

Université de Paris 8  
ATI  
*M1 ATI.*

# L'IA generative pour les interactions dans les jeux videos

Veille Technologique

Combe-Ounkham Gabriel

2024-2025

# Résumé

L'utilisation d'Intelligence Artificielle (IA) pour les interactions dans les jeux videos est une idée emergeante mais encore peu applique.

[17] [16] [18] [14] [11] [4] [13] [3] [8] [12] [7] [2] [10] [9] [5] [6] [1] [19] [15]

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Énoncé du problème . . . . .	6
1.2	Motivation . . . . .	6
1.3	Méthodes . . . . .	6
1.4	Cahier des charges . . . . .	6
1.4.1	Besoins fonctionnels . . . . .	6
1.4.2	Besoins non-fonctionnels . . . . .	6
1.4.3	Contraintes . . . . .	6
<b>2</b>	<b>Technologies</b>	<b>7</b>
2.1	Labellimg et Roboflow . . . . .	7
2.2	État de l’art de la détection d’objet . . . . .	7
2.3	Langage de programmation . . . . .	8
<b>3</b>	<b>Développements Logiciel : Conception, Modélisation, Implémentation</b>	<b>9</b>
3.1	Développements logiciel . . . . .	9
3.1.1	Intelligence Artificielle . . . . .	9
3.1.2	Logiciel de suivi de seiches . . . . .	10
3.2	Modules . . . . .	11
3.2.1	Descripteurs . . . . .	11
3.2.2	Mesure de similarité . . . . .	11
3.2.3	Filtre à particule . . . . .	12
3.3	Structures de données . . . . .	12
3.4	Statistiques . . . . .	12
<b>4</b>	<b>Algorithmes et Analyse</b>	<b>13</b>
4.1	YOLO : une méthode basée sur les CNN . . . . .	13
4.2	Descripteurs . . . . .	13
4.2.1	Histogramme de gradient orienté . . . . .	13
4.2.2	HOG en cascade . . . . .	15
4.3	Mesures de similarité . . . . .	15
4.3.1	Distance de Bhattacharyya . . . . .	15
4.3.2	Cosine similarity . . . . .	16
4.4	Filtre à particule . . . . .	16
<b>5</b>	<b>Analyse des résultats</b>	<b>20</b>
5.1	Métrique d’évaluation IoU . . . . .	20
5.2	Analyse et comparaison . . . . .	21
<b>6</b>	<b>Gestion du Projet</b>	<b>23</b>
<b>7</b>	<b>Bilan et Conclusions</b>	<b>25</b>
	<b>Appendices</b>	<b>27</b>

# Table des figures

3.1	Performances de notre modèle après entraînement. . . . .	9
3.2	Exemples de résultats obtenus par notre modèle. . . . .	10
3.3	Exemple d'une image renvoyée par le logiciel. . . . .	10
3.4	IoU (blanc) entre l'image de référence (vert) et notre résultat (rouge). . . . .	10
3.5	Diagramme UML de cas d'utilisation. . . . .	11
3.6	Diagramme UML des classes du module descripteur. . . . .	11
4.1	Bloc d'un HOG. . . . .	14
4.2	Déplacement d'un bloc. . . . .	15
4.3	HOG en cascade. . . . .	15
5.1	Intersection over Union (IoU) . . . . .	20
5.2	Exemple de résultats obtenus par notre filtre à particule. . . . .	21
5.3	Exemple de résultats obtenus par YOLOv7. . . . .	21
5.4	Trajectoire avec le filtre à particule sur les X . . . . .	22
5.5	Trajectoire avec le filtre à particule sur les Y . . . . .	22
5.6	Trajectoire avec YOLOv7 sur les X . . . . .	22
5.7	Trajectoire avec YOLOv7 sur les Y . . . . .	22
5.8	Exemple de résultats divergeant obtenus par notre filtre à particule. . . . .	22
6.1	Diagramme de Gantt du projet. . . . .	24
1	Schéma vrais/faux positifs/négatifs . . . . .	28
2	Tracé de la courbe Precision-Recall ou PR curve . . . . .	28
3	Diagramme UML des classes globales (lien vers l'image SVG, zoomer sur l'image SVG pour avoir plus de détails). . . . .	29
4	Diagramme UML de la classe Similarity (lien vers l'image SVG, zoomer sur l'image SVG pour avoir plus de détails). . . . .	30
5	Diagramme UML de la classe ParticleFilter (lien vers l'image SVG, zoomer sur l'image SVG pour avoir plus de détails). . . . .	31
6	Schéma basique d'un VAE. . . . .	32

# Liste des Algorithmes

1	Filtre à particule général . . . . .	17
2	Filtre à particule . . . . .	19

# 1 Introduction

## 1.1 Énoncé du problème

## 1.2 Motivation

## 1.3 Méthodes

## 1.4 Cahier des charges

### 1.4.1 Besoins fonctionnels

### 1.4.2 Besoins non-fonctionnels

### 1.4.3 Contraintes

## 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[wang\_yolov7\_nodate] à détecter des seiches dans une image. L'annotation a été répartie 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 ont été rajoutés 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*[ren\_faster\_2016], *RetinaNet*[lin\_focal\_2018], *SSD*[liu\_ssd\_2016], *YOLO*[redmon\_you\_2016] ou en encore R-FCN[dai\_r-fcn\_2016] (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 [sanchez\_review\_2020]). 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[wang\_yolov7\_nodate] 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 annexe 7).

## 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[wang\_yolov7\_nodate], qui utilise pytorch.

Python possède également une grande quantité de bibliothèques, 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'images et les opérations matricielles à des bibliothèques 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 celui-ci 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[wang\_yolov7\_nodate], qui peuvent être trouvés sur le github officiel de YOLOv7[yolov7\_github]. 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.

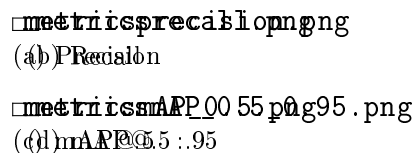


FIGURE 3.1 – Performances de notre modèle après entraînement.

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

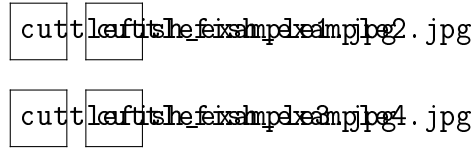


FIGURE 3.2 – Exemples de résultats obtenus par notre modèle.

### 3.1.2 Logiciel de suivi de seiches

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 de la 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.

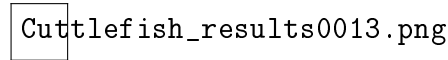


FIGURE 3.3 – Exemple d'une image renvoyée par le logiciel.

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 chapitre 5) entre une estimation à un instant  $t$  et la véritable valeur à ce même instant, comme montré dans la figure 3.4.

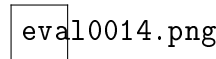


FIGURE 3.4 – IoU (blanc) entre l'image de référence (vert) et notre résultat (rouge).

## 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.

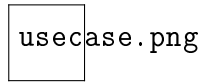


FIGURE 3.5 – Diagramme UML de cas d'utilisation.

### 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 employer 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.

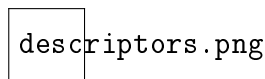


FIGURE 3.6 – Diagramme UML des classes du module descripteur.

### 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 dont l'utilisateur peut se servir 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 4.

### 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 5.

## 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 sont 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 différents 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 3 en annexe). Les scripts et classes de YOLOv7[wang\_yolov7\_nodate] 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 ([pp2pf]).

## 4 Algorithmes et Analyse

### 4.1 YOLO : une méthode basée sur les CNN

YOLO est un algorithme basé sur la régression qui classifie et prédit des bounding boxes pour une image entière en un seul passage [redmon\_you\_2016].

Dans un premier temps, un réseau de neurones traite l'image entière. Puis, l'image est divisée en cellules et dans chaque cellules on prédit s'il y a un objet ou non.

Comme mentionné dans le chapitre 2, YOLO surpasse les autres algorithmes de détection, comme R-CNN et fast R-CNN, en particulier en terme de vitesse. Cette vitesse est due au fait qu'il utilise un unique réseau de neurones et qu'il extrait des informations globales du contexte de l'image, tandis que d'autres algorithmes de détection utilisent un grand nombre de réseaux de neurones (jusqu'à plusieurs milliers).

Dans notre travail, nous avons utilisé la septième version de YOLO (YOLOv7) pour ses améliorations significatives en terme de précision et de vitesse [wang\_yolov7\_nodate].

### 4.2 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, ...).

#### 4.2.1 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 [dalal\_histograms\_2005]. 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 [dalal\_histograms\_2005].

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 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.

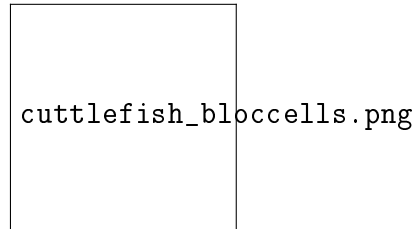


FIGURE 4.1 – Bloc d'un HOG.

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 :

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

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.

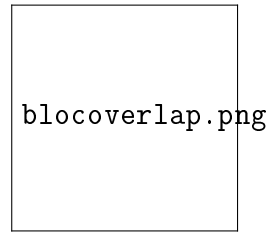


FIGURE 4.2 – Déplacement d'un bloc.

### 4.2.2 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 [qiang\_zhu\_fast\_2006]. De cette manière, on ajoute de l'information spatiale à notre descripteur.

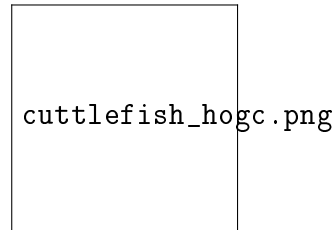


FIGURE 4.3 – HOG en cascade.

## 4.3 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.

### 4.3.1 Distance de Bhattacharyya

La distance de Bhattacharyya, comme définit dans l'article [battacharyya\_measure\_1960], 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 :

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

### 4.3.2 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 :

$$Sc(X, Y) = 1 - \frac{X \cdot Y}{\|X\| \|Y\|}$$

## 4.4 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 nième particule,  $w_n$  indique l'importance de la particule, ou le poids de celle-ci, et  $N$  est le nombre total de particules. Davantage d'informations peuvent être trouvées aux références suivantes [russell\_norvig] et [rlabbe].

Cet algorithme a, par exemple, pu être utilisé avec le descripteur HOG, comme dans ces articles [xu\_human\_2010], [kong\_particle\_2019], [qiang\_zhu\_fast\_2006] et [dalal\_histograms\_2005] mais il peut également être utilisé avec d'autres descripteurs, ou combinaison de descripteurs, 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 pixels, respectivement,  $(l^t, h^t)$  décrit la bounding box de notre cible en pixels 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 :

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

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

7     Ré-échantillonnage des particules pondérées par leur poids

8 **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 patches est ensuite donnée à une fonction de mesure de similarité, qui, pour chacun des patches, renvoie 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 :

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

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

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

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, défini par les formules suivante, comme énoncé dans [kong\_particle\_2019] :

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

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

On a alors :

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

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.

---

<b>Algorithme 2 : Filtre à particule</b>	
<b>Données :</b> Une vidéo sous-marine de seiche	
<b>Résultat :</b> La liste des positions et bounding box de la cible dans la vidéo	
1	<b>pour chaque</b> <i>Images</i> $t$ <b>de la vidéo faire</b>
2	<b>si</b> <i>Première image</i> <b>alors</b>
3	Détection de la seiche dans l'image grâce a YOLOv7
4	Calcul du descripteur de référence : $F_{ref}^0$
5	<b>sinon</b>
6	<b>pour chaque</b> <i>Particules</i> $p$ <b>faire</b>
7	Prédiction de la position (équation 4.3) et de la bounding box (équation 4.5) 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 coefficient de similarité entre $F_p^t$ et $F_{ref}^{t-1}$ : $coeff\_sim_p^t$
10	Calcul du poids de $p$ dans $t$ :
	$w_p^t = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(coeff\_sim_p^t)^2}{2\sigma^2}}$
11	<b>fin</b>
12	Normalisation des poids des particules :
	$w_p^t = \frac{w_p^t}{\sum_{i=0}^{NB_{particule}} w_i^t}$
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	<b>fin</b>
16	Mise à jour du descripteur de référence : $F_{ref}^t$
17	<b>fin</b>

---

# 5 Analyse des résultats

## 5.1 Métrique d'évaluation IoU

En général, Intersection over Union (IoU) (ou indice de Jaccard) est considéré comme la métrique la plus populaire pour l'évaluation de détection d'objet. Dans le domaine de la détection d'objet, IoU est utilisé pour mesurer la similarité entre la bounding box prédit  $B_p$  et la bounding box de la vérité terrain  $B_{gt}$ , en mesurant l'intersection (l'aire du chevauchement) pour  $B_p$  et  $B_{gt}$ , divisée par l'aire de leur union, qui est :

$$IoU = J(B_p, B_{gt}) = \frac{\text{aire}(B_p \cap B_{gt})}{\text{aire}(B_p \cup B_{gt})}$$

Comme illustré en figure 5.1 :

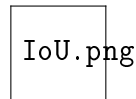


FIGURE 5.1 – Intersection over Union (IoU)

Dans notre cas, les bounding boxes de prédiction correspondent à la sortie de notre logiciel, ou à celle du réseau de neurones (YOLOv7). Les bounding boxes de la vérité terrain, quant à elles, correspondent à des bounding boxes annotées manuellement et qui englobe l'objet ciblé à détecter (i.e. la seiche).

Pour avoir la vérité terrain, nous avons utilisé LabelImg.

Afin de classer le résultat de la détection comme étant correcte ou non, nous comparons l'IoU à un seuil donné  $T$ . Donc, si  $IoU > T$ , alors nous pouvons considérer la détection comme étant correcte, autrement, la détection est considérée comme incorrecte.

Dans nos tests, nous avons fixé le seuil  $T$  à 0.5.

## 5.2 Analyse et comparaison

Les valeurs IoU sont calculées pour une séquence et deux approches (filtre à particule et YOLOv7), elles sont représentées dans la table 5.1.

Notre méthode avec filtre à particule a réussi à détecter la seiche dans 288 des 292 frames, équivalent à 98% du nombre total de frames, voir figure 5.2. L'IoU varie de 0.44 à 0.87 et l'IoU moyen est de 0.74.

YOLOv7 a réussi à détecter la seiche dans 292 des 292 frames, équivalent à 100% du nombre total de frames, voir figure 5.3. L'IoU varie de 0.60 à 0.90 et l'IoU moyen est de 0.76.

Méthode	Frames	Fréquence de détection	IoU min	IoU max	IoU mean
Filtre à particule	292	288 (98%)	0.44	0.87	0.74
YOLOv7	292	292 (100%)	0.60	0.90	0.76

TABLE 5.1 – Résultats pour une même séquence en utilisant notre filtre à particule et YOLOv7.

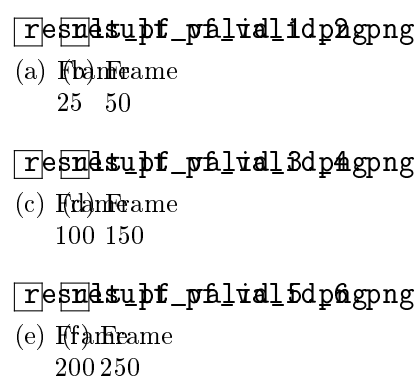


FIGURE 5.2 – Exemple de résultats obtenus par notre filtre à particule.

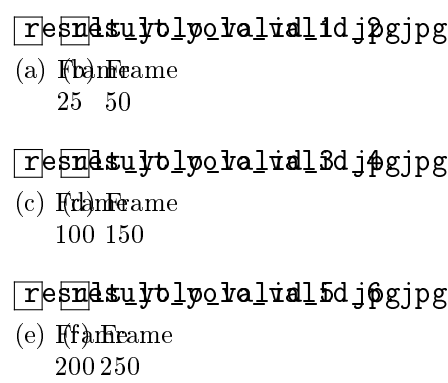


FIGURE 5.3 – Exemple de résultats obtenus par YOLOv7.

Les courbes de déplacement de la seiche dans les figure 5.4, 5.5, 5.6 et 5.7 illustrent un exemple de détection continue dans une séquence.

Dans la figure 5.4, nous comparons la trajectoire de la seiche suivi par le filtre à particule (couleur violette) aux positions définies manuellement (couleur or) dans la direction des x, tandis que dans la figure 5.5 nous comparons les deux trajectoires dans la direction des y.

De même, dans la figure 5.6 nous comparons la trajectoire de la seiche suivi par YOLOv7 (couleur bleu) aux positions définies manuellement (couleur or) dans la direction des x, tandis que dans la

figure 5.7 nous comparons les deux trajectoires dans la direction des y.

La distance entre la trajectoire automatique et manuelle indique la précision de l'algorithme, ce qui donne pour la séquence une erreur moyenne globale de 21 pixels pour le filtre à particule, calculé sur les 288 frames de la séquence où la seiche a été détectée, et 12 pixels pour YOLOv7, calculé sur les 292 frames de la séquence où la seiche a été détectée.

La méthode de détection avec filtre à particule semble détecter la seiche dans 98% de la séquence, et la méthode de détection avec YOLOv7 dans 100% de la séquence.

X\_displacement\_pf.png

FIGURE 5.4 – Trajectoire avec le filtre à particule sur les X

Y\_displacement\_pf.png

FIGURE 5.5 – Trajectoire avec le filtre à particule sur les Y

X\_displacement\_yolo.png

FIGURE 5.6 – Trajectoire avec YOLOv7 sur les X

Y\_displacement\_yolo.png

FIGURE 5.7 – Trajectoire avec YOLOv7 sur les Y

Cependant, la détection avec le filtre à particule peut diverger, comme illustré dans la figure 5.8. Cette divergence peut s'expliquer de plusieurs manières. Elle peut provenir d'une accumulation d'erreur lors du suivi, ou bien d'une paramétrisation du logiciel non optimale, ou encore de la seiche qui n'est que sur une petite portion de la bounding box, ce qui a pour effet de beaucoup plus représenter l'arrière plan et donc mettre un poids plus élevé sur ce qui ressemble à l'arrière plan lors de l'étape de calcul des poids.

resnatupf\_pfiyadval1dp2.png  
resnatupf\_pfiyadval3dp4.png

FIGURE 5.8 – Exemple de résultats divergeant obtenus par notre filtre à particule.

## 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 propositions 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.

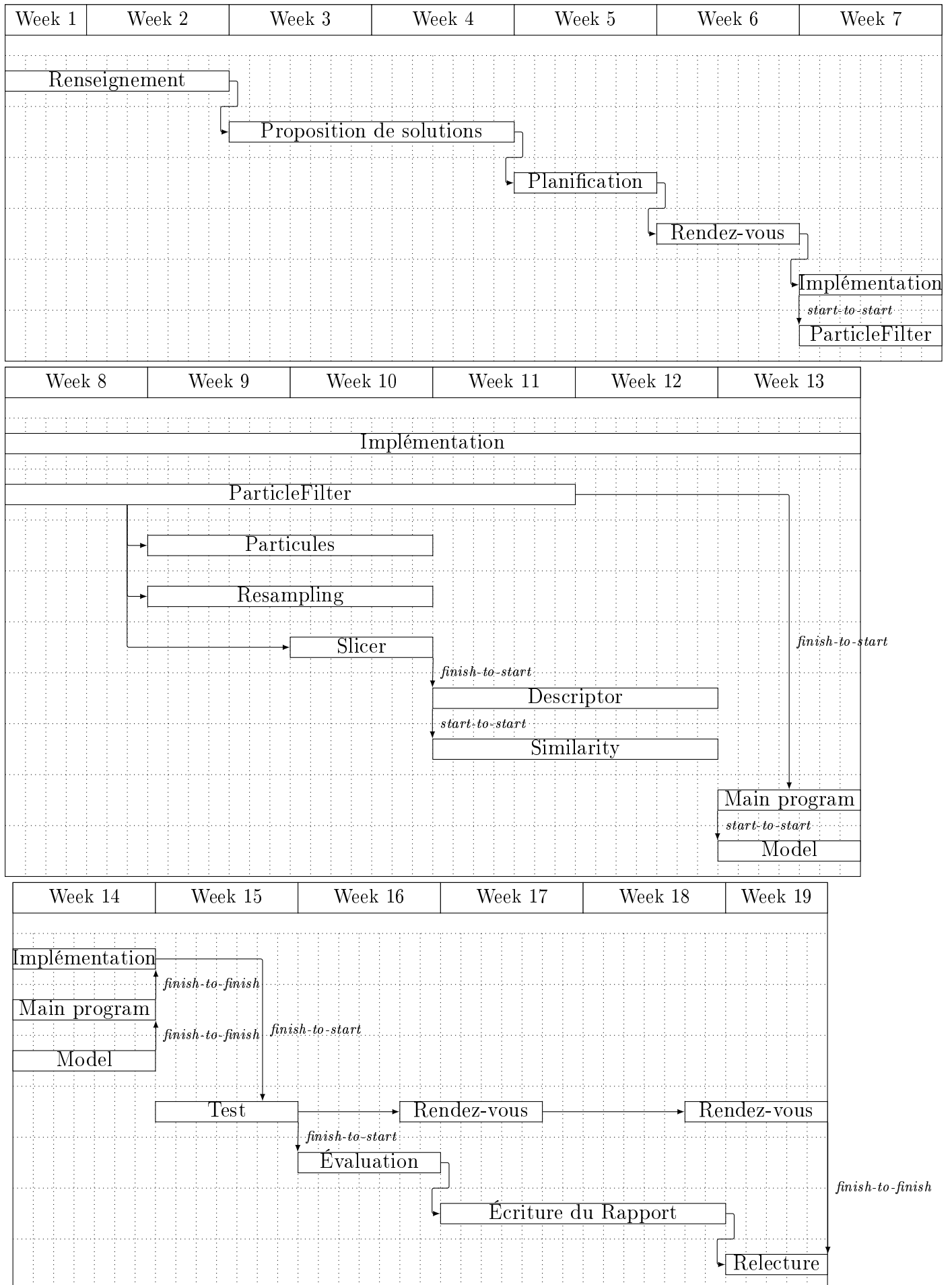


FIGURE 6.1 – Diagramme de Gantt du projet.



## 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 98% par rapport aux données de référence. Ce résultat est très satisfaisant et n'a qu'une différence de 2% 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] Nader AKOURY, Ronan SALZ et Mohit IYYER. « Towards Grounded Dialogue Generation in Video Game Environments ». In : ().
- [2] Akif Yasin AYAS et al. « Artificial Intelligence (AI)-Based Self-Deciding Character Development Application in Two-Dimensional Video Games ». In : *Bilgi ve İletişim Teknolojileri Dergisi* 5.1 (28 juin 2023), p. 1-19. ISSN : 2687-492X. DOI : 10.53694/bited.1247338. URL : <http://dergipark.org.tr/en/doi/10.53694/bited.1247338> (visité le 14/02/2025).
- [3] Steph BUONGIORNO et al. « PANGeA : Procedural Artificial Narrative Using Generative AI for Turn-Based, Role-Playing Video Games ». In : *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 20.1 (15 nov. 2024), p. 156-166. ISSN : 2334-0924, 2326-909X. DOI : 10.1609/aiide.v20i1.31876. URL : <https://ojs.aaai.org/index.php/AIIDE/article/view/31876> (visité le 14/02/2025).
- [4] Vikas CHAUDHARY et al. *Deep Learning in Gaming and Animations : Principles and Applications*. 1<sup>re</sup> éd. Boca Raton : CRC Press, 18 oct. 2021. ISBN : 978-1-003-23153-0. DOI : 10.1201/9781003231530. URL : <https://www.taylorfrancis.com/books/9781003231530> (visité le 14/02/2025).
- [5] Jason Patrick Winarto HARDIMAN et al. « AI-powered dialogues and quests generation in role-playing games using Google's Gemini and Sentence BERT framework ». In : *Procedia Computer Science* 245 (2024), p. 1111-1119. ISSN : 18770509. DOI : 10.1016/j.procs.2024.10.340. URL : <https://linkinghub.elsevier.com/retrieve/pii/S187705092403148X> (visité le 14/02/2025).
- [6] Dmitrii IARVOI, Richard HEBBLEWHITE et Phoey Lee TEH. « AI's Influence on Non-Player Character Dialogue and Gameplay Experience ». In : *Intelligent Computing*. Sous la dir. de Kohei ARAI. T. 1016. Series Title : Lecture Notes in Networks and Systems. Cham : Springer Nature Switzerland, 2024, p. 76-92. ISBN : 978-3-031-62280-9 978-3-031-62281-6. DOI : 10.1007/978-3-031-62281-6\_6. URL : [https://link.springer.com/10.1007/978-3-031-62281-6\\_6](https://link.springer.com/10.1007/978-3-031-62281-6_6) (visité le 14/02/2025).
- [7] James Hutson2 JEREMIAH RATCAN1\*. « Adaptive Worlds : Generative AI in Game Design and Future of Gaming, and Interactive Media ». In : (6 oct. 2024). Publisher : ISRG Publishers. ISSN : 2583-7672. DOI : 10.5281/ZENODO.13894497. URL : <https://zenodo.org/doi/10.5281/zenodo.13894497> (visité le 14/02/2025).
- [8] Niels JUSTESEN et al. *Human-like Bots for Tactical Shooters Using Compute-Efficient Sensors*. 30 déc. 2024. DOI : 10.48550/arXiv.2501.00078. arXiv : 2501.00078[cs]. URL : <http://arxiv.org/abs/2501.00078> (visité le 14/02/2025).
- [9] Benjamin KENWRIGHT. « Why Player-AI Interaction Research Will Be Critical to the Next Generation of Video Games ». In : ().
- [10] Jialu LI et al. *Unbounded : A Generative Infinite Game of Character Life Simulation*. 30 oct. 2024. DOI : 10.48550/arXiv.2410.18975. arXiv : 2410.18975[cs]. URL : <http://arxiv.org/abs/2410.18975> (visité le 14/02/2025).
- [11] Jiayun LU et Junhui WANG. « Research on the Application of Generative Artificial Intelligence in the Video Game Field ». In : ().

- [12] Antonio MACIÁ-LILLO et al. « Hybrid Architecture for AI-Based RTS Games ». In : *IEEE Transactions on Games* (2025), p. 1-14. ISSN : 2475-1502, 2475-1510. DOI : 10.1109/TG.2025.3533949. URL : <https://ieeexplore.ieee.org/document/10854898/> (visité le 14/02/2025).
- [13] Xinyu MAO et al. *Procedural Content Generation via Generative Artificial Intelligence*. 12 juill. 2024. DOI : 10.48550/arXiv.2407.09013. arXiv : 2407.09013[cs]. URL : <http://arxiv.org/abs/2407.09013> (visité le 14/02/2025).
- [14] Jeremiah RATICAN et James HUTSON. « Video game development 3.0 : AI-driven collaborative co-creation ». In : *Metaverse* 5.2 (17 déc. 2024), p. 2904. ISSN : 2810-9791. DOI : 10.54517/m2904. URL : <https://aber.apacsci.com/index.php/met/article/view/2904> (visité le 14/02/2025).
- [15] Dr Mariët THEUNE et Dr Lorenzo GATTI. « Affective Dialogue Generation for Video Games ». In : ().
- [16] Paul TOPRAC. « Automation of Game Development using AI and Other Tools for AAA Video Game Production ». In : ().
- [17] Tom TUCEK et al. « One Spell Fits All : A Generative AI Game as a Tool for Research in AI Creativity and Sustainable Design ». In : ().
- [18] Jiwen YU et al. *GameFactory : Creating New Games with Generative Interactive Videos*. 14 jan. 2025. DOI : 10.48550/arXiv.2501.08325. arXiv : 2501.08325[cs]. URL : <http://arxiv.org/abs/2501.08325> (visité le 14/02/2025).
- [19] Guanwei ZENG. « A review of AI-based game NPCs research ». In : *Applied and Computational Engineering* 15.1 (23 oct. 2023), p. 155-159. ISSN : 2755-2721, 2755-273X. DOI : 10.54254/2755-2721/15/20230827. URL : <https://www.ewadirect.com/proceedings/ace/article/view/4569> (visité le 14/02/2025).

# Appendices

## mAP (mean Average Precision)

La mAP est une mesure de précision pour les algorithmes de détection d'objet. La mAP correspond de manière générale à la moyenne de l'AP obtenu sur différents objets et à différents paramètres. Son calcul dépend du contexte et de la technique de comparaison.

L'Average Precision (AP) se calcule en se basant sur la précision, le rappel et l'Intersection over Union.

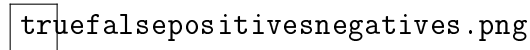


FIGURE 1 – Schéma vrais/faux positifs/négatifs

La précision correspond à la proportion de vrais positifs (VP) parmi l'ensemble des vrais positifs et des faux positifs (FP).

$$precision = \frac{VP}{VP + FP}$$

Le rappel correspond à la proportion de vrais positifs parmi l'ensemble des vrais positifs et des faux négatifs (FN).

$$rappel = \frac{VP}{VP + FN}$$

On trace ensuite la courbe représentant la précision en fonction du rappel :

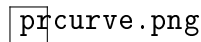


FIGURE 2 – Tracé de la courbe Precision-Recall ou PR curve

L'AP correspond à l'aire sous cette courbe Precision-Rappel (PR curve).

$$\int_0^1 f_{PR}(rappel) d(rappel)$$

## Diagramme UML global

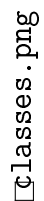
classes.png

FIGURE 3 – Diagramme UML des classes globales (lien vers l'image SVG, zoomer sur l'image SVG pour avoir plus de détails).

# Diagramme UML mesure de similarité


A small square icon with the text 'similarity.png' written vertically inside it, positioned over the title.

FIGURE 4 – Diagramme UML de la classe Similarity (lien vers l'image SVG, zoomer sur l'image SVG pour avoir plus de détails).

## Diagramme UML filtre à particule

particlefilter.

FIGURE 5 – Diagramme UML de la classe ParticleFilter (lien vers l'image SVG, zoomer sur l'image SVG pour avoir plus de détails).



# VAE (Variational Autoencoder)

Les VAE, ou Variational Autoencoder, sont des réseaux de neurones qui essaient 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 6.

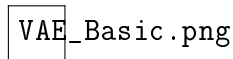


FIGURE 6 – Schéma basique d'un VAE.