

Université de Montpellier
Faculté des Sciences
L3 informatique.

Suivi de seiches dans des vidéos sous-marine

Projet de programmation 2

Vaisse Ariane Beldjilali Maxime Young Brun Luis-Miguel
Combe-Ounkham Gabriel

Encadré par
Houda Hammami

2022-2023

Résumé

Le suivi de seiche dans des vidéos sous-marine prisent en conditions réelles est difficile dû aux mouvements de la camera, au contraste, ou encore à la colorimétrie, qui peut varier d'une vidéo à l'autre.

Notre objectif est donc de tester plusieurs approches afin d'explorer et de proposer des solutions robustes pour le suivi de seiche en environnement non contrôlé.

Nous avons commencé par apprendre ce qui se fait de mieux dans le domaine du suivi d'objet, afin d'avoir une idée des différentes techniques ainsi que de leur efficacité.

Nous utilisons un filtre à particule comme base de notre algorithme de suivi, il utilisera plusieurs types de descripteurs qui seront comparés entre eux afin de déterminer les plus efficace. Leurs efficacités sont mesurées en utilisant la méthode de Pascal VOC, qui consiste à comparer la bounding box de référence avec celle estimée par notre méthode.

Le suivi est amélioré grâce à une étape de prédiction qui permet de gérer les cas d'occlusions et de déformation des seiches au cours de la vidéo.

Le modèle d'intelligence artificielle YOLOv7 est utilisé pour faire la détection initiale de seiches dans les premières frames de la vidéo de manière automatique.

Ce modèle sera également utilisé en tant qu'algorithme de suivi à part entière, et comparé avec nos solutions faites à la main.

Table des matières

1	Introduction	6
1.1	Énoncé du problème	6
1.2	Motivation	7
1.3	Méthodes	7
1.4	Cahier des charges	8
1.4.1	Besoins fonctionnels	8
1.4.2	Besoins non-fonctionnels	8
1.4.3	Contraintes	8
2	Technologies	9
2.1	LabelImg et Roboflow	9
2.2	État de l'art de la détection d'objet	9
2.3	Langage de programmation	9
3	Développements Logiciel : Conception, Modélisation, Implémentation	11
3.1	Développements logiciel	11
3.1.1	Intelligence Artificielle	11
3.1.2	Logiciel de suivi de seiche	12
3.2	Modules	14
3.2.1	Descripteurs	14
3.2.2	Mesure de similarité	15
3.2.3	Filtre à particule	15
3.3	Structures de données	16
3.4	Statistiques	16
4	Algorithmes et Analyse	17
4.1	Descripteurs	17
4.1.1	Histogramme de gradient orienté	17
4.1.2	HOG en cascade	18
4.2	Mesures de similarité	19
4.2.1	Distance de Bhattacharyya	19
4.2.2	Cosine similarity	19
4.3	Filtre à particule	19
5	Analyse des résultats	21
5.1	Méthodologie	21
5.2	Analyse et comparaison	21
6	Gestion du Projet	22
6.1	Planification	22
7	Bilan et Conclusions	24
	Appendices	26

Table des figures

1.1	Différentes apparences d'une même seiche dans une vidéo.	6
3.1	Performances de notre modèle après entraînement.	11
3.2	Exemples de résultats obtenus par notre modèle.	12
3.3	Exemple d'une image renvoyée par le logiciel.	13
3.4	IoU (blanc) entre l'image de référence (vert) et notre résultat (rouge).	13
3.5	Diagramme UML de cas d'utilisation.	14
3.6	Diagramme UML des classes du module descripteur.	15
3.7	Diagramme UML des classes du module mesure de similarité.	15
3.8	Diagramme UML des classes du module filtre à particule.	16
4.1	Bloc d'un HOG.	18
4.2	Déplacement d'un bloc.	18
4.3	HOG en cascade.	19
6.1	Diagramme de Gantt du projet.	23
1	IoU (blanc) entre deux bounding box.	27
2	Diagramme UML des classes global (image SVG).	28
3	Schéma basic d'un VAE.	29

Liste des Algorithmes

1	Filtre à particule général	20
2	Filtre à particule	20

1 Introduction

1.1 Énoncé du problème

Le suivi d'objet par vision par ordinateur a toujours été, et est encore, un problème suscitant beaucoup d'intérêt et qui fait l'objet de beaucoup de sujets de recherche.

En effet, le suivi d'objet apparaît dans plein de domaine, comme par exemple :

- L'aérospatiale, avec l'arrimage de module à l'ISS, ou encore le suivi de débris spatiaux pour ensuite les faire sortir d'orbites sensibles.
- Le militaire, avec le suivi de missile pour interception précise.
- L'astrophysique, avec le suivi de corps céleste.
- L'étude de population animal, comme l'étude de cycle migratoire ou du comportement de certaines espèces.

Le projet se place dans le contexte du suivi de seiche en milieu aquatique grâce à des vidéos. Les vidéos sous-marine sont sujettes à beaucoup de difficultés, comme la variation de couleur, de lumière et de contraste d'une image à l'autre, la dégradation de la qualité de la vidéo par des sédiments, les mouvements instables du plongeur, ou encore l'arrière plan qui change.

Les solutions implémentées doivent essayer de prendre en compte toutes ces variations ainsi que la forme générale de la seiche au cours de ses mouvements (voir figure 1.1).

Le projet a pour objectif de fournir un outil qui propose des solutions qui répondent à toutes ces contraintes, pour suivre des seiches de façon non intrusive (pas de capteurs sur la seiche suivie), afin de limiter l'interaction humaine avec les seiches.

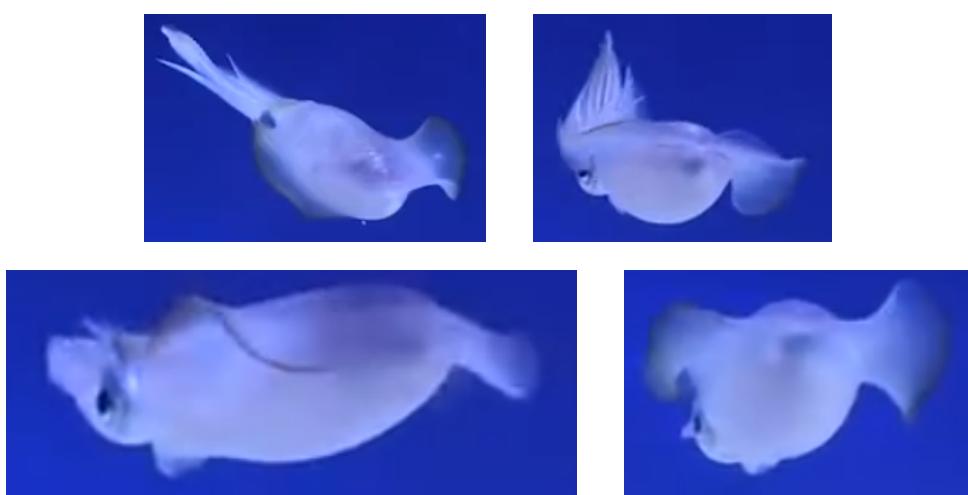


FIGURE 1.1 – Différentes apparences d'une même seiche dans une vidéo.

1.2 Motivation

Le suivi d'objet, de manière générale, étant un problème que l'on rencontre dans beaucoup de domaine lié à l'informatique, les précédents exemples n'étant qu'un petit aperçu, il est donc un problème de choix à étudier lors de notre parcours d'études.

De plus, il permet d'introduire des concepts et algorithmes fondamentaux, comme le concept de filtrage de signaux, ou d'espace de représentation, ou encore, les algorithmes de filtrage, de descripteurs, ou de mesure de similarité.

Ces concepts et algorithmes sont récurrents dans le monde de l'informatique et plus précisément quant il s'agit de faire du traitement du signal ou de la vision par ordinateur.

Ce familiariser des à présent avec ces différentes notions pourra grandement nous aider dans la suite de notre parcours.

1.3 Méthodes

Il existe beaucoup d'approches possible pour résoudre le problème de suivi d'objet, approches parmi lesquelles ont peut noter :

- *Intelligence Artificielle*

Cette approche est de plus en plus populaire, notamment avec des modèles comme YOLO[9] ou SSD[7]. Ces modèles peuvent directement donner la bounding box de l'objet suivi, sans avoir à faire de traitement sur la sortie du modèle.

Cependant, cette approche est peu résistante à l'occlusion de l'objet suivi.

- *Capteurs*

Cette approche utilise, par exemple, des IMUs ou marqueurs infrarouge, qui peuvent donner des informations sur l'accélération linéaire ou angulaire. Ces informations sont ensuite filtrées grâce à des algorithmes de filtrage, comme le filtre de Kalman (linéaire ou non), ou encore le filtre à particule.

Cependant, cette approche nécessite de poser des capteurs sur l'objet à suivre.

- *Photogrammétrie*

Cette approche utilise des descripteurs d'image pour extraire des features importantes et ensuite, matcher ces features avec d'autres images pour pouvoir estimer le déplacement de la caméra.

Cependant, cette approche est plus utilisée dans le cas où l'on cherche à savoir où est-ce que le caméraman se situe, plutôt qu'un objet qui se trouve dans une image (comme le SLAM).

- *Hybride*

Cette approche combine différentes parties des méthodes déjà présentées et est celle sur laquelle ce projet est basé.

On utilise l'intelligence artificielle pour détecter un objet d'intérêt à suivre dans une séquence d'image, la partie filtrage de l'approche avec des capteurs pour améliorer nos estimations de l'état de l'objet suivi, et enfin, la photogrammétrie pour récupérer les features intéressantes dans une image et les comparer avec les features d'une image de référence.

Cette approche est cependant assez sensible aux paramètres que nous lui donnons, ainsi qu'à certaines caractéristiques des images données en entrée, comme le contraste, la résolution ou encore la colorimétrie.

Le détail de cette approche sera donné en partie 3.

1.4 Cahier des charges

1.4.1 Besoins fonctionnels

Les besoins peuvent être séparés en 5 catégories :

1. Le besoin d'une intelligence artificielle pour effectuer la détection initiale.
2. Le besoin de descripteurs pour récupérer un vecteur qui décrit une image donnée.
3. Le besoin de mesures de similarité pour comparer un vecteur caractéristique d'une image avec un descripteur de référence.
4. Le besoin d'un filtre à particule permettant d'estimer certaines propriétés de la seiche que l'on suit.
5. Le besoin d'un programme principal permettant d'agencer chacune des parties ensemble.

Les différents descripteurs et mesures de similarité devront pouvoir être utilisés par le filtre à particule afin de mettre à jour l'état de chacune des particules. Par extension, le filtre à particule devra être modulable, afin de fonctionner avec ces différents descripteurs et mesures de similarité, ainsi que de répondre aux demandes du programme principal.

Le programme principal sera chargé de l'initialisation des différents modules ainsi que de l'affichage de données clefs (visualisation de résultats).

1.4.2 Besoins non-fonctionnels

Les formats vidéos acceptés sont libres.

La résolution des vidéos est également libre, mais une préférence sera portée pour la résolution 640x640 (résolution utilisée pour entraîner l'intelligence artificielle).

Le programme doit pouvoir tourner sur Windows, Linux et OSX.

Le programme doit pouvoir sauvegarder le résultat obtenu en une vidéo et également sauvegarder les bounding box dans un fichier texte.

1.4.3 Contraintes

Aucun budget n'a été alloué pour le projet, le travail s'effectuera sur nos machines personnelles, ou sur les machines mises à disposition par l'université.

Le projet doit être complété en 4 mois, avec une vingtaine de jours supplémentaires pour la rédaction du rapport.

2 Technologies

2.1 LabelImg et Roboflow

LabelImg est un logiciel d'annotation d'image, qui permet entre autre d'annoter une bounding box que l'on dessine sur une image, avec un label que l'on définit, et d'enregistrer le tout sous différent format, comme le Pascal VOC, ou bien le format de YOLO.

Ce logiciel a été utilisé pour annoter la vidéo de référence pour que l'on puisse comparer nos résultats avec.

Roboflow est un site en ligne, qui permet de faire de la gestion de base de donnée pour l'entraînement d'intelligence artificielle, ainsi que de l'annotation collaborative.

Ce logiciel a été utilisé pour annoter des images de seiche afin de constituer une base de donnée suffisante, pour entraîner l'intelligence artificielle YOLOv7[14] à détecter des seiches dans une image. L'annotation a été repartie entre les membres du groupe pour accélérer le processus.

Une fois l'annotation terminée, un dataset a été créé et augmenté grâce à Roboflow, en ajoutant des images déjà annotées auxquelles a été rajouté du bruit, des rotations, ou des changement de contraste et de luminosité. Augmenter ainsi le dataset permet à l'intelligence artificielle d'être plus résistante à des variations de contraste ou de rotation des seiches dans une image.

2.2 État de l'art de la détection d'objet

Dans le cadre du suivi d'objet par la méthode du filtre à particule, on constate fréquemment un phénomène de divergence. Une manière connue pour palier à ce problème consiste à un indiquer une position initiale correcte au filtre à particule. On a donc choisi de détecter l'objet suivie, la seiche, sur la première image afin de réduire cette effet de divergence.

Pour procéder nous emploierons une méthode de détection d'objet à l'aide de réseau de neurones, en effet cette manière à montrer des bons résultats dans la littérature ces dernières années.

Parmi les différents candidats se démarquent *Faster R-CNN*[10], *RetinaNet*[6], *SSD*[7], *YOLO*[9] ou en encore *R-FCN*[2]. YOLO a largement été utilisés dans des projets similaires. En effet chacun de ces réseaux de neurones proposent un degré de précision proche dans la détection mais YOLO se démarque par sa vitesse d'exécution et sa souplesse d'implémentation [12]. De plus YOLO bénéficie d'un suivi régulier depuis sa mise en ligne et à connu de nombreuse mise à jour. Pour la version nous opterons pour YOLOv7[14] qui propose de meilleures performances tout en disposant de suffisamment de documentation récente. YOLOv8 a été jugé trop récent pour être choisi.

2.3 Langage de programmation

Le langage de programmation choisi est python, un langage très populaire et qui permet de faire du prototypage rapidement. C'est également un des langages les plus utilisé par les chercheurs en intelligence artificielle, notamment avec le framework pytorch.

Notre choix a été fait en partie pour cette aspect de prototypage rapide, mais aussi, du fait de notre

2 Technologies

utilisation du modèle YOLOv7[14], qui utilise pytorch.

Python possède également une grande quantité de librairie, comme OpenCV, pour le traitement d'image, Numpy, pour les opérations optimisées sur des tenseurs, ou Scipy, pour les calculs scientifiques. Cela nous a permis de nous concentrer sur les algorithmes et de laisser l'affichage d'image et les opérations matricielles à des librairies qui ont été optimisées pour cela.

Il a aussi été choisi par ça facilité de prise en main, et parce que tous les membres du groupe ont déjà programmé avec et le maîtrise bien.

3 Développements Logiciel : Conception, Modélisation, Implémentation

3.1 Développements logiciel

Dans le cadre du projet, plusieurs éléments ont été développé, à noter, une intelligence artificielle pour détecter des seiches dans une image, une base de donnée de seiche pour entraîner l'intelligence artificielle et un algorithme de filtre à particule utilisant des descripteurs et mesures de similarité pour calculer le poids de chaque particules.

3.1.1 Intelligence Artificielle

La base de donnée d'image de seiche est composé d'image provenant de vidéos de seiche prisent par des plongeurs en mer, et par des particuliers dans des aquariums.

Chaque image a été annotée à la main par les membres du groupe en utilisant le logiciel en ligne Roboflow, la base de donnée est constitué d'un total de 5175 images.

L'intelligence artificielle utilise le code et les poids pré entraîné de YOLOv7[14], qui peuvent être trouver sur le github officiel de YOLOv7[13]. Cette intelligence artificielle a été entraîné pour 300 cycles de la base de donnée, soit un total de 25 heures sans interruptions.

Les performances obtenues sont illustrées dans la figure 3.1 et des exemples sont donnés en figure 3.2.

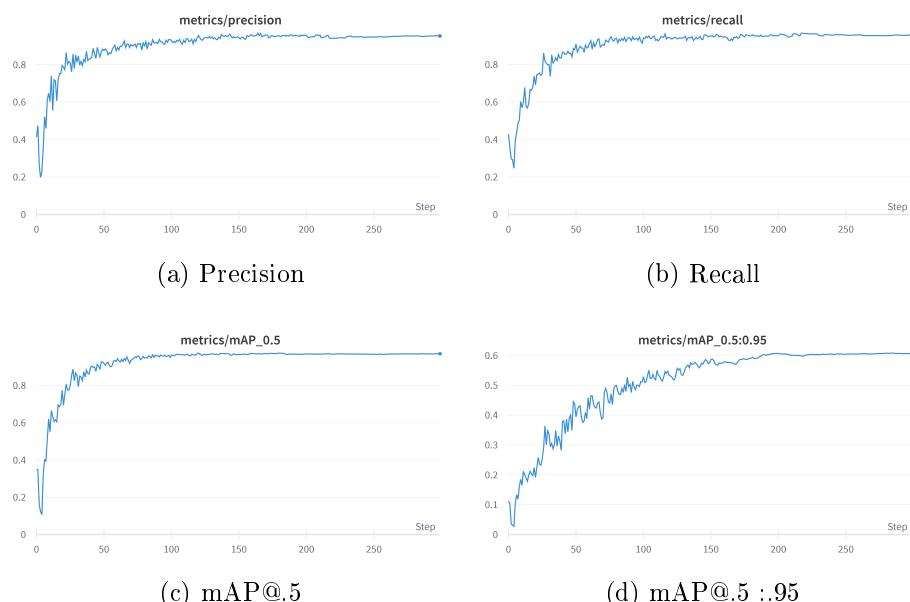


FIGURE 3.1 – Performances de notre modèle après entraînement.

Les différentes définitions peuvent être retrouvées en annexe (7).

Les résultats obtenus sont compilés dans le tableau suivant :

Précision	Recall	mAP@.5	mAP@.5 :.95
0.953	0.959	0.971	0.607



FIGURE 3.2 – Exemples de résultats obtenus par notre modèle.

3.1.2 Logiciel de suivi de seiche

Ce logiciel est composé d'un filtre à particule, de descripteurs, et de mesures de similarité, qui seront développés en partie 4.

Il prend en entrée une liste de paramètres pour configurer les différentes parties, et une vidéo. Après configuration du filtre à particule, du descripteur et de la mesure de similarité, la première image de la vidéo est donnée à notre intelligence artificielle, pour avoir une estimation de la position et bounding box d'une seiche que l'on souhaite suivre dans la vidéo.

Après le traitement de la première image par notre intelligence artificielle, chaque image de la vidéo est donnée au filtre à particule afin qu'il mette à jour toutes ses particules, et estime au mieux la position et la bounding box de la seiche suivie.

Une fois que le filtre à particule a fini de traiter une image, celle-ci est affichée avec la position

estimée en bleu et la bounding box estimée en vert, et les meilleures particules en rouge (voir figure 3.3). Le numéro de l'image traitée est affiché en haut à gauche de la fenêtre.



FIGURE 3.3 – Exemple d'une image renvoyé par le logiciel.

Le logiciel donne également la possibilité de sauvegarder la vidéo résultante, ainsi que les valeurs de la particule estimée à chaque itération du filtre.

Sauvegarder cette estimation permet, par la suite, d'effectuer une analyse des performances de nos solutions, en utilisant un script d'évaluation qui calcule l'IoU (voir annexe 7) entre une estimation à un instant t et la véritable valeur à ce même instant, comme montré dans la figure 3.4.

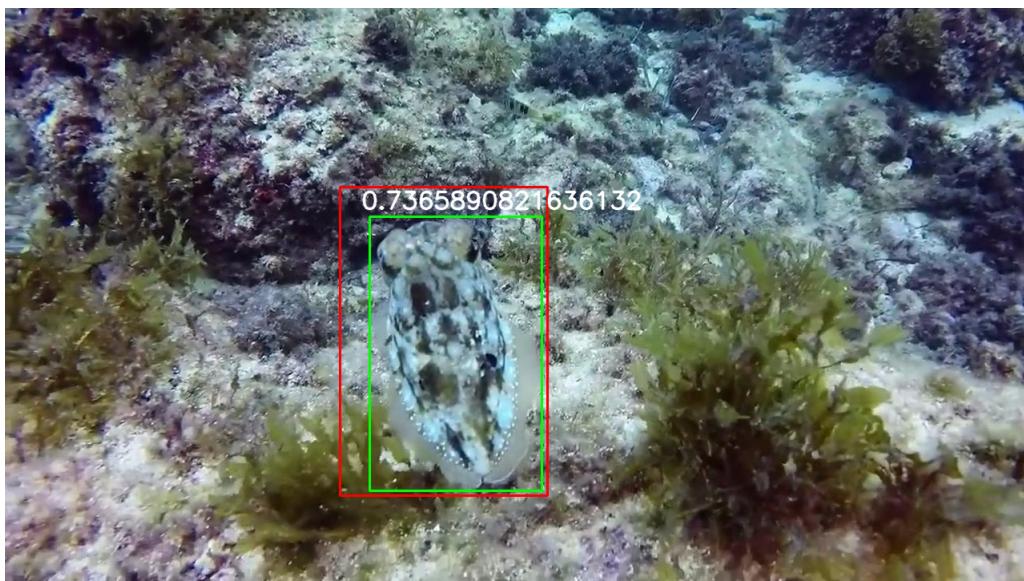


FIGURE 3.4 – IoU (blanc) entre l'image de référence (vert) et notre résultat (rouge).

3.2 Modules

Le diagramme UML de cas d'utilisation (figure 3.5) du projet est assez simple, et ce résume à une unique relation entre l'utilisateur et le logiciel, celle de lancer le programme avec une vidéo et les paramètres souhaités. Le reste des relations est effectuées en interne et aucune interaction externe n'est requise.

Le diagramme UML global des classes peut être retrouvé en annexe 7.

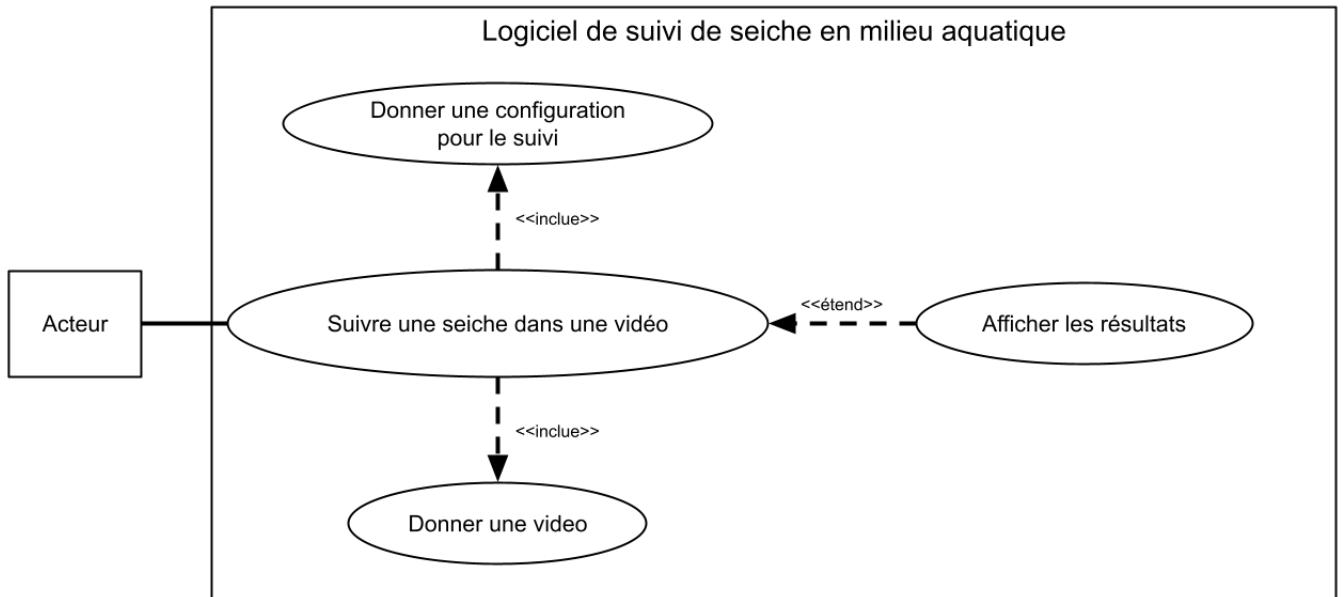


FIGURE 3.5 – Diagramme UML de cas d'utilisation.

3.2.1 Descripteurs

Le module descripteur est constitué d'une classe parent *Descriptor* qui possède 5 classes filles, *HOG*, *HOGCASCADE*, *HOGCOLOR*, *LBP* et *HOGCASCADELBP*. Ces classes sont les différents descripteurs que l'utilisateur peut utiliser dans le filtre à particule.

Toutes ces classes possèdent 2 méthodes communes, la méthode *update* qui permet de mettre à jour certains paramètres du descripteur, et la méthode *compute* qui permet d'effectuer le calcul du descripteur sur une liste d'image.

A noter, que les descripteurs *HOGCASCADE* et *HOGCOLOR* sont des variantes du descripteur *HOG* (*HOG* et *HOG* cascade seront détaillé en partie 4), et le descripteur *HOGCASCADELBP* est une combinaison entre un descripteur *HOG* cascade et *LBP*.

Ce module permet de calculer, pour chaque particule du filtre, un vecteur descripteur qui pourra, par la suite, être comparé avec un descripteur de référence pour estimé la similarité entre la particule et une particule de référence.

Le diagramme UML du module est donné en figure 3.6.

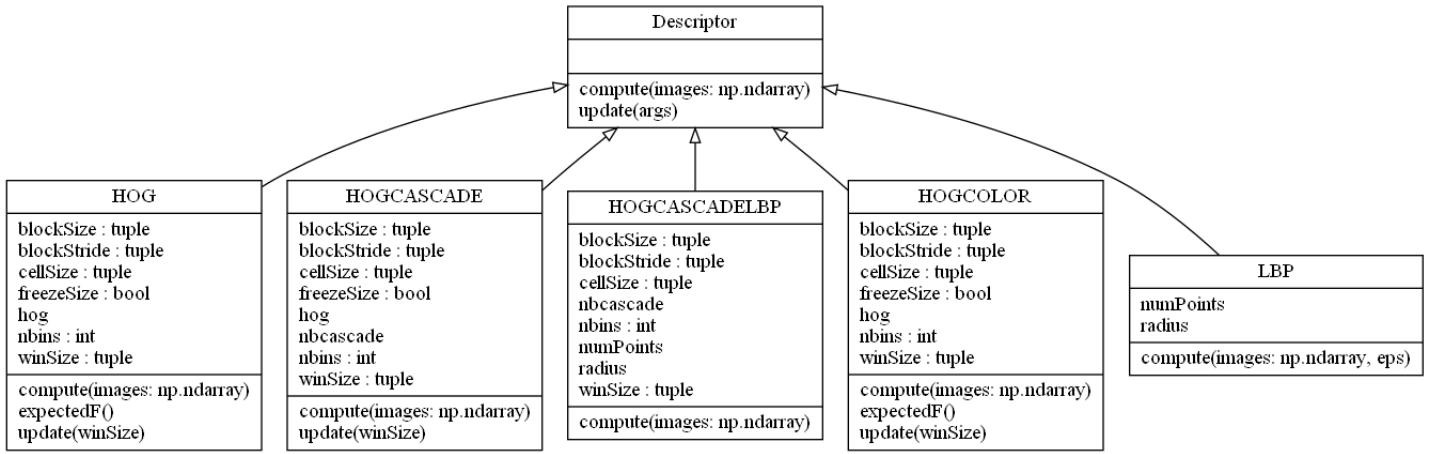


FIGURE 3.6 – Diagramme UML des classes du module descripteur.

3.2.2 Mesure de similarité

Le module mesure de similarité est constitué d'une classe parent `Similarity` qui possède 4 classes filles, `Bhattacharyya_sqrt`, `Bhattacharyya_log`, `Cosine_Similarity` et `Kullback_Leibler_Divergence`. Ces classes sont les différentes mesures de similarité que l'utilisateur peut utiliser dans le filtre à particule.

Toutes ces classes possèdent une méthode commune, la méthode `computeSimilarity` qui permet de calculer la similarité entre une liste de vecteurs descripteur et un vecteur descripteur de référence. Les mesures de similarité `Bhattacharyya_sqrt` et `Cosine_Similarity` seront détaillé en partie 4.

Ce module permet de calculer, pour chaque descripteur de chaque particule du filtre, un coefficient de similarité qui indique quelles particules sont les plus probables de représenter la position de la seiche à un instant t .

Le diagramme UML du module est donné en figure 3.7.

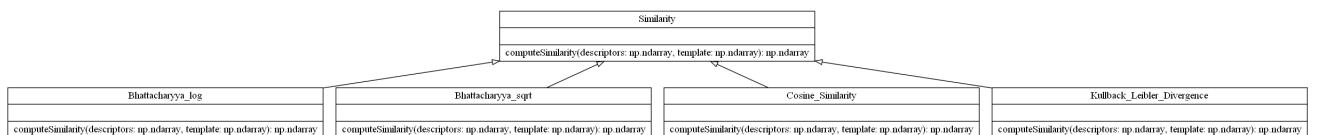


FIGURE 3.7 – Diagramme UML des classes du module mesure de similarité.

3.2.3 Filtre à particule

Le module filtre à particule est le plus conséquent, car il regroupe plusieurs autres modules, comme les modules descripteur et mesure de similarité. La classe `ParticleFilter` est la classe principale de ce module et implémente l'algorithme du filtre à particule. Elle fait appel aux classes `Particle`, `Descriptor`, `Slicer`, `Resampling` et `Similarity`, et les utilisent ensemble pour effectuer le suivi d'une seiche.

Ce module est responsable de fournir tous les résultats nécessaires au programme principale, pour qu'il puisse les afficher et/ou les sauvegarder pour être finalement évalués. Ce module est très flexible et permet d'être configuré avec différente structure de particule, différent descripteur, différente mesure de similarité, ou encore différente méthode de ré-échantillonnage.

Le diagramme UML du module est donné en figure 3.8.

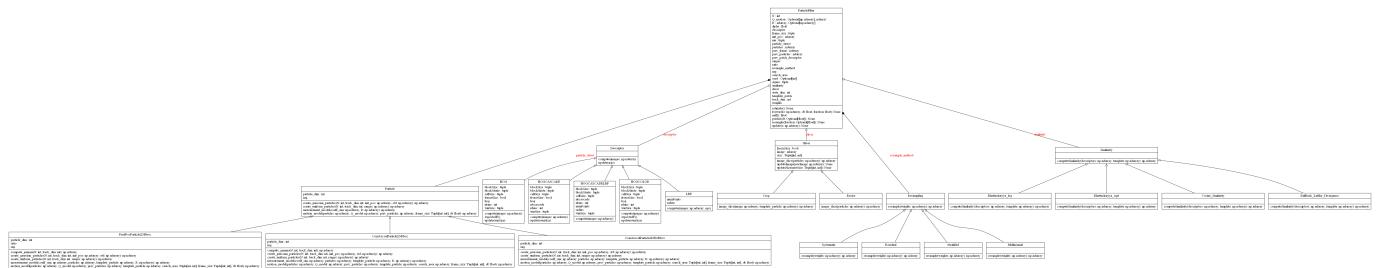


FIGURE 3.8 – Diagramme UML des classes du module filtre à particule.

3.3 Structures de données

Les structures de données principales du projet sont les tenseurs en tant que tableau multidimensionnel, pour ce faire, nous utilisons la librairie Numpy, qui permet de réaliser des opérations sur ces tableaux de façon optimisé.

Nous essayons de garder les données un maximum sous ce format, pour éviter les conversions et opérations qui pourraient ralentir le logiciel de suivi. C'est pour cela que la majorité des entrées des programmes réalisés demande des 'ndarray', qui est le type des tableaux Numpy.

Les images prisent en entrée des programmes sont transformées en tableau Numpy, selon la convention de OpenCV, c'est-à-dire avec les couleurs au format BGR et la hauteur de l'image comme première dimension du tenseur, et la largeur de l'image comme seconde dimension.

L'utilisateur du logiciel n'intervient qu'à un seul niveau dans le programme, le reste des entrées du logiciel est géré en interne afin de préserver au mieux l'intégrité des données traitées.

L'utilisateur peut uniquement donner des paramètres au logiciel lorsqu'il le lance, ces paramètres sont alors parser en différent arguments qui sont vérifiés par le programme, puis utilisés pour initialiser les différents module, afin de commencer le suivi d'une seiche.

3.4 Statistiques

Le projet compte un total de 25 classes réparties dans 10 scripts python (voir figure 2). Les scripts et classes de YOLOv7[14] utilisés dans le projet ne sont pas comptés.

Le projet compte en tout 1587 lignes de code.
L'entièreté du projet est en accès libre sur [github](#) ([4]).

4 Algorithmes et Analyse

4.1 Descripteurs

Pour pouvoir appliquer notre mesure de similarité entre deux images et calculer les poids de nos particules, on a besoin de pouvoir extraire de l'information de nos images. Ainsi nous allons utiliser des descripteurs.

Un descripteur est un morceau d'information extrait d'une image sous forme de vecteur. Il permettra de reconnaître un motif ou une structure spécifique au cours de notre vidéo. Pour calculer un descripteur on s'appuie sur l'information bas niveau de nos images (valeur des pixels, contours, gradients, ...).

4.1.1 Histogramme de gradient orienté

Nous utiliserons en particulier les histogrammes de gradients orientés (HOG) qui sont des descripteurs proposés par Navneet DALAL et Bill TRIGGS [3]. Ils correspondent à la distribution de l'orientation des contours locaux d'une image.

Les HOG sont calculés par DALAL et TRIGGS ainsi :

Une première étape de pré-traitement de l'image afin de normaliser les couleurs et d'appliquer une correction gamma à celle-ci puis la convertir en niveau de gris. Ici la normalisation du HOG est suffisante et cette étape est donc facultative, on s'est contenté de la conversion en niveau de gris.

L'étape suivante est le calcul de la carte des gradients en x et en y qui correspondent respectivement à la variation horizontal et vertical des gradients. On effectue une convolution centrée sur le pixel cible. Parmi les différents filtres possibles, les masques que nous utilisons sont $[-1, 0, 1]$ pour les gradients en x et $[-1, 0, 1]^T$ pour les gradients en y, aussi appelés filtres de Sobel. Ces masques se sont révélés plus performants dans la littérature [3].

À partir de ces deux cartes de gradients on pourrait calculer la carte des modules des gradients. Néanmoins, afin d'augmenter l'efficacité du descripteur, le HOG n'est pas directement calculé à partir de l'image.

En effet l'image est divisée en "cellules", elles-mêmes regroupées en "blocs". On choisit des cellules de 6×6 pixels et des blocs de 3×3 cellules.

Pour chaque blocs, on calcule le HOG de chaque cellule. Le HOG du bloc correspond à la concaténation du HOG des cellules le composant. Pour une meilleure invariance face à l'illumination et à l'ombrage on effectue une normalisation du HOG du bloc (norme euclidienne). Pour un vecteur \mathbf{x} , on a :

$$\sum_{k=0}^n (x_k)^2$$

Donc le HOG de l'image est la concaténation de l'ensemble des HOG de chacun de ses blocs.

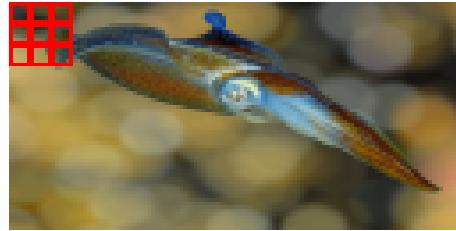


FIGURE 4.1 – Bloc d'un HOG.

Les histogrammes sont construits grâce au mesures des angles des gradients et à leurs intensités. De la même manière qu'avec la taille des blocs et des cellules, on doit choisir la structure de notre histogramme. Étant donnés que nous travaillons sur des angles, nous choisirons un histogramme à neuf classes et nous utiliserons des angles non signés (compris entre $[0; 180]$). Ainsi pour un pixel $p(x, y)$ l'angle est définis par :

$$\text{angle}_p = |\arctan(\vec{\nabla}y, \vec{\nabla}x) * \frac{180}{\pi}|$$

De même que son intensité par :

$$\text{intensite}_p = \sqrt{\vec{\nabla}x^2 + \vec{\nabla}y^2}$$

Pour chaque pixel, on répartit son intensité proportionnellement dans les deux classes les plus proche de l'angle qui lui est associé. Par exemple, prenons $\vec{\nabla}x = 10$ et $\vec{\nabla}y = 10$ on obtient un intensité d'environ 14.14 et un angle de 45. De cette manière la classe centrée en 40 recevra les trois quarts $(1 - |\frac{5}{20}|)$ et la classe centrée en 60 recevra le quart restant $(1 - |\frac{15}{20}|)$ de l'intensité.

Enfin pour affiner notre descripteur nous effectuerons le déplacement du bloc suivant :

$$\frac{T_{cellule} * T_{bloc}}{2}$$

$T_{cellule}$ est la taille en pixel de la cellule, T_{bloc} est la taille en nombre de cellule d'un côté du bloc. Ainsi on a : $\frac{6*3}{2} = 9$, de plus, un bloc faisant plus de 9 pixels de côté, on a un chevauchement des blocs et donc des pixels pris en compte plusieurs fois. Cela permet au vecteur de mieux caractériser l'image cible.

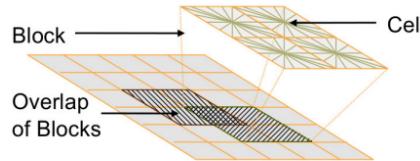


FIGURE 4.2 – Déplacement d'un bloc.

4.1.2 HOG en cascade

Pour enrichir notre vecteur caractéristique on peut utiliser HOG en cascade qui combine les HOG de l'image à différentes résolutions. Pour procéder on calcule les HOG sur une successions de sous-régions inclusives et les ajoutons au HOG global, comme décrit dans l'article [8]. De cette manière on ajoute de l'information spatiale à notre descripteur.

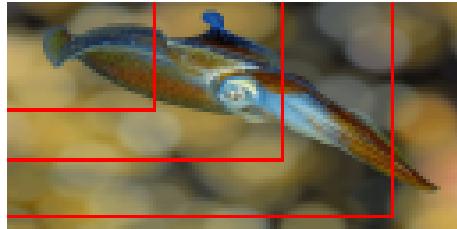


FIGURE 4.3 – HOG en cascade.

4.2 Mesures de similarité

4.2.1 Distance de Bhattacharyya

4.2.2 Cosine similarity

4.3 Filtre à particule

Le filtre à particule, aussi connu sous le nom de bootstrap, ou algorithme de condensation, est basé sur une méthode de Monte Carlo, c'est à dire utiliser un ensemble de particule discrète pour représenter la densité de probabilité associé au système que l'on souhaite modéliser. L'idée principale du filtre à particule est d'approcher une distribution, impossible ou difficile à estimer directement, grâce à un ensemble d'échantillons pondérés $S = \{(X_n, w_n) | n = 1 \dots N\}$, où X_n indique la nième particule, w_n indique l'importance de la particule, ou le poids de celle-ci, et N est le nombre total de particule. Plus d'information peuvent être trouvé aux références suivantes [11] et [1].

Cet algorithme a, par exemple, pu être utilisé avec le descripteur HOG, comme dans ces articles [15], [5], [8] et [3], mais il peut également être utilisé avec d'autre descripteur, ou combinaison de descripteur, comme HOG cascade combiné avec LBP (Local Binary Pattern).

Les particules que le filtre utilise peuvent être définies de plusieurs façon, nous avons choisi de définir une particule, comme un point en 2D, la vitesse et l'accélération de ce point, et la demi largeur et demi hauteur de la bounding box délimitant notre cible, mais il est possible de changer cette définition. A un instant t , une particule est donc représentée par un vecteur à 8 dimensions :

$$X^t = [x^t, \dot{x}^t, \ddot{x}^t, y^t, \dot{y}^t, \ddot{y}^t, l^t, h^t]^T$$

Où $(x^t, y^t)^T$, $(\dot{x}^t, \dot{y}^t)^T$ et $(\ddot{x}^t, \ddot{y}^t)^T$ sont les coordonnées, vitesses et accélération du centre de notre cible en pixel, respectivement, (l^t, h^t) décrit la bounding box de notre cible en pixel avec la demi largeur et la demi hauteur.

Bien qu'il existe plusieurs variantes de cet algorithme, le principe général reste le même (voir figure 1). Il s'agit d'utiliser les particules pour déterminer l'état de notre système à un instant t , en faisant une prédiction de nos particules dans le temps, et en combinant cette prédiction avec une observation z (résultats de capteurs, ou images d'une vidéo, par exemple) afin de mettre à jour les poids de chacune des particules. Une fois les poids mis à jour, on vérifie combien de particule participent réellement à l'estimation de l'état de notre système, et si ce nombre est en dessous d'un certain seuil, on effectue une étape de ré-échantillonnage en gardant les particules de poids fort, et en supprimant celles de poids faible. Finalement, on peut estimer l'état de notre système, en prenant la moyenne pondérée de nos particules (MLE, Maximum Likelihood Estimation) ou en prenant comme estimation la particule de poids le plus fort (MAP, Maximum A Posteriori).

Algorithme 1 : Filtre à particule général

Données : Une observation z
Résultat : Une estimation de l'état de notre système
1 pour chaque *Particules p faire*
2 fin

Algorithme 2 : Filtre à particule

Données : Une vidéo sous-marine de seiche
Résultat : La liste des positions et bounding box de la cible dans la vidéo
1 pour chaque *Images t de la vidéo faire*
2 si Première image **alors**
3 Détection de la seiche dans l'image grâce a YOLOv7
4 Calcul du descripteur de référence : F_{ref}^0
5 sinon
6 pour chaque *Particules p faire*
7 Prédiction de la position et de la bounding box de p dans t en fonction de ses états précédents.
8 Calcul du descripteur du patch correspondant à p : F_p^t
9 Calcul du poids de p dans t :

$$w_p^t = D_B(F_p^t, F_{ref}^{t-1})$$

10 fin
11 Ré-échantillonnage des particules pondérées par leurs poids
12 Estimation de la position et bounding box de la seiche dans t :

$$X^t = \frac{1}{NB_{particule}} * \sum_{p=0}^{NB_{particule}} x_p^t * w_p^t$$

13 fin
14 Mise à jour du descripteur de référence : F_{ref}^t
15 fin

5 Analyse des résultats

5.1 Méthodologie

5.2 Analyse et comparaison

6 Gestion du Projet

6.1 Planification

Le projet a été divisé entre les 4 membres du groupe pour la partie implémentation des solutions et écriture du rapport, le reste, à savoir, la planification du projet, la répartition des tâches et les proposition de solutions à implémenter ont été décidé par tous les membres du groupe.

La planification c'est faite grâce à un diagramme de Gantt et a été répartie sur les 5 mois accordés, soit du 5 Janvier 2023 au 12 Mai 2023, comme montré sur la figure 6.1.

Aucun changement majeur n'a été effectué au cours du projet et toutes les parties respectent l'architecture qui a été définie au début du projet.

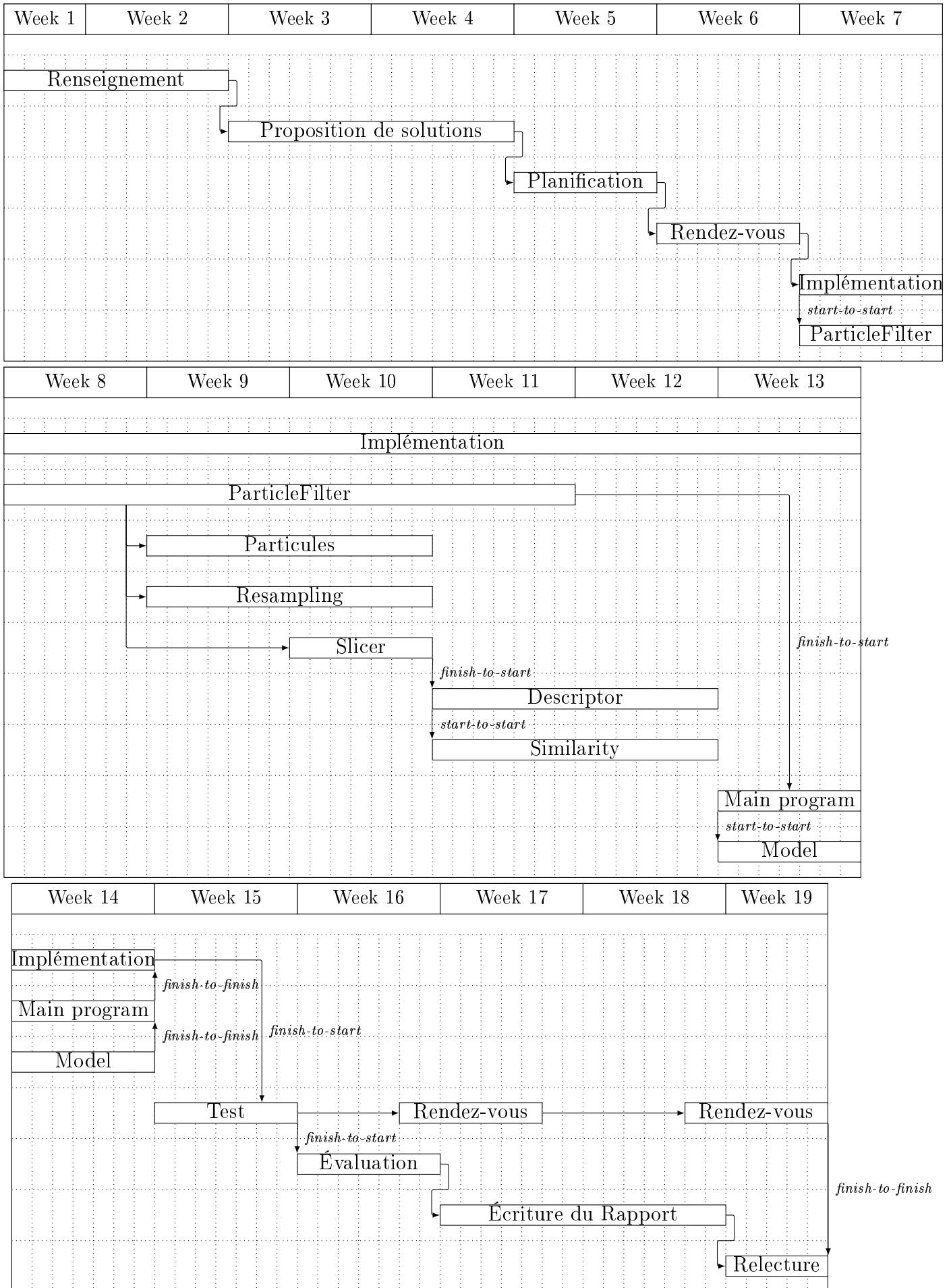


FIGURE 6.1 – Diagramme de Gantt du projet.

7 Bilan et Conclusions

Au terme de ce projet, nous disposons d'un logiciel de suivi de seiche en milieu aquatique non contrôlé, qui peut utiliser différente combinaison de descripteur et mesure de similarité.

Les résultats montrent que sur des vidéos prisent en conditions réelles, et avec une configuration du logiciel adéquate, le suivi ce fait avec une efficacité d'environ 75% par rapport aux données de références. Ce résultat est très satisfaisant et n'a qu'une différence de 1% avec la méthode qui utilise uniquement YOLOv7.

Cependant, notre méthode reste très dépendante de la configuration donnée par l'utilisateur, ainsi qu'aux caractéristiques des vidéos, comme le mouvement de la caméra, le contraste, la luminosité, ou encore l'arrière plan qui peut prendre une trop grande proportion dans la bounding box et faire diverger le suivi. Mais, avec une configuration adéquate, nous arrivons à obtenir un suivi satisfaisant pour les cas problématiques définis dans l'introduction, comme par exemple les mouvements de la caméra ou la déformation de la seiche au cours de la vidéo.

Les perspectives d'amélioration tournent principalement autour des réseaux de neurones et du temps réel.

En effet, le suivi actuel ce fait en post-processing et peut demander un certain temps en fonction de la configuration donnée au logiciel. On pourrait donc essayer d'accélérer et d'améliorer le suivi en combinant l'algorithme du filtre à particule avec des réseaux de neurones, ou bien utiliser exclusivement des réseaux de neurones.

On pourrait par exemple entraîner un VAE (7) sur des images de seiches, et récupérer la partie encodeur du VAE pour l'utiliser comme un descripteur à la place de HOG. En faisant cela, on s'assure que le vecteur descripteur sortant de l'encodeur représente bien l'image que l'on a donnée en entrée.

On pourrait également utiliser uniquement des réseaux de neurones, comme YOLOv7 qui a prouvé son efficacité dans la partie 5 et qui traite chaque image en environ 2ms, ce qui est amplement suffisant pour faire du temps réel.

Bibliographie

- [1] Roger Labbe (RLABBE). *Kalman and Bayesian Filters in Python*. 2014. URL : <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>.
- [2] Jifeng DAI et al. *R-FCN : Object Detection via Region-based Fully Convolutional Networks*. arXiv :1605.06409 [cs]. Juin 2016.
- [3] N. DALAL et B. TRIGGS. « Histograms of Oriented Gradients for Human Detection ». In : *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. T. 1. San Diego, CA, USA : IEEE, 2005, p. 886-893.
- [4] Combe-Ounkham GABRIEL et al. *PP2 Particle Filter*. 2023. URL : https://github.com/gabriel-combe/PP2_Particle_Filter.
- [5] Xiaofang KONG et al. « Particle filter-based vehicle tracking via HOG features after image stabilisation in intelligent drive system ». In : *IET Intelligent Transport Systems* 13.6 (juin 2019), p. 942-949.
- [6] Tsung-Yi LIN et al. *Focal Loss for Dense Object Detection*. arXiv :1708.02002 [cs]. Fév. 2018.
- [7] Wei LIU et al. « SSD : Single Shot MultiBox Detector ». In : t. 9905. 2016, p. 21-37.
- [8] QIANG ZHU et al. « Fast Human Detection Using a Cascade of Histograms of Oriented Gradients ». In : *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*. T. 2. New York, NY, USA : IEEE, 2006, p. 1491-1498.
- [9] Joseph REDMON et al. *You Only Look Once : Unified, Real-Time Object Detection*. Mai 2016.
- [10] Shaoqing REN et al. *Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks*. arXiv :1506.01497 [cs]. Jan. 2016.
- [11] Stuart RUSSEL et Peter NORVIG. *Artificial Intelligence : A Modern Approach*. Pearson, 2021. Chap. 14, p. 509-517. ISBN : 1-292-40113-3.
- [12] S A SANCHEZ, H J ROMERO et A D MORALES. « A review : Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework ». In : *IOP Conference Series : Materials Science and Engineering* 844 (juin 2020), p. 012024.
- [13] Chien-Yao WANG, Alexey BOCHKOVSKIY et Hong-Yuan Mark LIAO. *YOLOv7*. 2022. URL : <https://github.com/WongKinYiu/yolov7>.
- [14] Chien-Yao WANG, Alexey BOCHKOVSKIY et Hong-Yuan Mark LIAO. « YOLOv7 : Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors ». In : () .
- [15] Fen XU et Ming GAO. « Human detection and tracking based on HOG and particle filter ». In : *2010 3rd International Congress on Image and Signal Processing*. Yantai, China : IEEE, oct. 2010, p. 1503-1507.

Appendices

mAP (mean Average Precision)

IoU (Intersection over Union)

L'IoU signifie Intersection over Union et est un calcul utilisé dans la technique de comparaison Pascal VOC. L'IoU est exprimé ainsi :

$$\text{IoU} = \frac{\text{Aire de l'intersection}}{\text{Aire de l'union}}$$

L'IoU, dans notre cas, ce calcul sur deux bounding box, comme dans la figure 1.

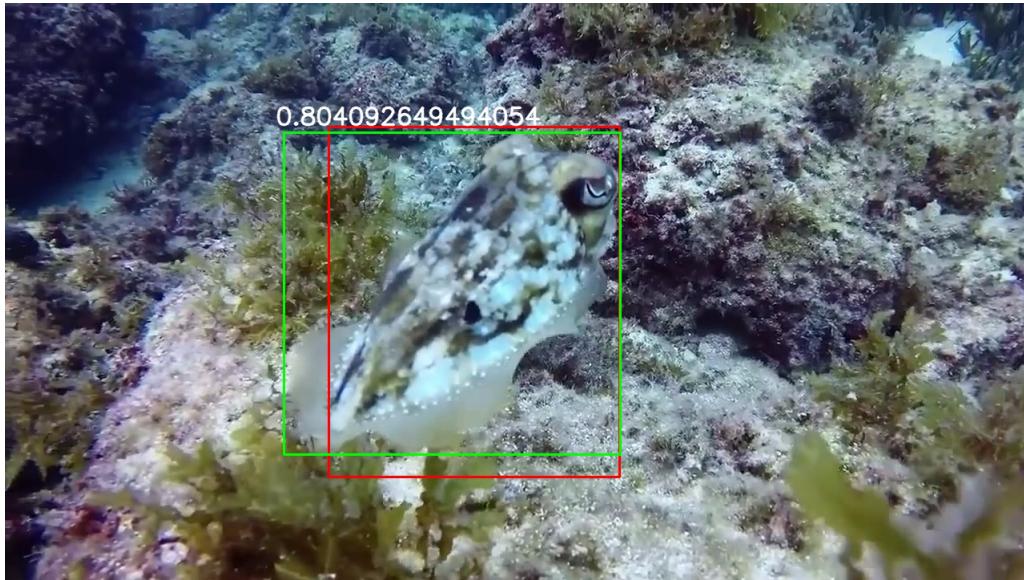


FIGURE 1 – IoU (blanc) entre deux bounding box.

Diagramme UML global

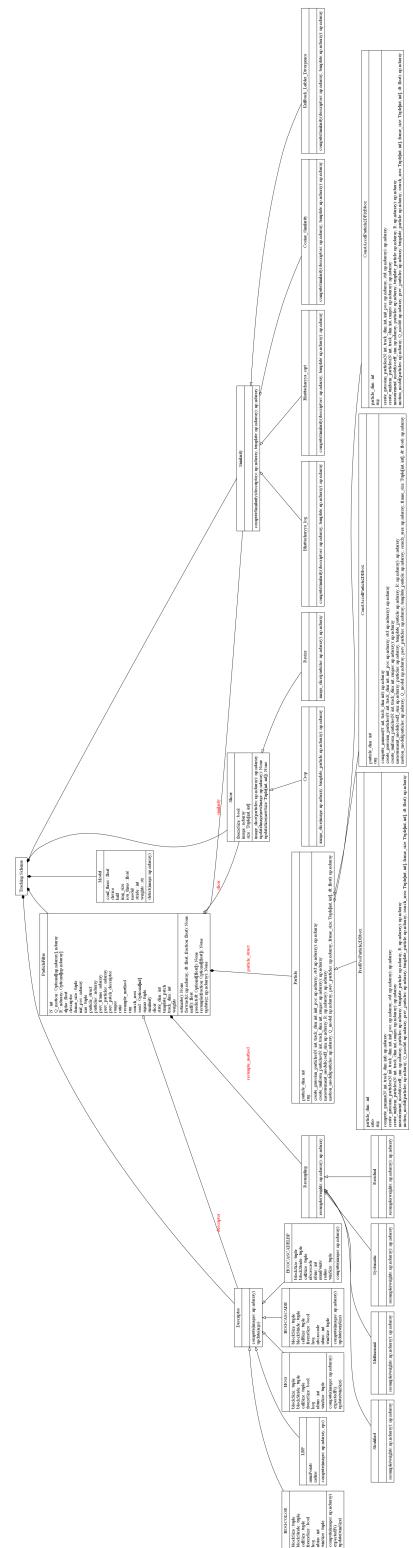


FIGURE 2 – Diagramme UML des classes global (image SVG).

VAE (Variational Autoencoder)

Les VAE, ou Variational Autoencoder, sont des réseaux de neurones qui essayent d'estimer une distribution probabiliste avec seulement un nombre limité d'échantillon provenant de cette distribution.

Ce genre de réseau de neurones est divisé en deux parties :

— *Encodeur*

Pour une donnée x , l'encodeur va projeter/compresser x dans un espace de représentation plus petit (espace latent).

L'encodeur a pour objectif de représenter x de façon optimisée, et de conserver les informations les plus importantes.

— *Décodeur*

Pour une représentation issue de l'espace latent, le décodeur va essayer de décompresser/reconstruire x en minimisant les pertes d'information.

Le décodeur a pour objectif de produire une donnée x' la plus similaire possible à x .

Le schéma global est illustré en figure 3.

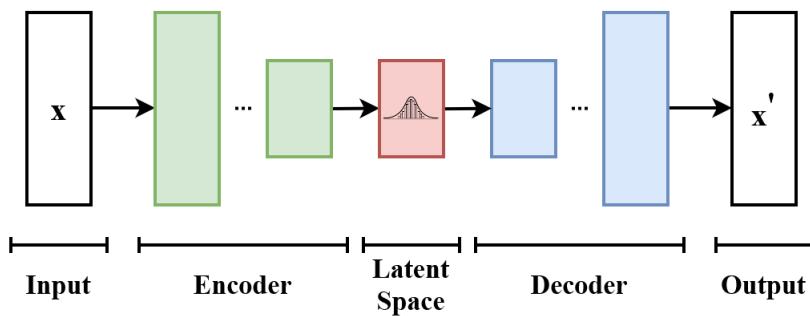


FIGURE 3 – Schéma basic d'un VAE.