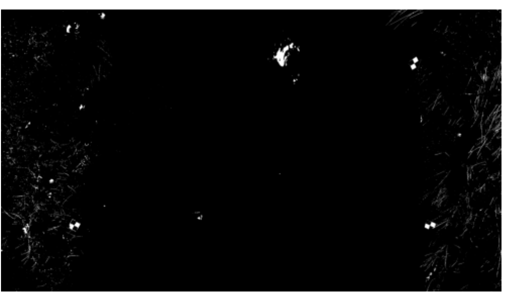
Pour la première étape, nous allons exploiter les couleurs du damier de référence. Comme nous allons travailler uniquement avec les couleurs blanches ou noires, nous allons d'abord convertir notre image du format RGB en noir et blanc. Cela permet un gain significatif en termes de mémoire et en temps d'exécution. La conversion se fait simplement en prenant la moyenne pondérée des 3 composantes de couleur primaires [31] :

$$Y = 0.3 * R + 0.59 * G + 0.11 * B \tag{5.1}$$

Nous avons maintenant l'image au format numérique sous la forme d'un tableau de valeurs allant de zéro (noir) à 255 (blanc). Ensuite, nous allons définir un seuil sur les valeurs que nous conservons. Tous les pixels ayant une intensité supérieure à 235 seront mis à 1, tandis que les autres seront définis à zéro. Nous obtenons ainsi une image binaire qui représente les parties de l'image avec un niveau de blanc suffisamment élevé. La valeur seuil de 235 a été déterminée par essai-erreur, c'est celle qui nous a donné les meilleurs résultats. Il est possible de modifier cette valeur lors de l'implémentation, notamment en utilisant l'adaptative thresholding. Cette idée est expliquée et développée dans la Section 5.5.6. Voici un exemple illustrant le résultat obtenu. On peut observer qu'il reste encore beaucoup de bruit et d'éléments non pertinents dans la recherche des balises.



(a) Image convertie en noir et blanc

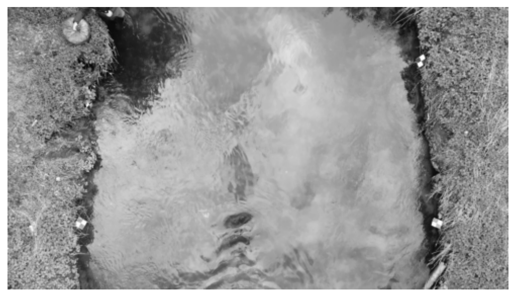


(b) Image après le thresholding des pixels dont l'intensité > 235

FIGURE 5.14 – Résultat après la première étape

La balise est également composée de couleur noire, il est intéressant d'analyser les résultats après une thresholding des valeurs proches du noir. Nos expériences ont montré que la détection des parties noires ne fournit que peu d'informations concernant la détection des balises. En effet, les parties noires de la balise ont tendance à présenter une variation de couleur. Cette variation est due à leur forte sensibilité à la luminosité. Les carrés uniformes prennent des nuances qui s'éloignent de la couleur noire pure. Les formes obtenues sont donc moins précises, ce qui rend la suite de notre algorithme moins efficace. Il serait possible de combiner les deux informations provenant des deux thresholds possibles. Cependant, le threshold bas sur les intensités ne fournit pas d'informations suffisamment qualitatives sur les balises. Il perturberait les informations contenues dans le thershold élevé des intensités.

Nous pouvons voir sur la Figure 5.15 que la balise en bas à gauche n'a pas du tout été détectée. Les carrés noirs de celle-ci reflètent si intensément le ciel que leur teinte vire au bleu. Cela entraîne une intensité plus élevée et la zone n'est pas détectée via le thresholding. Nous pouvons expliquer ce phénomène par le fait que le noir, étant dépourvu de toute couleur, a tendance à absorber davantage les longueurs d'ondes de la lumière. Au contraire, Le blanc est blanc car il est composé de l'ensemble du spectre lumineux. Ainsi, la couleur blanche reflète la totalité des rayons. Le thresholding est donc plus robuste lorsqu'il est utilisé pour conserver uniquement les intensités les plus élevées de l'image.



(a) Image convertie en noir et blanc

(b) Image après le thresholding des pixels dont l'intensité < 30

FIGURE 5.15 – Thresholding sur les intensités basses

5.5.4 Post-traitement de l'image

Dépendamment de l'image à traiter, le résultat du thresholding va varier. Sur des images très lumineuses, les zones obtenues sont nombreuses. Ici, le bruit se situe surtout sur les deux berges. Afin de corriger le bruit, nous allons utiliser un concept puissant dans le traitement d'images : la morphologie mathématique.

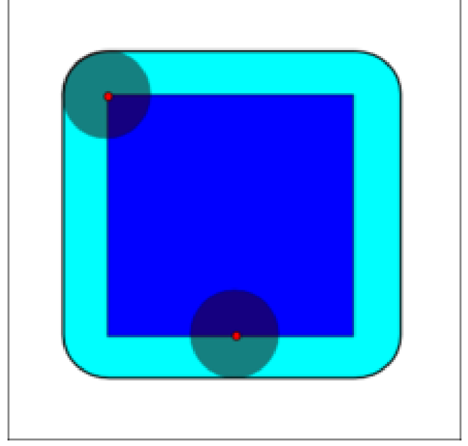
Mathématiquement, une image binaire $f : [x, y] \in \mathbb{Z}^2 \mapsto f[x, y] \in \{0, 1\}$ est équivalent à un ensemble de positions F dans $\mathbb{Z}^2$ [24] :

$$f \longleftrightarrow F := \Big\{[p_x, p_y] \in \mathbb{Z}^2 : f[p_x, p_y] = 1\Big\} \subset \mathbb{Z}^2 \tag{5.2}$$

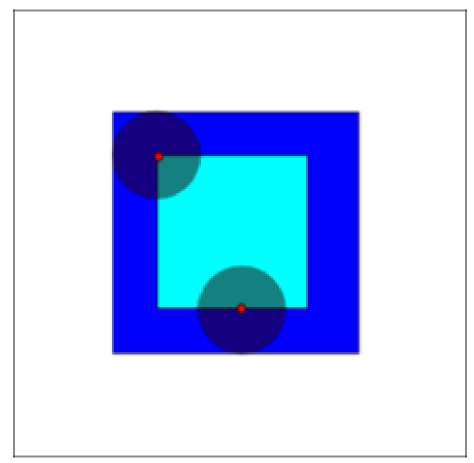
Sur base de cet ensemble, nous pouvons définir un certain nombre d'opérations. En premier lieu, la dilation est utilisée pour élargir toutes les parties blanches de l'image binaire (c'est à dire les positions qui correspondent à 1 dans l'image binaire). Nous avons une image f, son ensemble associé F ainsi qu'un noyau K. F et K représentent tous les deux une image binaire. Formellement, l'opération de la dilation peut être définie par la formule suivante (formule définie pour chaque pixel [x,y] de f)[24] :

$$(f \oplus K)[x, y] = \Big\{[x + k_x, y + k_y] : [x, y] \in F, (k_x, k_y) \in K\Big\} \tag{5.3}$$

Intuitivement, il s'agit de toutes les zones atteignables par le noyau K en déplaçant son centre partout dans F. Pour simplifier davantage, la dilation épaissit toutes les formes. Si l'image binaire est un texte écrit, cela reviendrait à appuyer plus fort sur le stylo pour épaissir les traits du texte. L'image suivante (Figure 5.16 à gauche) permet de mieux comprendre le résultat de cette opération de dilation.



(a) Dilation du carré bleu foncé avec comme noyau un disque



(b) Érosion du carré bleu foncé avec comme noyau un disque

FIGURE 5.16 – Opérateurs morphologiques de base ([54])

Nous pouvons maintenant introduire l'opération d'érosion. Il s'agit de l'opérateur dual de la dilation. Avec un ensemble F représentant l'image binaire f, K un noyau et $K_{[a,b]}$ le noyau centré en $[a, b]$, l'opération érosion se définit comme ceci [24] :

$$(f \ominus K)[x, y] := \left\{[a, b] \in \mathbb{Z}^2 : K_{[a,b]} \subset F\right\} \tag{5.4}$$

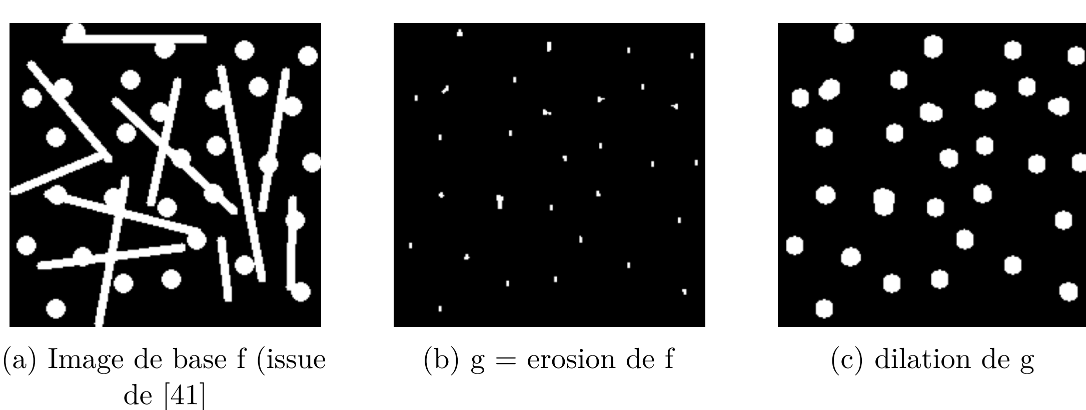
Intuitivement, l'érosion peut être visualisée comme une réduction des bords de l'objet , ce qui entraîne son amincissement. Il s'agit de tous les endroits atteignables par le centre du noyau en le déplaçant partout dans F.

Enfin, en utilisant ces deux opérateurs, nous pouvons les combiner pour en créer un nouveau. Nous définissons l'opérateur d'ouverture comme étant une suite des deux. L'ouverture de f par le noyau K est le résultat de l'érosion de f par K, suivie de la dilation de ce résultat par K [24] :

$$f \circ K := (f \ominus K) \oplus K \tag{5.5}$$

Grâce à cette opération, il nous est maintenant possible de filtrer certains aspects d'une image. La Figure 5.17 nous permet de visualiser en détail le processus. L'image f est binaire, elle contient des 0 pour l'arrière-plan et des 1 pour les parties en premier plan. Nous utilisons un noyau sous la forme d'un disque de diamètre de 11 pixels. L'ouverture se décompose en deux étapes. Tout d'abord, nous appliquons l'opérateur d'érosion. À chaque endroit où le noyau peut être placé, nous enregistrons les coordonnées de son centre. Ces coordonnées forment la nouvelle image g. C'est pourquoi nous trouvons un point dans l'image g à chaque emplacement des cercles dans f. Il est important de noter que dans certains endroits,

le croisement de deux barres permet de créer un espace suffisamment grand pour placer un cercle, ce qui laisse également un point. Pour la deuxième étape, nous appliquons l'opérateur de dilatation sur g. Nous plaçons le noyau sur chaque point de l'image g. Tous les points couverts par ce noyau sont enregistrés dans l'image finale. La dilatation a donc pour effet d'agrandir tous les points pour les ramener à la taille originale des cercles dans f. La combinaison de ces deux opérateurs pour former l'opérateur d'ouverture nous a permis de filtrer les lignes afin de ne conserver que les cercles.



(a) Image de base f (issue de [41]

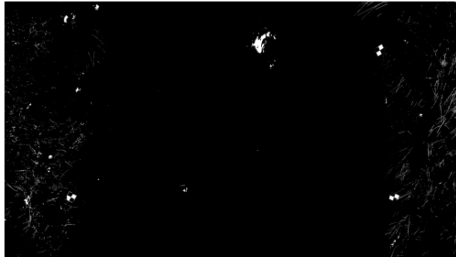
(b) g = erosion de f

(c) dilation de g

FIGURE 5.17 – Ouverture d'une image f, visualisation en deux étapes ([54])

Revenons maintenant à notre propre implémentation. Nous avions obtenu l'image thresholdée qui contenait du bruit. En utilisant un noyau corculaire avec un diamètre suffisamment grand, nous pouvons appliquer l'opérateur d'ouverture sur l'image. Cela aura pour effet de supprimer les bruits blancs tout en conservant les formes blanches assez grandes. Cette étape est importante pour la suite afin d'obtenir des résultats pertinents. La difficulté réside dans le choix de la taille appropriée pour le noyau. Un noyau trop petit laissera encore du bruit, tandis qu'un noyau trop grand supprimera des blocs de blanc importants et entraînera une perte d'information. L'implémentation actuelle utilise un noyau circulaire avec un diamètre de 8 pixels. Idéalement, cette valeur devrait dépendre de la résolution de l'image. Plus la résolution est élevée, plus ce paramètre devrait être grand. Cependant, nous avons choisi de le laisser constant à cette valeur car la résolution reste généralement autour du format full hd (1920x1080 pixels).

Pour reprendre notre exemple, voici le résultat de l'ouverture appliquée à notre image d'exemple. On peut constater que l'image est plus propre et plus claire. Cependant, ce n'est pas uniquement une question d'esthétique. Cette étape revêt une grande importance pour éviter que le reste de l'algorithme ne soit surchargé par un trop grand nombre d'objets présents sur l'image.



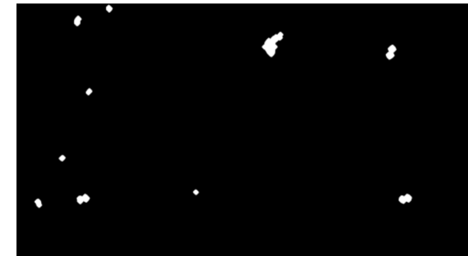
(a) Image après le thresholding des pixels dont l'intensité > 235

(b) Image après ouverture avec comme noyau utilisé un disque de 8 pixels de diamètre

FIGURE 5.18 – Résultat de l'opérateur ouverture appliqué sur l'image

Il nous reste cependant une dernière étape de post-traitement. En effet, nous remarquons que, dans cet exemple, les parties blanches des balises sont bien détectées. Le problème réside dans le fait que les carrés blancs ne se rejoignent pas par l'un de leurs coins au centre de la cible pour chaque balise. Les deux formes ne possèdent donc aucun pixel les reliant, l'algorithme les traite donc comme deux objets distincts. Nous allons donc utiliser l'opération de dilation pour légèrement étendre chaque forme et ainsi relier les deux carrés blancs de chaque repère. Pour cela, un noyau circulaire de 3 pixels de diamètre est nécessaire afin de ne pas trop déformer les objets. Nous effectuons un total de 12 itérations afin de d'élargir suffisamment les traits. Notons que 1 itération avec un noyau 12x plus grand n'obtiendrait pas le même résultat et alerterait de manière trop prononcée les formes de l'image. Notre objectif est de conserver des formes relativement fidèles à celles présentes sur l'image de base. Voici le résultat de la dilation sur l'image filtrée. Nous obtenons un total de 10 objets assez espacés dans l'espace, ce qui permettra une analyse correcte de chacun d'entre eux.

(a) Image après l'opération d'ouverture



(b) Image après dilation pour joindre les carrés uniformes des balises

FIGURE 5.19 – Dilation pour joindre les carrés blancs des balises

5.5.5 Analyse de similarité des formes

A ce stade ci de l'implémentation, nous avons un certain nombre d'objets qui ont comme point commun d'avoir une partie uniforme blanche. Il s'agissait des deux premiers aspects mis en avant dans l'analyse du repère de référence dans la Section 5.5.2. Nous avons maintenant besoin de décider quels sont les 4 objets les plus probables de correspondre aux repères placés dans le monde réel. Nous allons pour cela utiliser la dernière caractéristique intrinsèque au damier de référence : sa forme. Pour rappel, les deux carrés blancs uniformes se rejoignent en un seul point au centre du pattern. Il s'agit d'une forme peut commune ce qui nous permettra de trier correctement nos objets. Une première idée serait d'utiliser un algorithme de pattern matching comme présenté dans la Section 5.3.1. Malheureusement, cet algorithme n'est pas robuste face aux rotations et scaling du pattern. Or, nous n'avons pour l'instant aucune idée sur la position précise de chaque balise.

Pour faire face à ce problème, nous allons introduire les Hu-moments d'une image [21] : un outil qui permet de décrire précisément une forme contenue dans une image tout en étant invariant aux rotations, scaling et translations. La suite des explications est largement inspirée de l'excellent article issu du site learnopencv [29]. Pour la suite des explications, considérons une image binaire I. L'intensité du pixel à la position (x,y) peut être donnée par I(x,y). Notons que $I(x, y) = \{0, 1\}$. Le moment le plus simple que l'on pourrait imaginer serait de prendre la somme de toutes les intensités :

$$M = \sum_x \sum_y I(x, y) \tag{5.6}$$

Pour une image binaire, ce moment représente le nombre de pixels blanc de l'image. Il peut aussi être vu comme l'aire de la région blanche. Ce moment présente déjà des propriétés intéressantes. Sur l'exemple ci-dessous Figure 5.20, nous pouvons remarquer que le moment basique des deux images de S sera très similaire. Le moment basique de la lettre K sera quant-à-lui différent. Cela n'est pas suffisant, il est en effet facile de créer deux image distinctes qui possèdent un moment basique similaire.

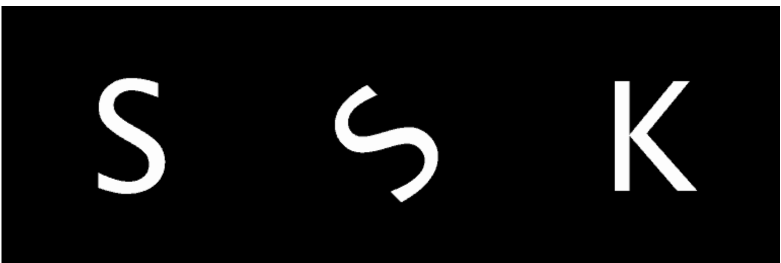


FIGURE 5.20 – 3 images binaires. De gauche à droite : la lettre S, même image mais avec une rotation, la lettre K

Nous allons complexifier le moment afin d'y rajouter des informations supplémentaires. Définissons des nouveaux moments qui combinent l'intensité du pixel avec la position de celui-ci :

$$M_{ij} = \sum_x \sum_y x^i y^i I(x, y) \tag{5.7}$$

Ces moments sont appelés "raw image moments" et peuvent être utilisés pour des images en noir et blanc. Le fait de mêler l'information de l'intensité à celle de la position permet de capturer une notion de forme dans l'image. En utilisant ces raw moments, nous pouvons calculer le centroïde $(\overline{x}, \overline{y})$ de chaque objet via la formule suivante :

$$\overline{x} = \frac{M_{10}}{M_{00}} \qquad \overline{y} = \frac{M_{01}}{M_{00}} \tag{5.8}$$

Avec ce point centroïde, nous pouvons construire de nouveaux moments qui sont invariables aux translations. Il suffit de soustraire les centroïdes dans la formule des raw moments. Dans la théorie mathématique, ils sont appelés "moments centraux" et sont définis comme ceci :

$$\mu_{ij} = \sum_x \sum y(x - \overline{x})^i (y - \overline{y})^j I(x, y) \tag{5.9}$$

Il est possible de rendre ces moments centraux invariants au scaling en les normalisant. On obtient la formule des moments centraux normalisés :

$$\eta_{ij} = \frac{\mu_{i,j}}{\mu_{00}^{((i+j)/2)+1}} \tag{5.10}$$

Il ne nous reste plus qu'une propriété à obtenir : l'invariance aux rotations. C'est ici qu'intervient le fantastique travail de Hu. Les moments de Hu constituent un set de 7 nombres calculés en utilisant les moments centraux normalisés. Les six premiers moments ont été démontrés comme étant invariants à la translation, au

scaling, à la rotation et à la réflexion. En revanche, le signe du septième moment change en cas de réflexion de l'image. La fondation théorique de ces moments peut être trouvée dans le papier original [21]. Voici comment sont calculés les 7 hu-moments :

$$\begin{aligned}
h_1 &= \eta_{20} + \eta_{02} \\
h_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
h_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
h_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
h_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
h_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
h_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{5.11}$$

Fort heureusement pour nous, la librairie OpenCV dispose de fonctions utiles pour calculer les hu-moments [32]. Calculons maintenant les hu-moments de différentes images binaires de lettres afin de comparer leurs valeurs. Nous pouvons constater sur le tableau Figure 5.21 que les différentes images du S obtiennent des moments similaires. A l'inverse, la lettre K nous donne des moments bien différents de ceux calculés pour les images de S. Notons que le dernier moment change de signe quand l'image est inversée. Cette propriété peut avoir des applications intéressantes.

id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0		2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2		2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

FIGURE 5.21 – Résultat sur plusieurs images binaires qui représentent chacune une lettre

Nous avons maintenant un moyen robuste de comparer deux images binaires qui ont subi soit une translation, un scaling, une rotation ou une inversion. OpenCV possède une fonction intéressante pour analyser la similarité entre deux images appelée : distanceMatching() [32]. Il s'agit simplement de la distance entre les deux vecteurs des hu-moments respectifs des deux images. La formule pour la distance entre deux images est la suivante (H_i^B représente le i_{eme} hu-moment de l'image B) :

$$D(A, B) = \sum_{i=0}^{6} |H_i^B - H_i^A| \tag{5.12}$$

Retournons à présent sur notre implémentation pratique. Pour chaque objet actuellement détecté de l'image, nous allons donc calculer et stocker la matching distance entre cet objet et le damier de référence. Pour obtenir les 4 objets les plus similaires, il ne nous reste plus qu'à trier les objets par leur score de distance. Nous prenons ensuite les 4 objets ayant la plus grande similitude avec la cible, c'est à dire qui ont le score de distance le plus faible. Nous retournons ensuite les coordonnées des centroïdes de chacun des 4 objets. Voici ci-dessous la visualisation pour la dernière étape de notre implémentation Figure 5.22 :

(a) Image après la dilation

(b) Résultat final, 4 croix rouges indiquant les GCPs considérés comme les plus probables

FIGURE 5.22 – Dernière étape de notre implémentation sur la frame de Limelette

Nous remarquons que 3 des 4 balises ont été repérées et indiquées avec succès. La dernière balise se trouvant en haut à gauche de l'image n'a pas été sélectionnée. Si nous l'analysons plus en détail (voir Figure 5.23), nous pouvons remarquer que cette balise est à moitié immergée dans l'eau. Son deuxième carré blanc possède donc une couleur plus sombre. Son intensité étant trop faible par rapport à un blanc net, l'étape du thresholding ne le prend donc pas en compte. Nous avons comme résultat pour cette balise un seul carré blanc. Cela va tromper l'algorithme de distance matching pendant sa comparaison avec la cible de référence. L'algorithme va alors donner un score de similitude plus important. Ici, il va préférer prendre une partie très claire sur la rivière qu'il estimera comme plus similaire avec la cible de référence. Nous pouvons constater sur cet exemple que notre implémentation est efficace quand la balise est bien visible dans son ensemble. Si ce n'est pas le cas, l'algorithme ne garanti pas de la trouver. Dans la Section 5.5.8, nous analyserons plus en détail les résultats de notre implémentation sur un certain nombre d'images prises dans des conditions variées.

FIGURE 5.23 – Zoom sur la balise qui n'a pas été trouvée

5.5.6 Approfondissements possibles de la méthode

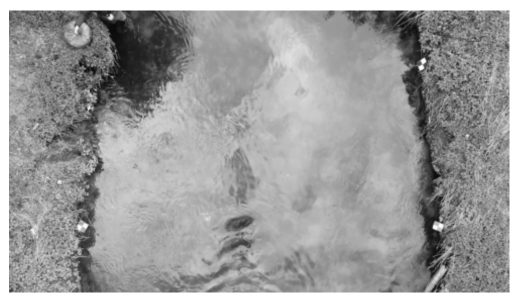
Pour pallier au manque de robustesse de la routine que nous avons implémentée, certaines autres pistes ont été explorées. Malheureusement, ces nouveaux modules n'ont pas encore donné de bons résultats mais mériteraient tout de même un approfondissement.

Détection de la rivière

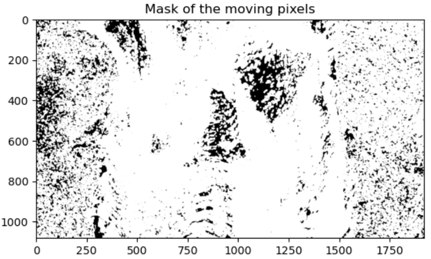
Le constat est que les balises sont localisées sur les berges, proches de l'eau. La première idée d'amélioration de la méthode est de détecter la rivière sur l'image. Ensuite, il faudrait imposer que les balises trouvées soient localisées dans une zone proche de la rivière mais jamais à l'intérieur. La méthode permettant d'obtenir ce résultat est illustrée sur la Figure 5.24.
Pour détecter la rivière, la première étape consiste à localiser les pixels mobiles sur les images, c'est-à-dire les pixels dont l'intensité change (jusqu'à un certain seuil) d'une frame de la vidéo à l'autre (Figure 5.24b). L'image binaire des pixels mobiles est ensuite traitées à l'aide des opérateurs morphologiques introduits dans la Section 5.5.4. Ensuite, il est possible d'extraire le plus grand groupe de pixels mobiles en définissant un groupe de pixels comme étant ceux qui se touchent au moins en diagonale (Figure 5.24c). Enfin, ce plus grand groupe est supprimé de l'image initiale pour empêcher l'algorithme de détection des GCP de trouver une balise dans la rivière.

Cette méthode part du principe que la rivière sera la plus grande partie mobile

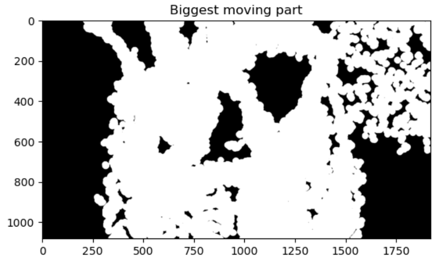
de l'image, ce qui pourrait ne pas être le cas si la rivière ne prend pas une place proéminente sur la vidéo et que des arbres en arrière-plan deviennent la plus grande partie mobile par exemple.



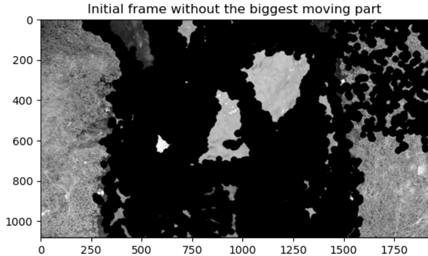
(a) Image initiale de la vidéo convertie en noir et blanc



(b) Masque des pixels mobiles en utilisant un seuil d'intensité de 20



(c) Détection du plus grand groupe de pixels mobiles



(d) Première frame de la vidéo où la plus grosse partie mobile est retirée

FIGURE 5.24 – Illustration de la méthode implémentée pour détecter la rivière sur l'exemple de Limelette

Malheureusement, ce module n'est pas encore abouti. Comme l'illustrent les tests réalisés (figures disponibles dans l'Annexe C), la rivière n'est pas détectée correctement. Non seulement il n'est pas possible de supprimer la rivière de la zone de recherche des balises, mais il n'est pas non plus possible d'imposer que les balises trouvées soient proches de la rivière.

En observant les tests, nous avons pu déterminer que lorsque la vidéo n'est pas stabilisée au préalable, il est impossible pour la méthode de différencier un pixel

mobile à cause du mouvement de la rivière d'un pixel mobile à cause du mouvement de la caméra. De plus, un réglage différent des paramètres peut augmenter l'efficacité de la méthode sur certains jeux de données. Ce réglage est cependant encore manuel et donc non exploitable dans une version aboutie de la méthode.

On peut alors conclure qu'en manipulant les opérateurs morphologiques et leurs paramètres, ou en ajoutant des étapes de pre-processing, cette méthode pourrait s'avérer efficace.

Contrainte sur la forme délimitée par les 4 balises

L'objectif de cette méthode est d'imposer que la forme trouvée soit un quadrilatère raisonnable. La notion de quadrilatère raisonnable n'est pas encore proprement définie, mais l'idée est que la forme devrait ressembler plus à un rectangle qu'à un triangle, comme illustré sur la Figure 5.25. Dans le cas d'une forme pas suffisamment raisonnable, il va falloir décider quels sont les points parasites. Une idée est de regarder quels sont, parmi les points détectés, ceux qui ressemblent le plus à la balise d'un point de vue de la forme et d'exclure les autres.
Les résultats présentés dans la Section 5.5.8 montrent que le matching de forme n'est pas encore totalement au point et que cette technique n'est donc pas encore exploitable.

Adaptive thresholding

Lors de la sélection de la palette de couleur (voir Section 5.5.3), un seuil de détection du blanc est choisi grâce à une méthode d'essai-erreur. Cependant, en fonction de l'illumination de la scène filmée, ce seuil devrait être adapté. En effet, comme l'illustre la Figure 5.26, pour différentes illuminations d'une même scène, la détection des pixels blancs varie pour un même seuil. Ainsi que pour une même illumination de la scène, la détection varie pour différents seuils.

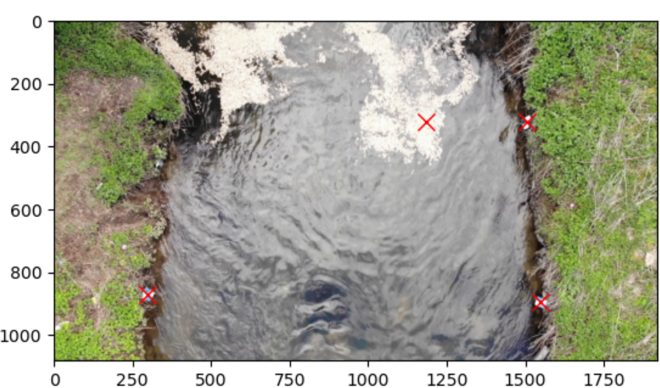
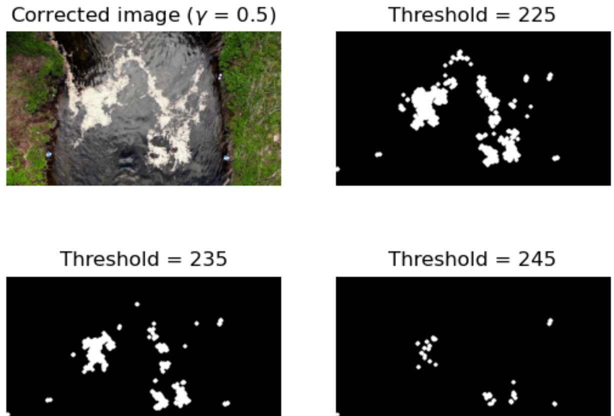
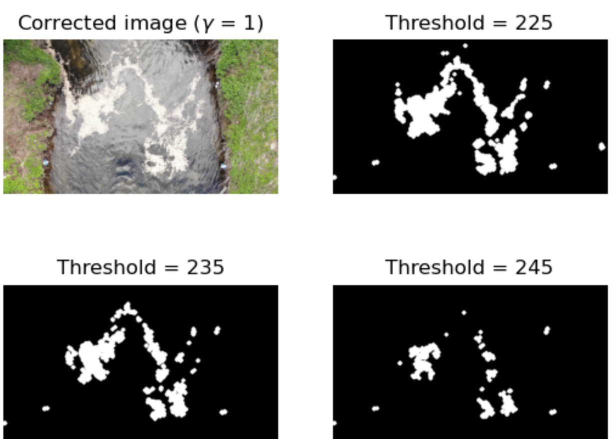


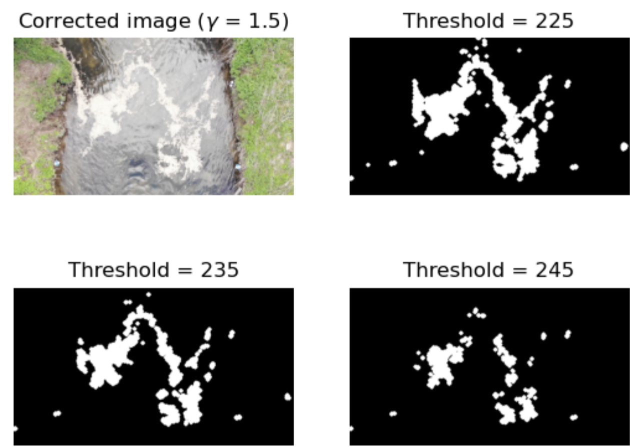
Figure 5.25 – Illustration des balises trouvées pour la vidéo de Limelette en utilisant la méthode implémentée dans `riverApp`. On observe que les 4 points trouvés forment un contour plus semblable à un triangle qu'à un rectangle.



(a) Adaptative thresholding pour une correction assombrie $\gamma = 0.5$



(b) Adaptative thresholding pour l'image originale $\gamma = 1.0$



(c) Adaptative thresholding pour une correction éclaircie $\gamma = 1.5$

FIGURE 5.26 – Illustration de l'effet du changement du seuil de filtre de la palette des blancs sur la première frame de la vidéo de Limelette

Une piste d'amélioration de la méthode serait alors de calculer automatiquement un seuil optimal sur le filtre de la palette des blancs. Un critère d'optimalité possible serait d'imposer que la méthode détecte une certaine proportion de l'image comme étant des pixels blancs.

5.5.7 Tri des balises dans le sens horlogique

Une fois les balises détectées, il est nécessaire de les ordonner afin qu'elles puissent être utilisées correctement par le programme. Le tri des balises permet avant tout au programme de traiter les distances renseignées par l'utilisateur. Pour cela, une liste de règle a été établie pour décider comment les balises vont être triées. Dans le fichier `data_format.md` disponible dans le répertoire `.git` (voir Section 6.1), l'utilisateur peut prendre connaissance de ces règles de tri et renseigner les longueurs en conséquence. La règle principale est : "à partir du premier point $P1$, les balises sont numérotées dans le sens horlogique autour du centre de gravité du quadrilatère formé par les 4 balises".
Pour trouver le point $P1$, les règles se basent sur les 4 quadrants dessinés à partir du centre de gravité et numérotés comme sur la Figure 5.27.

- Si un point unique se trouve dans le quadrant $Q2$, alors ce point est $P1$.
- Si deux points se trouvent dans le quadrant $Q2$, alors s'ils sont alignés verticalement, le point le plus haut est $P1$, sinon c'est le point le plus à gauche qui est $P1$.
- Si aucun point ne se trouve dans le quadrant $Q2$, alors le point le plus haut du quadrant $Q3$ est $P1$.

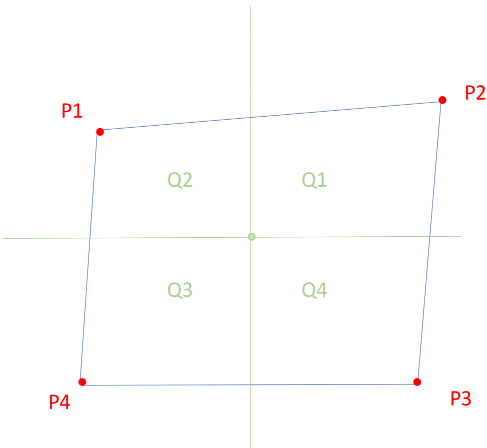


FIGURE 5.27 – Illustration des concepts géométriques utiles pour le tri des balises

5.5.8 Résultats et précision de l'implémentation

Cette section présente les résultats détaillés obtenus avec notre méthode implémentée sur les 4 jeux de données exploitables. On peut d'ores et déjà remarquer que la routine n'est pas encore totalement au point, mais présente tout de même des résultats encourageants. Notons que pour l'entièreté de cette section, le score donné à chaque point détecté réfère à la distanceMatching (telle que définie dans l'Eq. (5.12)) entre la forme détectée et la forme de la balise de référence (voir Annexe A). De plus, il est important de noter que le thresholding de la palette des blancs ainsi que paramétrage des étapes de post-processing ont été calibrés pour l'exemple de Limelette. Pour cette raison, cet exemple présente les meilleurs résultats, ce qui est encourageant pour le développement de la méthode.

Video Globe Challenge 1

Les scores de distanceMatching entre chaque balise détectée et la balise de référence sont présentés dans le Tableau 5.1. La Figure 5.28 présente le résultat visuel obtenu avec la méthode de détection ainsi que le résultat de l'implémentation de la routine de tri des balises.

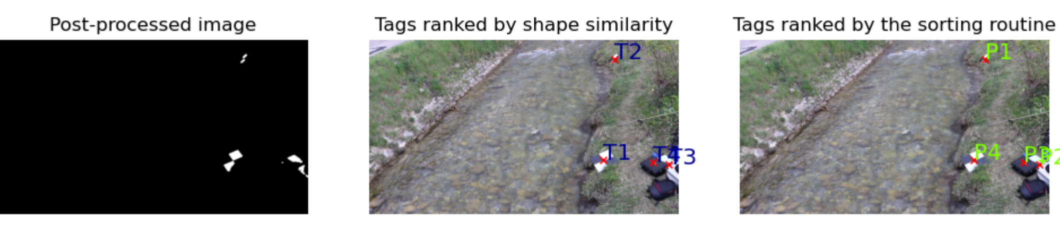


FIGURE 5.28 – Résultats de la méthode de détection automatique des balises sur le jeu de données du Vidéo Globe Challenge 1. La première figure montre la première frame après les étapes de post-processing. La seconde figure montre les balises triées par ordre croissant de distanceMatching (définie dans la Eq. (5.12)). La dernière figure montre les balises triées avec la méthode définie dans la Section 5.5.7.

Dans cet exemple, on peut voir qu'il n'y a que deux balises standards sur l'image (Figure 5.28). Ces deux balises sont bien visibles et on peut détecter la forme du damier à l'oeil nu. Dans ce cas, l'algorithme va bien détecter ces deux balises comme étant les deux points les plus probables avec un bon score de distanceMatching() (Tableau 5.1). En effet, après le post-processing de l'image (image de gauche), les

Balise	T1	T2	T3	T4
matchingDistance()	0.026	0.130	0.219	0.300

Table 5.1 – MatchingDistance entre la balise de référence (Annexe A) et les balises détectées automatiquement pour la vidéo du Video Globe Challenge 1. La numérotation des balises correspond à la numérotation de l'image centrale de la Figure 5.28.

deux damiers se distinguent bien.
Pour ce qui est des deux balises de la rive gauche, l'algorithme ne va pas les détecter du tout. En raison de la taille trop grande du noyau dans l'étape d'érosion, ces deux points sont filtrés et ne seront pas traités par l'étape de correspondance de forme. Cet exemple illustre l'importance de l'utilisation de balises standards mais également l'importance de l'affinage des paramètres des étapes de post-processing. Comme l'algorithme est programmé pour détecter 4 balises, il détecte les deux vraies balises comme étant les deux points les plus probables, mais il va également détecter deux autres points parasites comme étant les troisième et quatrième plus probables. Néanmoins, avec un score bien moins bon que pour les deux vraies balises, ces balises pourraient être rejetées par un module de détection de quadrilatère raisonnable.

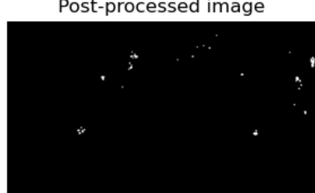
Noirath

Les scores de distanceMatching entre chaque balise détectée et la balise de référence sont présentés dans le Tableau 5.2. La Figure 5.29 présente le résultat visuel obtenu avec la méthode de détection ainsi que le résultat de l'implémentation de la routine de tri des balises.

Balise	T1	T2	T3	T4	T5	T6
matchingDistance()	0.0678	0.0755	0.0962	0.137	0.170	0.176

Table 5.2 – MatchingDistance entre la balise de référence (Annexe A) et les balises détectées automatiquement pour la vidéo de Noirath. La numérotation des balises correspond à la numérotation de l'image centrale de la Figure 5.29

Dans cet exemple, on peut remarquer que les quatre balises sont visibles sur l'image (Figure 5.29). Même si les quatre balises ne sont pas bien visibles, on peut tout de même voir les damiers à l'oeil nu. Pourtant, l'algorithme ne va distinguer qu'une seule des quatre balises et l'unique balise détectée est le point avec la moins bonne disanceMatching() (Tableau 5.2).



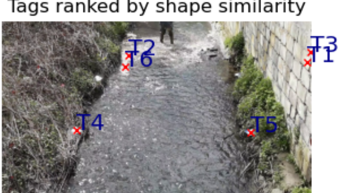


FIGURE 5.29 – Résultats de la méthode de détection automatique des balises sur le jeu de données de Noirath. La première figure montre la première frame avec les étapes de post-processing. La seconde figure montre les balises triées par ordre croissant de distanceMatching (définie dans l'Eq. (5.12)). La dernière figure montre les balises triées avec la méthode définie dans la Section 5.5.7

Comme la luminosité de l'image est importante, l'algorithme va devoir traiter une grande quantité de potentielles balises à partir de l'image post-processée (image de gauche). À cause de cette grande quantité de parties blanches, l'algorithme a de plus grandes chances de ne pas détecter l'intégralité des balises. Cet exemple illustre alors l'importance de l'adaptative thresholding de la palette des blancs.
De plus, l'image après le post-processing présente une proportion importante de carrés blancs et les formes en damier des balises ne sont plus du tout distinguables. Cet exemple illustre alors l'importance du paramétrage des étapes du post-processing de l'image qui devraient accorder plus d'importance à la conservation des formes.

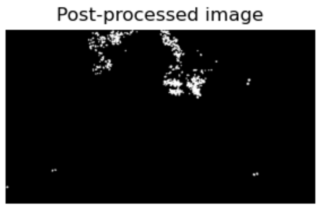
Limelette

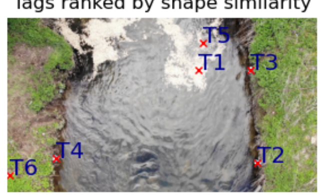
Les scores de distanceMatching entre chaque balise détectée et la balise de référence sont présentés dans le Tableau 5.3. La Figure 5.30 présente le résultat visuel obtenu avec la méthode de détection ainsi que le résultat de l'implémentation de la routine de tri des balises.

Balise	T1	T2	T3	T4	T5	T6
matchingDistance()	0.000	0.032	0.118	0.151	0.290	0.297

TABLE 5.3 – MatchingDistance entre la balise de référence (Annexe A) et les balises détectées automatiquement pour la vidéo de Limelette. La numérotation des balises correspond à la numérotation de l'image centrale de la Figure 5.30.

Dans cet exemple, on peut distinguer les quatre balises (Figure 5.30). Les balises sont bien visibles et la forme des damiers l'est également à l'oeil nu. L'algorithme





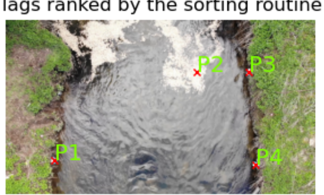


Figure 5.30 – Résultats de la méthode de détection automatique des balises sur le jeu de données de Limelette.La première figure montre la première frame avec les étapes de post-processing. La seconde figure montre les balises triées par ordre croissant de distanceMatching (définie dans l'Eq. (5.12)). La dernière figure montre les balises triées avec la méthode définie dans la Section 5.5.7

Figure 5.31 – Illustration du cluster détecté comme ayant la meilleure distanceMatching() pour la vidéo de Limelette. Intuitivement, on peut voir que la forme ne correspond pas du tout au damier. Cet exemple illustre les limites de la détection de la forme avec la distanceMatching().

va détecter trois des quatre balises avec de bons scores de distanceMatching() comparés aux autres balises non retenues (T5 et T6)(Tableau 5.3). La quatrième balise détectée correspond à un paquet de copeaux sur la rivière. Ce point obtient un score de distanceMatching() de zéro et est donc considéré comme parfaitement équivalent à une balise.
Pour les trois balises détectées, on peut remarquer que les formes des damiers sont bien distinguables sur l'images post-processed (image de gauche).
Cet exemple illustre alors l'une des limites de la méthode de reconnaissance de la forme. En effet, si une forme aléatoire (Figure 5.31) peut avoir un si bon score de distanceMatching(), alors elle sera toujours prise comme la balise la plus probable même si elle n'y ressemble pas.

Cependant, cet exemple illustre également le potentiel de l'ajout d'un module qui retire la rivière de la zone de recherche de balises. En effet, les copeaux blancs sont utilisés dans de nombreux exemples pour faciliter l'analyse PIV. Comme ces copeaux ne sont pas un phénomène naturel, il y a peu de chance de retrouver une forme qui obtiendrait un aussi bon score de matchingDistance() sur les berges.

Rosière

Les scores de distanceMatching entre chaque balise détectée et la balise de référence sont présentés dans le Tableau 5.4. La Figure 5.32 présente le résultat visuel obtenu avec la méthode de détection ainsi que le résultat de l'implémentation de la routine de tri des balises.

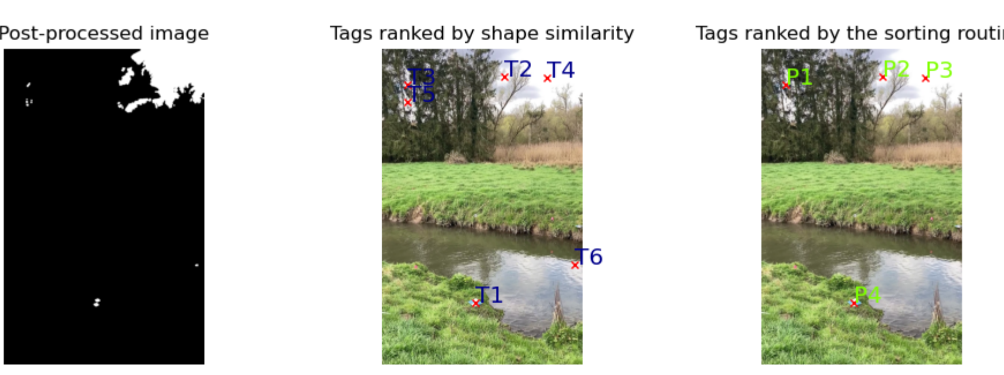


FIGURE 5.32 – Résultats de la méthode de détection automatique des balises sur le jeu de données de Rosière. La première figure montre la première frame avec les étapes de post-processing. La seconde figure montre les balises triées par ordre croissant de distanceMatching (définie dans l'Eq. (5.12)). La dernière figure montre les balises triées avec la méthode définie dans la Section 5.5.7

Balise	T1	T2	T3	T4	T5	T6
matchingDistance()	0.094	0.115	0.137	0.166	0.180	0.265

TABLE 5.4 – MatchingDistance entre la balise de référence (Annexe A) et les balises détectées automatiquement pour la vidéo de Rosière. La numérotation des balises correspond à la numérotation de l'image centrale de la Figure 5.32

Pour cet exemple, deux balises sont visibles sur l'image (Figure 5.32). Les deux balises sont clairement visibles et les formes des damiers sont distinguables

à l'oeil nu. Cependant, l'algorithme n'en détecte correctement qu'une seule mais avec le meilleur score de distanceMatching() (Tableau 5.4). En effet comme on peut le distinguer sur l'image de gauche, la forme du damier est bien distinguable sur l'image post-processed.
Les autres balises détectées sont trois balises dans le ciel. Ces balises sont donc loin de la rivière et l'exemple illustre parfaitement la pertinence de l'ajout d'un module qui imposerait que les balises soient proche de la rivière. De plus, les quatre balises détectées ne forment pas un quadrilatère raisonnable. L'ajout d'un tel module aurait également sa pertinence dans ce cas-ci.

Chapitre 6
Version finale du projet

Ce chapitre a pour but de dresser un état des lieux de l'avancement du projet `riverApp`. Le projet a été récupéré en début d'année dans l'état où il a été introduit dans le Chapitre 2. Depuis lors, différentes modifications ont été apportées comme expliqué dans le Chapitre 4 et dans le Chapitre 5. Ainsi, le projet `riverApp` ne ressemble plus à ses versions précédentes, tant d'un point de vue de l'implémentation du code que de son utilisation.

Étant donné que le projet n'est pas encore terminé et que différentes parties prenantes devront encore s'y intéresser, il est essentiel de faciliter sa prise en main future. En effet, la complexité du code source du projet `riverApp` a été un obstacle majeur au lancement des développements (comme expliqué dans le Chapitre 3). Dans cette nouvelle version, les différentes étapes sont plus facilement identifiables au sein du code source et peuvent être consultées aisément par les futurs développeurs.

Dans ce chapitre, la Section 6.1 est consacrée à la description du projet `riverApp` dans sa nouvelle version et peut être considérée comme un guide pour les développeurs. Cette section présente l'organisation du code source. Ensuite, un récapitulatif des modifications apportées au projet est présenté dans la Section 6.2. Enfin, la Section 6.3 permet de mettre en évidence les désavantages et les limites du projet `riverApp`. Cette dernière section clôture le travail et sert donc de base pour de futurs sujets de recherche.

6.1 Description du projet `riverApp` final

6.1.1 Structure basée sur `pyOrc`

En tout premier lieu, il est important de noter que le code source de `riverApp` repose désormais sur les fonctions implémentées dans l'API du projet `PyOrc`.

De plus, afin de rester cohérent avec l'architecture de `PyOrc`, le paradigme en pipeline est désormais mis en oeuvre pour exécuter le code du fichier *main.py* de `riverApp`, comme expliqué dans la Section 4.3.

Le répertoire `.git` actuel est structuré conformément à ce qui est décrit dans la Figure 6.1. Les dossiers pertinents pour le développement sont présentés dans les paragraphes suivants.

Tout d'abord, le dossier **dev** contient les quatre principales étapes du pipeline de calcul, telles que décrites dans la Section 4.3. On y trouve également les implémentations des nouvelles fonctionnalités introduites dans le Chapitre 5.

Ensuite, le dossier **example/pyorc_examples** est destiné à l'apprentissage de l'utilisation des fonctions de pyorc. Le dossier **example/riverapp_examples**, quant à lui est le dossier dans lequel il est recommandé de mettre toutes les données relatives aux exemples analysés avec la `riverApp`. Il est important de noter que tous les fichiers ajoutés à ce dossier seront automatiquement ignorés par `Git` lors de la mise en ligne. Cela permet à l'utilisateur d'y déposer des données localement sans encombrer l'espace en ligne dédié au répertoire. Ce dossier contient également le manuel descriptif du formatage des données pour une utilisation optimale des fonctions de l'application.

De plus, le dossier **pyorc** contient l'API de `PyOrc` ainsi que toutes les fonctions implémentées par ses développeurs. Si une fonction de `PyOrc` ne répond pas aux attentes d'utilisations, il est possible de la trouver dans ce dossier et de la modifier. C'est l'un des principaux avantages d'avoir fork le projet `PyOrc` pour reconstruire `riverApp`, comme expliqué dans la Section 4.1.

Enfin, le fichier `main.py` est le fichier principal de `riverApp`. Dans ce fichier, les données sont chargées pour être utilisées dans l'application et les fonctions du pipeline sont appelées (pour une explication du paradigme du pipeline, voir la Section 4.3). C'est ce fichier que l'utilisateur doit exécuter pour utiliser l'application dans son ensemble. Une version simplifiée du fichier `main.py` est présentée dans la

Code 6.1.

6.1.2 Interface graphique

En parallèle du changement de coeur du projet (passage du coeur riverApp à un coeur basé sur `PyOrc`), une refonte graphique est prévue. Elle permettra d'améliorer l'expérience utilisateur et de profiter de la puissance de `PyOrc`. Cette partie du travail sera réalisée par Arnaud Meunier, étudiant à l'EPHEC en Technologies de l'Informatique. La refonte est documentée dans son rapport de fin d'études [30].

```
riverApp
├─ .github/workflow
│  └─ # github manager
├─ dev
│  ├─ gcp_detection_trials
│  │  └─ # files for pattern recognition algorithm
│  ├─ utils
│  │  ├─ GCP_detection.py
│  │  └─ reference.png
│  ├─ cam_creation.py
│  ├─ mask_and_plot_piv.py
│  ├─ stab_and_piv.py
│  └─ transect.py
├─ docs
│  └─ # pyorc documentation
├─ envs
│  └─ # environment files
├─ example
│  ├─ pyorc_example
│  │  ├─ # files to learn to use pyorc library
│  │  └─ notebook_first_use.ipynb
│  └─ riverapp_example
│     ├─ img
│     │  └─ # images used for data format
│     ├─ VGC1
│     │  └─ # all data related with VGC1
│     └─ data_format.md
├─ pyorc
│  └─ #all the pyorc functions
├─ tests
│  └─ # pyorc tests
├─ .gitignore
├─ CHANGELOG.md
├─ Dockerfile
├─ LICENSE
├─ README.md
├─ TRADEMARK.md
├─ environment.yml
├─ main.py
└─ setup.py
```

FIGURE 6.1 – Arbre des différents fichiers et dossiers d'intérêt sur le répertoire `Git` du projet `riverApp`. Les commentaires en gris réfèrent à des parties du répertoire non développées dans cet arbre.

```python
import packages

if __name__ == "__main__":

    ###########
    # Load data file (video, dim and bathy)
    ###########

    # Data variables such as the paths to files, waterlevel, etc
    [...]

    # Checking if the data files are available
    [...]

    # Loading the data from files
    video = cv2.VideoCapture(video_filepath)
    dimension = np.loadtxt(dimension_filepath, delimiter=",")
    bathy = pd.read_csv(bathy_filepath)

    ###########
    # Process the video file and create a Video object
    ###########

    # create the Video object
    video = pyorc.Video(video_filepath, start_frame=start_frame, end_frame=
        end_frame, stabilize="fixed" if to_stabilize else None, freq=freq, h_a=
        water_level)

    ###########
    # RiverApp processes in pipeline
    ###########

    # 1 = cam creation, 2 = piv process, 3 = mask application, 4 = transect choice
    step = 3

    # Start at the camera calibration
    if step == 1:
        cam_config = cam_creation.cam_create(video, directory, dimension,
            water_level)
    else:
        cam_config = pyorc.load_camera_config(directory + "cam_config.json")

    video.camera_config = cam_config

    # Start at the piv computation
    if step <= 2:
        stab_and_piv.process_piv(directory, video)

    # Start at the post-process of the velocity field
    if step <= 3:
        ds = xr.open_dataset(directory + "piv.nc")
        mask_and_plot_piv.mask_and_plot(directory, ds, video)

    # Start at the discharge computation
    if step <= 4:
        ds = xr.open_dataset(directory + "piv_masked.nc")
        transect.transect(ds, video, directory, bathy_filepath)
```

Code 6.1 – Fichier *main.py* épuré. La fonction principale du projet riverApp est structurée en pipeline composé de quatre étapes clés, après avoir chargé la vidéo : (i) la caméra doit être calibrée ; (ii) l'analyse PIV est réalisée ; (iii) le résultat est filtré ; (iv) le débit est estimé.

6.2 Modifications supplémentaires apportées

Cette section présente les contributions supplémentaires apportées au projet `riverApp`. Les deux principales modifications qui ont été apportées sont décrites dans le Chapitre 4 et dans le Chapitre 5. Cependant, d'autres modifications ont également été apportées en parallèle. Ce sont ces autres modifications mineures qui seront présentées dans les sections suivantes.

6.2.1 Formatage des données

Dans la nouvelle version de l'application `riverApp`, il est nécessaire de fournir les données dans un format spécifique. Le format de chaque fichier de données est expliqué dans le fichier *data_format.md* disponible dans l'Annexe B. Ce fichier a pour but d'expliquer les différentes données à inclure et la manière de les inclure afin que le programme les comprenne correctement.

6.2.2 Création d'un *Jupyter Notebook* de première utilisation de `pyOrc`

Afin de permettre aux futurs développeurs de l'application de se familiariser avec l'utilisation de l'API de `PyOrc`, ce notebook reprend les principales étapes de calcul du débit. La différence par rapport à l'utilisation du fichier `main.py` est que ce notebook reprend l'intégralité du processus de calcul du débit. Pour une meilleure clarté du code source, le fichier `main.py` se contente d'appeler des sous-fichiers, ce qui ne facilite pas l'appréhension des fonctions de `PyOrc`.

6.2.3 Création d'un fichier `__init__.py` pour `pyOrc`

La création de ce fichier `__init__.py` permet de charger l'ensemble de l'API de `PyOrc` dans le fichier `main.py`. En effet, comme expliqué dans [53], l'utilisation d'un fichier nommé `__init__.py` permet de regrouper les différents fichiers de l'API qui contiennent des fonctions. Cette approche résout les problèmes de conflits de chemin (*path*) que le développeur pourrait rencontrer lors du chargement des fichiers dans le fichier `main.py`.

6.3 Limites et désavantages

Cette section vise à répertorier les limites de `riverApp`. Les principales limites sont liées à deux freins majeurs dans le développement de l'application. Tout d'abord, le premier frein important est lié à l'utilisation des données (voir la Section 6.3.1). Ensuite, le deuxième frein majeur est dû à la complexité du système de géométrie virtuelle `PyOrc` ainsi qu'à son implémentation (voir la Section 6.3.2). De plus, une série d'autres pistes d'amélioration sont explorées dans la 6.3.3.

6.3.1 Limites liées aux données utilisables

Tout d'abord, une des limites de `riverApp` est liée à la disponibilité de jeux de données exploitables. En effet, la plupart des jeux de données disponibles ne sont pas complets et donc inutilisables.

Ensuite, l'application manque de robustesse lorsque les données ne sont pas prises dans de bonnes conditions ou ne sont pas correctement reportées.

Par exemple, lors de la détection des GCP, la routine implémentée (Section 5.5) est sensible à la visibilité des balises. Si un damier ne peut pas être clairement vu à l'oeil nu, l'algorithme ne pourra pas non plus le détecter.
Un autre exemple est lors du renseignement de la section d'intérêt. Cette étape permet de décider de la section de la rivière sur laquelle la vitesse de l'eau sera intégrée. Or cette intégration utilise la bathymétrie et si la section choisie ne correspond pas à la section sur laquelle la bathymétrie a été calculée, le résultat peut être faussé. En effet, la bathymétrie d'une rivière n'est pas toujours constante et faire cette hypothèse peut avoir un impact important sur le résultat de débit calculé.

La piste d'amélioration pour cette section serait alors de développer une liste de règles de bonne pratique pour la prise de mesures ainsi qu'une série de données obligatoires pour l'utilisation de `riverApp`. Cette liste de règles serait proposée à l'utilisateur avant qu'il ne récolte ses données afin que celles-ci soient prises dans des conditions optimales. Certains points d'attention à avoir sont listés dans le Tableau 6.1. Il faut cependant mentionner qu'un objectif de l'application est de pouvoir être utilisée dans de nombreuses conditions. La liste de règles de bonne pratique ne devrait donc pas être trop restrictive [1].

1. Il faut bien comprendre que l'objectif n'est pas de cacher le manque de robustesse de l'application en imposant une prise de mesures plus précise, mais simplement de permettre de recueillir des résultats satisfaisants malgré le développement encore précoce de l'application

Suite à cette liste de règles, il faudrait développer un plus large ensemble de mesures sur différentes rivières dans différentes conditions. Cet ensemble de mesures complètes permettrait alors de tester de manière plus robuste les nouvelles implémentations.

Donnée	Point d'attention
Vidéo	Utilisation de exactement quatre balises standards
Vidéo	Placement des balises de manière à ce que le damier soit bien visible
Bathymétrie	Noter la section à laquelle correspond la prise de mesure de la bathymétrie
Distances	Au moins 5 des 6 distances entre les balises doivent être mesurées.
...	...

TABLE 6.1 – Liste non exhaustive de points d'attention à porter lors de la prise de mesures en rivière.

Une autre piste intéressante est de reprendre un jeu de données d'Internet et de le convertir au bon format actuel. Nous avons par exemple trouvé cet article scientifique [40] qui introduit un jeu de plusieurs vidéos. C'est d'ailleurs sur ce jeu là que le projet `PyOrc` a été testé [55] et approuvé comme fournissant des données pertinentes pour les cours d'eau. Une telle ressource adaptée au bon format pour `riverApp` serait un avantage conséquent pour les prochaines équipes qui vont collaborer et développer le code de ce projet.

6.3.2 Limites liées à l'implémentation de `pyOrc`

Depuis l'implémentation de `riverApp` avec l'API de `PyOrc`, de nombreuses nouvelles fonctions sont utilisées. Bien que ces fonctions aient été implémentées par les chercheurs de la TUDelft, elles sont totalement open-source et il est possible de retrouver le code de leurs implémentations directement sur le répertoire `.git` de `riverApp`.
Malheureusement, pour cette toute première implémentation croisée, l'API de `PyOrc` n'est pas encore maîtrisée dans tous ses détails. Certaines fonctions et certains paramètres restent des boîtes noires.

Dans cette première version du code, telle qu'illustrée dans la Section 4.3, la principale incompréhension concerne la configuration de la caméra et plus particulièrement l'interprétation du référentiel local construit à partir des balises.
Un premier exemple de résultats incompris concerne la position de la caméra qui doit être renseignée dans le référentiel local. Sans ce paramètre, la configuration de la caméra ne pourra pas être créée. Ce paramètre n'influence pas directement le

résultat mais suscite néanmoins des interrogations.

Un second exemple concerne le sens des axes du référentiel local. Ces axes sont définis en fonction de l'ordre dans lequel les balises sont renseignées. Cependant, selon le sens d'écoulement de la rivière le débit calculé peut être négatif dans certains cas.
Une compréhension approfondie du fonctionnement du système de référentiel local dans `PyOrc` pourrait permettre de résoudre ces erreurs et d'obtenir des résultats cohérents, indépendamment du sens d'écoulement de la rivière.

Une autre source d'incompréhension concerne le renseignement de la section d'intérêt. Comme illustré sur la Figure 6.2, la section d'intérêt est sélectionnée de manière plus large que la rivière elle-même. Dans ce cas, le programme peut trouver des vecteurs vitesse non nuls sur les berges, alors qu'aucun vecteur vitesse n'est normalement calculé dans ces zones lors de l'analyse PIV. Cette situation peut entraîner des erreurs dans le calcul du débit et nécessite une modification.

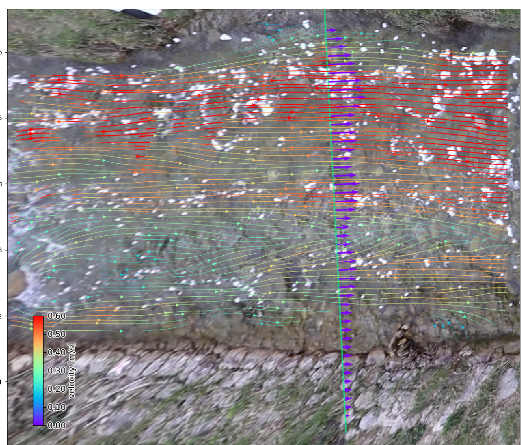


FIGURE 6.2 – Illustration de la section d'intérêt calculée sur la vidéo du Vidéo Globe Challenge 1 grâce au programme `riverApp`. La section d'intérêt dépasse les bords de la rivière et la vitesse est alors non nulle sur les berges.

6.3.3 Autres pistes de développement

Implémentation des différentes techniques de calcul du niveau d'eau

Lors de la transition vers `PyOrc`, l'application `riverApp` a perdu les fonctions qui permettent de calculer le niveau d'eau. Il est nécessaire de réimplémenter ces fonctions dans la nouvelle version de `riverApp`.

Amélioration de la routine de détection des GCP

L'implémentation de la détection des GCP (Ground Control Points) fonctionne si les balises sont bien visibles. Cependant, dans des configurations où les damiers ne sont pas correctement distinguables, l'algorithme peine à trouver des solutions exactes. Une série d'idées d'améliorations de la routine est proposée dans la Section 5.5.6.

Annexes

A Balise standard

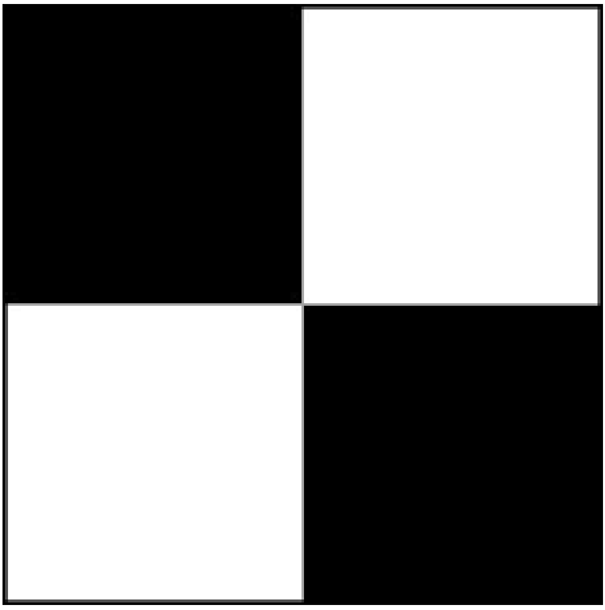


Figure A.1 – Balise standard utilisée pour la mesure en rivière.

B data_format.md

Load files in the appropriate directory (recommended)

First, create a folder in

```
./examples/riverapp_examples/your_river/
```

where **your_river** is the name of your dataset Note that everything you add in the **riverapp_examples** will be ignored by git thanks to **.gitignore**

The three following files must exists in order to launch the process.

```
./examples/riverapp_examples/your_river/video.mp4
./examples/riverapp_examples/your_river/dimension.txt
./examples/riverapp_examples/your_river/bathymetry.txt
```

Files format (mandatory)

Then format your files with the following specifications.

Video file

Should be a video of the river with four visible tags.

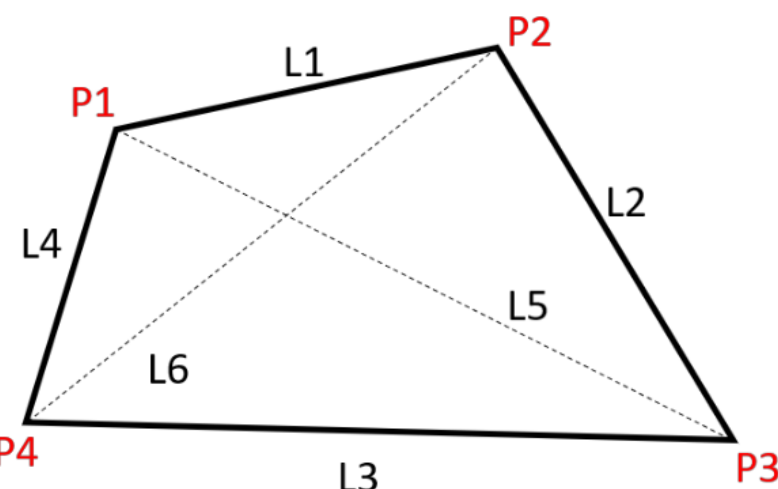
Dimension file

A file containing one line with the 6 distances :

```
L_1, L_2, L_3, L_4, L_5, L_6
```

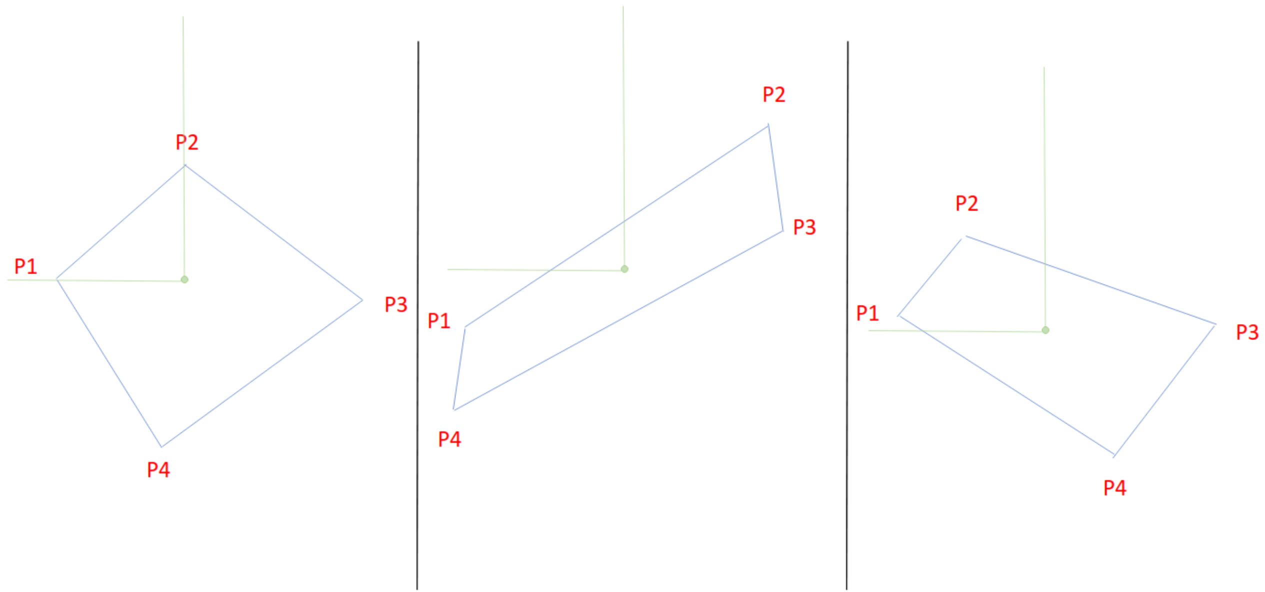
where these distances refers to the distances between the reference points P1,P2,P3 and P4 (P1 top left corner then clockwise order)
Note that the number should be separated with a comma
Note that a line starting with '#' will be ignored

The points should be labeled as in the following display



As you see it on the picture, the top right point should always be the first one. Then the following go on in clockwise order.
Note that the position of the points doesn't depend on the orientation of the river but on the view of the camera. Therefore, it doesn't matter if the river inflow through L1 or L2 (or even L3 or L4) as long as the points display on the video with these specific positions.

There might be ambiguity on which point is the top right corner as the following scenarii show on image.
In case of ambiguity (i.e. not exactly one point in the second quadrant), take the left point as **P1**.
If an ambiguity remains, see the sorting rules in the **sort_src()** in **dev/utils/GCP_detection.py** and note that the first principle of the rule is: a unique point in the second quadrant around the centroid of the polygon is always **P1**.

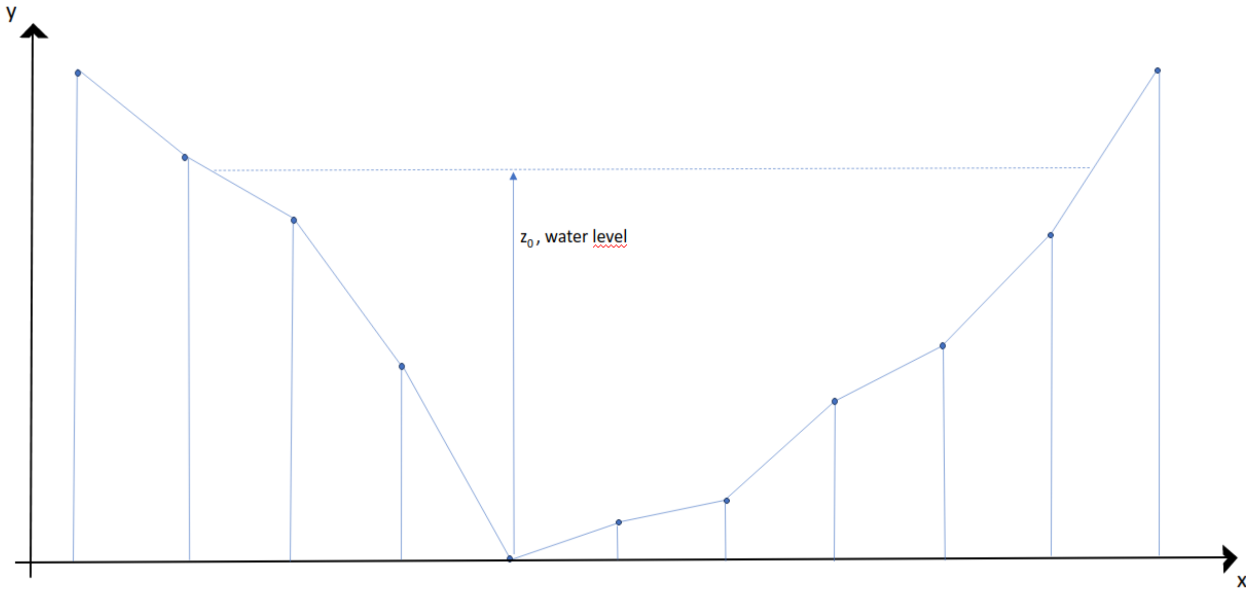


Bathymetry file

A file containing the (x,y) coordinates refering to the depth of the river along a section of the river

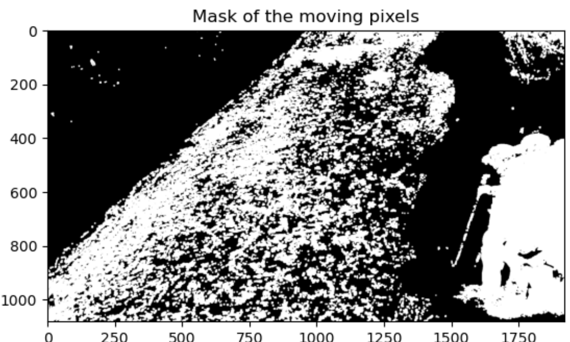
```
x,y
x1,y1
x2,y2
...
```

Note that the the first line of that file should be **x,y**
Note that the number should be separated with a comma (**,**)

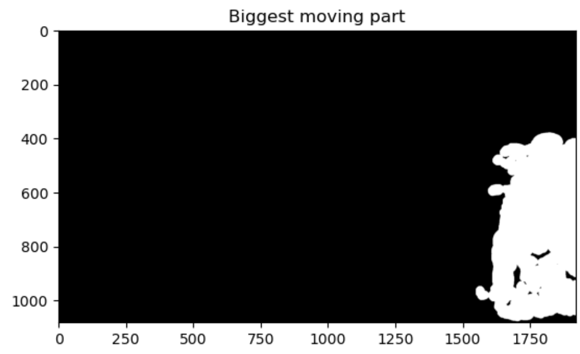


Note that the x-coord refer to the coordinate along the section of the river and the y-coord refer to the depth of the river as shown on the picture.
Note that the deepest point of the river should have a y-coord equal to zero. Note that the water level will therefore be taken from that point.

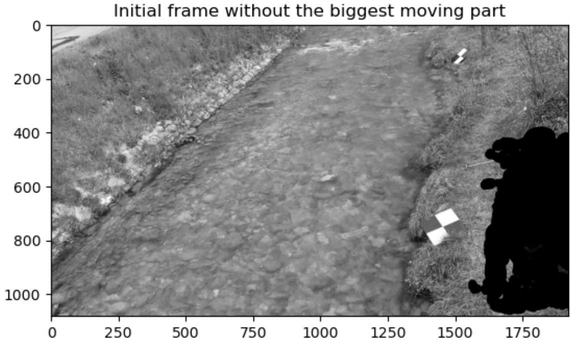
C Tests supplémentaires sur la détection de la rivière



(a) Masque des pixels mobiles en utilisant un seuil d'intensité de 20

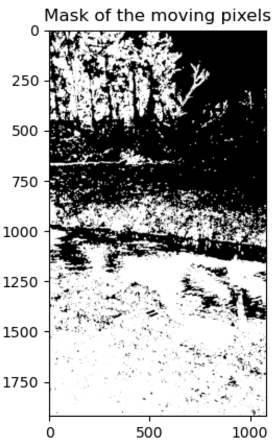


(b) Détection du plus grand groupe de pixel mobile

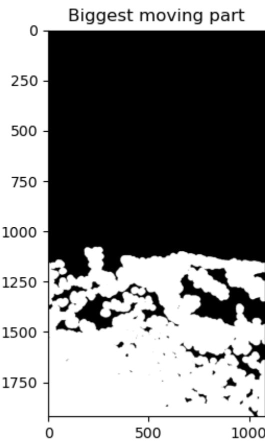


(c) Première frame de la vidéo où la plus grosse partie mobile est retirée

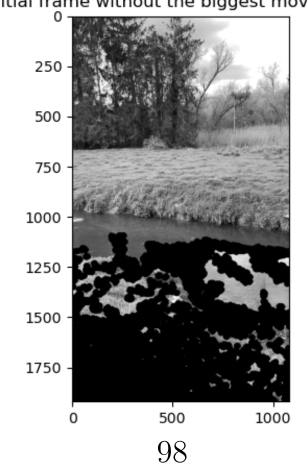
FIGURE A.2 – Illustration de la méthode implémentée pour détecter la rivière sur l'exemple du Video Globe Challenge 1



(a) Masque des pixels mobiles en utilisant un seuil d'intensité de 20

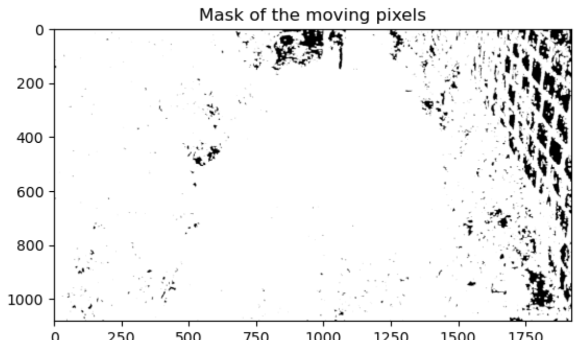


(b) Détection du plus grand groupe de pixel mobile

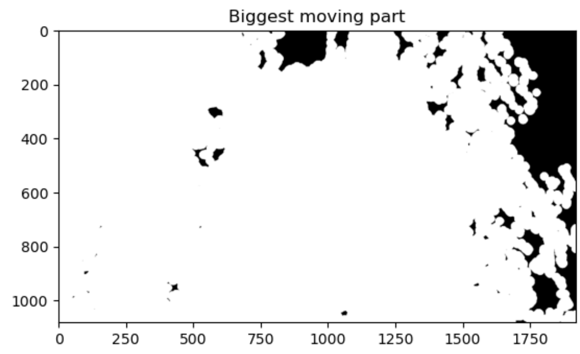


(c) Première frame de la vidéo où la plus grosse partie mobile est retirée

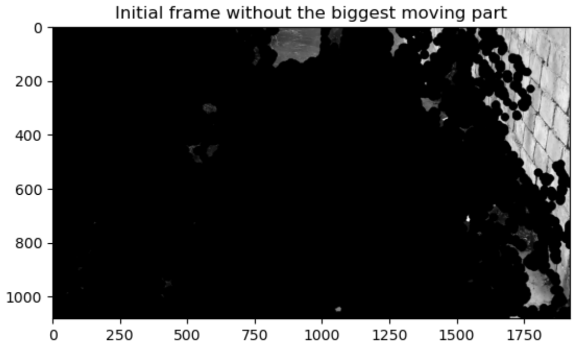
Figure A.3 – Illustration de la méthode implémentée pour détecter la rivière sur l'exemple de Rosière



(a) Masque des pixels mobiles en utilisant un seuil d'intensité de 20



(b) Détection du plus grand groupe de pixel mobile



(c) Première frame de la vidéo où la plus grosse partie mobile est retirée

FIGURE A.4 – Illustration de la méthode implémentée pour détecter la rivière sur l'exemple de Noirath

Bibliographie

[1] Rosebrock ADRIAN. *Multi-template matching with OpenCV*. 2021.

[2] AISHWARYA SINGH. *SIFT Algorithm | How to Use SIFT for Image Matching in Python*. 2019. URL : https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/.

[3] Herbert BAY, Tinne TUYTELAARS et Luc VAN GOOL. "SURF : Speeded up robust features". In : t. 3951. Juill. 2006, p. 404-417. ISBN : 978-3-540-33832-1. DOI : 10.1007/11744023_32.

[4] Raphaël BOURSICAUD et al. "Gauging extreme floods on YouTube : application of LSPIV to home movies for the post-event determination of stream discharges : Application of LSPIV to Flood Home Movies". In : *Hydrological Processes* 30 (mai 2015). DOI : 10.1002/hyp.10532.

[5] Michael CALONDER et al. "BRIEF : Binary Robust Independent Elementary Features". In : t. 6314. Sept. 2010, p. 778-792. ISBN : 978-3-642-15560-4. DOI : 10.1007/978-3-642-15561-1_56.

[6] John CANNY. "A computational approach to edge detection". In : *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), p. 679-698.

[7] Kharagpur CE IIT. *Flow Dynamics in Open Channels and Rivers*. CE IIT, Kharagpur.

[8] Chin-Sheng CHEN et al. "An accelerating CPU based correlation-based image alignment for real-time automatic optical inspection". In : *Computers Electrical Engineering* 49 (2016), p. 207-220. ISSN : 0045-7906. DOI : https://doi.org/10.1016/j.compeleceng.2015.09.010.

[9] Local Devices CONSORTIUM. *pyOpenRiverCam 0.4.3*. 2023. URL : https://localdevices.github.io/pyorc/index.html.

[10] J. Coz et al. “Image-based velocity and discharge measurements in field and laboratory river engineering studies using the free FUDAA-LSPIV software”. In : *Proceedings of the International Conference on Fluvial Hydraulics, RIVER FLOW 2014* (août 2014), p. 1961-1967. doi : 10.1201/b17133-262.

[11] Deepanshu Tyagi. *Introduction to BRIEF(Binary Robust Independent Elementary Features)*. 2019. url : https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6.

[12] Michael Dougherty et Christopher Haring. *FluvialGeomorph User Manual*. 2021. url : https://github.com/FluvialGeomorph/FluvialGeomorph-toolbox.git.

[13] Célestin Dufromont et Pierre Laurent. “Mesures de débit en rivière à partir de techniques d’imagerie : développement d’une application”. Master’s Thesis. UCLouvain, 2022.

[14] Bruno Fahy. “Namur : la tortue de la Citadelle emballée, l’oeuvre d’un plaisantin”. In : *RTBF Info* (août 2015).

[15] Pasquale Ferrara, Rudolf Haraksim et Laurent Beslay. “Performance Evaluation of Source Camera Attribution by Using Likelihood Ratio Methods”. In : *Journal of Imaging* 7.7 (2021). doi : 10.3390/jimaging7070116.

[16] Edouard de Grand Ry et Charlotte Everaert. “Mesures de débit en rivière à partir de techniques d’imagerie”. Master’s Thesis. UCLouvain, 2021.

[17] Gwennaëlle Gribaumont. “Namur, capitale culturelle”. In : *L’Eventail* (24 oct. 2022).

[18] Christophe Halbardier. “Namur : la Ville veut acheter l’œuvre de Jan Fabre”. In : *Le Soir* (1er sept. 2015).

[19] Fang Han et al. “Using Edge SIFT Points for Simple Shape Object Matching”. In : (nov. 2013). doi : 10.2991/icmt-13.2013.66.

[20] Amin Hashem. *How To Correct Lens Distortion – DxO Optics Pro 11 Is The Answer*. url : https://ehabphotography.com/how-to-correct-lens-distortion-dxo-optics-pro-11-is-the-answer/.

[21] Ming-Kuei Hu. “Visual pattern recognition by moment invariants”. In : *IRE Transactions on Information Theory* 8.2 (1962), p. 179-187. doi : 10.1109/TIT.1962.1057692.

[22] Hydrométrie en Wallonie. *Réseaux de mesure*. Article. Service public Wallon.

[23] Cardio Logic Inc. *TECHNOLOGY : Orthorectification*. url : http://www.cardiofx.com/technology/.

[24] Laurent JACQUES. *Slides issues du cours d'Image Processing (LELEC2885)*. Fév. 2022.

[25] Magali JODEAU et al. "Laboratory and field lspiv measurements of flow velocities using Fudaa-LSPIV a free user-friendly software". In : (mars 2017).

[26] Ebrahim KARAMI, Siva PRASAD et Mohamed SHEHATA. *Image Matching Using SIFT, SURF, BRIEF and ORB : Performance Comparison for Distorted Images*. 2017.

[27] J. LE COZ et al. "Image-based velocity and discharge measurements in field and laboratory river engineering studies using the free FUDAA-LSPIV software". In : *River Flow*. Lausanne, Switzerland, sept. 2014, p. 7.

[28] David LOWE. "Distinctive Image Features from Scale-Invariant Keypoints". In : *International Journal of Computer Vision* 60 (nov. 2004), p. 91-. DOI : 10.1023/B:VISI.0000029664.99615.94.

[29] Satya MALLICK et Krutika BAPAT. *Shape Matching using Hu-Moments*. URL : https://learnopencv.com/shape-matching-using-hu-moments-c-python/ (visité le 30/06/2023).

[30] Arnaud MEUNIER. "Développement de l'interface graphique de riverApp". Master's Thesis. EPHEC, 2023.

[31] OPENCV. *OpenCV : Color Conversion*. URL : https://docs.opencv.org/4.x/de/d25/imgproc_color_conversions.html#color_convert_rgb_gray%5C%20documentation (visité le 29/05/2023).

[32] OPENCV. *OpenCV : Contours : More functions*. URL : https://docs.opencv.org/3.4/d5/d45/tutorial_py_contours_more_functions.html (visité le 30/05/2023).

[33] OPENCV. *OpenCV :Detection of ArUco Markers*. URL : https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (visité le 29/05/2023).

[34] OPENCV. *Template Matching in OpenCV 4.7.0*, 2023. URL : https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html.

[35] OPENCV. *OpenCV - Open Computer Vision Library*. 1999. URL : https://opencv.org/ (visité le 10/02/2023).

[36] OPENCV 3.4.19-DEV. *Canny Edge Detection*. URL : https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[37] OPENCV 3.4.19-DEV. *FAST Algorithm for Corner Detection*. URL : https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html.

[38] OpenCV 3.4.19-dev. *Introduction to SURF (Speeded-Up Robust Features)*. url : https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html.

[39] Antoine Patalano, Carlos Marcelo García et Andrés Rodríguez. "Rectification of Image Velocity Results (RIVeR) : A simple and user-friendly toolbox for large scale water surface Particle Image Velocimetry (PIV) and Particle Tracking Velocimetry (PTV)". In : *Computers Geosciences* 109 (2017), p. 323-330. issn : 0098-3004. doi : https://doi.org/10.1016/j.cageo.2017.07.009.

[40] M. T. Perks et al. "Towards harmonisation of image velocimetry techniques for river surface velocity observations". In : *Earth System Science Data* 12.3 (2020), p. 1545-1559. doi : 10.5194/essd-12-1545-2020.

[41] A. Walker R. Fisher S. Perkins et E. Wolfart. *Morphology - Opening*. url : https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm (visité le 29/05/2023).

[42] Kenneth Reitz. *Structuring Your Project - The Hitchhiker's Guide to Python*. url : https://docs.python-guide.org/writing/structure/ (visité le 23/05/2023).

[43] Simon Renard. "Development of an application for river discharge measurement". Master's Thesis. UCLouvain, 2020.

[44] OpenPIV team Revision. *Basics of the PIV algorithms*. 2014. url : https://openpiv.readthedocs.io/en/latest/src/piv_basics.html#Basics-of-the-PIV-algorithms.

[45] Johan Rockström et al. "Safe and just Earth system boundaries". In : *Nature* (mai 2023), p. 1-10. doi : 10.1038/s41586-023-06083-8.

[46] Edward Rosten et Tom Drummond. "Machine Learning for High-Speed Corner Detection". In : t. 3951. Juill. 2006. isbn : 978-3-540-33832-1. doi : 10.1007/11744023_34.

[47] Ethan Rublee et al. "ORB : an efficient alternative to SIFT or SURF". In : nov. 2011, p. 2564-2571. doi : 10.1109/ICCV.2011.6126544.

[48] Julian Friedrich Sauterleute et Julie Charmasson. "A computational tool for the characterisation of rapid fluctuations in flow and stage in rivers caused by hydropeaking". In : *Environmental Modelling Software* 55 (2014), p. 266-278. issn : 1364-8152. doi : https://doi.org/10.1016/j.envsoft.2014.02.004.

[49] Rainbow Sensing. *OpenRiverCam*. 2023. url : https://rainbowsensing.com/index.php/programs/#openrivercam.

[50] Sofiane Sahir. *Canny Edge Detection Step by Step in Python — Computer Vision*. url : https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.

[51] Andreas Tausendfreund, Dirk Stöbener et Andreas Fischer. “In-Process Measurement of Three-Dimensional Deformations Based on Speckle Photography”. In : *Applied Sciences* 11.11 (2021). issn : 2076-3417. doi : 10.3390/app11114981.

[52] TENEVIA. *TENEVIA CAMFLOW vitesses de surface et débits par caméra*. url : https://www.tenevia.com/fr/sensor/cam-flow-mesure-de-debit-par-camera-lspiv-discharge/ (visité le 02/05/2023).

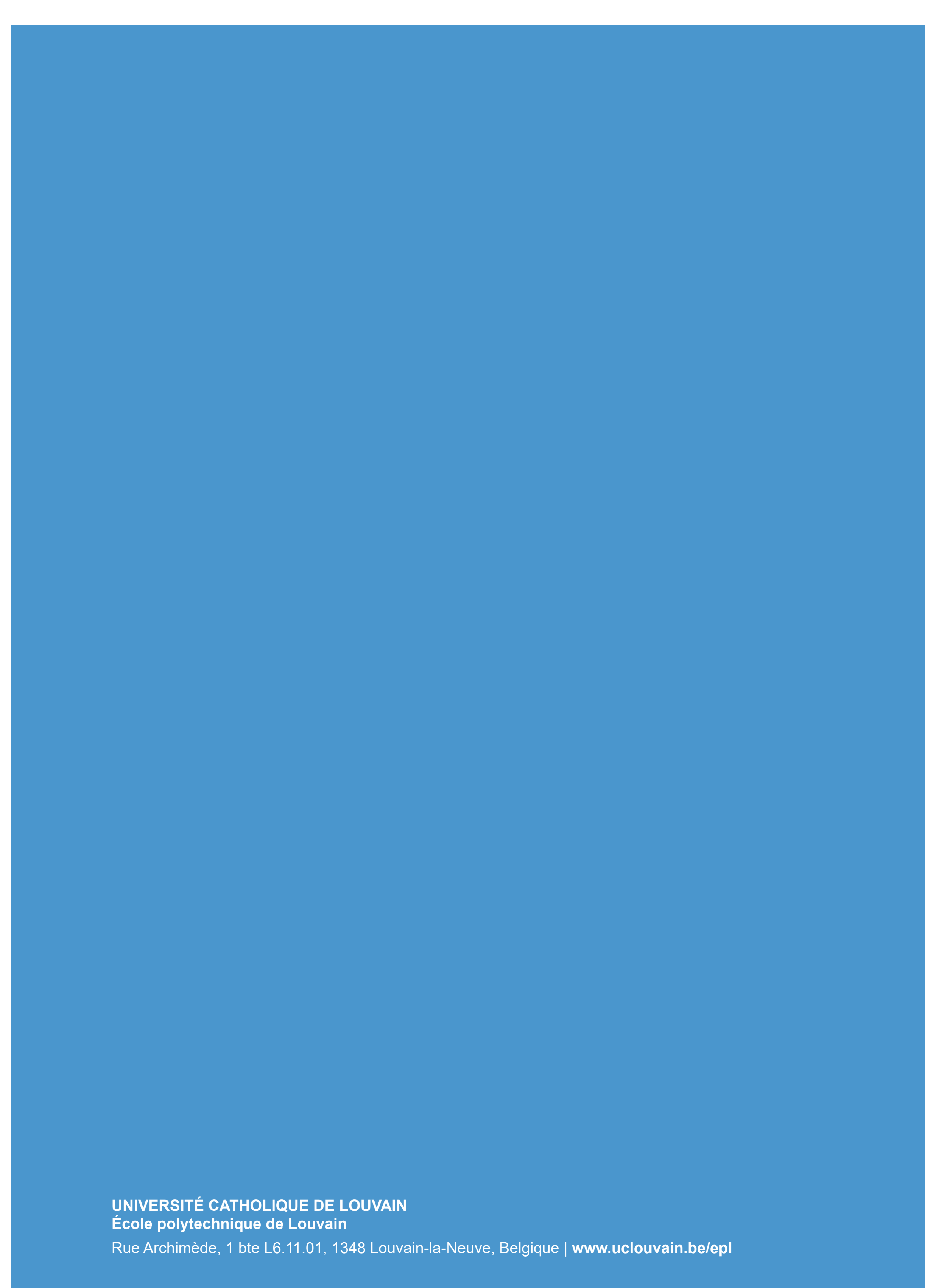
[53] Varshita, Sher. *Understanding Python imports, __init__.py and pythonpath — once and for all*. url : https://towardsdatascience.com/understanding-python-imports-init-py-and-pythonpath-once-and-for-all-4c5249ab6355.

[54] Wikipedia. *Mathematical morphology - Wikipedia*. url : https://en.wikipedia.org/wiki/Mathematical_morphology (visité le 29/05/2023).

[55] Hessel Winsemius. *Linkedin Post de Hessel Winsemius*. 2022. url : https://www.linkedin.com/posts/hessel-winsemius-a53b1b13_pyopenrivercam-surface-velocities-activity-6950527387784626176-bz-m?utm_source=linkedin_share&utm_medium=android_app (visité le 02/06/2023).

[56] Hessel Winsemius. *Towards free and open-source river flow observations with cameras : pyOpenRiverCam*. 2022. url : https://www.linkedin.com/pulse/towards-free-open-source-river-flow-observations-hessel-winsemius?trk=public_profile_article_view (visité le 02/06/2023).

[57] zsiki. *Find ArUco markers in digital photos*. url : https://github.com/zsiki/Find-GCP (visité le 29/05/2023).


UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**