

Université de Montpellier  
Faculté des Sciences  
*L3 Informatique.*

# Suivi de seiches dans des vidéos sous-marines

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-marines prises en conditions réelles est difficile. Ceci est dû aux mouvements de la caméra, 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 efficaces. Leur efficacité est mesurée 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

<b>1</b>	<b>Introduction</b>	<b>6</b>
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
<b>2</b>	<b>Technologies</b>	<b>9</b>
2.1	LabelImg et Roboflow . . . . .	9
2.2	État de l'art de la détection d'objet . . . . .	9
2.3	Langage de programmation . . . . .	10
<b>3</b>	<b>Développements Logiciel : Conception, Modélisation, Implémentation</b>	<b>11</b>
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
<b>4</b>	<b>Algorithmes et Analyse</b>	<b>17</b>
4.1	Descripteurs . . . . .	17
4.1.1	Histogramme de gradient orienté . . . . .	17
4.1.2	HOG en cascade . . . . .	19
4.2	Mesures de similarité . . . . .	19
4.2.1	Distance de Bhattacharyya . . . . .	19
4.2.2	Cosine similarity . . . . .	19
4.3	Filtre à particule . . . . .	20
<b>5</b>	<b>Analyse des résultats</b>	<b>24</b>
5.1	Méthodologie . . . . .	24
5.2	Analyse et comparaison . . . . .	24
<b>6</b>	<b>Gestion du Projet</b>	<b>25</b>
<b>7</b>	<b>Bilan et Conclusions</b>	<b>27</b>
	<b>Appendices</b>	<b>29</b>

# 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.	26
1	IoU (blanc) entre deux bounding box.	30
2	Diagramme UML des classes globales (image SVG).	31
3	Schéma basique d'un VAE.	32

# Liste des Algorithmes

1	Filtre à particule général . . . . .	21
2	Filtre à particule . . . . .	23

# 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 domaines, comme par exemple :

- L'aérospatial, avec l'arrimage de modules à l'ISS, ou encore le suivi de débris spatiaux pour ensuite les faire sortir d'orbites sensibles.
- Le militaire, avec le suivi de missiles pour interception précise.
- L'astrophysique, avec le suivi de corps célestes.
- L'étude de populations animales, comme l'étude de cycles migratoires ou du comportement de certaines espèces.

Le projet se place dans le contexte du suivi de seiches en milieu aquatique grâce à des vidéos. Les vidéos sous-marines 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.

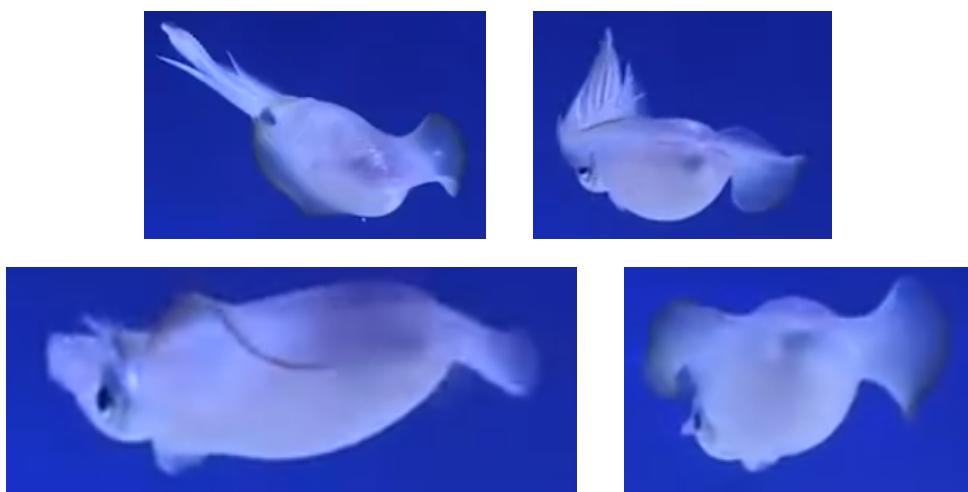


FIGURE 1.1 – Différentes apparences d'une même seiche dans une vidéo.

## 1.2 Motivation

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

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

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

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

## 1.3 Méthodes

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

### — *Intelligence Artificielle*

Cette approche est de plus en plus populaire, notamment avec des modèles comme YOLO[10] ou SSD[8]. 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 infrarouges, 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'images, 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 se charge 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 différents formats, comme le Pascal VOC, ou bien le format de YOLO.

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

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

Ce logiciel a été utilisé pour annoter des images de seiches afin de constituer une base de données suffisante, pour entraîner l'intelligence artificielle YOLOv7[15] 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 changements 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 suivi, la seiche, sur la première image afin de réduire cet effet de divergence.

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

Parmi les différents candidats se démarquent *Faster R-CNN*[11], *RetinaNet*[7], *SSD*[8], *YOLO*[10] ou en encore *R-FCN*[3] (voir tableau 2.1). YOLO a largement été utilisé dans des projets similaires. En effet chacun de ces réseaux de neurones propose un degré de précision proche dans la détection mais YOLO se démarque par sa vitesse d'exécution et sa souplesse d'implémentation (voir article [13]). De plus, YOLO bénéficie d'un suivi régulier depuis sa mise en ligne et a connu de nombreuses mises à jour. Pour la version nous opterons pour YOLOv7[15] qui propose de meilleures performances tout en disposant de suffisamment de documentation récente. YOLOv8 a été jugé trop récent pour être choisi.

Modèles	mAP	Papier
RetinaNet	52.1	SpineNet : Learning Scale-Permuted Backbone for Recognition and Localization
YOLOv7	<b>51.4</b>	YOLOv7 : Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors
Faster R-CNN	43.9	LIP : Local Importance-based Pooling
DeformConv-R-FCN	37.5	Deformable Convolutional Networks
SSD512	28.8	SSD : Single Shot MultiBox Detector

TABLE 2.1 – Comparaison de la précision de plusieurs modèles (mAP \*).

## 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 utilisés par les chercheurs en intelligence artificielle, notamment avec le framework pytorch.

Notre choix a été fait en partie pour cet aspect de prototypage rapide, mais aussi, du fait de notre utilisation du modèle YOLOv7[15], qui utilise pytorch.

Python possède également une grande quantité de librairies, 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 sa facilité de prise en main, et parce que tous les membres du groupe ont déjà programmé avec et le maîtrisent bien.

# 3 Développements Logiciel : Conception, Modélisation, Implémentation

## 3.1 Développements logiciel

Dans le cadre du projet, plusieurs éléments ont été développés, à noter, une intelligence artificielle pour détecter des seiches dans une image, une base de données de seiches 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 particule.

### 3.1.1 Intelligence Artificielle

La base de données d'images de seiches est composée d'images provenant de vidéos de seiches prises 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 données est constituée d'un total de 5175 images.

L'intelligence artificielle utilise le code et les poids pré-entraînés de YOLOv7[15], qui peuvent être trouvé sur le github officiel de YOLOv7[14]. Cette intelligence artificielle a été entraînée pour 300 cycles de la base de données, 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.

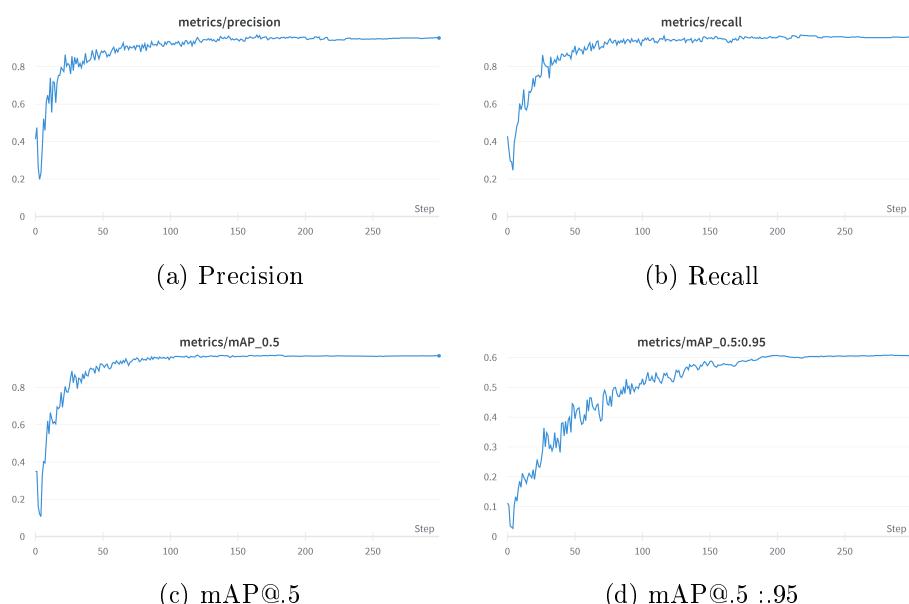


FIGURE 3.1 – Performances de notre modèle après entraînement.

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 renvoyée 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.

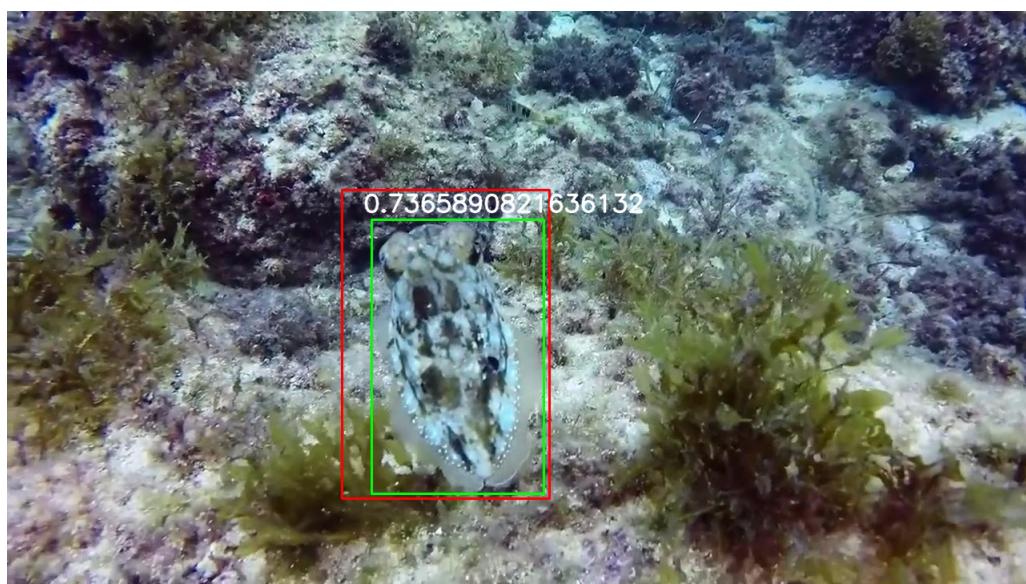


FIGURE 3.4 – IoU (blanc) entre l'image de référence (vert) et notre résultat (rouge).

## 3.2 Modules

Le diagramme UML de cas d'utilisation (figure 3.5) du projet est assez simple, et se 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 effectué en interne et aucune interaction externe n'est requise.

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

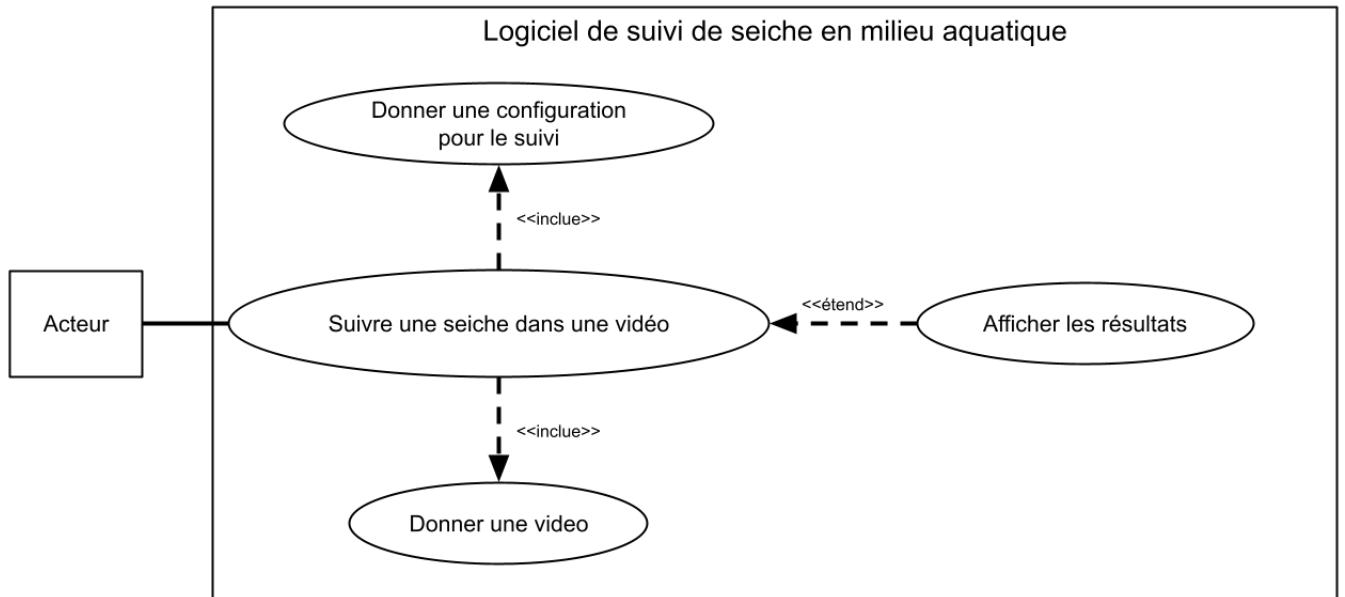


FIGURE 3.5 – Diagramme UML de cas d'utilisation.

### 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 deux 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'images.

A noter, que les descripteurs *HOGCASCADE* et *HOGCOLOR* sont des variantes du descripteur *HOG* (*HOG* et *HOG* cascade seront détaillés 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 estimer la similarité entre la particule et une particule de référence.

Le diagramme UML du module est donné en figure 3.6.

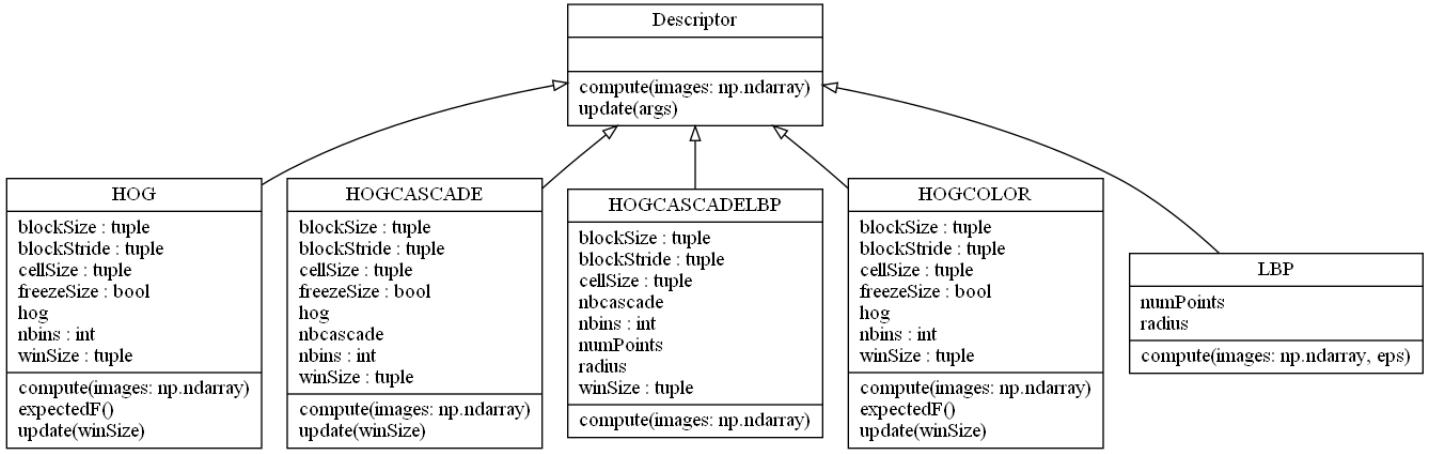


FIGURE 3.6 – Diagramme UML des classes du module descripteur.

### 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 détaillées 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.

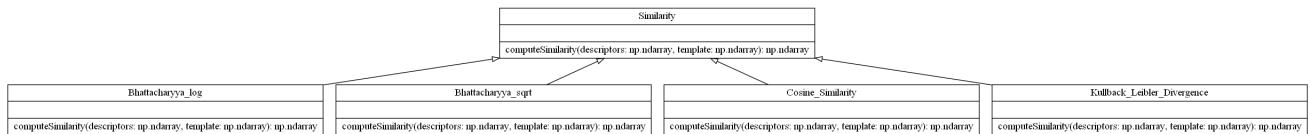


FIGURE 3.7 – Diagramme UML des classes du module mesure de similarité.

### 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 utilise ensemble pour effectuer le suivi d'une seiche.

Ce module est responsable de fournir tous les résultats nécessaires au programme principal, 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 différentes structures de particules, différents descripteurs, différentes mesures de similarité, ou encore différentes méthodes de ré-échantillonnage.

Le diagramme UML du module est donné en figure 3.8.

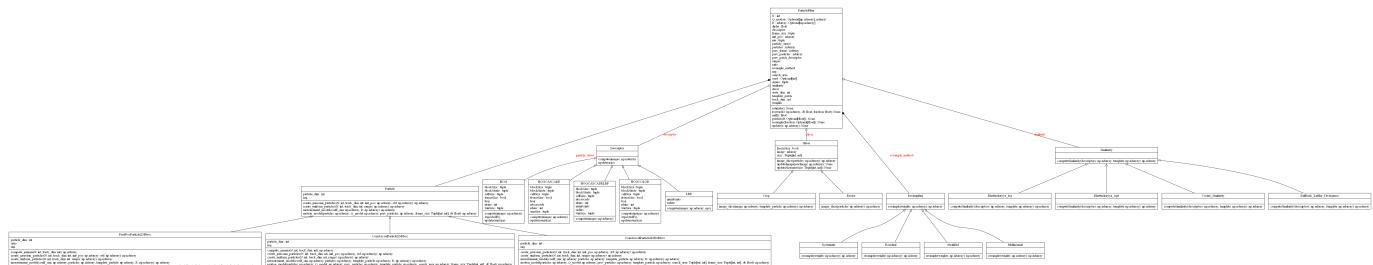


FIGURE 3.8 – Diagramme UML des classes du module filtre à particule.

### 3.3 Structures de données

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

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 prises 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 parsés en différent arguments qui sont vérifiés par le programme, puis utilisés pour initialiser les différents modules, 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 en annexe). Les scripts et classes de YOLOv7[15] 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](#) ([5]).

# 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 [4]. 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 consiste au 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 horizontale et verticale 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 [4].

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  $6 \times 6$  pixels et des blocs de  $3 \times 3$  cellules.

Pour chaque bloc, 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  $x$ , 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 aux 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 donné 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 défini par :

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

De même que son intensité par :

$$intensite_p = \sqrt{\vec{\nabla}x^2 + \vec{\nabla}y^2}$$

Pour chaque pixel, on répartit son intensité proportionnellement dans les deux classes les plus proches de l'angle qui lui est associé. Par exemple, prenons  $\vec{\nabla}x = 10$  et  $\vec{\nabla}y = 10$  on obtient une 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 pixels de la cellule,  $T_{bloc}$  est la taille en nombre de cellules 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.

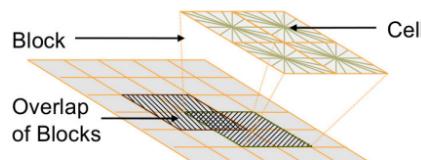


FIGURE 4.2 – Déplacement d'un bloc.

### 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 succession de sous-régions inclusives et les ajoutons au HOG global, comme décrit dans l'article [9]. De cette manière, on ajoute de l'information spatiale à notre descripteur.

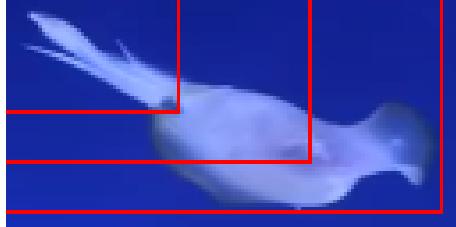


FIGURE 4.3 – HOG en cascade.

## 4.2 Mesures de similarité

Après avoir obtenu les vecteurs descripteurs de chaque particule, nous les comparons avec le vecteur descripteur de référence.

Pour comparer deux vecteurs descripteur, nous utilisons des mesures de similarité, qui nous indiqueront si les deux vecteurs sont plus ou moins similaires. La mesure de similarité donne 0 si les vecteurs sont exactement similaires, et 1 si ils sont complètement opposés.

### 4.2.1 Distance de Bhattacharyya

La distance de Bhattacharyya, comme définit dans l'article [2], permet de calculer une distance entre deux lois de probabilité discrètes. Elle est définie pour deux lois  $X$  et  $Y$  par la formule :

$$D_B(X, Y) = \sqrt{1 - BC(X, Y)}$$

ou bien encore, par la formule :

$$D_B(X, Y) = -\ln(BC(X, Y))$$

Où  $BC(X, Y)$  est appelé coefficient de Bhattacharyya et est défini par la formule :

$$\sum_{k \in X \text{ ou } k \in Y} \sqrt{X(k) * Y(k)}$$

### 4.2.2 Cosine similarity

La similarité cosinus (Cosine similarity), est une distance, un score, basé sur le cosinus de l'angle entre deux vecteurs.

Donc, à partir d'un vecteur descripteur  $X$  et du vecteur descripteur de référence  $Y$ , nous pouvons calculer le cosinus de l'angle formé par les deux vecteurs, qui est donné par la formule :

$$Sc(X, Y) = \frac{X \cdot Y}{\|X\| \|Y\|}$$

## 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 particules discrètes pour représenter la densité de probabilité associée 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  $n$ ème particule,  $w_n$  indique l'importance de la particule, ou le poids de celle-ci, et  $N$  est le nombre total de particules. Plus d'information peuvent être trouvées aux références suivantes [12] et [1].

Cet algorithme peut, par exemple, être utilisé avec le descripteur HOG, comme dans ces articles [16], [6], [9] et [4], mais il peut également être utilisé avec d'autres descripteurs, ou combinaison de descripteur, comme HOG cascade combiné avec LBP (Local Binary Pattern).

Les particules que le filtre utilise peuvent être définies de plusieurs façons. 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 également 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 algorithme 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 puis en ajoutant du bruit, 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 particules 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     Prédiction de  $p$  grâce à un modèle de prédiction  $f$  et  $\nu$  du bruit blanc :  $\hat{X}_p^t = f(X_p^{t-1}, \nu)$

3     Traitement de l'observation grâce à un modèle de mesure  $h$  et  $n$  le bruit de l'observation :  $z^t = h(z, n)$

4     Combinaison de  $\hat{X}_p^t$  avec  $z^t$  pour mettre à jour le poids de  $p$  :  $w_p^t$

5 **fin**

6 Normalisation des poids des particules :

$$w_p^t = \frac{w_p^t}{\sum_{i=0}^{NB_{particule}} w_i^t}$$

7     **si** le nombre de particule effective est inférieur à un certain seuil **alors**

8         Ré-échantillonnage des particules pondérées par leur poids

9     **fin**

9 Estimation de l'état de notre système :

$$(MLE) \quad X^t = \frac{1}{NB_{particule}} * \sum_{p=0}^{NB_{particule}} X_p^t * w_p^t$$

ou

$$(MAP) \quad X^t = \arg \max_p w_p^t$$

---

Dans notre cas, le modèle de mesure sera effectué en calculant la similarité entre un descripteur de référence  $F_{ref}$  et les descripteurs de chaque particule. Au préalable, chaque particule se voit associer un patch de l'image courante qui est récupéré en utilisant la position et la bounding box de la particule, puis en dimensionnant le patch obtenu à une dimension fixée en amont par l'utilisateur. Cette liste de patchs est ensuite donnée à une fonction de mesure de similarité, qui, pour chacun des patchs, renvoi un coefficient de similarité ( $coeff\_sim$ ).

Ces coefficients de similarité sont ensuite utilisés pour calculer le poids de chaque particule, grâce à une distribution gaussienne centrée en 0 et de déviation standard  $\sigma$  (dans notre cas  $\sigma = n$ , le bruit de l'observation) :

$$w_p^t = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(coeff\_sim_p)^2}{2\sigma^2}}$$

Après estimation de l'état de notre système, nous calculons le descripteur associé à cet état et remplaçons le descripteur de référence  $F_{ref}$  par celui-ci.

Le modèle de prédiction, quant à lui, utilise la méthode des différences finies d'ordre 3, c'est-à-dire que nous considérons les trois derniers états de la particule pour calculer sa position, vitesse et accélération actuelle. Cependant, d'autres modèles auraient pu être choisis, comme le modèle de prédiction utilisant les équations physique du mouvement, qui a aussi été implémenté.

La méthode des différences finies est basée sur les formules suivante :

$$\begin{aligned} \dot{x}^t &= \frac{x^t - x^{t-1}}{dt} \\ \dot{y}^t &= \frac{y^t - y^{t-1}}{dt} \end{aligned} \tag{4.1}$$

$$\begin{aligned} \ddot{x}^t &= \frac{\dot{x}^t - \frac{x^{t-1} - x^{t-2}}{dt}}{dt} = \frac{\dot{x}^t - \dot{x}^{t-1}}{dt} \\ \ddot{y}^t &= \frac{\dot{y}^t - \frac{y^{t-1} - y^{t-2}}{dt}}{dt} = \frac{\dot{y}^t - \dot{y}^{t-1}}{dt} \end{aligned} \tag{4.2}$$

$$\begin{aligned} x^{t+dt} &= \frac{1}{2} \cdot \ddot{x}^t \cdot dt^2 + \dot{x}^t \cdot dt + x^t + \nu \\ y^{t+dt} &= \frac{1}{2} \cdot \ddot{y}^t \cdot dt^2 + \dot{y}^t \cdot dt + y^t + \nu \end{aligned} \tag{4.3}$$

Avec  $dt$  l'intervalle de temps entre chaque image de la vidéo, et  $\nu$  le bruit gaussien pour la prédiction.

Au cours de la vidéo, il arrive que la cible se rapproche ou s'éloigne de la caméra, si les dimensions de la bounding box sont fixes. On peut perdre en qualité sur le suivi, car la bounding box peut inclure trop d'éléments parasites, ou au contraire, n'inclure qu'une petite partie de la cible.

Pour palier à ce problème, nous effectuons également une prédiction sur les dimensions de la bounding box, définie par les formules suivante, comme énoncé dans [6] :

$$\gamma = \begin{cases} -0.05, & 0 \leq p \leq 0.2 \\ -0.0125, & 0.2 \leq p \leq 0.4 \\ 0, & 0.4 \leq p \leq 0.6 \\ 0.0125, & 0.6 \leq p \leq 0.8 \\ 0.05, & 0.8 \leq p \leq 1 \end{cases} \tag{4.4}$$

Où  $p$  est un réel aléatoire entre  $[0,1]$ .

On a alors :

$$\begin{aligned} l^t &= l^{t-1} * (1 + \gamma) + \nu \\ h^t &= h^{t-1} * (1 + \gamma) + \nu \end{aligned} \quad (4.5)$$

En assemblant chacune des parties précédentes, on obtient alors l'algorithme final du filtre à particule, dont un exemple avec une configuration spécifique est donné dans l'algorithme 2.

---

### 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 (équation 4.3) et de la bounding box (équation 4.5) de  $p$ 
8             dans  $t$  en fonction de ses états précédents
9             Calcul du descripteur du patch correspondant à  $p$  :  $F_p^t$ 
10             Calcul du coefficient de similarité entre  $F_p^t$  et  $F_{ref}^{t-1}$  :  $coeff\_sim_p^t$ 
11             Calcul du poids de  $p$  dans  $t$  :

$$w_p^t = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(coeff\_sim_p^t)^2}{2\sigma^2}}$$

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

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

15         fin
16     fin

```

---

## **5 Analyse des résultats**

### **5.1 Méthodologie**

### **5.2 Analyse et comparaison**

# 6 Gestion du Projet

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 décidées par tous les membres du groupe.

La planification s'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.

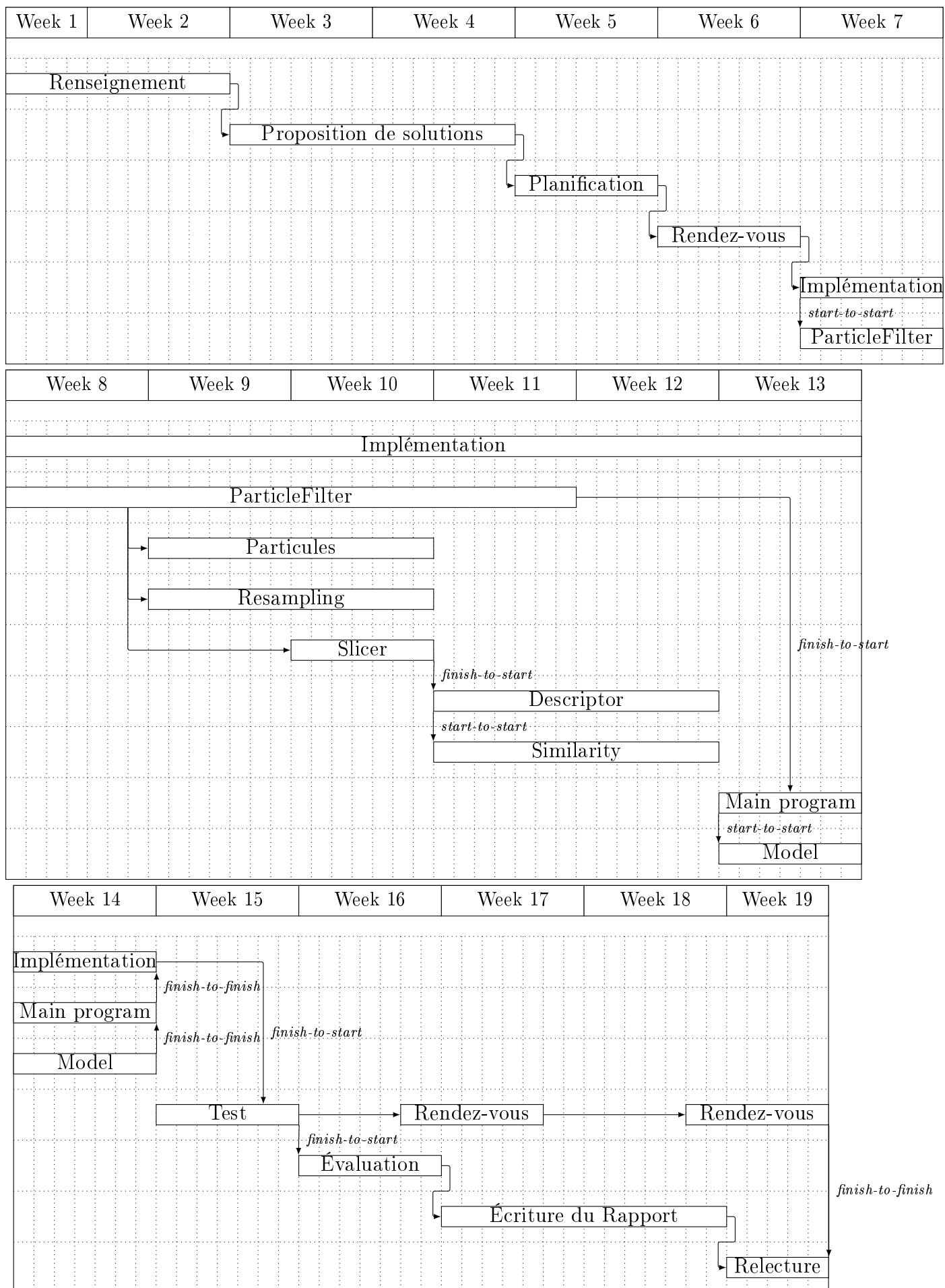


FIGURE 6.1 – Diagramme de Gantt du projet.

## 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 différentes combinaisons de descripteurs et de mesures de similarité. Les résultats montrent que sur des vidéos prises en conditions réelles, et avec une configuration du logiciel adéquate, le suivi se fait avec une efficacité d'environ 75% par rapport aux données de référence. 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 se 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] A. BHATTACHARYYA. « On a Measure of Divergence between Two Multinomial Populations ». In : *Sankhyā : The Indian Journal of Statistics (1933-1960)* 7.4 (1960), p. 401-406.
- [3] Jifeng DAI et al. *R-FCN : Object Detection via Region-based Fully Convolutional Networks*. arXiv :1605.06409 [cs]. Juin 2016.
- [4] 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.
- [5] Combe-Ounkham GABRIEL et al. *PP2 Particle Filter*. 2023. URL : [https://github.com/gabriel-combe/PP2\\_Particle\\_Filter](https://github.com/gabriel-combe/PP2_Particle_Filter).
- [6] 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.
- [7] Tsung-Yi LIN et al. *Focal Loss for Dense Object Detection*. arXiv :1708.02002 [cs]. Fév. 2018.
- [8] Wei LIU et al. « SSD : Single Shot MultiBox Detector ». In : t. 9905. 2016, p. 21-37.
- [9] 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.
- [10] Joseph REDMON et al. *You Only Look Once : Unified, Real-Time Object Detection*. Mai 2016.
- [11] Shaoqing REN et al. *Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks*. arXiv :1506.01497 [cs]. Jan. 2016.
- [12] Stuart RUSSEL et Peter NORVIG. *Artificial Intelligence : A Modern Approach*. Pearson, 2021. Chap. 14, p. 509-517. ISBN : 1-292-40113-3.
- [13] 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.
- [14] Chien-Yao WANG, Alexey BOCHKOVSKIY et Hong-Yuan Mark LIAO. *YOLOv7*. 2022. URL : <https://github.com/WongKinYiu/yolov7>.
- [15] 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 : () .
- [16] 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, se calcule sur deux bounding box, comme dans la figure 1.

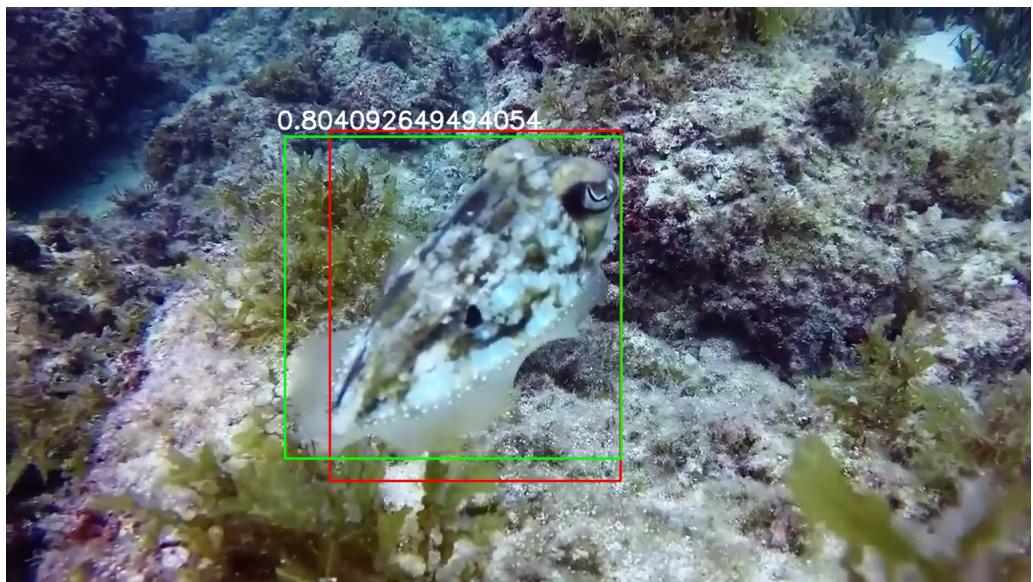


FIGURE 1 – IoU (blanc) entre deux bounding box.

# Diagramme UML global

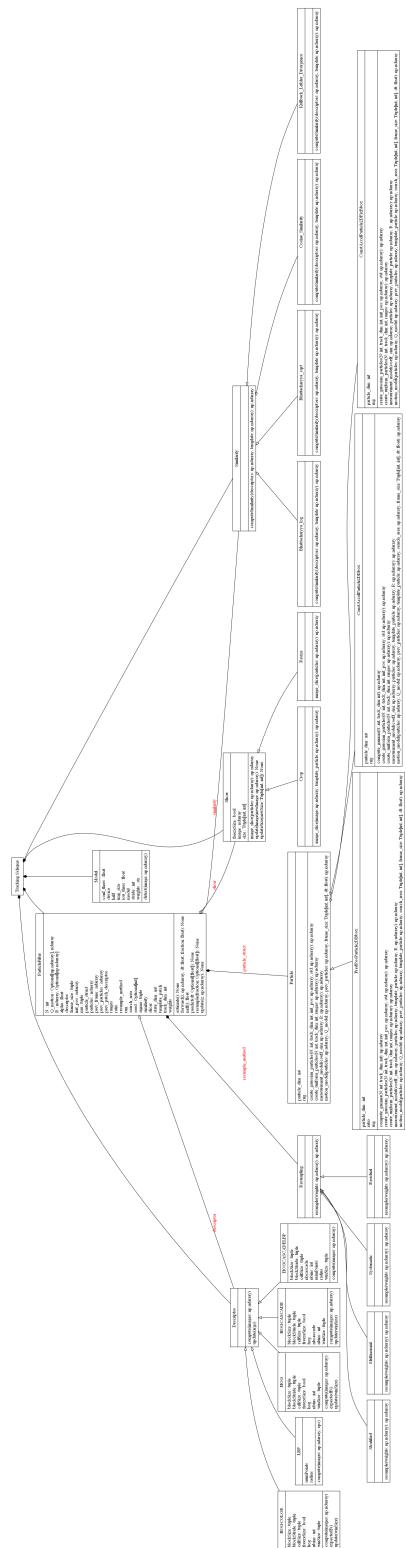


FIGURE 2 – Diagramme UML des classes globales (image SVG).

# 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'échantillons provenant de cette distribution.

Ce genre de réseaux 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.

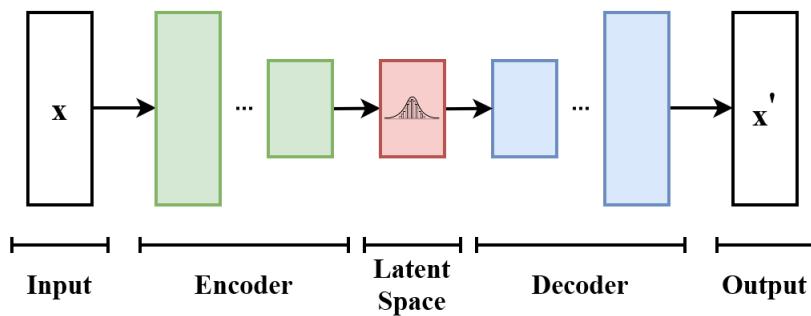


FIGURE 3 – Schéma basique d'un VAE.