

INTRODUCTION AU TRAITEMENT DES IMAGES

PROJET IMAGE

Détection de drones en temps réel dans une vidéo à fond statique à l'aide d'un réseau de neurones convolutionnel

Étudiant :

CARRERA Hanus Alex

21211797

M1 ROB – UM4RBI11

Projet Image

30 octobre 2025

Résumé

Ce projet reprend et étend la méthode présentée dans l'article *Real-Time and Accurate Drone Detection in a Video with a Static Background*[1]. L'approche repose sur deux étapes : la détection d'objets en mouvement par soustraction de fond, puis leur classification en drone, oiseau ou arrière-plan via un réseau MobileNetV2[2]. La démo est implémentée en Python avec OpenCV[3] et PyTorch[4]. La *pipeline* a été complétée avec un classifieur utilisant également MobileNetV2[2], entraîné et testé avec des données issues du dataset Drone-detection-dataset[5]

1 Introduction et contexte

Depuis quelques années, les drones sont devenus des outils incontournables dans de nombreux domaines. Leur accessibilité, leur maniabilité et leur faible coût ont largement contribué à leur diffusion. Cependant, cette popularité s'accompagne également de **risques de sécurité importants**.

Il est donc nécessaire de développer des systèmes capables de détecter et d'identifier rapidement ces drones. Parmi les nombreuses approches existantes (radar, capteurs acoustiques, signaux radio-fréquences), la vision par ordinateur offre une solution particulièrement intéressante grâce à son coût réduit et à sa capacité à identifier visuellement la nature d'un objet. Toutefois, cette tâche reste difficile : les drones sont souvent petits, rapides, et visuellement similaires aux oiseaux.

Le but de ce travail est de reproduire les méthodes utilisées dans l'article **Real-Time and Accurate Drone Detection in a Video with a Static Background**[1].

2 Chaîne de traitement proposée

Notre article utilise une méthode simple à mettre en œuvre, abordable (comparée au radar, par exemple), qui est silencieuse, n'émet pas d'ondes ni de signaux, et qui peut être déployée *hors ligne* n'importe où, à condition d'avoir une caméra statique et un ordinateur avec le classifieur connecté.

Voici la chaîne de traitement qui est proposée dans l'article.

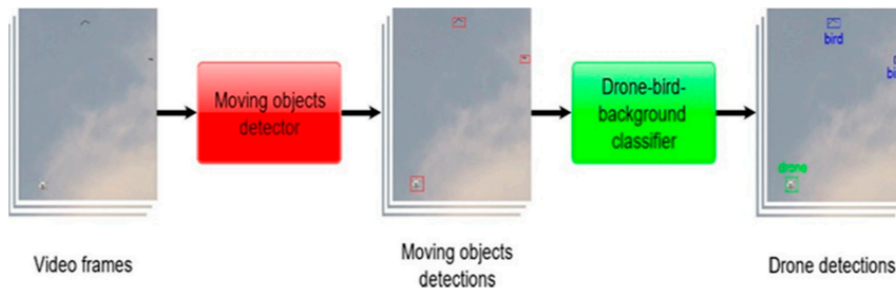


FIGURE 1 – Chaîne de traitement proposée[1]

Nous regarderons tout d'abord la partie de traitement de données qui permettra de générer de Régions d'Intérêt (*Regions of Interest* - (*ROI*) en anglais) et ensuite présenterons le classifieur qui permet de déterminer si la *ROI* s'agit d'un oiseau, drone ou arrière plan.

2.1 Détection d'objets en mouvement

Puisque la classification est un calcul qui peut être coûteux et lent, le but de notre *pipeline* génératrice de *ROI* est d'alléger le travail du classifieur en *réduisant la taille de la zone à classifier*. Au lieu de regarder dans **toute l'image**, nous n'allons que regarder à des endroits très spécifiques, aux *ROI*.

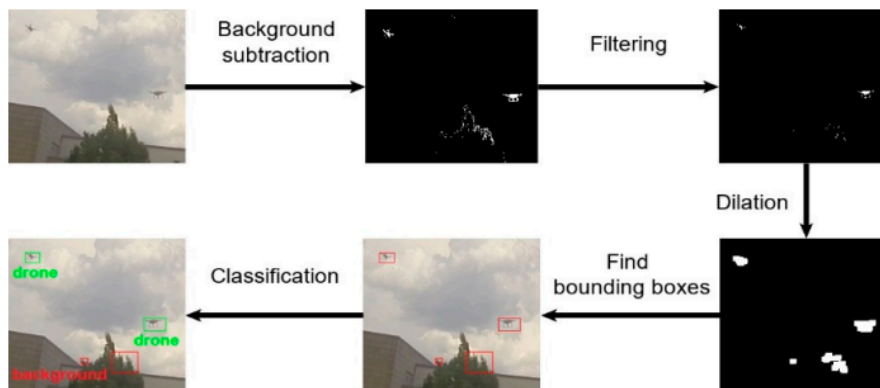


FIGURE 2 – Toutes les étapes de l'algorithme de détection de drones proposé[1]

Traduction :

Soustraction de fond > Filtrage > Dilatation > Trouver boîtes englobantes > Classification

Voici la sous-chaîne proposée par l'article, qui traite les images afin de trouver les *ROI*.

Nous mentionnerons également quelques petites améliorations proposées par la suite.

Détection de mouvement par soustraction de fond

Dans l'article, plusieurs méthodes sont mentionnées en ce qui concerne la détection de mouvement, comme le *optical flow*, le *edge detection* [...] [1], mais explique en citant [6] en quoi elles ne seraient pas adaptées à un scénario réel. La méthode la plus adaptée et qui donnerait les meilleurs résultats serait donc celle de mouvement par soustraction de fond. Cette méthode est exploitable uniquement dans le cas d'une caméra statique, puisque tout mouvement de la caméra ou mise au point serait faussement détecté comme mouvement d'un objet (tout l'écran serait un "objet en mouvement"). Nous proposerons par la suite des solutions pour des micro-mouvements/changements de mise au point possibles.

La soustraction de fond est une méthode classique et efficace dont le principe repose sur la comparaison entre l'image à l'instant actuel et une image de référence de fond.

Notre image de référence est un modèle statistique du fond. Les montagnes ne bougeant pas, si jamais un pixel à l'endroit de la montagne varie fortement (intensité) à cause de la présence d'un drone, alors nous verrons que cette différence sera importante

Mathématiquement, cela peut être écrit comme :

$$\text{mouvement} = \text{Vrai} \quad \text{si } |I_{(t)}(x, y) - M(x, y)| > \text{seuil} \quad \text{sinon Faux}$$

Où $I_{(t)}(x, y)$ est l'intensité lumineuse du pixel en x, y à l'instant t , $M(x, y)$ la valeur moyenne et seuil un seuil¹

La moyenne est calculée sur une fenêtre. Cela permet de faire face aux phénomènes tels que les changements d'illumination. Plus la fenêtre est large, plus elle va agir comme un filtre passe-bas et plus nous aurons un modèle précis. Cela a également un inconvénient : nous ne détectons que des variations, mais que se passe-t-il si notre objet s'arrête pendant quelques secondes ? Nous proposerons une solution à cette problématique par la suite.

Ensuite, si un pixel change beaucoup entre les deux images, cela signifie qu'un objet s'est déplacé à cet endroit, et nous allons avoir un "1" logique à cet endroit (blanc pour une image monochromatique). Sinon, on considère qu'il fait partie du fond statique, et on lui associe le "0" (noir).

Dans notre cas, nous avons utilisé OpenCV[3] et la méthode `cv2.createBackgroundSubtractorMOG2()` qui permet d'extraire les zones en mouvement.

Filtrage

Ce modèle est très sensible à la valeur du seuil et aux petites perturbations, telles que le mouvement de l'herbe causé par le drone par exemple. Nous aurons donc beaucoup de bruit. L'auteur de l'article[5] propose de filtrer ces données mais ne détaille pas comment.

Nous proposons d'appliquer un simple filtre gaussien. Ainsi, les pixels activés individuels (en "1") seront mis à zéro et ne resteront que les grandes variations/regroupement de pixels.

Dilatation

Afin de connecter nos groupes de pixels ensembles et de réduire le nombre de *ROI*, l'auteur[1] propose de réaliser plusieurs opérations de dilatation. Le résultat final est l'apparition de *blobs*/tâches. Voici un exemple concret de notre démo. Ce phénomène est également visible dans 2.

1. mentionné plus tard



FIGURE 3 – Résultat dilatation et boîte englobantes

Trouver boîtes englobantes

Une fois nous récupérons les tâches, la prochaine étape consiste à trouver des **carrés** tel que celles-ci sont englobées. Ces carrés sont nos *ROI*. Plus il y a de tâches, plus il y a de *ROI* à classifier, d'où l'importance du filtrage et la dilatation. La raison pour laquelle il faut des *ROI* carrées et non rectangulaires est parce que notre classifieur s'attend à des images carrées. La dernière étape avant classification est celle du redimensionnement. Ces boîtes sont visibles dans la figure 5, en rouge.

3 Améliorations

Nous venons de résumer la *pipeline* proposée par l'article. Voici quelques améliorations que nous avons proposées afin de remédier à certains problèmes récurrents.

3.1 Disparition des ROI

Nous avons vu que si notre objet reste statique pendant une durée suffisamment importante telle que le nouveau modèle du fond inclut l'objet (par exemple si le drone fait un vol stationnaire), alors nous allons *perdre de vue* l'objet. Nous avons certaines fois également des cas où d'un *frame* à un autre les variations ne sont pas très importantes et donc nous n'aurons pas de *ROI* à ce moment, ce qui est dérangeant.

Time-out

Notre première solution est, dans le cas où notre *ROI* disparaît spontanément, de la laisser là où elle était pendant *timeout frames*. Plus cette valeur est grande, plus on laissera du temps à la *pipeline* pour redétecter l'objet en mouvement. La valeur de ce *time out* est de l'ordre de 3-5 frames, ce qui à l'oeil nu est quasiment imperceptible.

Distance et âge des ROI

Nous avons remarqué que certaines fois, plusieurs *ROI* étaient générées les unes à côtés des autres, ou que certaines fois, nous avions des *ROI* qui apparaissaient pendant quelques fractions de secondes (mouvement d'un arbre, herbe...). Ceci a un impact sur la cadence (*frame rate*).

Ainsi, nous avons un seuil de distances pour les *ROI*. Les *ROI* plus jeunes vont être absorbées par celles qui sont plus âgées, l'idée étant de réduire au maximum le nombre de *ROI*. Cette méthode a un très bon avantage : lorsque la caméra bouge, le système voit sa cadence réduite d'un facteur 10 mais uniquement pendant quelques fractions de secondes, sans perdre de vue notre objet initial. Nous pourrions également choisir d'envoyer au classifieur uniquement les *ROI* âgées de plus de certains *frames*, mais ceci ne semble pas nécessaire à notre échelle.

3.2 Multidéttection ou monodéttection

Certaines fois, les objets seront très lointant. Il faudra donc adapter nos seuils. Malheureusement, ceci aura un impact sur le bruit et peut rendre notre *pipeline* instable. Un mode *mono* est proposé. Il adapte les seuils et ne prend en compte que la *ROI* plus grande et âgée. Une autre alternative est de modifier nos seuils dynamiquement, tant que nous n'avons pas de *ROI*... Dans notre cas le mode *multi* a une performance satisfaisante, mais il est vrai que pour les objets lointants, il a tendance à les perdre de vue pendant quelques secondes.

4 Classifieur

Pour le classifieur, nous reprenons le même que celui de l'article, à savoir le CNN MobileNetV2[2]. Il s'agirait du meilleur compromis entre performance et facilité à l'entraîner. Le CNN aura pour but classifier 3 classes : **1. Drone**, **2. Oiseau**, **3. Fond** (le reste). Les auteurs ont entraîné ce modèle *from scratch* (depuis zéro) et utilisé plusieurs datasets, pour avoir :

- 10 155 images de drones
- $1921 + 2651 = 4572$ images de oiseaux
- 9348 d'arrière plan

Pour des raisons pratiques et surtout par contrainte de temps et puissance de calcul², contrairement aux auteurs, nous avons décidé de faire du *transfer-learning* et d'utiliser un seul dataset [5].

Nos données étaient tirées de plusieurs vidéos statiques des drones et oiseau (dont les vérités terrains étaient annotée auparavant). Un paramètre permettait de réduire le nombre de données en ne prenant que toutes les n frames (sinon temps de training extrêmement important). Ceci est pertinent puisque les variations entre deux *frames* proches seront minimales. De plus, pour extraire des images pour le "fond", nous cherchions de façon aléatoire dans l'image, de façon à ce que nous soyons hors-vérité terrain (si la vérité terrain se trouve à droite de l'image, typiquement, le "fond" choisi sera à gauche).

Le nombre d'images de fond est généralement plus important, puisque le fond peut être très variant (ciel, mur, arbre...).

- **Drone** : 8961
- **Oiseau** : 3807
- **Fond** : 12710

Même si le nombre d'exemples est important, il faut considérer que les variations sont dans certains cas très petites. Le nombre de données *utiles* est donc bien moins important.³

Pour les essais vidéos, 3 vidéos de oiseaux et 3 videos de drones furent isolées lors de l'apprentissage.

Nous avons ajouté l'affichage de matrice de confusion, mais malheureusement nous n'avons sauvegardé que l'avant dernière version.

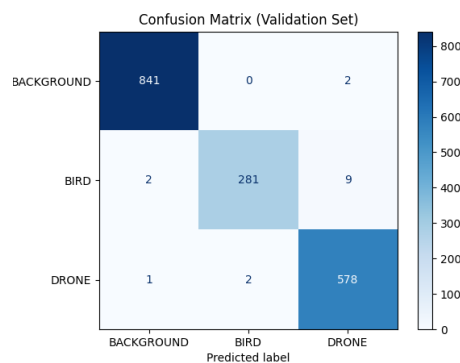


FIGURE 4 – Ancienne Matrice de confusion

2. Tensorflow n'est plus compatible avec CUDA pour les nouvelles versions sur Windows et donc pas de GPU

3. Ceci aura un impact sur la classification

Cette version utilisait moins de données et n'isolait pas les videos test.

Cette matrice semblerait indiquer que notre classifieur aurait une très bonne performance, mais en pratique ceci est uniquement vrai dans le cas où le drone/oiseau est clairement loin d'un fond (forêt, montagne). Nos vidéos de test ont été choisies pour mettre en évidence cette problématique. En effet, dès que le drone s'approche d'un fond, alors la détection bascule de *drone* à *fond*. L'explication la plus probable est le fait que la classe *fond* est bien plus présente dans la base d'entraînement et que nous n'avons pas assez de données.

Certaines fois, d'une *frame* à un autre, notre détection basculé temporellement de *drone* à *oiseau* par exemple. Pour remédier à ce problème et avoir une sorte de filtre passe-bas de classes, nous moyennons notre classe sur une fenêtre de 5 *frames*. Ceci permet d'éviter les sauts et rend la *pipeline* bien plus robuste.

Ceci dit, relativement parlant, la performance globale, vu les contraintes de temps est plutôt satisfaisante. Nous avons des détections stables et correctes, sauf dans les cas spécifiques.

Cette problématique de fond est également abordée par les auteurs de l'article[5]. Lorsque l'objet vole près d'un autre objet en mouvement (de l'herbe en mouvement qui serait du *fond*), alors la détection est fortement détériorée.

5 Résultats

Les vidéos résultantes sont postées dans un drive, dont le lien est référence dans le GitHub du projet[7], dans le Readme.md.

12 vidéos furent générées. On *run* la *pipeline* avec nos 6 vidéos (3 pour drone et 3 pour oiseau) dans les 2 modes, *mono* et *multi*. Nous constatons que la performance entre *mono* et *multi* n'est pas très différente, il aurait été sensé être plus agressifs avec nos paramètres (en effet certaines fois nous perdons les objets lointant, même en mode *mono*). Ceci dit il y a quand même une amélioration, notamment pour la vidéo **V_BIRD_045**.

La vidéo la plus intéressante est **MULTI_V_BIRD_019.mp4**. On retrouve deux oiseaux, une caméra qui bouge, dont la mise au point varie mais on arrive quand même à voir des résultats satisfaisant et à suivre l'oiseau.

MULTI_V_DRONE_027 présente également cette remise au point qui auparavant, *avant* nos "améliorations" stagné le *frame rate* et déstabilisé nos détections.

Désormais, notre *pipeline* est bien stable et fortement moins impactée par les perturbations extérieures

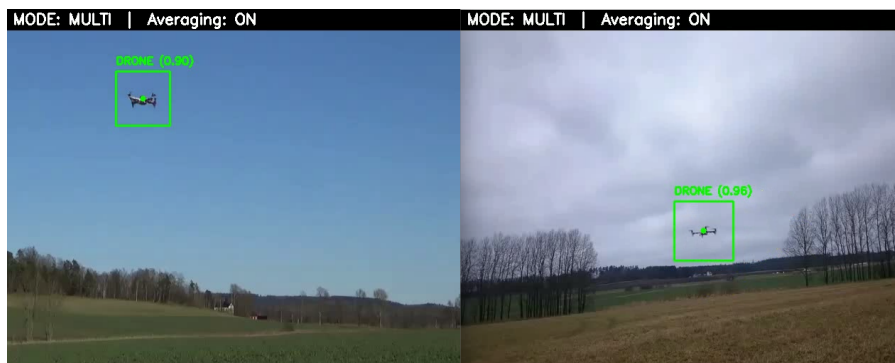


FIGURE 5 – Résultats de classification

Une dernière vidéo a été ajoutée au drive (**main.mp4**) qui montre en temps réel également les tâches.

6 Conclusion

Ce travail a permis de concevoir et d'implémenter un système complet de détection de drones en temps réel, inspiré de la méthode proposée par Seidaliyeva et al. (2020)[1]. L'approche repose sur une chaîne de traitement modulaire combinant la soustraction de fond pour la détection du mouvement

et l'utilisation d'un réseau MobileNetV2[2] pour la classification des objets en trois catégories : drone, oiseau et arrière-plan.

L'implémentation a été réalisée en Python, à l'aide des bibliothèques OpenCV[3] et PyTorch[4], avec une attention particulière portée à la simplicité, la robustesse et la performance en temps réel. Plusieurs améliorations ont été ajoutées par rapport à la méthode d'origine, notamment un mode multi-objet, la suppression des régions qui se chevauchent.

Les résultats obtenus montrent que ce type de pipeline léger et modulaire permet d'obtenir une détection rapide et suffisamment précise, tout en restant facilement adaptable à différents environnements. Cependant, certaines limites persistent, notamment la sensibilité à la lumière variable, aux mouvements d'arrière-plan et aux objets de petite taille.

Pour la suite, plusieurs pistes d'amélioration sont envisageables : intégrer un suivi temporel plus avancé, utiliser des modèles de deep learning plus récents comme YOLOv8[8] pour la détection directe, ou encore combiner la vision avec d'autres capteurs (acoustiques, radar ou infrarouge) pour renforcer la fiabilité du système.

En conclusion, ce projet démontre qu'il est possible de concevoir un système de détection de drones simple, efficace et réutilisable, tout en s'appuyant sur des outils open source et une architecture adaptable aux besoins réels du terrain.

Bibliographie

- [1] U. S. et al, “Real-time and accurate drone detection in a video with a static background,” *Sensors*, 2020.
- [2] Google AI Vision Team, “Mobilenetv2.” <https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet>.
- [3] OpenCV Team, “Opencv : Open source computer vision library.” <https://opencv.org/>, 2024.
- [4] PyTorch Foundation, “Pytorch : An open source machine learning framework.” <https://pytorch.org/>, 2024.
- [5] F. Svanström, “Dronedetectionthesis/drone-detection-dataset : First release,” May 2020.
- [6] S. T. Arunachalam, R. Shahana, R. Vijayasri, and T. Kavitha, “Flying object detection and classification using deep neural networks,” *International Journal of Engineering Trends and Technology (IJETT)*, vol. 67, 2019.
- [7] A. Carrera, “Imageprojet : Drone and bird detection pipeline in python.” <https://github.com/alexcrerra/ImageProjet>, 2025.
- [8] Ultralytics, “Ultralytics yolov8.” <https://github.com/ultralytics/ultralytics>, 2023. Accessed November 2025.