Département Sciences du Numérique

Rapport Projet Long - Master PSMSC
UE Research Methodology

Joël Ky

19 Mars 2021

Sommaire

1 Abstract **3**

2 Introduction **4**

3 Travaux précédents **5**
3.1 Navigation à hautes altitudes 5
3.2 Navigation à basses altitudes 6

4 Descriptif détaillée des solutions **7**
4.1 YOLO 7
4.2 DisNet 9
4.3 Architecture améliorée pour la prédiction de distance 12
4.4 Assimilation de données : Filtre de Kalman Etendu 14
4.5 AirSim & Unreal Engine 16

5 Limites & Futurs travaux **18**
5.1 Limites 18
5.2 Futurs travaux 18

6 Conclusion **20**

Table des figures

1 YOLO Architecture 7
2 Objet Detector 8
3 Architecture DisNet 9
4 Variation de la précision en fonction du nombre de couches . . . 10
5 Variation de la précision en fonction du nombre de neurones . . . 11
6 Courbes d'apprentissage 11
7 Erreur absolue du modèle 12
8 Architecture 13
9 Génération de la distance 14
10 RMSE des deux architectures 15
11 Régression linéaire sur l'écart-type de l'assimilation 16
12 Environnement de simulation Unreal Engine & Mixamo 17

1 Abstract

The positioning of drones in front of a person is a subject that is still open and not much covered in the literature.
The aim of this paper is to present the various works in the field of drones positioning. We will then give further explanation about some recent works about Object detection such as the well-know YOLO (You Only Look Once) solution for object detection. In order to deal with the positioning of a drone we explore solutions about distance estimation using Artificial Intelligence and images from the drone camera. We will then present the DisNet solution and an improvement of this solution in this paper.
To enhance our results from distance estimation, we focus on Data assimilation, to correct errors incrementally.
We explore AirSim and Unreal Engine tools to create an environment of simulation with realistic landscapes and persons in order to train and test our future solution.

Keywords : *Unmanned Aerial Vehicle, Human Detection, AirSim, DisNet, Data Assimilation, Deep Learning*

2 Introduction

Les drones sont de plus en plus utilisés, notamment grâce à leurs multiples applications. Ces outils s'avèrent très pratiques pour des missions de surveillance et aussi de sauvetage. C'est dans cette optique que la société SII Group, dans le cadre de son projet de recherche DROOPI, nous avons eu à prototyper un drone de sauvetage et de secourisme. Ce drone doit pouvoir se positionner devant une personne en détresse et entrer en communication avec elle par l'intermédiaire d'un chat-bot.
Le positionnement du drone requiert de pouvoir détecter une personne en détresse par l'intermédiaire de la caméra du drone. L'objectif de ce rapport est de présenter la revue bibliographique préalable des différentes solutions ou articles scientifiques dans le domaine, permettant de répondre à cette problématique. Nous présenterons donc entre autres les différents travaux précédents réalisés dans le cadre de la navigation des drones à hautes altitudes et à basses altitudes.
La deuxième partie consistera à une présentation détaillée de solutions de détection de personne et d'estimation de distance. Nous aborderons aussi dans cette partie la solution de l'assimilation de données, notamment le filtre de Kalman ainsi que les analyses des avantages de cette solution pour notre projet. La situation sanitaire actuelle ne permettant pas des tests en situation réelle, nous étudierons aussi les différents outils permettant de récréer des environnements ainsi que des paysages réalistes permettant l'implantation et les tests de notre solution.
La troisième partie consistera à analyser les limites des différents travaux ainsi que les potentiels futurs travaux pouvant être réaliser dans le but de compenser ces limites.

3 Travaux précédents

Dans le cadre de ce projet, il existe peu ou pas d'articles traitant exactement du positionnement du drones face à une personne. Les articles existant sont sur la reconnaissance des personnes à l'aide d'un drone depuis de hautes altitudes. On arrive ainsi à détecter des personnes dans des environnements compliqués ou très boisés. D'autres articles sont basés sur la détection depuis des altitudes plus basses. Nous allons donc diviser cet état de l'art en 2 parties, les solutions basées sur une navigation du drone à plus hautes altitudes et des solutions sur des navigations à basses altitudes.

3.1 Navigation à hautes altitudes

De nombreux articles traitent de la situation du drone, survolant à plusieurs dizaines de mètres dans le cadre de mission de recherche et de sauvetage.
Une des techniques utilisés par l'article [1] c'est d'utiliser des techniques de traitement d'images en appliquant des filtres pour détecter les personnes. L'idée serait de pouvoir détecter les gradients d'images. Les filtres permettent de convertir les images de l'espace RGB vers l'espace YCbCr, qui est mieux adapté pour la reconnaissance des hommes. L'article [2] effectue de la détection de personnes depuis les hautes altitudes et détecter une zone à côté de la personne afin de positionner le drone. Cette solution est utilisée dans le cadre des livraisons à domicile. Ce système utilise les coordonnées GPS couplées à des réseaux de neurones pour détecter une personne dans la zone de livraison Cette solution fut prototypée sur une Raspberry Pi3. La solution de l'intelligence artificielle a été aussi employée par l'article [3]. Dans cet article, on utilise un algorithme de traitement d'images basés sur des réseaux de neurones profonds appelés ScoreMap to ROI. Cette solution couplée à une assistance d'experts a fourni des résultats très intéressants.
La plupart de ses solutions basées sur de la navigation à haute altitude, nécessite que le drone soit capable d'effectuer une phase de descente afin de se positionner à une hauteur raisonnable (relativement égale à la taille de la victime), si on envisage le cadre de notre projet, qui est de permettre une communication et une assistance de la personne en détresse. Il est donc nécessaire de voir comment cette phase de descente peut être implantée.
L'article [4] proposent des méthodes pour effectuer un atterrissage du drone en se basant sur les informations visuelles fournies par la caméra du drone. Cette solution permet aux drones d'atterrir en se basant sur des marqueurs qui seront détectées depuis 30m de hauteur. La solution proposée, basée sur les estimations d'angles permet d'atterrir avec une erreur moindre (0,11m). Cette solution est interessante mais de notre point de vue il serait intéressant d'effectuer le processus de descente pas-à-pas en se basant sur les images acquises par la caméra.
La solution [2] utilise la détection en s'affranchissant de la contrainte du besoin d'un marqueur. Contrairement à [4], cette solution se base sur l'information des coordonnées GPS de la position de la personne et amorce la descente à une certaine distance de la personne détectée. Cette solution repose sur des algo-

rithmes embarqués de détection comme YOLO, SSD et de traitement d'images (OpenCV).

3.2 Navigation à basses altitudes

Pour ce qui concerne les solutions basées sur une navigation à basses altitudes, on a la solution de l'entreprise SII Group, qui permet de détecter une personne en détresse à partir d'une altitude relativement basse (2m). Cette solution est basée sur la librairie PoseNet [5]. Cette solution permet d'estimer la pose, c'est-à-dire détecter à l'aide des techniques de vision par ordinateur les formes, les figures, les personnes. Cet algorithme appliqué dans le cas des personnes, permet d'estimer où se trouver les points clés du corps humain.
La solution de détection qui reste la meilleure dans le cadre des détections d'objets est la solution YOLO (You Only Look Once) [6]. Cette architecture de réseaux de neurones convolutifs (24 couches) est très efficace pour de la détection en temps réel. A partir de la détection effectuée, on désirerait avoir une information permettant au drone de pouvoir se déplacer et/ou se positionner en face de la personne détectée.
Nous avons donc remarqué que certains articles arrivent à estimer la distance en se basant sur les boxes de détection prédite par les architectures comme YOLO. La solution DisNet, basée sur un réseau de neurones simple, [7] arrive à estimer la distance entre une caméra mono-oculaire et l'objet détecté.
L'article [8] a essayé d'améliorer la solution fournie par DisNet en se basant sur un extracteur de caractéristiques, une régression de distance et un classificateur pour estimer la distance à partir d'une image RGB. Cette solution réussie mieux là où la solution DisNet avait du mal notamment pour les objets qui sont très proches de la caméra.

4 Descriptif détaillée des solutions

Les solutions présentées dans les travaux relatifs au domaine nécessitent des moyens plus importants notamment pour les solutions impliquant la navigation à de hautes altitudes. Dans la suite du projet et afin de continuer dans la lancée des travaux effectués par la société SII dans le projet, nous allons passé de façon plus explicite en revue les méthodes impliquant de la navigation à basses altitudes. Les solutions de navigation à hautes altitudes pourront faire l'objet de futurs travaux.

4.1 YOLO

L'architecture YOLO est inspirée de l'architecture GoogLeNet [9] pour la classification d'images. Le réseau est constitué de 24 couches convolutionnelles suivie de 2 couches entièrement connectées. Mais contrairement à l'architecture GoogLeNet, il utilise une couche de réduction suivie par 3*3 couches convolutionnelles.

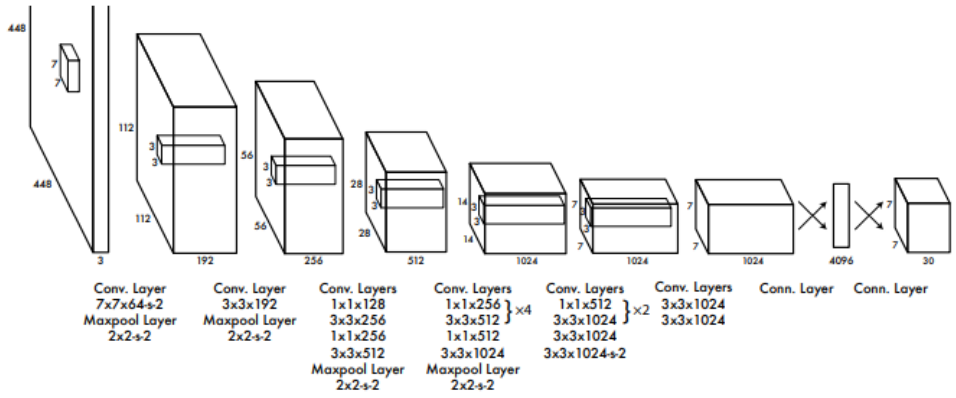


FIGURE 1 – YOLO Architecture

Le modèle YOLO a été pré-entraîné sur le dataset ImageNet[10] de 1000 classes pendant environ une semaine. Les résultats obtenus étaient de 88% pour la métrique *top-5 accuracy*. Ces résultats furent améliorés par l'utilisation d'un taux d'apprentissage décroissant, et en utilisant de l'augmentation de données et la technique du *Dropout*.
Le modèle YOLO s'avère très performant et très rapide en termes d'inférence parce que contrairement aux autres architectures de détection d'objets, il nécessite qu'une seule évaluation du réseau contrairement aux méthodes basées sur des classificateurs. L'utilisation de la technique de *Non-maximal suppression*, permet de corriger les détections multiples et apporte 2-3% de précision à l'architecture

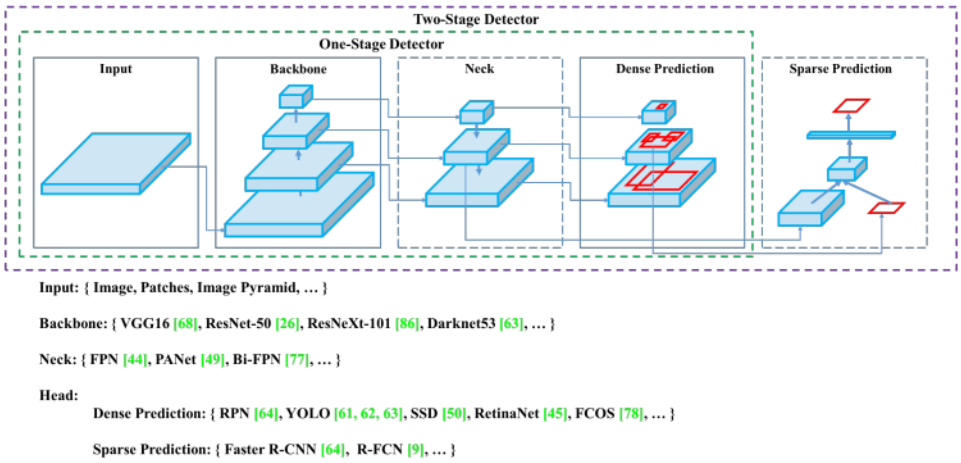


FIGURE 2 – Objet Detector

Cependant le modèle YOLO impose des contraintes spatiales qui limite le nombre d'objets avoisinants que le modèle peut détecter. Le modèle a ainsi du mal sur les petits objets qui apparaissent en groupe.

L'architecture YOLO comme la majorité des modèles de détections d'objets basés sur des réseaux de neurones convolutifs ne sont pas efficaces pour des applications en temps réel tant ils nécessitent une certaines puissances de calcul (GPU plus puissants). Dans l'optique d'améliorer les performances du modèle YOLO, le modèle YOLO v4 [11] fut conçue afin de répondre à ce problème et être de ce fait plus efficace pour ce qui est des applications dans les systèmes de production et minimiser les calculs parallèles. L'architecture YOLOv4 devient ainsi plus facile à entraîner et à tester.
Une architecture classique d'un modèle de détection d'objets est composée de différentes parties :

- **Input** : Images d'entrée
- **Backbones** : Architecture de base (généralement un réseau de neurones convolutifs),(VGG16[12],ResNet,CSPDarknet53[13]...)
- **Neck** : Couches de réseaux de neurones entre la "tête" et l'architecture de base du modèle utilisée pour récolter les caractéristiques des images des différentes étapes (PAN [14],FPN[15]...)
- **Heads** : architecture de réseaux de neurones permettant la détection soit en une étape (YOLOv3, SSD..) ou en deux étapes (Faster R-CNN[16], Mask R-CNN[17]...)

L'architecture YOLOv4 est constitué donc :
- Backbone : CSPDarknet53,

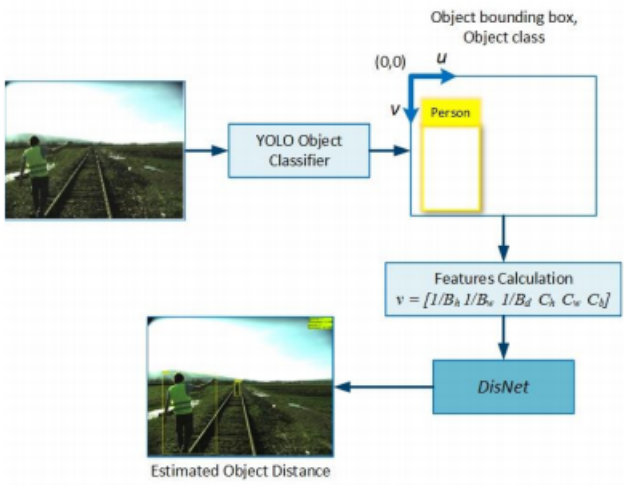


FIGURE 3 – Architecture DisNet

- Neck : SPP [18] (Spatial Pyramid Pooling), PAN (Path Aggregation Network)
- Head : YOLOv3

Afin de rendre cette architecture plus adaptée à l'entraînement et l'inférence sur une machine équipée d'un unique GPU, des améliorations furent adoptées. Ce fut entre autre une nouvelle méthode d'augmentation de données qui combine 4 images d'entraînement, la sélection des hyper-paramètres par des algorithmes génétiques et la modification de méthodes existantes comme la Cross mini Batch Normalization.

4.2 DisNet

La plupart des applications de détection d'objets en vision par ordinateur sont effectuées via des images venant de deux caméras stéréo. L'estimation des cartes de profondeur via une image reste difficile vu que l'information sur la profondeur des images reste ambiguë. La méthode DisNet permet d'estimer la distance entre objets en se basant sur une image sans aucune autre information sur la scène ou provenant des caméras.
L'architecture DisNet sur l'estimation de distance est basée sur l'état de l'art des modèles de détection d'objets, YOLO. Les boxes de détection prédites par YOLO, sont utilisés pour calculer les caractéristiques utilisées pour estimer la distance.

La méthode DisNet est une méthode d'apprentissage supervisé basé. Pour l'entraînement 2000 instances de boxes prédites par le modèle YOLO, collectées manuellement ont été utilisées. Ces boxes de différents objets enregistrés par des caméras RGB, sont associées à différentes distances mesurées par scanner laser entre la caméra et l'objet. Les objets utilisés dans le cadre de la méthode

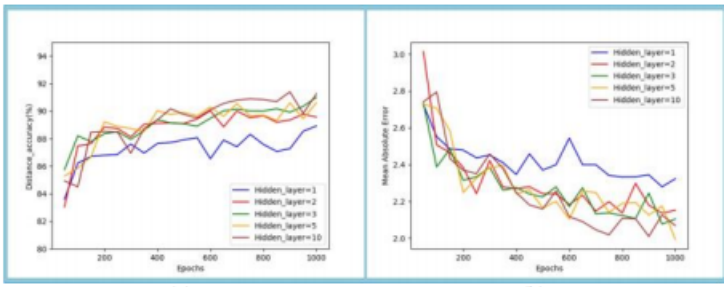
DisNet sont les objets que l'on est susceptible de rencontrer sur une scène de chemin de fer, notamment des personnes ce qui correspond à notre cas d'usage dans notre projet. Pour chaque box prédite pour un objet, un vecteur d'instance v est calculé avec :

$$v = [1/B_h, 1/B_w, 1/B_d, C_h, C_w, C_b]$$

- B_h : hauteur du cadre de sélection en pixels/image,
- B_w : largeur du cadre de sélection en pixels/image,
- B_d : diagonale du cadre de sélection en pixels/image,
- C_h : hauteur moyenne d'un objet d'une classe particulière,
- C_w : largeur moyenne d'un objet d'une classe particulière,
- C_b : épaisseur moyenne d'un objet d'une classe particulière.

Les valeurs C_h, C_w, C_b représentent les grandeurs réelles moyennes pour une classe. Par exemple pour la classe *person*, qui nous intéresse en particulier, $C_h = 175$cm, $C_w = 55$cm, $C_b = 30$cm. Ces informations permettront de mieux estimer la distance entre la caméra et l'objet en fonction des classes.

Le modèle de la méthode DisNet est composé de 3 couches de 100 neurones. Le nombre approprié de couches et de neurones fut déterminé par évaluation de l'erreur absolue moyenne (*Mean Absolute Error*) après entraînement du modèle sur 1000 époques en divisant l'ensemble de données en données d'entraînement (80%) et en données de test (20%). Ce modèle fut entraîné par retropropagation avec l'optimiseur Adam. Les expérimentations ont montré qu'augmenter le nombre de couches ou de neurones n'amélioraient pas les performances comme le montre les figures.



(a) (b)

Figure 5. (a) Distance Estimation Accuracy and (b) Mean Absolute Error achieved for different numbers of hidden layers

FIGURE 4 – Variation de la précision en fonction du nombre de couches

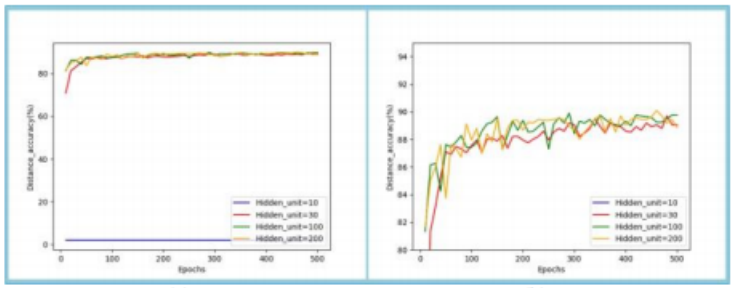


Figure 6. Distance Estimation Accuracy achieved for different number of hidden neurons per hidden layer in 3-hidden layers neural network DisNet

FIGURE 5 – Variation de la précision en fonction du nombre de neurones

Nous avons décidé de réaliser l'entraînement du modèle DisNet sur l'ensemble des données collectées manuellement dans le cadre de l'élaboration du modèle avec l'architecture TensorFlow. Les conditions de l'entraînement furent les mêmes que celles utilisées dans l'article [7]. Nous avons donc effectué l'entraînement sur 80% du dataset en 1000 époques, l'optimiseur Adam, un taux d'apprentissage $lr = 10^{-4}$. En utilisant comme fonction de coût l'erreur moyenne absolue, on obtient comme courbes d'apprentissage la figure suivante :

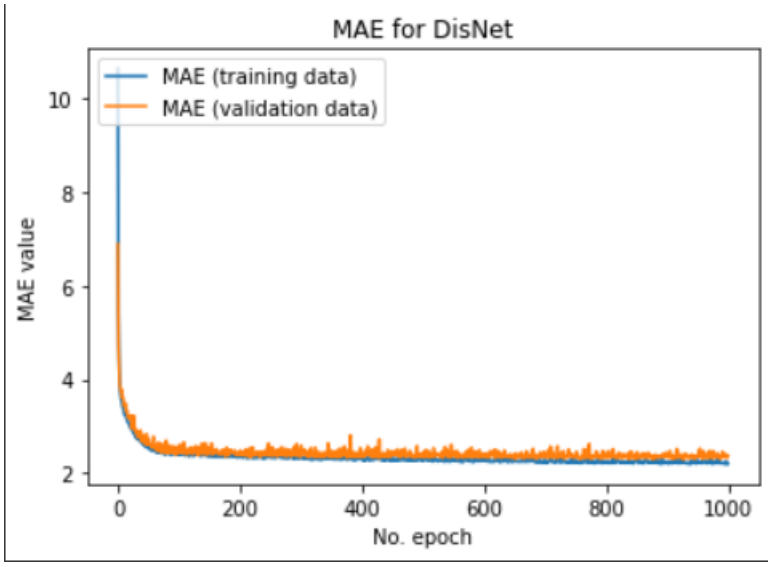


FIGURE 6 – Courbes d'apprentissage

Nous avons décidé d'analyser les erreurs de prédiction de notre modèle sur l'ensemble des données en traçant l'erreur absolue des prédictions sur l'ensemble

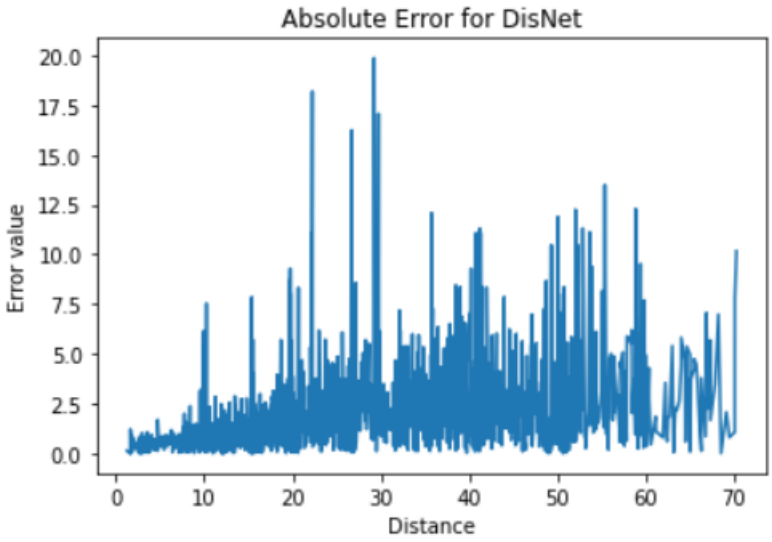


FIGURE 7 – Erreur absolue du modèle

des données disponible. On peut remarquer que l'erreur maximum du modèle est de l'ordre de 20m. Les erreurs varient en fonction de la distance réelle entre l'objet et la caméra.

4.3 Architecture améliorée pour la prédiction de distance

L'estimation de distance avec des méthodes comme celles de DisNet présente des limites. On peut noter entre autres que ces méthodes ont de mauvaises performances pour les objets localisés sur les côtés de la caméra ou les objets situés à plus de 40 mètres.
Dans le but de pouvoir compenser ces limites et aussi améliorer les performances dans l'estimation de distance, un modèle basé sur des réseaux de neurones convolutifs, qui permet d'estimer les distances à partir d'images RGB et des boxes prédites délimitant les objets et complété par un régresseur de point clé pour une meilleure estimation de la distance spécifique à l'objet.
L'architecture de base est composé de 3 composants :

- **Extracteur de caractéristiques** : Nous exploitons pour cela les architectures de réseaux convolutifs populaires (VGG16, ResNet50...).
- **Un régresseur de distance** : La carte de caractéristiques collectée à partir de l'extracteur est utilisée pour générée un vecteur qui sera passé au régresseur afin de prédire la distance. Le régresseur est constituée de 3 couches de réseaux entièrement connectées et d'une fonction d'activation(*SoftPlus*) afin de s'assurer que la distance prédite est positive.
- **Un classificateur multiclasse** : Pour le classificateur multiclasse, on

fait comme précédemment pour attribuer un label à l'objet prédit. Le classificateur est constitué d'une couche entièrement connectée complétée par une fonction d'activation (*SoftMax*).

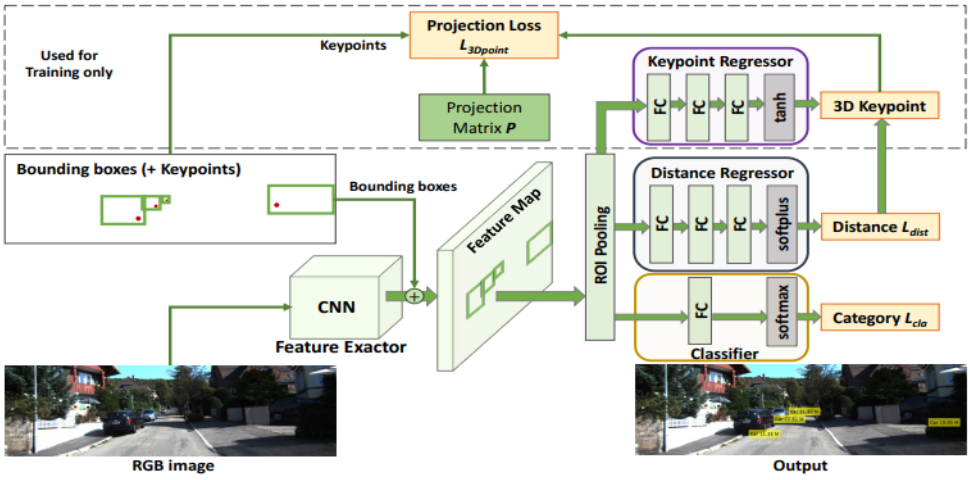


FIGURE 8 – Architecture

Les trois composants du modèle de base ont été entraîné simultanément avec une fonction de coût minimisant à la fois le classificateur multiclasse et le régresseur de distance. Le modèle a été entraîné avec l'optimiseur Adam avec un taux d'apprentissage $lr = 10^{-3}$ qui décroît exponentiellement après 10 époques. Le classificateur n'est utilisé que durant la phase d'entraînement. Utiliser le classificateur permet à notre modèle d'apprendre des caractéristiques (taille,forme) qui seront utilisés pour mieux estimer les distances.

L'architecture de base a elle seule ne suffit pas pour obtenir la précision désirée. Elle a donc été améliorée en utilisant un régresseur de point clé, pour optimiser les performances du modèle de base en ajoutant une contrainte de projection.

Le régresseur de point clé prédit une position approchée dans le repère 3D de la caméra. En utilisant la distance prédite par le régresseur de distance comme la coordonnée Z dans le repère 3D de la caméra, il n'aura plus qu'à prédire les coordonnéees (X, Y) à l'aide du régresseur de points clés. Le point 3D ainsi généré sera projeté dans l'espace de la caméra à l'aide de la matrice de projection de la caméra P. On aura plus qu'à minimiser l'erreur entre les véritables coordonnées (X, Y) et les coordonnées projetées. Afin d'améliorer la prédiction, la fonction de coût est complétée par un paramètre de régularisation afin d'ajouter une pénalisation.
Le modèle ainsi formé a été entraîné dans les mêmes conditions que l'architec-

ture de base sur la plateforme PyTorch sur une machine Xeon E5-2603 CPU et NVIDIA Tesla K80 GPU.
La principale difficulté dans cette solution aura été de constituer la base de données d'entraînement. Une base de données fut créé à partir des datasets KITTI[19] et nuScenes[20]. Ce dataset ne comporte que des images 2D ainsi que les boxes délimitants (2D et 3D) ; il manque donc l'information sur la distance. Cette information est donc générée par segmentation des points de l'objet à partir du nuage de point velodyne correspondant en utilisant ses paramètres de boxes 3D délimitants de l'objet. On trie ensuite tous les points segmentés en fonction de leurs valeurs de profondeur et enfin la n-ième valeur de profondeur de la liste triée est choisie comme la distance réelle associée à l'objet donné.

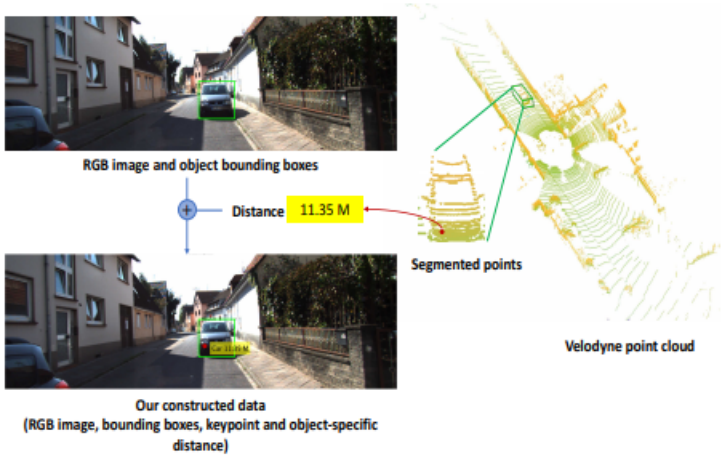


FIGURE 9 – Génération de la distance

Le modèle ainsi obtenu est évalué sur le dataset KITTI en utilisant comme métrique d'évaluation la racine carrée de l'erreur moyenne absolue ($RMSE$). On observe les performances de l'architecture de base et de l'architecture améliorée sur la figure ci-dessous.

$$RMSE = \sqrt{\frac{1}{N}\sum_{d\in N} \|d_i - d_i^*\|^2}$$

On peut remarquer que le modèle est plus performant sur les objets proches de la caméra mais que les performances se dégradent au fur et à mesure que la distance entre la caméra et les objets augmentent.

4.4 Assimilation de données : Filtre de Kalman Etendu

La solution précédente est très intéressante mais contrairement à la solution de DisNet nécessite d'avoir à disposition un dataset ou d'être à mesure de le

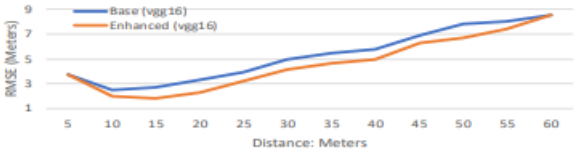


FIGURE 10 – RMSE des deux architectures

reconstituer. Face à cette difficulté, nous avons décidé de nous intéresser à la solution de l'assimilation de données.
Nous souhaitons avoir une bonne estimation de la distance entre le drone et les personnes en détresse pendant que le drone continue à avancer de façon autonome. Etant donné que nos informations sur la distance ne sont pas exactes, nous allons à chaque estimation de notre modèle corriger cette prévision en utilisant les équations du filtre de Kalman étendu

$$
\begin{aligned}
x^a &= x_B + K(y - Hx_B) \\
K &= BH^T(R + HBH^T)^{-1} \\
P^a &= (I - KH)P^f \\
x^f &= Mx^a \\
P^f &= MP^aM^T
\end{aligned}
$$

where :
- x_B : état apriori du système,
- K : matrice de gain,
- y : observation du système,
- H : matrice d'observation,
- B : matrice de covariance de l'état apriori du système,
- R : matrice de covariance du bruit
- P^a : matrice d'assimilation à l'étape d'analyse,
- P^f : matrice d'assimilation à l'étape de prédiction
- x^a : état du système à l'étape d'analyse,
- x^f : état du système à l'étape de prédiction.

Pour appliquer ces équations, nous avons besoin de déterminer l'écart-type B lié à la mesure de la distance par la méthode DisNet. Des expérimentations ont été réalisées afin de déterminer les bonnes valeurs de l'écart-type pour notre filtre d'assimilation.

Nous avons réalisé différentes régressions linéaires sur les erreurs du modèle DisNet sur l'ensemble de données à notre disposition afin de trouver les bons coefficients nous permettant d'avoir le bon écart-type en fonction de la distance d prédite. On arrive a approcher par la droite ci-après les valeurs des différents

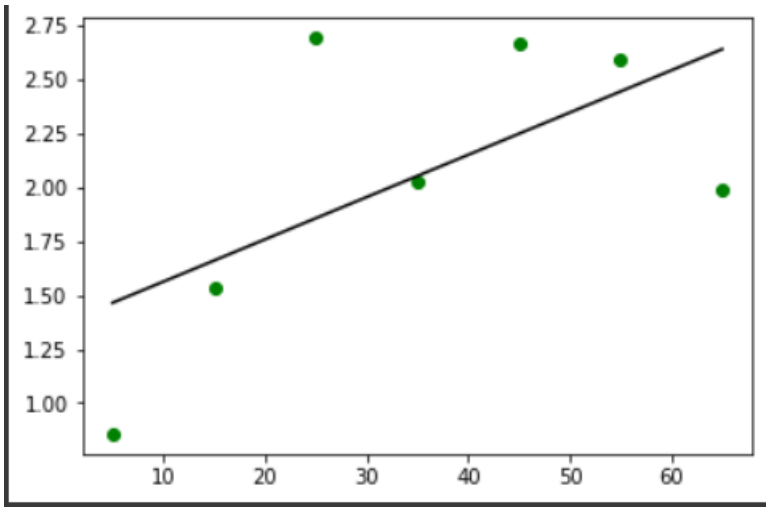


FIGURE 11 – Régression linéaire sur l'écart-type de l'assimilation

écart-types.

4.5 AirSim & Unreal Engine

En raison du contexte sanitaire, nous étions dans l'impossibilité d'effectuer les tests et les expérimentations sur de réels drones. Nous avons donc du envisager des plateformes permettant des simulations dans des environnements réalistes.
La plateforme AirSim [21] (Aerial Informatics and Robotics Simulation) est un environnement de simulation pour les drones ou les voitures construits sur la plateforme Unreal Engine [22] de Epic Games. Cette plateforme a été développée par Microsoft dans le cadre des expériences mêlant du Deep-Learning, de la vision par ordinateur et de l'apprentissage par renforcement. La plateforme AirSim nous permet d'avoir accès à tous les composants liés au fonctionnement du drone à savoir les capteurs, la caméra... ce qui permet de concevoir des algorithmes de commandes du drone notamment pour la navigation et la collecte d'informations provenant des différents capteurs et de la caméra. Toutes ces données seront utiles et nous permettront d'expérimenter et développer nos solutions.
Les environnements générés sont très réalistes comme le montre l'image ci-dessous. Nous avons ensuite cherché un moyen de rajouter à notre environnement de simulation des personnes à animer afin de simuler une personne en détresse. La solution à notre disposition est l'outil Mixamo (`https://www.mixamo.com/`).

FIGURE 12 – Environnement de simulation Unreal Engine & Mixamo

5 Limites & Futurs travaux

Les solutions présentées tout au long de cet article présentent un avantage pour la réalisation du projet DROOPI. Cependant quelques limites sont à relever.

5.1 Limites

Les solutions de détection d'objets comme YOLO souffrent de certaines anomalies de détection qui dépendent de la distance et des angles de dépression. Comme le montre l'article [23], les performances décroîent très rapidement quand la distance entre la caméra et l'objet devient très importante. De résultats semblables sont aussi visibles quand les angles de dépression induits par la hauteur et l'inclinaison de la caméra de par sa position sur le drone.

La méthode DisNet dans le cas de l'estimation de distance entre la caméra et des personnes supposent que la taille moyenne d'une personne est de 175 cm. Ainsi, pour des personnes de taille différentes, l'estimation de distance conduirait à des résultats différents. Une autre particularité de la méthode DisNet, est sa dépendance des boxes prédites par le modèle de détection d'objets, YOLO. La taille des boxes de détection varie en fonction de la distance mais aussi de la position de la personne. Pour une personne, dont les bras sont en l'air ou une personne qui ne se trouve pas en position debout, on obtiendra des boxes plus ou moins grands ce qui influencera la prédiction de la distance.

La solution de l'architecture améliorée de l'article [8] nécessite d'avoir un dataset à disposition avec une information sur la distance associée aux images et aux cartes de vérité terrain. La difficulté reste l'obtention de l'information sur la distance qui est obtenue par une estimation de nuages de points. La création d'une telle base de données s'avère fastidieuse et pas à notre portée.

5.2 Futurs travaux

Notre étude s'est limitée uniquement à la situation où le drone vole à basses altitudes. De futurs travaux s'avèrent nécessaires pour des situations de vols à hautes altitudes et dans les situations de vol de nuit ou par temps de mauvaise visibilité.
La méthode de l'apprentissage par renforcement qui s'avère très efficace pour les algorithmes d'apprentissage de navigation autonome est une solution aussi qu'il faudrait envisager. Ce sujet reste cependant encore très ouvert et de nombreuses recherches à ce sujet sont toujours en cours comme le montre l'article[24].
Dans le cadre d'avoir des résultats aussi pertinents pour le cas des drones, des solutions embarquées se doivent d'être développées. Les modèles de détection d'objet comme YOLO sont trop lourdes pour être déployées directement sur un drone. Des solutions comme YOLO Tiny, plus légères peuvent être utilisées mais

ont des performances plus limitées en termes de détection d'objet notamment si la distance entre la caméra du drone et la personne à détecter est de plus de 40 mètres. Les solutions d'apprentissage par renforcement sont plus adaptées pour une utilisation embarquée, mais nécessiterait des études plus approfondies.

6 Conclusion

Nous avons passé en revue les différentes techniques qui pourraient nous être utiles dans le cadre du projet DROOPI, notamment dans le domaine du positionnement d'un drone face à une personne en détresse. Nous avons notamment passé en revue l'état de l'art en terme de détection de personnes. La solution YOLO s'avère avoir les meilleures performances en termes de détection de personnes et en termes de rapidité d'inférence comme le montre l'amélioration de YOLO, YOLOv4 qui permet d'avoir des résultats très rapides et notamment effectuer du temps réel, ce qui est une performance importante pour notre solution de sauvetage et d'assistance à l'aide d'un drone.

La solution la plus réalisable pour le positionnement serait de se baser sur une estimation de la distance. Les méthodes d'estimation de la distance sont basées sur la méthode DisNet [7] qui arrive à estimer des distances à partir des boxes délimitant les objets prédites par les algorithmes de détection d'objets. Cette méthode s'avère efficace mais présente des lacunes pour les personnes situées proches de la caméra ou dans des angles de vision extrêmes.

Une solution pour y remédier serait d'utiliser l'architecture améliorée de l'article [8] qui arrive à compenser ces limites par des méthodes de projection. Le principal inconvénient de cette méthode reste la constitution d'une base de données assez importante. Nous nous sommes donc intéressés à la méthode de l'assimilation de données. La méthode du filtre de Kalman étendu qui permet l'application dans les cas non-linéaires, nous permet de corriger l'erreur lors de l'application de la méthode DisNet afin d'avoir de meilleurs résultats.
Les différentes plateformes ou outils de simulation existants pour effectuer les expérimentations dans le cadre de ce projet sont la bibliothèque AirSim et le logiciel Unreal Engine qui offre des algorithmes et interfaces de simulations réalistiques pour des applications du Deep-Learning ou de navigation autonomes. De futurs travaux, notamment dans l'étude des solutions par apprentissage par renforcement, plus adaptée pour les solutions de navigation autonome.

Références

[1] Jingxuan Sun, Boyang Li, Yifan Jiang, and Chih-yung Wen. A camera-based target detection and positioning uav system for search and rescue (sar) purposes. *Sensors*, 16(11), 2016.

[2] Safadinho David, Ramos João, Ribeiro Roberto, Filipe Vítor, Barroso João, and Pereira António. Uav landing using computer vision techniques for human detection. *Sensors*, 20(3), 2020.

[3] Gotovac Sven, Zelenika Danijel, Marušić Željko, and Božić Štulić Dunja. Visual-based person detection for search-and-rescue with uas : Humans vs. machine learning algorithm. *Remote Sensing*, 12(20), 2020.

[4] Jamie Wubben, Francisco Fabra, Carlos T. Calafate, Tomasz Krzeszowski, Johann M. Marquez-Barja, Juan-Carlos Cano, and Pietro Manzoni. Accurate landing of unmanned aerial vehicles using ground pattern recognition. *Electronics*, 8(12), 2019.

[5] S. Yu, B. Park, and J. Jeong. Posnet : 4x video frame interpolation using position-specific flow. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3503–3511, 2019.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection, 2016.

[7] Muhammad AbdulHaseeb, JianyuGuan, Danijela Ristić-Durrant, and Axel Gräser. Disnet : A novel method for distance estimation from monocular-camera. 2018.

[8] Jing Zhu, Yi Fang, Husam Abu-Haimed, Kuo-Chin Lien, Dongdong Fu, and Junli Gu. Learning object-specific distance from a monocular image, 2019.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[11] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4 : Optimal speed and accuracy of object detection, 2020.

[12] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection, 2015.

[13] Vishal Mandal, Abdul Rashid Mussah, and Yaw Adu-Gyamfi. Deep learning frameworks for pavement distress classification : A comparative analysis, 2020.

[14] Chaoyue Wang, Chang Xu, Chaohui Wang, and Dacheng Tao. Perceptual adversarial networks for image-to-image transformation. *IEEE Transactions on Image Processing*, 27(8) :4066–4079, Aug 2018.

[15] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6) :1137–1149, 2017.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science*, page 346–361, 2014.

[19] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[20] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes : A multimodal dataset for autonomous driving, 2020.

[21] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim : High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[22] Epic Games. Unreal engine.

[23] Hwai-Jung Hsu and Kuan-Ta Chen. Face recognition on drones : Issues and limitations. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, DroNet '15, page 39–44, New York, NY, USA, 2015. Association for Computing Machinery.

[24] Guillem Muñoz, Cristina Barrado, Ender Çetin, and Esther Salami. Deep reinforcement learning for drone delivery. *Drones*, 3(3), 2019.

[25] A. Carrio, J. Tordesillas, S. Vemprala, S. Saripalli, P. Campoy, and J. P. How. Onboard detection and localization of drones using depth maps. *IEEE Access*, 8 :30480–30490, 2020.

[26] Fatih Gökçe, Göktürk Üçoluk, Erol Şahin, and Sinan Kalkan. Vision-based detection and distance estimation of micro unmanned aerial vehicles. *Sensors*, 15(9) :23805–23846, 2015.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch : An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[28] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow : A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.