Titre: Title:	Perchage automatique de drones basé sur la vision artificielle
Auteur: Author:	Louison Greffier
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Greffier, L. (2019). Perchage automatique de drones basé sur la vision artificielle [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4056/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Perchage automatique de drones basé sur la vision artificielle

LOUISON GREFFIER
Département de génie mécanique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie mécanique

Octobre 2019

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Perchage automatique de drones basé sur la vision artificielle

présenté par **Louison GREFFIER**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Marek BALAZINSKI, président
Sofiane ACHICHE, membre et directeur de recherche
Maxime RAISON, membre et codirecteur de recherche
Abolfazl MOHEBBI, membre

DÉDICACE

À ma famille pour leur grand soutien moral et leur présence tout au long de mon projet de recherche.

À ma copine pour avoir été présente dans tous mes projets et surtout pour m'avoir soutenu dans les moments difficiles.

Aux carichoux pour tous ces bons moments passés au Canada, pour leur soutien moral et leur amusement parce que j'adore la rigolade !

À mes amis de l'INSA de Lyon pour ces Skypes pas piqué des hannetons et les meilleures soirées raclettes qui soient.

Aux sportifs pour les belles parties de wallyball et de badminton.

REMERCIEMENTS

Je remercie mon directeur Sofiane Achiche et mon codirecteur Maxime Raison de m'avoir permis de réaliser ce beau projet de recherche et de m'avoir montré comment acquérir une démarche scientifique.

Je remercie également M. Marek Balazinski et M. Abolfazl Mohebbi d'avoir accepté de participer à mon jury de maîtrise recherche.

Merci aussi à tous mes collèges du laboratoire A-530 pour leur soutien et pour les discussions fructueuses et/ou plaisantes. Je tiens particulièrement à remercier mon collège Dominique Beaini de m'avoir partagé ses connaissances dans le domaine de l'intelligence artificielle qui m'ont aidé à faire avancer mon projet de recherche.

J'en viens enfin à remercier les membres de MecaSupport qui m'ont permis d'avoir accès aux logiciels nécessaires au bon déroulement de mon travail.

RÉSUMÉ

L'utilisation de l'intelligence artificielle et de la vision se développe considérablement dans l'industrie des drones, notamment pour la saisie ou le dépôt d'objets ou encore pour l'atterrissage. Le perchage de drone, étroitement lié à ces tâches, commence également à se développer. Cette faculté permettrait aux drones de réaliser de nouvelles tâches mais aussi de combler leurs inconvénients tels que leur faible durée de vol ou le fait qu'ils soient fragiles. Par exemple, cela assurerait à un drone de se poser en cas de fin de batterie ou de mauvaises conditions climatiques. Les techniques actuelles de perchage ou de saisie d'objet effectuées à partir de la vision artificielle et d'un système de préhension ajouté au drone, se basent seulement sur la détection d'objets. Les objets/supports, lors des tests, sont sélectionnés en avance par les chercheurs afin d'avoir une bonne concordance avec le préhenseur. Ainsi, dans le cas où le support possède une forme complexe ou encore des dimensions trop différentes par rapport à celles du préhenseur, le drone le détectera et essayera de s'y percher sans succès.

L'objectif de cette maîtrise recherche est de développer un système de détection d'objets, par vision par ordinateur, qui selon les caractéristiques du préhenseur du drone, détecte les objets, leur attribue un score de concordance et renvoie le support idéal. Le score de concordance, que nous avons établi et nommé "CSP" (Concordance Support-Préhenseur) dans ce mémoire, se détermine à partir de la comparaison entre l'intervalle d'ouverture du préhenseur et les dimensions réelles des objets. La réalisation d'une comparaison ainsi que l'utilisation d'un score de concordance pour la détermination d'un support adéquat est une procédure que nous avons élaborée.

Partant d'un algorithme de détection basé sur la classification et la segmentation des objets détectés, la solution proposée dans ce mémoire a été développée en 3 phases : 1. Entrainement supervisé du réseau de neurones de l'algorithme de détection pour de nouvelles classes d'objets adaptées au perchage de drone. 2. Conception d'un algorithme, permettant de réaliser une comparaison entre les dimensions des objets détectés et celles du système de préhension dans le but de déterminer le support idéal pour le perchage du drone. 3. Évaluation des performances du modèle de détection complet, regroupant les deux algorithmes, à partir de tests réalisés sur un ensemble d'objets avec des paramètres et des conditions d'environnement différents entre chaque test.

Cette évaluation a démontré la précision et la fiabilité de notre système dans la détermination

du support idéal à partir de la vision artificielle. Nous avons effectué un ensemble de 36 tests avec pour chaque test un paramétrage différent. De plus, chaque test a été effectué avec un nombre de 10 répétitions soit avec une configuration des paramètres identique dans le but d'avoir des résultats plus fiables. Ces tests ont été réalisés avec une caméra ayant une résolution 640×480 et une carte graphique GTX 1080Ti (GPU). La performance globale de notre système obtient un taux de succès de **84.17 %** dans la détermination du support idéal. Ce taux de succès atteint même les **100 %** dans le cas d'une différence de diamètre d'au moins 20 mm entre les objets détectés. La vitesse d'exécution du modèle, dans le cas d'un seul objet détecté par image, prend en moyenne **0.42** seconde pour l'analyse d'une image, ce qui correspond à **2.38** fps. Le fait d'obtenir un temps d'exécution efficace dans le traitement de chaque objet permet ainsi de garder une fluidité dans la détection lors du déplacement de la caméra et ainsi d'éviter les mauvaises détections et/ou mauvaises segmentations.

ABSTRACT

The use of artificial vision and artificial intelligence in drones applications is experiencing rapid growth, particularly in pick and drop applications but also in drone landing. Drone perching, being closely linked to applications stated above, has also begun to emerge. This research would allow the drones to realize new tasks but also to mitigate their disadvantages such as their short flight time or their fragility. Current perching and object grasping approaches are carried out by utilizing artificial vision and robotic grippers integrated with the drones. The target objects or their handles are chosen in a way to match the geometry of the gripper. Thus, in case where supports have complex form or dimensions too differents in comparison with gripper dimensions, perching won't be avaiblable.

The purpose of this study is to develop a detection system that uses computer vision to return ideal support for drone perching. The ideal support corresponds to the object with the best matching score. The matching score, that we have developed and called "CSP" (Concordance Support-Préhenseur) in this study, is determined from the comparison between the gripper opening and the real object dimension. The implementation of this comparison as well as the utilization of matching score to the determination of the best support is a method that we created.

The suggested solution in this study is based on a detection algorithm which uses the classification and segmentation of objects. The several stages of the uses method are enumerated as follows : 1. Supervised training of neural network detection algorithm for new objects classes adapted to drone perching. 2. Conception of an algorithm, allowing to realize a comparison between object dimensions and grasping system dimensions in order to determine the ideal support. 3. Evaluation of the global system performances thanks to tests performed on a set of objects.

This evaluation has shown the precision and the reliability of our system in the ideal support determination based on artificial vision. We used a set of 36 tests with, for each test, different parameter values and different environmental conditions. Moreover, each test has been performed with a number of 10 repetitions for a given configuration in the purpose to obtain more reliable results. These tests have been made using a camera with a resolution of 640×480 and a graphic processor unit GTX 1080Ti (GPU). The global performance of our system obtains a success rate of **84.17 %** in the ideal support determination. This success

rate even reaches **100 %** in the case where the difference between the diameter of each object is at least 20 mm. The execution speed of this model, with only one detected object by image, takes on average **0.42** second for the analysis of one image, which corresponds to **2.38** fps. To maintain fluidity and thus to avoid wrong detections and/or segmentations during camera motion, an efficient execution time is needed in the treatment of each object.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

Sigle :	Signification :
AP	Average Precision
CNN	Convolutional Neural Network
CSP	Concordance Support-Préhenseur
DCOD	Distance Caméra-Objet de Détection
FAEO	Facteur Appartenance à un Élément de l'Objet
FC	Facteur de Calibration
FCN	Fully Convolutional Network
FCRP	Facteur de Conversion Réalité-Pixels
FD	Facteur de Distance
FDEP	Facteur de Distance Entre Pixels
FLOPS	Floating Point Operations Per Second
FPS	Frames Per Second
FSAB	Facteur de Sécurité d'Arrêt de Boucle
GB	Giga Bits
GPS	Global Positioning System
GPU	Graphics Processing Unit
GT	Ground Truth
IA	Intelligence Artificielle
LSP	Largeur Suffissante Perchage
mAP	mean Average Precision
N/A	Not Available
PR	Précision-Rappel
RAM	Random Access Memory
RCNN	Region Convolutional Neural Network
ROI	Region Of Interest
SVM	Support Vector Machine
UAL	Unité Arithmétique et Logique
UCT	Unité Centrale de Traitement
YOLO	YOU Only Look Once

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

Actuellement en plein essor, l'intelligence artificielle représente l'avenir en terme d'avancées technologiques. Elle a pour but d'imiter les aptitudes de l'être humain afin que le système l'utilisant puisse apprendre de lui-même et ainsi réaliser des tâches complexes. La vision artificielle correspond à l'une des branches de l'intelligence artificielle (IA). Elle permet le traitement des images acquises via un système d'acquisition comme une caméra, pour ensuite effectuer les opérations souhaitées. Cette branche de l'IA s'est développée, grâce à ses performances, dans de nombreux domaines tels que la robotique, l'imagerie médicale ou encore la sécurité. Dans le domaine de la robotique, les cas d'applications aux drones civils basés sur la vision artificielles ont explosé [5, 6, 7, 8, 1].

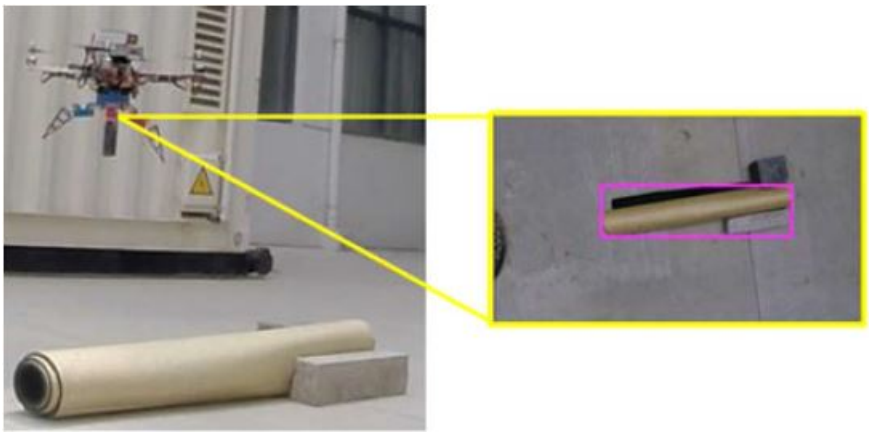


Figure 1.1 Perchage de drone basée sur la vision artificielle[1, Fig. 7a]

Les drones civils présents sur le marché possèdent des performances de stabilité, de vitesse et d'altitude surprenantes. Cependant, ils présentent deux inconvénients majeurs : une durée de vol très limitée allant de 10 à 30 minutes [9, 10] et des pièces fragiles. Selon les cas d'applications, lors de mauvaises conditions environnementales ou encore pour pouvoir recharger les batteries (panneaux solaires), il devient nécessaire d'assurer un atterrissage sécuritaire pour le drone afin d'éviter sa perte (crash, défaillance, batteries vides). Le drone doit également se poser dans des endroits adaptés pour ne pas gêner ou être dangereux pour autrui. Ainsi, dans le cas d'un drone autonome, l'atterrissage repose sur deux aspects :

- Posséder un système de préhension permettant au drone de se poser sur un support
- Détecter des supports potentiels pour s'y poser

Cependant, afin d'obtenir les meilleures chances de perchage du drone, il est important que le système de préhension assure le meilleur maintien possible sur le support pour éviter au drone de tomber. Plusieurs travaux sur le perchage de drone ou la saisie d'objet utilisent un seul support ou un objet qui est choisi d'avance pour qu'il coïncide le plus possible avec le système de préhension du drone [11, 1]. Pour le travail présenté en [1], c'est le préhenseur qui vient s'adapter au support. Dans chacun de ces cas, le drone détecte l'objet à saisir sans connaitre ses dimensions et sa forme. Ainsi si le support est de forme complexe ou ayant des dimensions incompatibles, le préhenseur ne sera pas adapté et n'arrivera pas à le saisir. Pour éviter ce genre de complication tout en obtenant le meilleur maintien possible, une méthode consisterait à détecter le support idéal i.e ayant la meilleure concordance avec le système de préhension.

L'objectif de ce projet de recherche est de développer un système de détection utilisant la vision et l'intelligence artificielles qui permet, à partir des caractéristiques du préhenseur du drone, de déterminer le support idéal pour qu'il puisse s'y poser.

Ce mémoire se décompose comme suit : Le chapitre 2 présente une revue critique de la littérature. Le chapitre 3 énonce la rationnelle du projet de recherche. Le chapitre 5 présente la création du modèle de détection adapté au cas du perchage de drone. Le chapitre 6 présente l'algorithme "Concordance Support-Préhenseur" (CSP) qui permet l'obtention d'un score de concordance entre le préhenseur et les supports détectés afin de déterminer le meilleur support pour le perchage ; dans ce projet de recherche, nous avons élaboré l'ensemble de l'algorithme CSP. Le chapitre 7 présente les résultats de l'ensemble des tests réalisés. Le chapitre 8 présente une interprétation de l'ensemble des résultats obtenus ainsi que la validation du système de détection dans la détermination du support idéal pour le perchage du drone. Enfin, le chapitre 9 correspond à la conclusion de ce mémoire.

CHAPITRE 2 REVUE CRITIQUE DE LA LITTÉRATURE

Ce chapitre présente une revue critique de la littérature sur la vision et l'intelligence artificielle utilisées dans le domaine des drones. Dans ce domaine, la vision artificielle permet au drone d'analyser puis de traiter les images obtenues à partir de son système d'acquisition. Il existe aujourd'hui un grand nombre de programmes informatiques permettant la détection d'objets via différents moyens de traitements d'images. Afin de déterminer, grâce à la vision du drone, les supports adéquats pour son perchage, il est indispensable de connaître les caractéristiques de son système de préhension. De ce fait, nous parlerons de systèmes développés pour permettre aux drones de venir se percher sur un support. Enfin la dernière partie de cette revue, sera consacrée aux applications de drones, existantes ou en cours de développement, utilisant des technologies en vision artificielle.

2.1 Programmes de traitement d'images pour la détection d'objets

Jusqu'à cette dernière décennie, la détection et la classification d'objets étaient réalisées par des programmes n'utilisant pas de réseau de neurones artificiels, dont le plus connu est le SVM (Support Vector Machine) [12]. Le SVM était le programme qui obtenait le meilleur score en terme de détection et de classification. Cependant ce dernier fut détrôné, en 2012, par le travail, utilisant un réseau de neurones artificiels présenté en [13]. Cette distinction a servi de tremplin à une nouvelle génération d'algorithmes. Ces derniers emploient généralement des techniques dites d'apprentissage profond (Deep Learning en anglais) basées sur des réseaux de neurones à convolutions (CNN) [14], aboutissant à de puissants programmes d'intelligence artificielle. L'utilisation de réseaux de neurones n'est pas obligatoire pour détecter des objets. Il existe bien sûr une multitude de méthodes plus simples, le SVM par exemple. Néanmoins, aujourd'hui et pour les années à venir, les réseaux de neurones restent la méthode la plus prometteuse en terme de détection car évoluant sans cesse. En effet, chaque nouvelle version des outils de détection existants obtient de meilleurs résultats, présenté dans les tables finales en [15]. C'est pour cela que dans cet état de l'art nous nous concentrerons d'avantage sur les algorithmes utilisant les réseaux de neurones. L'amélioration significative de leurs performances leur a permis de s'étendre à de nouveaux domaines et à fortiori à de nouveaux cas d'applications. Nous retrouvons par exemple les applications pour l'aide aux personnes handicapées dans le domaine médical [16, 17] ou encore les applications liées aux systèmes mécatroniques que ce soit dans leur conception [18, 19] ou la prédiction de leur comportement [20]. Dans cette section, avant de parler des différents types d'outils de

détection, nous effectuerons un résumé sur le fonctionnement des réseaux de neurones. Puis nous parlerons des différents types d'outils permettant la détection d'objets. Nous avons, tout d'abord, ceux effectuant la détection d'objets par encadrement à l'aide de boîtes englobantes. Puis, ceux utilisant la segmentation sémantique. Enfin, nous terminerons avec ceux utilisant la segmentation d'instances.

2.1.1 Généralités sur le fonctionnement d'un réseaux de neurones

Afin de détecter et catégoriser les objets souhaités, le réseau de neurones à convolutions doit passer par une phase d'apprentissage avant d'être utilisé comme détecteur. Le but de cette dernière est de lui apprendre à reconnaitre une ou plusieurs classes d'objets. Pour cela, nous lui fournissons un ensemble d'images contenant les objets en question que nous aurons encadrés et classifiés au préalable. Ces encadrements sont effectués par nos soins et sont appelés "La vérité sur le terrain" ("Ground Truth" en anglais) qui est le terme technique utilisé dans le domaine de l'IA. Ces "Ground Truth" sont généralement représentés par des rectangles mais peuvent aussi bien être des cercles ou encore des polygones. Et pour chaque "Ground Truth" défini, nous lui associons la classe de l'objet.

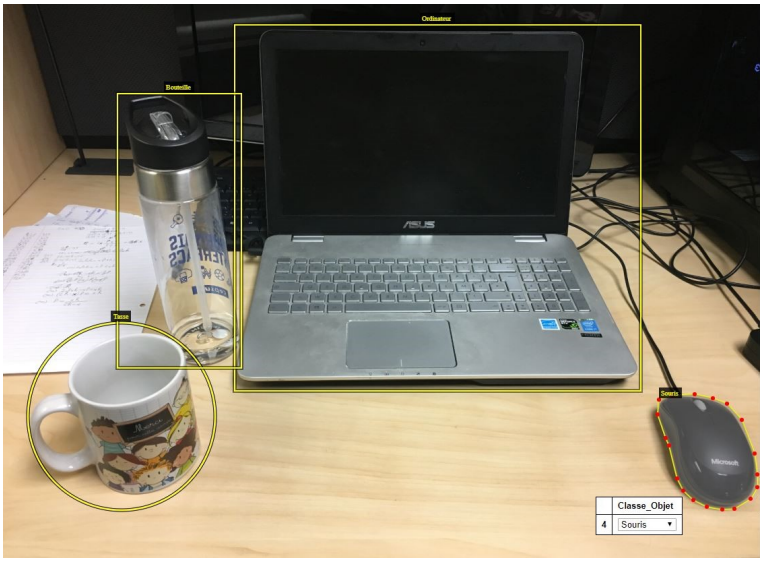


Figure 2.1 Annotations des objets dans l'image (Ground Truth)

L'encadrement des objets dans les images permet ainsi au réseau de neurones de savoir où il doit aller chercher l'information. Cet ensemble représente ainsi la source de savoir pour notre réseau de neurones. Partant de celle-ci, il va analyser chacune des informations données grâce à l'encadrement et à la classification, dans le but de créer ses propres données pour caractériser chacun des objets. L'encadrement lui permet de définir les caractéristiques de l'objet qu'elles soient générales ou spécifiques comme par exemple la forme, la taille, la couleur ou encore

les spécificités des bords ou des coins. La classification, quant à elle, permet d'associer les caractéristiques définies précédemment à une classe d'objet. Lorsque la phase d'apprentissage est terminée, le réseau de neurones est prêt pour détecter les objets souhaités. La détection peut se faire soit sur d'autres images, mais cette fois-ci sans présence des "Ground Truth", ou soit sur une vidéo en temps réel. Lors de la détection, il va chercher à retrouver dans l'image les caractéristiques de chaque objet qu'il a appris puis, selon ces dernières, il pourra déterminer la classe de l'objet trouvé. Il reste à noter que plus le nombre d'images apportées dans l'étape d'apprentissage est élevé et plus l'efficacité de détection du modèle sera élevée [21].

Enfin, pour évaluer un outil de détection sur ses performances, nous déterminons d'un côté le nombre moyen d'images qu'il peut traiter par seconde puis, d'un autre côté, le "mAP" (mean Average Precision) qui constitue la moyenne des précisions moyennes ("AP") de chacune des classes étudiées. Pour déterminer l'AP, nous devons passer par l'obtention des valeurs pour les paramètres de "Précision" et de "Rappel" en faisant varier la valeur du seuil de détection. Les paramètres cités précédemment ont des valeurs qui varient entre 0 et 1. Pour chaque détection d'objet, un score(probabilité) allant également de 0 à 1, est donné à chacune des classes d'objets. Ainsi, pour chaque objet détecté, l'outil de détection renverra en sortie le meilleur score et la classe associée à ce dernier. Le seuil permet de sélectionner les objets ayant un score de détection supérieur ou égal à sa valeur. Ainsi pour chaque valeur de seuil de détection, variant de 0 à 1, nous calculons les paramètres "Précision" et "Rappel". Afin de ne pas trop rentrer dans les détails, nous parlerons seulement des définitions de ces paramètres. Pour savoir à quoi ils sont dûs et comment les calculer, la référence [22] concernant les mesures de performances traite ces informations. Le paramètre "Rappel" correspond au pourcentage d'objets correctement détectés sur l'ensemble des objets qu'il fallait détecter dans la base de données. Le paramètre "Précision", quant à lui, correspond au pourcentage d'objets correctement détectés sur l'ensemble des objets effectivement détectés. Une fois ces valeurs acquises, l'AP de chaque classe est donné par le calcul de la moyenne des valeurs obtenues pour le paramètre "Précision". Enfin il reste plus qu'à effectuer la moyenne de l'ensemble des "AP" pour obtenir le "mAP" de notre outil de détection. Le nombre de "fps" (image par seconde ou "frame per second" en anglais) ainsi que le "mAP" sont deux données essentielles pour évaluer la performance de chaque algorithme d'IA.

2.1.1.1 Librairies pour l'apprentissage profond

Comme les réseaux de neurones artificiels permettent la réalisation de tâches complexes et possèdent des architectures sophistiquées, il est nécessaire d'avoir des ressources importantes pour les utiliser mais aussi pour obtenir de bonnes performances sur la précision de détection et sur la rapidité d'exécution. Afin de concevoir un algorithme basés sur les réseaux de neurones, il est important d'utiliser les bibliothèques spécifiques au domaine de l'apprentissage profond.

Pour réaliser et faciliter la conception de programme utilisant les réseaux de neurones artificiels, des librairies ont été spécialement créées dans les divers langages de programmation. Dans cette maîtrise de recherche, le langage de programmation **Python** a été choisi. Il correspond aujourd'hui, au programme le plus populaire dans le domaine de l'apprentissage machine [23] et il est facile d'utilisation. Dans cette section, les bibliothèques présentées seront donc essentiellement liées à Python [24].

Tout d'abord nous avons les librairies dites de bas niveau. Dans ces dernières, nous retrouvons, d'une part, la librairie **CUDA** qui permet l'utilisation des processeurs graphiques "GPU" dans le domaine de l'apprentissage profond et d'autre part, la librairie **cuDNN** qui accélére le fonctionnement du réseau. Les processeurs graphiques "GPU" ont une importance fondamentale dans l'utilisation des réseaux de neurones. Ces derniers apportent les ressources en calculs nécessaires et permettent d'atteindre de meilleures performances comme par exemple une augmentation considérable de la vitesse d'exécution [25, 26, 27] par rapport aux processeurs "CPU". Ensuite, nous retrouvons les librairies de haut niveau dont les plus connues sont les suivantes :

- **Tensorflow**
- **Keras**
- **Pytorch**
- **Caffe**

Ce sont des bibliothèques logicielles pour Python dont le domaine d'application correspond à l'apprentissage machine et à fortiori celui de l'apprentissage profond qui en est un sous-domaine. Les plus utilisées aujourd'hui sur Python sont Tensorflow et PyTorch. Tensorflow est basée sur Keras et a été créée par Google alors que PyTorch est basée sur Torch qui a été conçue par Facebook. Ces dernières permettent d'effectuer tous types de calculs, principalement tensoriels, nécessaires dans l'utilisation des réseaux de neurones que ça soit pour la détection ou pour l'apprentissage.

2.1.1.2 "Transfert learning" : modèles pré-entrainés

Le cas basique de l'apprentissage d'un réseau de neurones est d'entrainer chacune de ses couches ainsi que les liens qui les unissent. Afin d'obtenir une architecture optimisée permettant l'obtention de bonnes performances de détection et de classification, cela demande une base de données d'apprentissage très importante mais aussi un temps d'entrainement de même envergure. Les données d'apprentissage correspondent à des images contenant les objets que nous souhaitons détectés par la suite. L'ensemble des objets sélectionnés peut être regroupé en une ou plusieurs classes. Aujourd'hui de nombreuses bases de données sont utilisées principalement pour le domaine de l'apprentissage profond. Les plus utilisées, notamment pour les compétitions d'apprentissage profond dans la détection et la classification d'objets, sont énumérées ci-dessous :

- **COCO** [28]
- **ImageNet** [29]
- **SUN** [30]
- **PASCAL VOC** [31]

Ainsi pour chaque nouvelle classe d'objet que nous souhaitons détecter, il faudra ré-entrainer notre réseau de neurones afin qu'il définisse et apprenne les caractéristiques propres à cette nouvelle catégorie.

Comme énoncé précédemment, l'apprentissage d'un réseau demande une importante base de données mais surtout un temps d'entrainement extrêmement long. Cependant, dans le cas où nous avons accès qu'à une petite base de données et/ou que nous souhaitons réaliser un apprentissage en un temps record, il existe une méthode permettant cela tout en gardant la même efficacité du programme en terme de détection. Cette méthode qualifiée de "Transfert learning" se base sur l'utilisation de modèles pré-entrainés [24]. Un modèle pré-entrainé correspond à un réseau de neurones artificiels qui a été entrainé au préalable sur un problème de classification générale. Autrement dit, le réseau de neurones possède, grâce à un entrainement réalisé par des spécialistes du domaine, une architecture déjà optimisée. Ceci représente un avantage conséquent pour les utilisateurs souhaitant travailler sur un problème particulier avec des catégories d'objets spécifiques. En effet, ils peuvent directement utiliser des modèles pré-entrainés et ainsi éviter les entrainements longs et demandant d'importante ressources de calculs. Dans ce genre de cas, un apprentissage pour de nouvelles classes reste nécessaire mais l'utilisation d'un modèle pré-entrainé permet d'apprendre beaucoup plus vite tout en gardant les mêmes performances [24].

Dans ce projet de recherche, la recherche de programmes de détection utilisant des modèles pré-entrainés a été privilégié afin d'établir un modèle de détection, adapté au perchage de drone, de façon plus rapide et plus simple tout en ayant de bonnes performances.

2.1.2 Détection d'objets par encadrement

La plupart des programmes d'IA que nous retrouvons pour la détection et la classification de tous types d'objets, possèdent une architecture utilisant les boîtes englobantes, présenté en [15]. Le réseau de neurones, après l'analyse de l'image qui est la donnée d'entrée, va nous renvoyer en sortie une image avec des boîtes englobantes encadrant les objets prédits et leurs classes respectives . Ainsi nous avons accès d'une part, à la position de l'objet dans l'image et d'autre part, à sa classe.

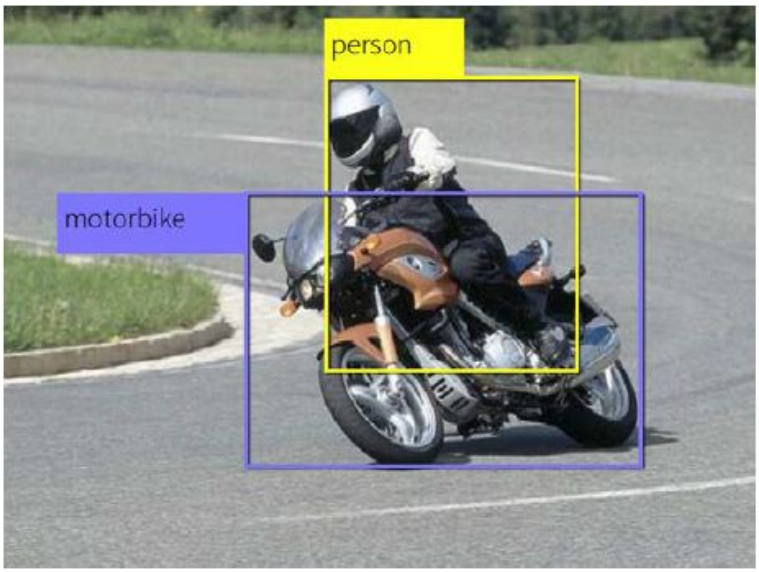


Figure 2.2 Détection d'objets par encadrement [2]

Chaque boîte englobante correspond à un rectangle dont les informations sur les coordonnées d'un point du rectangle, sa hauteur et sa longueur nous sont fournies en sortie de l'algorithme, présenté dans la partie 2 en [3]. Cependant nous avons une perte d'informations car nous ne pouvons pas déterminer avec exactitude où se trouve l'objet dans l'encadrement. De ce fait, nous avons accès qu'à une approximation des dimensions de l'objet qui sont les dimensions de la boîte englobante. Ainsi la détection d'objet par encadrement est principalement utilisée dans les domaines où seulement la recherche d'objets d'intérêts nous importe que ce soit dans une image ou lors d'un visionnage vidéo comme par exemple une caméra de surveillance [32, 33, 34, 35].

Aujourd'hui et selon les états de l'art il existe, parmi un grand nombre d'algorithmes utilisant les réseaux de neurones pour la détection d'objets par encadrement [15], deux grands types d'outils : ceux utilisant la méthode R-CNN (Region-Convolutional Neural Network) et ceux utilisant la méthode YOLO (You Only Look Once). Chaque méthode regroupe plusieurs programmes qui sont énoncés dans la table 1. ceux utilisant la méthode R-CNN (Region-Convolutional Neural Network) tels que R-CNN, Fast R-CNN, Faster R-CNN, R-FCN ou encore Mask-RCNN [36, 37, 38, 39, 40] et ceux utilisant la méthode YOLO tels que YOLO, YOLO tiny, YOLOv2 ou encore YOLOv3 [41, 3, 42, 43].

Tableau 2.1 Exemples de programmes pour chaque grands types d'outils

Méthode R-CNN	**Méthode YOLO**
- R-CNN [36] - Fast R-CNN [37] - Faster R-CNN [38] - R-FCN [39] - Mask-RCNN [40]	- YOLO [41, 3] - YOLO tiny [3] - YOLOv2 [42] - YOLOv3 [43]

Le premier type d'outil, R-CNN, est une méthode de classification basée sur la détermination de régions d'intérêts. Une première partie du réseau de neurones va prendre l'image comme entrée puis par utilisation d'une fenêtre glissante de détection, à plusieurs échelles, il va analyser l'ensemble de l'image. Lorsque cette étape est complétée il retourne, en sortie, des propositions de régions où des objets sont susceptibles de s'y trouver. Ensuite une deuxième partie du réseau prend ces régions proposées en entrée, puis à partir de la position de ces dernières, il crée d'autres régions dites d'intérêt. À partir de celles-ci, il va extraire des caractéristiques afin de les comparer à celles qu'il connaît, pour déterminer s'il y a un objet ou non [37].
Le deuxième type d'outil, YOLO, découpe l'image en une grille de taille $S \times S$. Puis de façon simultanée, il réalise d'une part la prédiction des objets dans l'image et d'autre part, détermine la classe de l'objet, comme montré sur la figure 2.3.

Pour la prédiction : le réseau de neurones va pour chacune des cellules de la grille, prédire B boîtes englobantes avec un score de confiance (probabilité d'avoir l'objet dans cette dernière).
Pour la classification : l'outil va, pour chacune des cellules, nous donner la probabilité de la

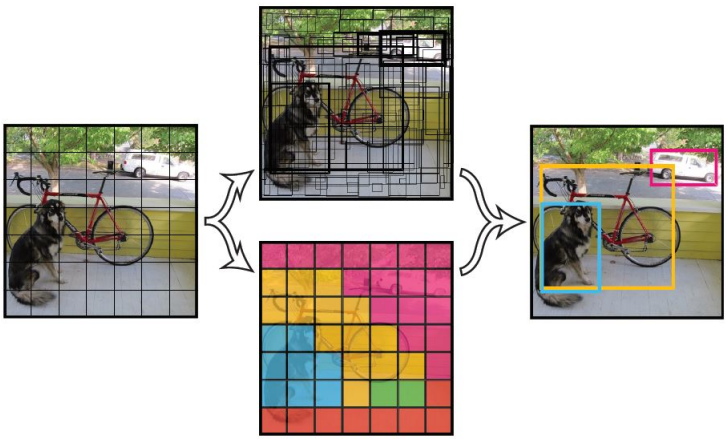


Figure 2.3 Architecture de YOLO [3, Fig. 2]

classe de l'objet présent dans cette dernière. Enfin, le programme va nous donner en sortie un vecteur rassemblant toutes les informations obtenues aux étapes précédentes. Chacune des boîtes englobantes obtenues regroupe cinq informations : les coordonnées de son point de départ, sa longueur, sa hauteur et le numéro de la classe prédite.
Chacune de ces deux méthodes a son avantage et son inconvénient, présentés en tableau 2.2.

Tableau 2.2 Avantages et inconvénients de chaque grands types d'outils

	Méthode R-CNN	**Méthode YOLO**
Avantage	Meilleure précision	Détection en temps réel
Inconvénient	Procédé lent	Précision inférieure
Exemple (voir table 3 en [43])	Faster R-CNN : 5 fps et mAP-50 = 59.1%	Yolov3-608 : 20 fps et mAP-50 = 57.9%

Un outil de détection peut être qualifié de détecteur en temps réel dans le cas où il n'y a pas de latence perceptible par rapport à l'action réalisée [44]. Dans le cas du déplacement d'un drone, le détecteur doit être en mesure de n'avoir aucune latence sur sa détection lors de ce mouvement. Selon la littérature, l'outil de détection ayant le plus bas fps (image par seconde ou "frame per second" en anglais) et étant qualifié de détecteur en temps réel correspond à "Faster R-CNN" avec un nombre de 5 fps [38]. La méthode YOLO est donc qualifiée de détecteur en temps réel, comme présenté en [3], car sa détection atteint, peu importe la version de YOLO, un minimum de 20 fps [41]. Il a notamment été utilisé avec un drone dans

le projet de recherche présenté en [1] dans le but de se percher sur un support. De plus, il possède une précision presque aussi bonne que Faster R-CNN.

Les outils utilisant la détection par encadrement couvrent aujourd'hui un grand nombre de domaines d'applications ; des exemples sont donnés dans le tableau 2.3.

Tableau 2.3 Exemples de domaines d'applications utilisant la détection par encadrement

Domaines d'applications		
Automobile	**Médecine**	**Surveillance**
- Détection de la signalisation [33] - Détection des piétons [34] - Assistance par temps de broullard [45]	- Détection de cancer [46] - Détection d'anomalies dans les poumons [47]	- Suivi du trafic [32] - Repérage de fissures [48] - Détection d'avions [35]

Des méthodes d'amélioration et d'optimisation sont constamment en développement afin de parfaire ce type d'outil. Notamment une méthode sur des boîtes englobantes en 3D, et non en 2D comme dans l'ensemble des programme de détection, permet d'avoir une meilleure estimation des délimitations de l'objet détecté ou encore pourrait permettre de déterminer ses dimensions [49].

2.1.3 Détection d'objet par segmentation sémantique

Un autre type de programme dans le domaine de l'IA se base sur la détection d'objets par leur segmentation. La segmentation d'image est une opération dont l'objectif est de regrouper les pixels possédant des caractéristiques communes comme la couleur ou la texture. Cette segmentation classique est aussi appelée segmentation sémantique. Le détecteur, utilisant cette méthode, identifie la catégorie d'objet associée à chaque pixel de l'image.

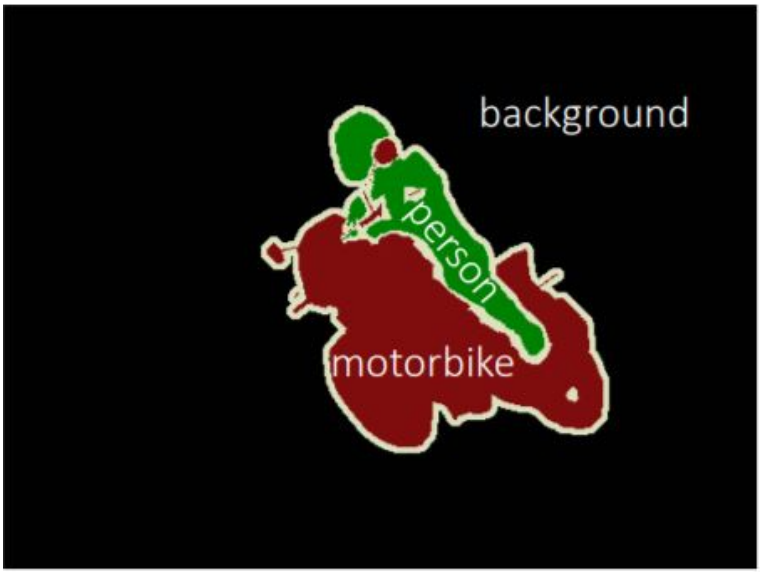


Figure 2.4 Détection d'objets par segmentation [2]

Il existe une multitude de méthodes de segmentation qui dépendent toutes des critères d'homogénéité que nous souhaitons définir [50, 51]. Les plus connues étant la segmentation par couleur, celle par texture ou encore celle par contour qui est un peu plus complexe que les autres [52]. Une grande difficulté dans l'utilisation de la segmentation reste la détermination des seuils pour la différenciation entre les régions. Comme toute méthode de détection, la segmentation possède des avantages et des inconvénients. Ces derniers sont énoncés dans le tableau 2.4.

Tableau 2.4 Avantages et inconvénients de la segmentation sémantique

Segmentation sémantique	
Avantages	**Inconvénients**
- Objet dissocié de l'arrière-plan et des objets de classes différentes - Objets définis en région de pixels donnant accès à leurs dimensions	- Sensible aux changements d'environnement (contraste, ombres, bruitages, ...) - Objets de même classe qui sont côte à côte : segmentation de l'ensemble et non de chaque objet

Dans le cas du deuxième inconvénient énoncé dans le tableau 2.4, il nous est alors difficile de déterminer le nombre d'objets présents dans une même région et faussant ainsi les données sur la dimension d'un objet, comme le montre l'image au centre de la figure 2.5.

L'arrivée des réseaux de neurones artificiels a permis pour ce type d'outil d'augmenter considérablement les résultats de segmentation et donc de détection des objets [53, 54, 55, 56]. En effet, l'algorithme présenté par Hou, Qibin et al. (2017) basé sur l'utilisation d'un réseau de neurones entièrement connectés (FCN) possède de très bonnes performances que ce soit sur le plan de la rapidité d'exécution avec 80ms par image ou sur la précision des résultats avec des prédictions proches de la réalité ("Ground Truth", en anglais) [57]. La segmentation possède également de nombreux domaines d'applications avec notamment le domaine médical qui l'utilise depuis longtemps [58]. L'utilisation des réseaux de neurones permet aussi d'obtenir plus facilement un procédé à partir d'un autre comme par exemple l'obtention de la segmentation d'objets à partir de l'exploitation de leurs boîtes englobantes [59].

2.1.4 Détection d'objet par segmentation d'instances

Nous avons vu dans la partie précédente, que la segmentation classique i.e la segmentation sémantique a un grand défaut car elle s'opère par rapport aux classes d'objets et donc ne pouvant différencier des objets similaires côte-à-côte. Un autre type de segmentation plus avancé a vu le jour il y a seulement quelques années : la segmentation d'instances [28], résolvant ainsi ce défaut.

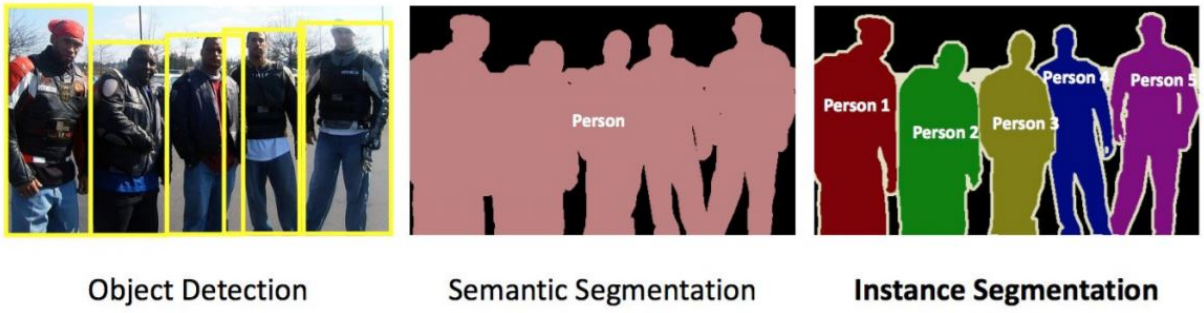


Figure 2.5 Détection d'objets par segmentation d'instances [4]

La segmentation d'instances permet d'obtenir d'une part une dissociation des objets par rapport à l'arrière-plan (cas d'origine) et d'autre part une dissociation entre chacun des objets présents dans l'image qu'ils soient de la même catégorie ou non. Ceci est dû au fait qu'elle identifie directement l'objet et la classe de chaque pixel de l'image. Ainsi cela nous donne une définition précise de chacun des objets détectés.

La segmentation d'instances se définit comme étant une association entre les deux méthodes vues précédemment : la détection d'objets par encadrement et la segmentation sémantique. De ce fait, nous avons accès avec plus de certitude aux dimensions des objets. Les premiers algorithmes utilisant ce type de segmentation sont apparus dans le courant des années 2014/2015 avec par exemple l'algorithme DeepMask [60]. Cette méthode a très vite progressé avec l'apparition d'autres algorithmes toujours plus performants les uns que les autres [61, 62, 63, 64, 65, 66, 40]. Cette progression a permis d'obtenir des algorithmes performants et donnant des résultats de détection et de segmentation précis, comme avec l'algorithme Mask R-CNN. Aujourd'hui, celui-ci est devenu une bonne référence dans l'état de l'art de la segmentation d'instances [40]. Cet algorithme est de plus en plus utilisé comme par exemple dans le domaine du bâtiment [67] ou encore dans le domaine médical [68] grâce à la précision obtenue notamment sur les masques d'objets.

Pour en revenir aux dimensions des objets, l'obtention de tels résultats en termes de détection et de segmentation nous permet de les déterminer via le traitement du masque de chaque objet qui est obtenu en sortie d'algorithme. Ces informations nous permettent ainsi d'accéder à de nouveaux cas d'applications, comme c'est le cas pour cette maîtrise de recherche.

2.2 Systèmes de préhension

Comme évoqué au début de cet état de l'art, les caractéristiques du système de préhension du drone constituent des données essentielles pour la configuration de l'algorithme de vision. C'est pour cela que nous verrons, dans cette partie, un certain nombre de systèmes, tous différents, afin de voir les possibilités de configurations pour l'algorithme. Chaque préhenseur possède ses avantages comme ses inconvénients, c'est pour cela qu'il est important de sélectionner un système de préhension adéquat à notre cas d'application [69]. Dans cette sélection de systèmes de préhension, nous pouvons les ranger en deux grandes catégories :

1. Systèmes de préhension basiques et adaptatifs
2. Système de préhension inspirés de la nature

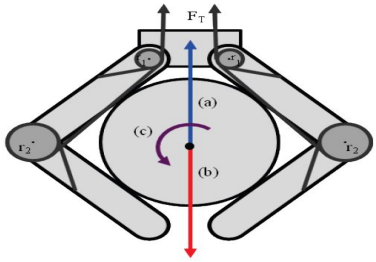
La première catégorie, allant des systèmes de préhension communs tels qu'une simple pince (limitée à deux doigts) à des systèmes de préhension adaptatifs [70] et donc plus avancés, regroupe des préhenseurs qui sont aujourd'hui très utilisés notamment dans le domaine de la robotique car étudiés depuis de nombreuses années et donc parfaitement connus en terme de conception et surtout de contrôle. Les préhenseurs adaptatifs de cette catégorie permettent

de couvrir un grand nombre de tailles et de formes de support. Enfin, dans la deuxième catégorie, nous verrons que de nombreux drones s'inspirent de la nature que ce soit pour la conception, le contrôle, la vision ou encore le système de préhension de ce dernier. Nous parlerons donc de systèmes de préhension adaptatifs inspirés de la nature.

2.2.1 Préhensions basiques et adaptatives

Les systèmes de préhension basiques sont composés principalement de pinces avec deux parties (ou "doigts") mobiles. Chaque partie peut être réalisée en un seul bloc (figure 2.6a) [71] ou en plusieurs parties (figure 2.6b) [70] pour permettre une meilleure préhension et une meilleure adaptation selon la taille de la cible à attraper. Enfin il est possible d'ajouter un bras articulé entre le drone et la pince afin d'obtenir un système de préhension qui peut couvrir une zone de travail beaucoup plus importante (figure 2.6c) [11, 72].

(a) Pince avec chaque partie en un seul bloc [71, Fig. 1]



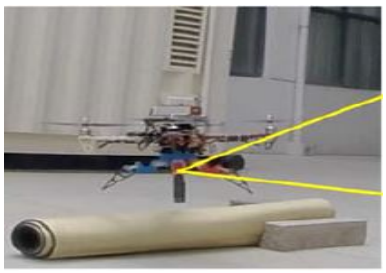
(b) Pince avec chaque partie articulée [70, Fig. 3]

(c) Bras articulé et pince avec chaque partie articulée [11, Fig. 7c]

Figure 2.6 Systèmes de préhension adaptatifs basiques

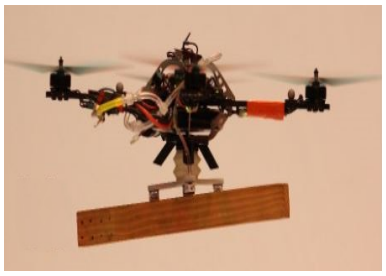
En effet, pour le cas des drones, si son système de préhension ne possède pas de bras articulé alors c'est à lui de s'adapter et de s'ajuster pour atteindre la cible. En présence d'un bras articulé le drone peut atteindre rapidement une position proche de la cible afin que le préhenseur puisse attraper l'objet ciblé. L'avantage de ce genre de préhension est la réalisation de son contrôle, dû aux nombreux acquis et connaissances de ce type de système dans le domaine de la mécanique. Quant à son inconvénient, ce type de préhenseur est très vite limité à la taille et à la forme de l'objet ciblé. En effet, la pince possède une ouverture limite et selon certaines formes d'objet le maintien de l'objet peut être compromis durant le déplacement du drone. De plus, une pince avec des parties en un seul bloc (figure 2.6a) sera d'autant plus limitée pour la taille et le bon maintien de l'objet qu'avec des parties en plusieurs sections (figure 2.6b).

Les systèmes de préhension adaptatifs se basent sur leur flexibilité à s'ajuster à leurs cibles, quelque soit leurs tailles ou leurs formes, afin de permettre un maintien optimal. C'est cette capacité à s'adapter qui leur apportent un avantage considérable par rapport aux préhenseurs basiques. Différents exemples de ce type de préhenseur sont présentés avec la figure 2.7.



(a) Pince avec 4 doigts [1, Fig. 7b]

(b) Préhenseur magnétique [73, Fig. 8 (2)]



(c) Préhenseur à ventouses [74, Fig. 10h]

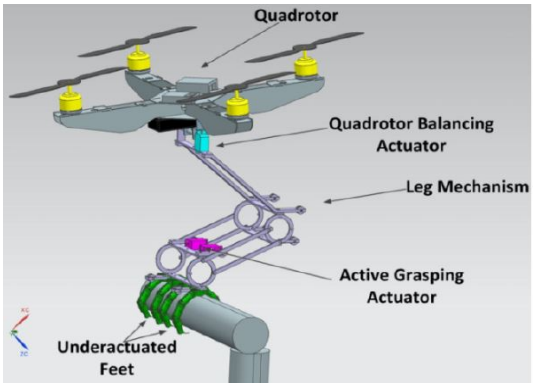
Figure 2.7 Systèmes de préhension adaptatifs avancés

Tout d'abord, nous retrouvons des pinces avec plusieurs "doigts" articulés généralement au nombre de trois ou quatre (figure 2.7a) [1]. Le fait d'avoir plus de deux doigts permet au système de préhension d'assurer une prise ferme et stable notamment pour des objets de forme ronde ou cylindrique par exemple. Cependant, plus un préhenseur possède de doigts et donc d'actionneurs, plus son contrôle est complexe.

Ensuite, nous avons les préhenseurs magnétiques (figure 2.7b) ou encore ceux utilisant des ventouses (figure 2.7c). Le préhenseur magnétique a comme avantages de pouvoir attraper, tout objet métallique peu importe sa taille (en comparaison avec celle du drone), sa forme et son poids car il est relativement puissant sur le maintien avec ce type d'objet [73]. Son inconvénient est bien évidemment son incompatibilité avec les objets non métalliques mais aussi les surfaces sales qui risquent de provoquer un maintien instable [69]. Concernant le préhenseur utilisant des ventouses, il peut également saisir tous types d'objets quelque soit la taille (ici aussi en comparaison avec celle du drone), la forme ou le matériau [74]. Il peut aussi saisir des objets assez lourds pour le drone mais à condition d'adapter la taille des ventouses car de trop petites ventouses pourraient ne pas suffire pour un objet avec un certain poids. L'inconvénient pour ce type de préhenseur apparaît lorsqu'un objet est poreux ou mouillé [69].

2.2.2 Préhension adaptative inspirée de la nature

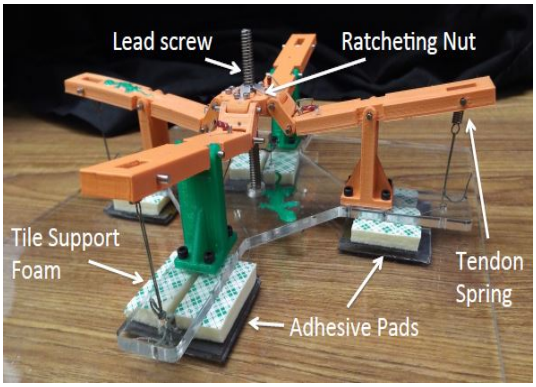
Depuis de nombreuses années et encore plus aujourd'hui avec l'apparition des drones civils, nous nous inspirons de la nature dans le but de concevoir des systèmes plus performants et plus robustes. En effet, la nature par ses millions d'années d'évolution reste une bonne source d'inspiration et cela dans la majorité des domaines. La conception de préhenseurs pour les drones n'y a pas échappée.



(a) Préhenseur inspiré des oiseaux modèle 1 [75, Fig. 1]

(b) Préhenseur inspiré des oiseaux modèle 2 [76, Fig. 1]

(c) Préhenseur inspiré de la main humaine [77, Fig. 7]



(d) Préhenseur adhésif inspiré du gecko [78, Fig. 6]

Figure 2.8 Systèmes de préhension adaptatifs avancés inspirés de la nature

Dans ce genre de systèmes de préhension, nous retrouvons, tout d'abord, les préhenseurs inspirés par les oiseaux (figures 2.8a et 2.8b) [76, 75]. En effet, les oiseaux possèdent des jambes et des pattes conçues pour être performantes dans les tâches de saisie que ce soit pour attraper une proie/nourriture ou encore pour se percher sur un support. De plus, certains types d'oiseaux sont pourvus d'un tendon à l'arrière de la cheville qui, lorsque la jambe est

pliée sous le poids de l'oiseau, provoque la contraction automatique des doigts des pattes permettant une saisie passive du support [76, 75].
Ensuite, nous avons les préhenseurs qui sont inspirés de l'homme (figure 2.8c) [77]. La main humaine bien que complexe permet une saisie d'un grand nombre d'objets avec des tailles et des formes différentes tout en assurant un maintien stable grâce notamment à son pouce opposable.
Enfin, il existe le préhenseur adhésif inspiré directement du gecko appartenant à la famille des lézards (figure 2.8d) [78].

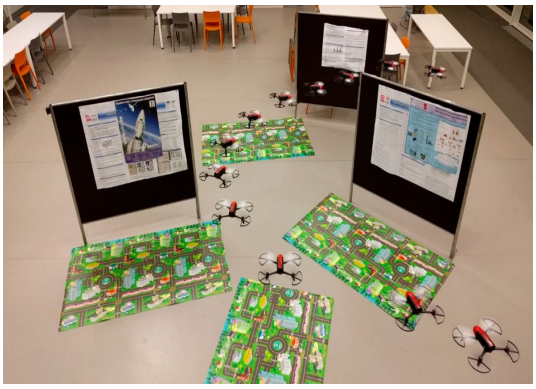
2.3 Applications de drones basées sur la vision artificielle

Nous avons pu voir, à travers les parties précédentes de cet état de l'art, les outils qui ont permis de propulser l'utilisation des drones durant ces dernières années. De plus, l'évolution perpétuelle de ces outils offre aux drones l'accès à de plus en plus de domaines d'applications, leurs assurant un avenir certain dans les années futures. Dans cette dernière partie, nous allons voir différents cas d'application de drones utilisant la vision artificielle pour réaliser leur tâche.
Nous pouvons distinguer quatre principaux types d'applications :

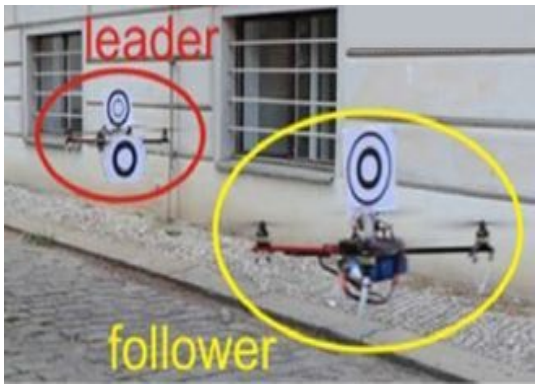
- Suivi de cible (figures 2.9a)
- Saisie et dépôt d'objets (figure 2.9b)
- Atterrissage de drones (figure 2.9c)
- Évitement d'obstacles (figures 2.9d)

2.3.1 Suivi de cible

Pour ce qui est du suivi de cible, les drones n'ont pas de système de préhension car seulement la vision artificielle est nécessaire dans ce genre d'applications. Dans cette catégorie, nous avons deux sous-catégories : d'un côté, il y a le suivi de cible en mouvement que nous cherchons à traquer [5, 81] et de l'autre côté nous avons le suivi d'un meneur où plusieurs drones suivent la trajectoire d'un drone leader (figure 2.9b) [82, 6]. Ce cas d'application peut se révéler très utile pour les deux catégories. En effet, le suivi de cible dangereuse par des drones, par exemple, permettrait d'éviter de prendre des risques aux humains. Cela permettrait également d'utiliser ce genre d'application pour détecter et intercepter d'autres drones : tels que des drones survolant des zones interdites. Pour le suivi d'un drone leader, il suffit seulement d'un pilote de drones pour atteindre une zone souhaitée (sans utilisation de GPS), utile notamment pour le déplacement de plusieurs drones dans un bâtiment.



(a) Évitement d'obstacles [79, Fig. 7]



(b) Suivi d'un drone meneur par un autre drone [6, Fig. 12]

(c) Saisie et dépôt d'objets par un drone [7, Fig. 16]

(d) Atterrissage de drone sur un support en mouvement [80, Fig. 16]

Figure 2.9 Cas d'applications de drones utilisant la vision artificielle

2.3.2 Saisie et dépôt d'objet

Pour le genre d'application réalisant la saisie et/ou la dépôt d'objets, l'utilisation de la vision et d'un système de préhension sont nécessaires. Le drone par sa vision artificielle détecte un objet d'intérêt puis avec son système de préhension l'attrape. Enfin, il l'apporte jusqu'à un endroit, défini par l'utilisateur, pour le déposer (figure 2.9c) [11, 7]. Ce type d'applications possède un réel potentiel d'utilisation puisque des projets de drones livreurs de colis sont en cours de développement avec par exemple "Prime Air" d'Amazon [83]. De plus des compétitions, comme la compétition MBZIRC [7] sur la préhension d'objet, sont mises en place dans le but de toujours repousser les limites de performances pour avoir à la fois une vitesse de détection plus rapide et une précision plus accrue.

Cependant, dans le cas d'un préhenseur de type pince comme en [11], l'objet a attrapé est choisi au préalable afin que son diamètre corresponde avec le diamètre d'ouverture du préhenseur. Ainsi dans le cas d'un objet ayant un diamètre trop grand, si le drone le détecte alors la préhension échouera.

2.3.3 Atterrissage de drones

Le dernier cas d'applications représente tout ce qui s'apparente à l'atterrissage de drone (figure 2.9d). L'atterrissage de drone reste encore aujourd'hui quelque chose de complexe. Ceci est principalement dû au fait que les drones sont composés en grande partie de pièces fragiles. Nous retrouvons, ici encore, l'utilisation de la vision artificielle mais selon les cas d'applications nous avons la présence au non d'un système de préhension.

Une étude, présentée en [84], s'est basée sur l'établissement d'un système biomimétique adaptatif, inspiré des oiseaux, pour que le drone puisse s'adapter à tous types de terrains : en pente, avec des obstacles ou des trous. La vision du drone va lui permettre d'analyser le terrain de la zone où il se trouve pour repérer l'endroit le moins accentué. Une fois la tâche réalisée, il peut s'y poser en toute sécurité grâce à son système adaptatif.

Deux autres études se sont quant à elles focalisées sur l'atterrissage de drones sur une zone précise. Cette zone est recouverte d'un ou plusieurs symboles afin de permettre à la vision artificielle du drone, entrainée pour reconnaitre ces symboles, de détecter cette zone. La première étude utilise la détection de simples carrés de pixels de différentes tailles [8]. La deuxième utilise une détection de contours afin de détecter des lignes spécifiques du symbole étudié [80]. Ainsi, le drone peut se poser sur un support, qu'il soit fixe ou en mouvement (figure 2.9d), possédant ce type de symboles.

L'étude, décrite en [1], a été menée afin de permettre à un drone de se poser sur des supports cylindriques de longueurs et de diamètres différents qui ont été détectés au préalable. Tout d'abord, la vision artificielle du drone utilise le réseaux de neurone YOLO [3] pour avoir une détection des supports en temps réel. Puis grâce à son système de préhension adaptatif (évoqué dans la partie 2 de cet état de l'art) il se pose sur le support détecté. Cependant, dans cette étude, le support utilisé pour les tests est sélectionné au préalable afin que ses dimensions soient contenues dans l'intervalle d'ouverture du système de préhension du drone. Ainsi, cela représente un réel problème dans le cas où le drone survole une zone en extérieur. En effet, dans cette configuration, lorsqu'il souhaitera se poser sur un support, il pourra détecter des objets d'intérêts mais il ne pourra pas déterminer si les dimensions du support

en question coïncideront avec celles de son préhenseur.

Cette catégorie d'applications est très importante car l'atterrissage de drones représente la solution aux différentes faiblesses d'un drone. Aujourd'hui, un drone avec une caméra est limité par sa durée de vol avec un temps moyen d'une trentaine de minutes [9, 10]. Une autre de ces faiblesses est sa fragilité due à des composants fragiles comme ses hélices et ses bras. Pour certains drones un peu moins performants, il y a aussi la résistance au vent qui peut vite le déstabiliser le rendant instable et provoquant son crash. Ainsi, en cas de vent violent, ou pour éviter de casser des pièces pendant l'atterrissage ou encore lorsqu'il n'a plus de batterie, l'atterrissage sécuritaire du drone est essentiel.

2.3.4 Évitement d'obstacles

L'évitement d'obstacles (figure 2.9a) représente un défi majeur dans le déplacement automatique de drones. Pour chacune des applications énoncées précédemment ou pour de grands projets tel que "Prime Air" d'Amazon, il est important que le drone puisse atteindre son objectif tout en s'assurant que le chemin pour y parvenir est libre. L'évitement d'obstacles est donc indispensable pour le pilotage automatique du drone. Différents systèmes de détection ont été développés pour ce genre d'application. Pour le système d'acquisition nous retrouvons, d'une part, ceux utilisant une caméra monoculaire [85, 86, 87, 88, 89, 79] et d'autre part, ceux utilisant une caméra stéréoscopique [90, 91]. La caméra stéréoscopique permet une représentation 3D de l'espace tandis que la monoculaire se limite à une représentation 2D. Pour ce qui est des méthodes de détection d'obstacles, nous en retrouvons principalement trois. Tout d'abord, nous retrouvons la méthode du flux optique correspondant au mouvement apparent des objets créé à partir du mouvement entre la caméra et les objets présents dans l'environnement [86, 89, 91]. Ensuite, nous avons l'utilisation des réseaux de neurones (CNN) pour détecter directement les objets présents [79]. Enfin, une autre méthode consiste à détecter les régions saillantes dans une image, autrement dit elles représentent les régions qui se détachent du reste de l'image à partir de ses caractéristiques comme les contrastes, les couleurs ou encore les bords [88, 87].

CHAPITRE 3 RATIONNELLE DU PROJET DE RECHERCHE

En résumé, selon l'état de l'art, il existe de nombreux cas d'applications de drone utilisant la vision artificielle pour réaliser des tâches variées et complexes. Pour ce qui est du perchage, nous en sommes encore au commencement. Le défi dans le perchage automatique de drone est de trouver, selon le système de préhension utilisé, un support adapté. En effet, dans la littérature, les projets sur le perchage de drone utilisent des supports choisis au préalable par les gérants dans le but d'assurer une concordance avec le préhenseur. Ainsi, le problème générale dans le perchage de drones reste la détermination automatique d'un support adapté. Même dans le cas où le préhenseur peut s'adapter sur un grand intervalle de diamètres, un support de diamètre trop petit ou trop grand peut entrainer l'échec du perchage. Le but de notre projet de recherche est donc de permettre, à partir de la vision du drone, de réaliser un calcul de concordance entre les supports détectés et le préhenseur utilisé pour arriver à combler ce manque. Le cas idéal serait d'obtenir la même chose mais avec une détection en temps réel.

3.1 Problématique

Dans la littérature, nous avons pu voir que la problématique principale que nous rencontrons dans le perchage des drones, est le problème de concordance qu'il peut y avoir entre les dimensions des objets/supports détectés et celles du préhenseur utilisé. Une mauvaise concordance entre les deux pourrait faire échouer le perchage du drone.
Cette problèmatique soulève cependant une sous-problématique. D'une part, un drone possède une vitesse de déplacement élevée, de l'ordre de 60 km/h [92]. D'autre part, la détection et la segmentation d'objets basées sur l'utilisation de réseaux de neurones ont un temps de traitement relativement long pouvant aller à plus d'une seconde par image. Néanmoins, ce temps d'exécution peut être écourté mais au détriment de la précision. Ainsi, il faut réussir à obtenir une vitesse de détection suffisante pour assurer un retour d'informations assez rapide par rapport aux déplacements de la caméra. En même temps, il faut atteindre une précision de détection suffisante qui assure la bonne caractérisation des dimensions des objets détectés. Pour le travail présenté dans ce projet de maîtrise, ces problématiques représentent un défi de taille. En effet, il devient rapidement difficile d'assurer à la fois une vitesse de détection en temps réel et une précision de détection permettant la bonne caractérisation des dimensions des objets détectés et à fortiori assurant la meilleure concordance possible avec le préhenseur du drone et donc le meilleur taux de succès pour son perchage.

3.2 Objectifs du projet de recherche

3.2.1 Objectif général

L'objectif général de ce projet de recherche est de développer un système de détection de pose optimale pour le perchage automatique de drones par vision et intelligence artificielle. Pour atteindre cet objectif, il est nécessaire de passer par la réalisation de plusieurs sous-objectifs.

3.2.2 Objectifs spécifiques

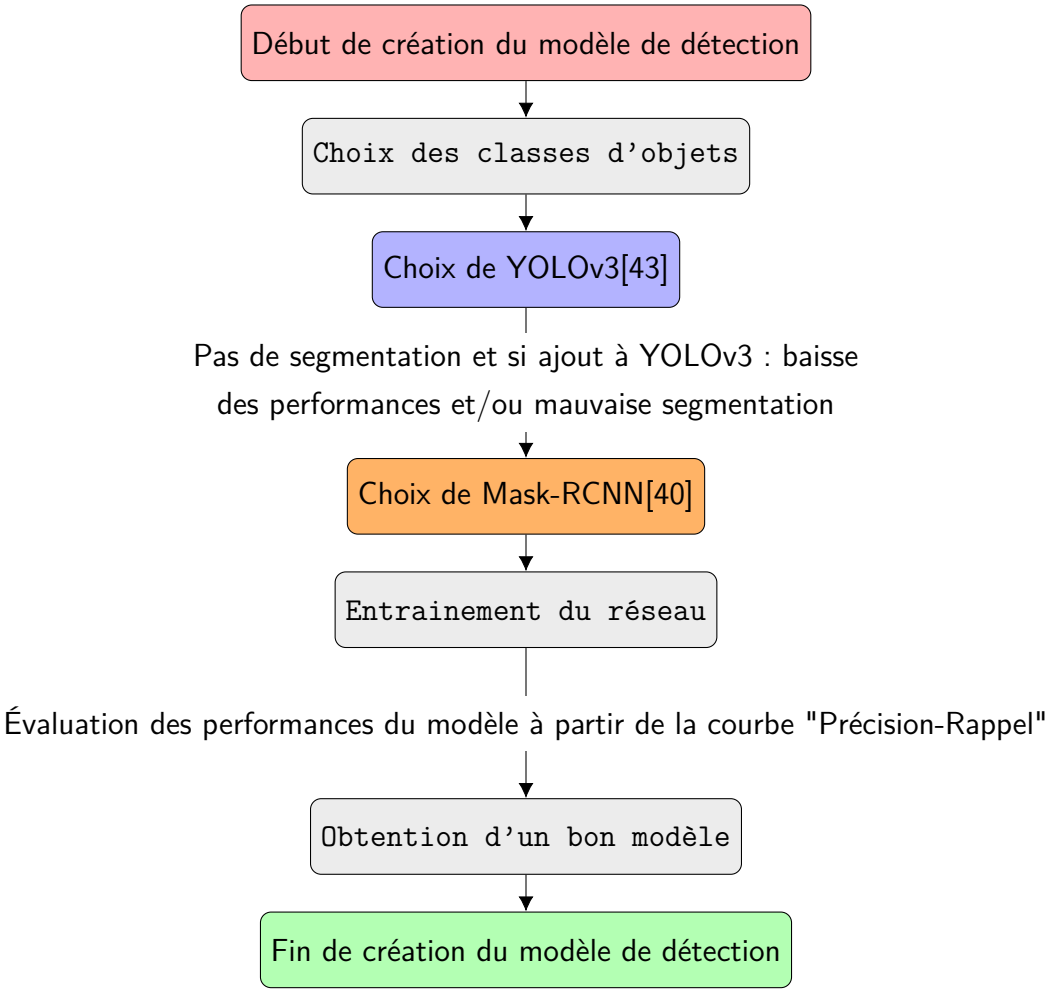
1. Développer un outil de détection pour de nouvelles classes d'objets représentant des supports potentiels pour le perchage du drone.
2. Développer un algorithme permettant de déterminer les dimensions des objets détectés.
3. Développer un algorithme qui permet d'identifier le support le plus adéquat pour le perchage de drones.
4. Valider le système de détection du support idéal à partir de tests expérimentaux.

CHAPITRE 4 CONCEPTION DU MODÈLE DE DÉTECTION ADAPTÉ AU PERCHAGE DE DRONE

Le premier objectif spécifique de ce projet correspond au développement d'un outil de détection, en temps réel, pour de nouvelles classes d'objets adaptées pour le perchage du drone. Cette partie du mémoire se consacre donc à la mise en œuvre d'un modèle de détection adapté au perchage de drone. Ce modèle de détection doit, à partir des images vidéos de la caméra, détecter les objets d'intérêts. Ces objets seront appris par le modèle afin qu'il puisse les reconnaitre en tout temps lors du survol du drone.

Pour la conception de ce modèle, nous sommes partis d'un algorithme de détection existant : Mask-RCNN, et nous l'avons adapté au cas du perchage de drones. Puis, à partir de ses paramètres d'entrainement et de détection, nous avons établi une meilleure configuration dans le but d'améliorer les performances de détection. Dans l'article de Mask-RCNN [40], ce dernier atteint une vitesse de détection de 5 fps mais cette valeur provient d'une moyenne faite à partir du temps de traitement sur un ensemble d'images qui peuvent posséder plusieurs objets. Selon l'article, Mask-RCNN est considéré comme un détecteur en temps réel avec un nombre de 5 fps. Cette valeur représente donc la valeur limite entre une détection en temps réel ou non. Nos tests ont été réalisés dans le cas où nous avons un seul objet par image. Pour cette configuration, le modèle atteint une vitesse de détection de 6.9 fps $\geq$ 5 fps et correspond donc à un détecteur en temps réel. Ainsi, à partir des résultats obtenus nous avons valider ce premier objectif spécifique.

Les choix et les différentes étapes de cette partie sont résumés dans le diagramme ci-dessous :



Chaque choix fait et chaque étape réalisée seront judicieusement détaillés au travers des sections de cette partie.

4.1 Détection en temps réel

Comme vu dans la revue de littérature de ce mémoire, "Faster R-CNN" est qualifié de détecteur en temps réel avec un nombre de 5 fps [38]. De ce fait, et pour la suite de ce mémoire, un modèle de détection sera qualifié de détecteur en temps réel si son nombre de fps est supérieur ou égal à cette valeur.

Comme énoncé en début de partie, la conception du modèle a été réalisée pour le style de drone représenté sur la figure 2.6c à savoir un drone avec un bras et une pince articulés. Partant de cela, la première étape consiste à définir les classes d'objets pouvant correspondre

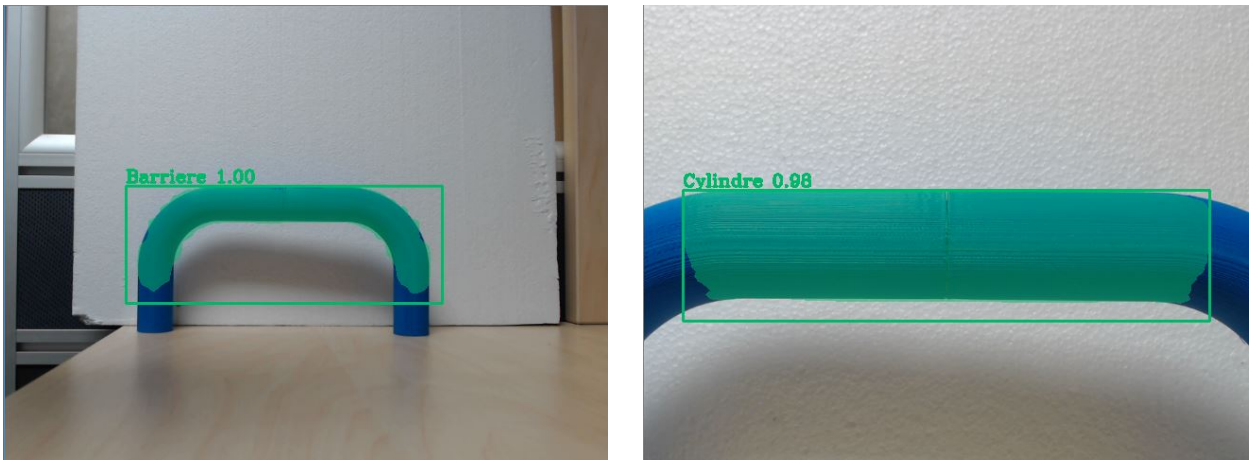
avec le cas du perchage et que nous retrouvons également dans notre quotidien. De plus, nous avons vu dans la littérature que les systèmes de préhension des drones ont généralement des objets cylindriques comme cible à saisir [70, 11, 1]. Même chose du côté de la nature avec les oiseaux qui se posent sur des branches, des lampadaires ou des barrières, comme présenté en [76]. Nous nous sommes donc inspirés de ce type de support pour ce porjet.

De ce fait, le choix des classes d'objets de ce projet de recherche a suivi cette direction. Les classes d'objets choisies sont les suivantes :

1. Classe 1 : Branche
2. Classe 2 : Cylindre
3. Classe 3 : Lampadaire
4. Classe 4 : Barrière

Nous avons choisi, ici, de nous limiter à ces 4 classes car elles regroupent un grand ensemble d'objets de forme cylindre, notamment avec la classe "Cylindre".
Les objets rectangulaires peuvent également fonctionner en terme de support de perchage, il suffit pour cela que les parties de la pince soient articulées. Parmi les différentes classes choisies, la classe "Cylindre" est la plus importante car elle permet d'assurer la détection des objets potentiels dans un grand nombre de cas. Par exemple, lors du survol du drone de la zone souhaitée pour le perchage, si le drone se rapproche d'un objet potentiel, l'objet risque de ne plus être visible dans son intégralité empêchant ainsi sa détection. Cependant comme l'ensemble des objets d'intérêts sont ici cylindriques chaque partie d'objet peut être vu comme un simple cylindre, comme le montre la figure 4.1.

Il est important de noter que pour la finalité de cette maîtrise, la classe de l'objet n'a pas réellement d'importance. Tout ce qu'il nous importe ici est que la vision du drone détecte un certain nombre de supports potentiels et retourne le support idéal pour le perchage. Cependant pour l'entrainement du réseau de neurones, il est important d'avoir plusieurs classes dans le but de distinguer les objets détectés. Cela permet d'obtenir un meilleur résultat d'apprentissage et à fortiori de détection. Effectivement, si toutes les classes sélectionnées étaient regroupées en une seule, il y aurait trop de caractéristiques à prendre en compte augmentant ainsi le risque de mauvaises détections, comme des objets complètement différents de ceux appris. Chaque classe regroupe des caractéristiques qui lui sont propres. Ici, même si elles ont la caractéristique de la forme cylindrique en commun, chacune de ces classes reste bien différente des autres.



(a) Barrière vue dans son intégralité (b) Zoom sur une partie de la barrière

Figure 4.1 Détection d'une barrière à différentes distances

4.2 Première approche avec l'outil YOLOv3

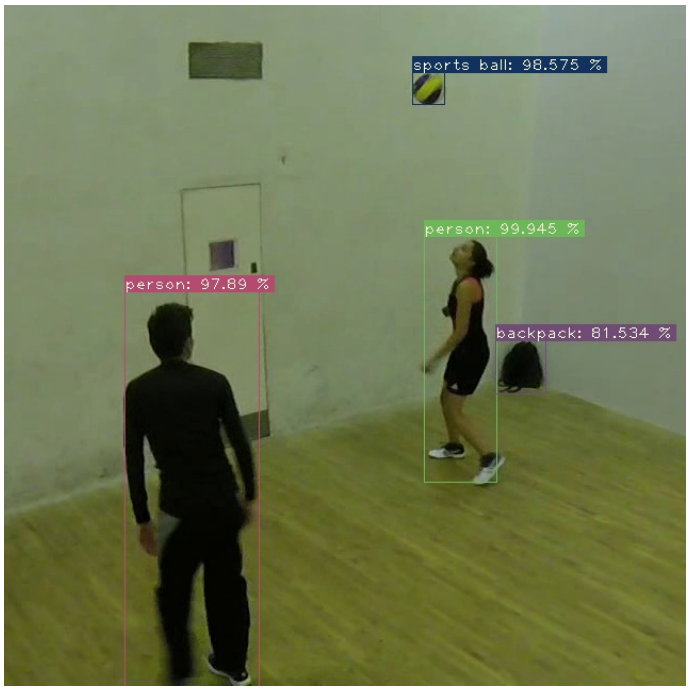
Dans un premier temps, étant donné que le modèle de détection doit être à la fois précis et rapide dans son exécution, les outils de détection en temps réel constitue la première approche de ce projet. Selon l'état de l'art, le choix de l'outil YOLO était le plus adéquat. En effet, comme expliqué et montré sur la figure 2.3, l'architecture de YOLO lui permet d'effectuer les tâches de détection et de classification en parallèle, augmentant ainsi sa vitesse d'exécution.

La dernière version de YOLO : **YOLOv3-608** a été sélectionnée comme point de départ pour ce projet. Selon son article[43], cette version obtient les meilleures performances en terme de précision tout en gardant une bonne vitesse d'exécution.
Cette version prend en entrée de l'algorithme des images de dimensions 608×608 et réalise un temps de détection de 51ms par image avec, comme processeur graphique (GPU), la "TITAN X"[43]. Ces résultats de performances ont été réalisés en utilisant la base de données COCO [28]. COCO contient un ensemble d'images regroupées en 80 catégories d'objets. Le réseau de neurones de YOLOv3 a donc déjà été entrainé avec COCO et le modèle final a été rendu publique sur le site internet de J. Redmon qui est le concepteur de YOLO [41]. Ainsi cette version, comme toutes les autres versions de YOLO, possède un modèle pré-entrainé accessible à tous. La base de données COCO utilise, pour tester les performances de détection d'un programme, un ensemble de 40670 images.

L'algorithme de A. Kathusia [93], nous a permis d'utiliser YOLOv3 dans le langage de programmation souhaité à savoir Python, puisqu'il a implémenté YOLOv3 avec PyTorch. Nous avons ensuite cherché à réaliser une comparaison de cet algorithme avec celui de l'article [43] sur leur rapidité d'exécution. La comparaison de performances sur la précision de détection n'est pas nécessaire ici car nous utilisons le modèle pré-entrainé donnant les résultats de l'article. Ainsi, les seules performances susceptibles de changer sont celles en termes de vitesse d'exécution qui sont directement liées à l'ordinateur et au langage de programmation utilisés et non au modèle.

Pour les tests de performances, nous les avons réalisés d'une part sur un ensemble de 1000 images, suffisant pour y effectuer une moyenne, de la base de données test de 2017 de COCO [94] et d'autre part sur des images vidéos de dimensions 640×640. La figure 4.2 montre un exemple des tests sur les différents supports utilisés.

(a) Détection réalisée sur une des images de la base de données de 2017 de COCO[94]



(b) Détection réalisée sur image vidéo à partir d'un support vidéo personnel

Figure 4.2 Détection d'objets sur différents supports avec YOLOv3

Pour chaque type de support, nous avons cherché à obtenir le nombre d'images par seconde. Ce nombre est directement lié à la durée de traitement, pour la détection des objets, par image. Cette relation est décrite par l'équation 6.1, suivante :

$$fps\_obtenu = \frac{1}{X} \tag{4.1}$$

Avec "X" le temps de détection pour une image.
Dans ces 2 cas, nous avons déterminé le nombre de fps : un avec l'affichage de l'image et un sans. Sans l'affichage, cela correspond au résultat que l'on retrouve dans les articles. Cependant dans ce cas nous n'avons pas de retour d'informations quant à la bonne détection ou non des objets. Pour la conception du modèle de détection, il nous est nécessaire d'avoir un retour d'informations pour s'assurer de son bon fonctionnement.

Le tableau 4.1 regroupe l'ensemble des résultats obtenus ainsi que les résultats de l'article afin de réaliser une comparaison entre les deux.

Tableau 4.1 Comparaison des résultats obtenus avec l'algorithme utilisé et ceux de l'article

	Article	**Ensemble d'images de "COCO"**	**Images vidéos** (1000 images)
fps (sans affichage)	19.6	14.29	15.79
fps (avec affichage)	N/A	11.36	14.09
Comparaison (sans affichage)	100%	73%	81%

Les résultats de la comparaison faite dans le tableau 4.1, montrent que nous avons, pour l'ordinateur et la version de YOLO utilisés, une vitesse de détection plus lente de 27% pour le cas d'une base de données similaire. Cette différence notable est due au fait que la version utilisée dans ce projet est une adaptation sur Python. Le code original de YOLOv3 est réalisé avec le langage de programmation C++, qui est plus rapide dans l'exécution des programmes que Python [95, 96]. Ensuite, pour ce qui est de la détection à partir d'une vidéo (voir figure 4.2b), nous obtenons un meilleur résultat car dans ce cas la différence avec l'article est seulement de 19%. Cette différence est due au fait que dans la base de données de COCO, les images peuvent contenir beaucoup d'objets allant parfois à plus de 20 objets. Alors que

dans la vidéo sélectionnée, le nombre maximum de détection d'objets est limité à 4. Ainsi, le temps de détection moyen par image sera plus petit pour le cas de la vidéo d'où un meilleur nombre de fps. Enfin, pour ce qui est de la comparaison entre les nombres de fps avec et sans affichage, nous observons que le nombre de fps est meilleur sans l'affichage. Ce résultat est sans appel puisque l'affichage des images augmente le temps d'exécution pour chaque image traitée. Cependant grâce à ce retour d'informations nous pouvons déterminer la qualité de la détection : détection ou non des objets d'intérêts.

Nous pouvons ainsi en conclure qu'avec l'algorithme YOLOv3-608, nous avons une détection en temps réel puisque, dans chacun des cas, nous nous trouvons avec des valeurs de fps supérieures à 5. Il est important d'ajouter que nous pourrions augmenter le nombre de fps en réduisant la taille des images en entrée du réseau de neurones mais cela entrainerait une baisse de la précision des détections. Dans ce choix de dimensions d'image, nous réalisons à la fois une bonne précision tout en gardant une vitesse d'exécution satisfaisante. À la suite de ces tests de performances, l'objectif suivant était d'entrainer le réseau de neurones de YOLOv3 pour qu'il puisse détecter les classes d'objets que nous avons défini dans la section précédente. Le projet réalisé en [97] nous a aidé dans la réalisation de cette étape.

Pour obtenir le meilleur support de perchage, nous avons choisi de travailler sur les dimensions des objets. Cependant pour YOLOv3-608, l'accès aux boîtes englobantes n'est pas suffisant pour les obtenir. Grâce à ces boîtes nous avons accès à leur taille, à la position des objets d'intérêts dans les images vidéos mais nous avons aucunes informations précises sur les objets eux-mêmes. En effet, la taille des boîtes englobantes nous donnes qu'une dimension grossière des objets avec une marge d'erreur importante, comme montré par la figure 4.3.

Pour réaliser une comparaison avec les caractéristiques du système de préhension, il est donc indispensable d'avoir accès aux dimensions réelles des objets détectés. Pour cela, il faut d'abord connaître de façon précise leurs dimensions dans l'image. Puis à partir des caractéristiques de la caméra utilisée et de la distance entre celle-ci et l'objet nous pouvons arriver à déterminer les dimensions réelles et ainsi réaliser cette comparaison. Cette comparaison nous permettrait de définir l'objet le plus adéquat pour le perchage. La partie 2 de ce mémoire est dédiée à la conception d'un algorithme qui effectue cette comparaison. Nous reviendrons ainsi sur ce sujet et cela de manière plus détaillée dans cette partie.

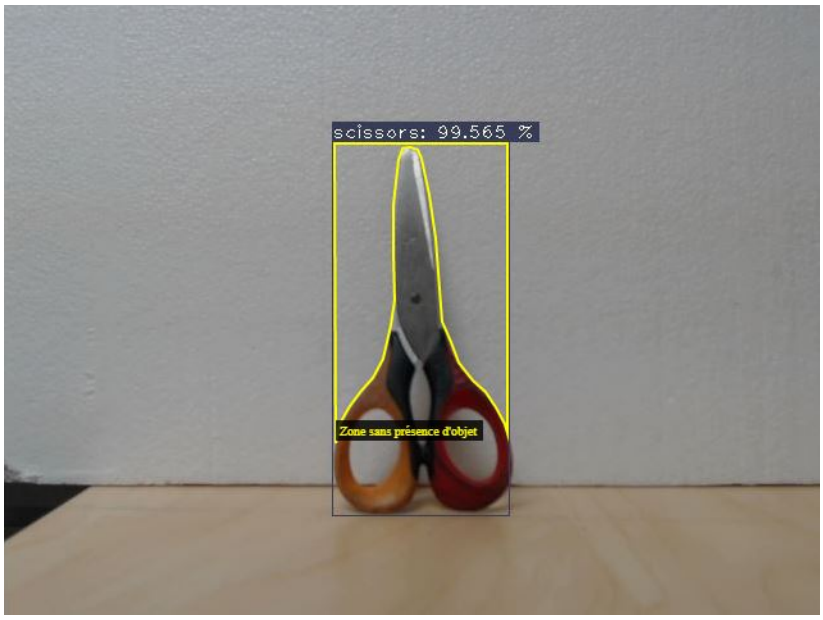


Figure 4.3 Perte d'informations sur les dimensions d'objet avec une détection seulement faite par encadrement

Pour accéder aux dimensions des objets dans l'image, la méthode de la segmentation sémantique apparaît comme la plus adéquate. En effet, la segmentation des objets permet de détourer l'objet du reste de l'image, assurant ainsi l'accès aux dimensions de celui-ci. Cependant, dans certain des cas, la segmentation sémantique seule n'est pas suffisante. Par exemple, la segmentation sémantique ne peut pas dissocier les objets de même catégorie entre eux. De ce fait, il faut donc opter pour un autre type de segmentation : la segmentation d'instances qui représente la fusion entre la détection d'objet par boîtes englobantes et la segmentation sémantique.

Ainsi pour avoir une segmentation d'instances, nous avions deux possibilités : soit utiliser YOLOv3 et y ajouter une phase réalisant la segmentation sémantique des objets après leur détection ou soit utiliser un programme de détection basé directement sur la segmentation d'instances comme Mask-RCNN. Chacune de ces possibilités présente un avantage et un inconvénient :

1. **YOLOv3 + segmentation sémantique**
    - Avantage : Bonne vitesse d'exécution (20 fps [43])-> détection en temps réel
    - Inconvénient : Segmentation basique et très approximative

2. **Mask-RCNN : segmentation d'instance**
    - Avantage : Précision de détection et de segmentation performantes
    - Inconvénient : Vitesse d'exécution lente (5 fps [40])-> limite d'une détection en temps réel

Dans la première possibilité, si nous voulons garder une vitesse de détection relativement intacte par rapport au cas sans segmentation, nous devons utiliser une segmentation basique i.e sans l'utilisation d'un réseau de neurones. Cependant une segmentation basique[52] réalisera une segmentation approximative des objets présents dans l'image et sera très sensibles aux changements d'environnement. Dans le cas d'un outil de détection basé sur la segmentation d'instance, ce dernier possède une architecture optimisée pour obtenir à la fois une bonne détection comme une bonne segmentation même si la vitesse de détection se voit diminuer par rapport à celle de YOLOv3.

Étant donné que l'objectif de ce projet de recherche est d'obtenir le meilleur support, parmi un certain nombre d'objets, la précision de détection ainsi que la segmentation sont plus importantes dans ce cas que la vitesse d'exécution. De plus, il est toujours possible d'ajouter d'autres GPU et donc d'augmenter la vitesse d'exécution du programme. C'est pour cette raison que nous avons opté pour la deuxième possibilité à savoir l'utilisation d'un outil de détection basé sur la segmentation d'instances. Selon l'état de l'art de ce mémoire, l'outil le plus adapté est Mask-RCNN du fait qu'il possède dans son réseau de neurone deux parties : une pour la détection et la classification d'objet avec l'utilisation de boîtes englobantes et une pour la segmentation avec l'utilisation de masques. Le choix de l'utilisation de Mask-RCNN pour la suite de ce projet, plutôt que de YOLOv3-608, a été fait.

4.3 Conception du modèle à partir de Mask-RCNN

Mask-RCNN est un outil de détection utilisant la segmentation d'instances [40], ce qui est très pratique si nous avons besoin de séparer chaque objet d'intérêt du reste de l'image (autres objets et l'arrière plan). Comme expliqué dans l'état de l'art de ce mémoire, Mask-RCNN utilise une fenêtre glissante qui passe au travers de l'ensemble de l'image pour détecter les objets souhaités. C'est un processus qui obtient une meilleure précision de détection mais dont le temps d'exécution est plus lent que la méthode utilisée par YOLO. Ayant choisi d'utiliser Mask-RCNN, le projet de A. Waleed [98] a été sélectionné comme base de création du modèle de détection. Cette version de Mask-RCNN implémenté en Python utilise comme librairie d'apprentissage profond : Tensorflow.

4.3.1 Architecture de Mask-RCNN

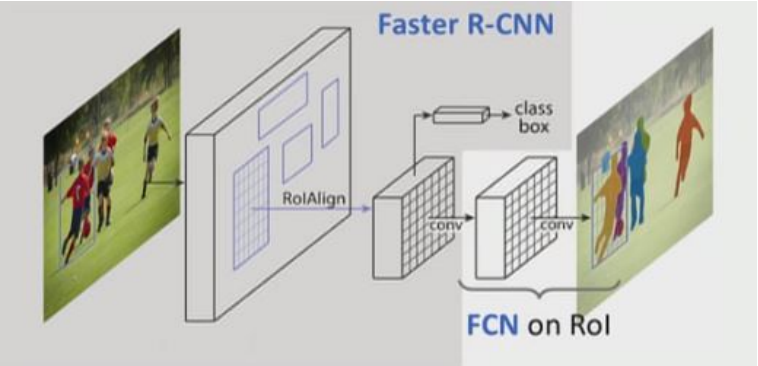


Figure 4.4 Architecture simplifiée de Mask-RCNN [4]

Comme le montre la figure 4.4, l'architecture de Mask-RCNN comprend 2 branches en parallèle :

- Une branche avec un réseau de neurones(CNN), qui effectue la détection par boîtes englobantes et la classification d'objets. Cette branche constitue la version précédente du programme soit Faster-RCNN[38]
- Une branche avec un réseau de neurones entièrement connecté(FCN), qui effectue la segmentation sémantique des objets (comme avec l'article[53])

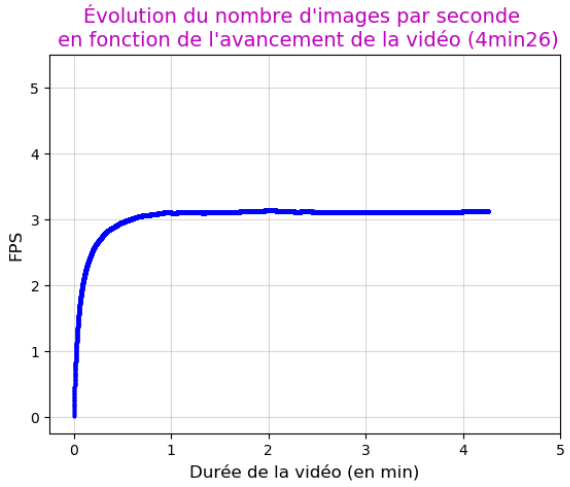
Cette architecture permet ainsi d'avoir à la fois la position des objets dans l'image mais également leurs masques grâce à la segmentation sémantique. De ce fait, cette architecture permet d'avoir, comme informations sur les objets détectés :

1. Par la classification et les boîtes englobantes
    - Connaître sa catégorie
    - Isoler chaque objet du reste de l'image
2. Par la segmentation sémantique soit l'accès au masque de chaque objet
    - Connaître sa forme
    - Connaître ses dimensions générales comme locales

4.3.2 Tests de performance en terme de vitesse d'exécution de Mask-RCNN

Comme effectué avec le programme YOLOv3, nous avons tout d'abord réaliser un test de performance pour connaître la vitesse de traitement d'images vidéos de Mask-RCNN avec

l'ordinateur utilisé. Ici aussi, un modèle pré-entrainé sur la base de données de COCO est disponible publiquement. Ainsi nous l'avons utilisé afin d'avoir un modèle déjà entrainé, un réseau de neurones optimisé et également de pouvoir faire une comparaison sur les mêmes données que lors du test sur YOLOv3. Ensuite, étant donné que l'algorithme Mask-RCNN de A. Waleed [98] est fait pour la détection d'images, nous avons sélectionné un algorithme adapté pour les vidéos et les caméras. Ce dernier est un projet réalisé par M. Jay [99]. Pour les tests de performances de vitesse de détection avec une vidéo, la même vidéo qu'avec YOLOv3 a été reprise. Nous avons pu remarquer pendant ces tests que, contrairement à YOLOv3 où le nombre de fps atteint rapidement sa valeur limite, le modèle de Mask-RCNN met du temps à s'établir dû au fait que son temps de traitement est 4 fois plus lent en comparaison des valeurs obtenues dans leurs articles respectifs : 20 fps pour YOLOv3 et 5 fps pour Mask-RCNN.



(a) Performance de Mask-RCNN en terme de vitesse de détection sur image vidéo

(b) Détection réalisée sur image vidéo à partir d'un support vidéo personnel

Figure 4.5 Performances de détection de Mask-RCNN sur images vidéo

À partir des résultats obtenus sur la figure 4.5a, nous avons construit le tableau récupitulatif 4.2 suivant :

Tableau 4.2 Comparaison des résultats obtenus sur images vidéos entre les algorithmes utilisés : Mask-RCNN et YOLOv3-608, et ceux de la littérature

	YOLOv3-608	**Mask-RCNN**
Article	20 fps	5 fps
Algorithme utilisé	14.09 fps	3.10 fps
Rapport Algorithme/Article	70%	62%

À partir du tableau 4.2, nous observons que la perte en temps d'exécution est un peu plus grande pour l'algorithme de Mask-RCNN que pour YOLOv3-608. Cette différence provient de la quantité de ressources en calculs nécessaire. Le réseau de neurones de Mask-RCNN se compose de 2 parties qui représente chacune un réseau de neurones, comme le montre la figure 4.4. Quant à lui, YOLOv3 possède un réseau de neurones composé en une seule partie. Par conséquent, YOLOv3 perdra moins de temps d'exécution que Mask-RCNN sur la comparaison entre les algorithmes "open source" et ceux des articles.

De plus, il est difficile de faire une bonne comparaison car nous ne connaissons pas la configuration utilisé, à savoir la valeur des différents paramètres, dans les articles [100, 43] pour Mask-RCNN et YOLOv3, respectivement. Par exemple pour Mask-RCNN, les paramètres PRE_NMS_LIMIT et POST_NMS_ROIS_INFERENCE ont une grande sensibilité sur la vitesse de détection mais au détriment de la précision de détection. Nous reviendrons, en fin de chapitre, sur ces deux paramètres pour les expliquer en détails et les modifier dans le but d'augmenter la vitesse de détection de Mask-RCNN tout en minimisant la perte de précision. Pour remplir le premier sous-objectif de ce projet de recherche, il devient donc nécessaire d'améliorer la vitesse de détection actuelle. En effet, un résultat de 3.1 fps ($\leq$ 5 fps) signifie que le modèle de Mask-RCNN ne permet pas une détection en temps réel. Une autre méthode pour améliorer la vitesse d'exécution de notre outil serait d'utiliser plusieurs "GPU".

Pour ce qui est des performances de vitesse, YOLOv3 qui atteint un nombre de fps égale à 14 est 4.5 fois plus rapide que Mask-RCNN, et donc le surpasse largement dans ce domaine. En revanche, nous pouvons constater en regardant la figure 4.5b que la précision de détection est meilleure avec Mask-RCNN puisque les scores de détection sont généralement meilleurs mais aussi dans la précision de la position des boîtes englobantes. Enfin le dernier aspect, et non des moindres, nous avons une segmentation précise des objets détectés.

4.3.3 Établissement du modèle de détection pour le perchage de drone

L'étape suivante consiste à entrainer Mask-RCNN afin qu'il puisse détecter les catégories d'objets choisies pour le perchage de drone. Pour son entrainement, le choix d'utiliser le modèle pré-entrainé "mask_rcnn_coco" a été fait dans le but d'avoir une architecture optimisée mais aussi pour avoir un entrainement plus rapide, comme énoncé dans le chapitre 5 de ce mémoire. De plus, comme il n'existe pas de base de données pour ces classes d'objets, la base de données pour l'entrainement de Mask-RCNN sera de petite taille. Sans l'utilisation d'un modèle pré-entrainé, une petite base de données ne peut fournir assez d'informations pour compléter un entrainement d'un réseau de neurones aussi complexe que celui de Mask-RCNN. La base de données utilisé dans ce projet contient : d'une part des images provenant du site PxHere[101] qui offre des images gratuites libre de droit et, d'autre part d'images prises directement dans la ville de Montréal. Une fois cette base de données entièrement constituée, nous l'avons décomposée en trois ensembles de données :

- Un ensemble de données pour l'entrainement ($\approx$ 80% de la base de données)
- Un ensemble de données pour la validation des paramètres du modèle à la suite d'une itération d'entrainement ($\approx$ 10% de la base de données)
- Un ensemble de données pour évaluer le modèle final ($\approx$ 10% de la base de données)

L'ensemble de données pour l'entrainement se compose de 75 images pour chaque catégorie d'objet. Afin d'avoir de meilleurs résultats de détection mais aussi une meilleure robustesse, une augmentation de cet ensemble de données a été faite. Une augmentation signifie, dans le domaine de l'apprentissage profond, une modification des images originales en réalisant divers opérations telles que :

- Rotation de l'image
- Changement de contraste
- Ajout de bruit
- Flouter l'image
- Prendre des parties d'image

Dans ce projet, nous avons choisi d'effectuer seulement une rotation de 90° et une modification du contraste pour rendre l'image originale à la fois plus sombre et à la fois plus claire. Ces choix s'appuient notamment sur les conditions de survol du drone en extérieur. En effet, les objets peuvent apparaître parallèles comme perpendiculaires à la direction de vol du drone et aussi, selon la météo le contraste peut varier rapidement. Cela permettra par la suite, durant

les tests, d'évaluer la robustesse du programme de détection face à ces changements. Ainsi pour une rotation et deux changements de contraste nous obtenons pour une image originale : 6 images d'entrainement. Au final, l'ensemble de données pour l'entrainement comptabilise un total de :

$$75\ \frac{images}{classe} \times 4\ classes\ d'objet \times 6 = 1800\ images \tag{4.2}$$

Ensuite, pour ce qui est de l'ensemble de données pour la validation : il se compose de 200 images originales. Cet ensemble n'a pas d'augmentation car comme le réseau de neurones ne réalise pas d'apprentissage avec celui-ci, cela n'a donc pas d'utilité. Enfin, pour le dernier ensemble de données, celui pour l'évaluation du modèle final : il se compose de 233 images. Ici aussi, aucune augmentation n'a été faite sur les images.
Pour résumé, la base de données utilisée pour l'entrainement du réseau de neurones de Mask-RCNN est composée d'un total de 2233 images. Cela reste faible pour le domaine de l'apprentissage profond mais l'utilisation d'un modèle pré-entrainé permet d'assurer le bon apprentissage du réseau [24].

Pour l'entrainement, les images des différents ensembles de données ont été redimentionnées en 640×512 dans le but de diminuer la durée d'entrainement et aussi le temps de détection par image. Il est important de noter que pour réaliser l'entrainement mais aussi pour la détection, le réseau de neurones de Mask-RCNN doit avoir en entrée des images ayant des dimensions divisible par 64 [100].

4.3.3.1 Détails des principaux paramètres d'entrainement de Mask-RCNN

Pour l'entrainement du réseau de neurones de Mask-RCNN, plusieurs paramètres rentrent en jeu.

Tableau 4.3 Principaux paramètres d'entrainement de Mak-RCNN

Paramètres d'entrainement	**Définition**
Type de couches du réseau	Pour Mask-RCNN, il est possible d'entrainer les premières, les suivantes ou la totalité des couches. Les premières couches du réseau permettent la détection de caractéristiques d'objets de bas niveau comme les bords, les coins. Les autres permettent la détection de caractéristiques de haut niveau comme une partie d'objet ou un objet entier.
Nombre d'epochs par type de couches : variable "EPOCHS"	Une "epoch" correspond, ici, à une itération d'apprentissage réalisée avec les ensembles d'entrainement et de validation. Ainsi chaque epoch est composée d'une succession d'itérations d'entrainement puis à la fin de ces dernières d'une succession d'itérations de validation.
Nombre d'itérations d'entrainement et de validation par epochs : variables "STEPS_PER_EPOCHS", "VALIDATION_STEPS"	Le nombre d'itération d'entrainement correspond au nombre d'images d'entrainement analysées : une image = une itération. De même pour le cas de la validation. Ces deux paramètres possèdent un maximum qui correspond à la taille de l'ensemble de données utilisé. Si cette limite est dépassée, le réseau sera entrainé sur les mêmes images provoquant une perte de performance pour la détection.

Paramètres d'entrainement	Définition
Nombre et taille des ancres par image entrainée : variable "RPN_TRAIN_ANCHORS_PER_IMAGE"	Les ancres ("anchors" en anglais) sont des propositions de régions, représentées par des boîtes 2D, pouvant contenir un objet. Leur score de confiance, associé à chacune d'elle, indique la probabilité de l'évènement. Les propositions de régions constituent le premier stage des réseaux de neurones de type RCNN.
Nombre de régions d'intérêt (ROI) par image entrainée : variable "TRAIN_ROIS_PER_IMAGE"	Le deuxième stage du réseau de neurones coïncide avec la classification et l'obtention des boîtes englobantes pour chaque objet détecté. Cette phase prend, en entrée, les propositions de régions. À partir des positions de celles-ci dans l'image, de nouvelles régions vont être créées au hasard à partir de ces positions : ce sont les régions d'intérêts.
Seuil pour les propositions de régions : variable "RPN_NMS_THRESHOLD"	Sa valeur varie entre 0 et 1.0. Pour une valeur donnée, seules les propositions de régions ayant un score de confiance supérieur au seuil sont sélectionnées.

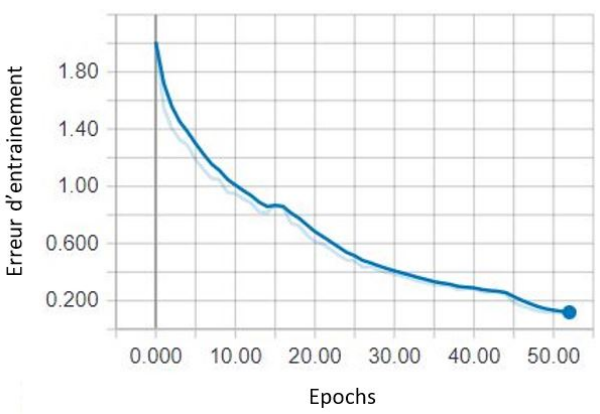
4.3.3.2 Évaluation des performances du modèle ayant les valeurs de paramètres par défaut

Afin d'avoir un modèle de départ et à fortiori un modèle de comparaison pour arriver à un meilleur modèle de détection, une sélection de valeurs par défaut a été faite. La configuration par défaut est la suivante :

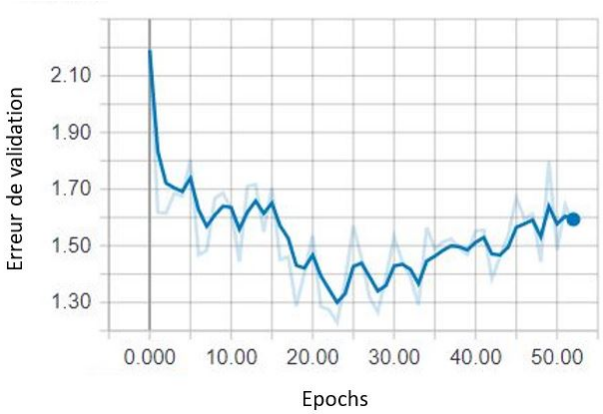
- EPOCHS = 15
- STEPS_PER_EPOCHS = 800
- VALIDATION_STEPS = 50
- RPN_TRAIN_ANCHORS_PER_IMAGE = 256
- TRAIN_ROIS_PER_IMAGE = 200
- RPN_NMS_THRESHOLD = 0.70

Les valeurs des 4 derniers paramètres sont les mêmes que celles données dans le projet proposé par A. Waleed [98]. La valeur des autres paramètres a été diminuée dans le but d'avoir des apprentissages moins longs. Durant l'apprentissage du réseau de neurones de Mask-RCNN,

chaque "epoch" (définition tableau 4.3) et ses caractéristiques sont sauvegardées. Cela permet de tracer, à l'aide de "TensorBoard", les différentes courbes d'analyses de l'apprentissage. "TensorBoard" est une boîte d'outil de visualisation relié à "TensorFlow". Ces courbes donnent accès à diverses informations pour savoir si le modèle est bien entrainé ou non. Les principales courbes sont celles montrant l'erreur d'entrainement et l'erreur de validation. Pour le modèle par défaut choisi, ces courbes sont données par les figures 4.6a et 4.6b respectivement.



(a) Courbe représentant l'erreur d'entrainement (Erreur en fonction des epochs)



(b) Courbe représentant l'erreur de validation (Erreur de validation en fonction des epochs)

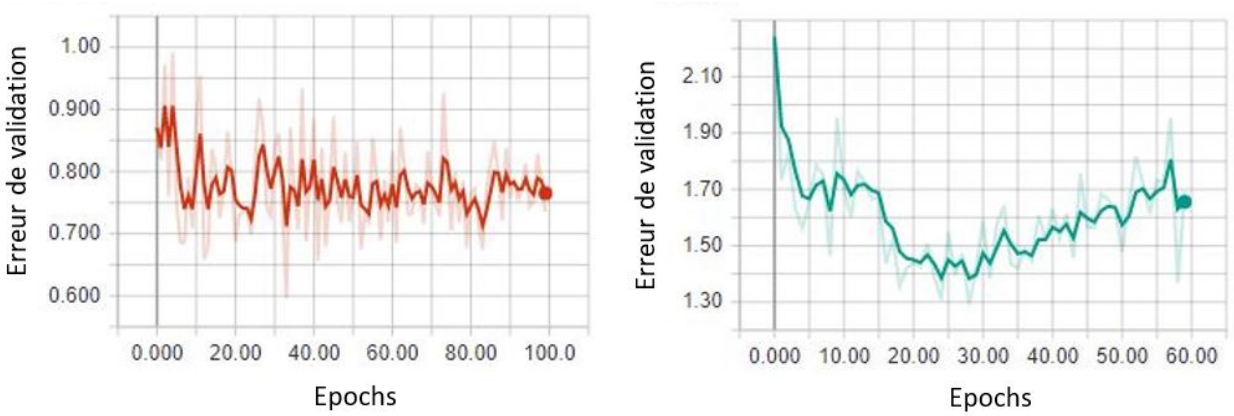
Figure 4.6 Courbes d'informations sur l'apprentissage du réseau de neurones de Mask-RCNN pour la configuration par défaut

Pour savoir si l'entrainement d'un réseau de neurones est bon, il faut que la courbe donnant l'erreur d'entrainement (figure 4.6a) diminue tout au long de l'entrainement jusqu'à tendre vers 0. La courbe donnant l'erreur de validation (figure 4.6b) nous informe sur la qualité du modèle. Les meilleurs performances de chaque apprentissage réalisé sont obtenues pour les points correspondant au minimum de la courbe.

Chaque apprentissage du réseau, avec les nouvelles catégories d'objets, a été réalisé avec la structure générale suivante :

1. Entrainement des premières couches sur "X" epochs : "X" correspondant à la valeur du paramètre "EPOCHS" ;
2. Entrainement de toutes les couches sur "n" $\times$ "X" epochs : "n" = [3 ou 4] selon la configuration sélectionnée.

Tout d'abord, les premières couches étant de bas niveau, leurs paramètres (poids des neurones et relation entre neurones) sont plus sensibles de varier lors d'entrainement fait à partir de nouvelles catégories d'objets. Cela permet d'avoir une variation notable dans la courbe représentant l'erreur de validation. Ensuite, nous avons remarqué que dans le cas où l'ensemble de l'entrainement est réalisé sur toutes les couches, il est difficile de repérer le minimum de la courbe car elle reste relativement constante. La structure proposée a été choisie à la suite de ces deux remarques constatées lors des premiers essais. La figure 4.7 témoigne de cette différence entre les deux structures.



(a) Courbe d'erreur de validation dans le cas d'un apprentissage avec seulement l'ensemble des couches

(b) Courbe d'erreur de validation dans le cas d'un apprentissage avec tout d'abord les premières couches puis l'ensemble des couches

Figure 4.7 Comparaison des courbes d'erreur de validation obtenues entre deux structures d'entrainement différentes

Grâce à la structure sélectionnée, nous obtenons ainsi, pour l'erreur de validation, une courbe de forme parabolique permettant l'accès à son minimum sans difficulté. Chaque point de courbe correspond ici à une epoch. Lorsque cette courbe commence à augmenter, cela signifie que nous sommes en présence d'un **sur-apprentissage**. Arrivé à ce stade, le réseau a passé en revue la totalité de la base de donnée d'entrainement à plusieurs reprises, passant alors à un apprentissage défini de : "par coeur". Ce phénomène est à éviter puisse qu'il fait diminuer les performances de détection du modèle[102], c'est pour cela qu'il faut choisir les points/epochs se trouvant avant ce phénomène.

Une fois l'apprentissage terminé, nous avons tracé les courbes "Précision-Rappel" pour chacune des epochs sélectionnée correspondant au minimum des courbes. La courbe "PR" pour le modèle avec la configuration par défaut est présentée par la figure 4.8.

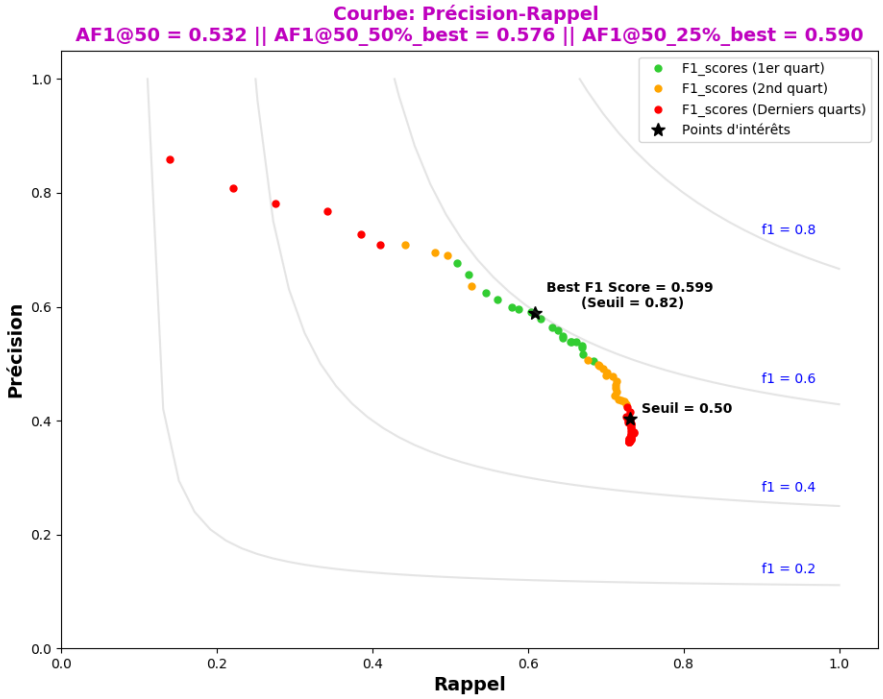


Figure 4.8 Courbe Précision-Rappel pour le modèle avec la configuration par défaut

La courbe "Précision-Rappel" permet d'évaluer la performance d'un modèle. Elle donne la précision d'un modèle en fonction de son rappel.

- Le paramètre "Rappel" correspond au pourcentage d'objets correctement détectés sur l'ensemble des objets qu'il fallait détecter dans la base de données
- Le paramètre "Précision" correspond au pourcentage d'objets correctement détectés sur l'ensemble des objets effectivement détectés

Pour savoir précisemment à quoi ils sont dûs et comment les calculer, la référence présentée en [22] concernant les mesures de performances traite ces informations.

Pour obtenir cette courbe, ces deux paramètres sont déterminés pour chaque variation du seuil de détection. Dans l'algorithme utilisé de Mask-RCNN , le seuil de détection correspond

à la variable nommée "DETECTION_MIN_CONFIDENCE". Le seuil de détection, variant de 0 à 1.0, signifie qu'en dessous de sa valeur, toutes les détections ayant un score de confiance inférieur ne sont pas affichées. Dans ce projet, le seuil de détection varie entre 0.20 et 0.99 avec un pas de 0.01. Étant donné qu'un score exact de 1.0 n'existe pas (détection parfaite), aucune détection d'objet ne sera affichée sur l'image finale. De plus, les scores de confiance affichés sont des scores arrondis i.e qu'il est possible d'avoir des scores, affichés sur l'image, égaux à 1.0 mais en réalité leurs valeurs sont légèrement inférieures. De ce fait, le tracé de la courbe commence à partir de 0.99.
Ensuite, dans cette étude, nous totalisons un ensemble de 5 classes composé en 4 classes d'objets et 1 classe correspondant à l'arrière-plan i.e le cas où il n'y a pas de détection. Le score de confiance est donc réparti entre les classes pour obtenir par addition un score maximum de 1.0. Par conséquent, cela signifie qu'une détection peut avoir au minimum un score de 0.20 (1.0 divisé par 5). Pour un seuil de détection compris entre 0 et 0.20, les paramètres de precision et de rappel auront donc la même valeur. Pour ces valeurs, les points obtenus deviennent non pertinent pour le tracé de la courbe. Au final, cette courbe comptabilise un total de 80 points pour son tracé.

Une fois la courbe tracé, nous pouvons déterminer le paramètre "F1_score" à partir des paramètres "Précision" et "Rappel". Ce dernier se calcule selon la formule suivante[22] :

$$F1\_score = 2 \times \frac{précision \times rappel}{précision + rappel} \tag{4.3}$$

Ce paramètre constitue l'information la plus importante dans l'évaluation d'un modèle car il prend en compte sa précision et son rappel. Plus le F1_score est élevé et plus le modèle est performant. Pour chaque point de la courbe le F1_score a été calculé. Puis, à partir des F1_scores calculés, nous avons créé 4 paramètres afin de caractériser le modèle :

1. Le meilleur F1_score de la courbe
2. Le F1_score moyen de l'ensemble des points (80 points)
3. Le F1_score moyen de la meilleure moitié (40 meilleurs points)
4. Le F1_score moyen du meilleur quart (20 meilleurs points)

À partir de ces 4 informations, nous pouvons déterminer les points forts et les points faibles d'un modèle. Le tableau 4.4, ci-dessous, regroupe les différentes informations obtenues selon le paramètre F1_score traité.

Tableau 4.4 Informations fournies sur le modèle selon le F1_score moyen obtenu

	Informations sur le modèle
Meilleur F1_score	Valeur du seuil de détection pour laquelle le modèle obtient la meilleure performance de détection.
F1_score moyen de l'ensemble des points	Performance générale du modèle.
F1_score moyen de l'ensemble de la meilleure moitié	Performance du modèle en enlevant les mauvais résultats dus aux valeurs de seuil extrêmes : ≥ 0.95 et ≤ 0.50.
F1_score moyen de l'ensemble du meilleur quart	Performance du modèle sur ses meilleurs résultats.

Étant donné que nous cherchons à obtenir la meilleure précision de détection possible, nous privilégierons donc un modèle ayant de meilleures valeurs pour les paramètres 1,3 et 4 que pour le paramètre 2. En effet, les paramètres 1,3 et 4 se basent directement sur les meilleurs F1_score du modèle et à fortiori sur la meilleure détection que peut apporter le modèle.

4.3.4 Meilleur modèle de détection

Une fois le choix d'un modèle de base effectué, l'objectif suivant a été de trouver un meilleur modèle afin d'avoir un plus grand nombre de détections d'objets mais aussi une meilleure précision de détection. Le fait d'avoir une meilleure précision permet d'augmenter les chances d'avoir une meilleure qualité de masque d'objet. Face aux grands nombres de configurations possibles et dû au fait que nous avons 6 paramètres à gérer, et qui en plus ont un éventail de valeur conséquent, nous avons cherché à obtenir non pas le meilleur modèle possible mais un bon modèle de détection qui possède des performances plus élevées que celui de base. Sachant qu'un entrainement du réseau de neurones de Mask-RCNN prend en moyenne 9 heures pour être effectué et que pour chaque epoch sélectionnée il faut 3 heures pour obtenir la courbe "Précision-Rappel", nous avons choisi de nous limiter dans le nombre de configurations pour éviter de passer trop de temps à trouver une excellente configuration.

L'ensemble des simulations, des paramétrages et des résultats effectués dans ce projet pour arriver à un meilleur modèle de détection est détaillé dans l'annexe A.

4.3.4.1 Évaluation des performances du nouveau modèle et comparaison avec celles du modèle de base

Une fois l'ensemble de simulations réalisé, nous avons pu aboutir à un meilleur modèle de détection ayant des performances supérieures à la configuration de base.

La nouvelle configuration obtenue pour ce modèle est la suivante :

- EPOCHS = 15
- STEPS_PER_EPOCHS = 1200
- VALIDATION_STEPS = 100
- RPN_TRAIN_ANCHORS_PER_IMAGE = 512
- TRAIN_ROIS_PER_IMAGE = 50
- RPN_NMS_THRESHOLD = 0.70

Cette configuration concerne les paramètres d'entrainement du réseau de neurones. Pour le nouveau modèle de détection, la courbe "Précision-Rappel" correspondante est celle présentée sur la figure 4.9.

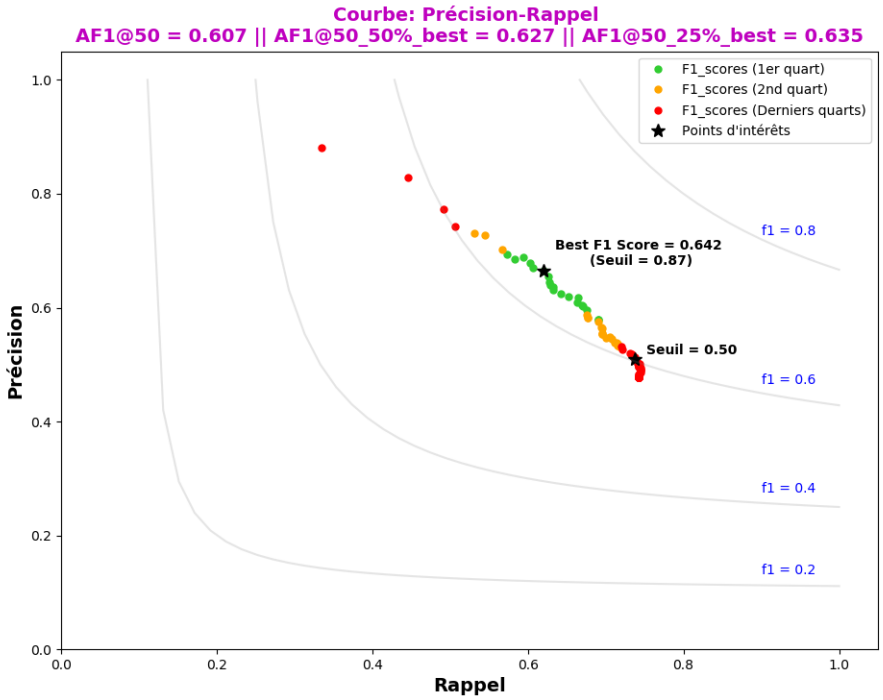


Figure 4.9 Courbe Précision-Rappel pour le modèle avec la configuration finale

Pour ce qui est de la comparaison des résultats de performances entre les deux modèles à savoir ceux des figures 4.8 et 4.9, nous pouvons nettement voir une amélioration des performances. Pour les différents paramètres sur le F1_score nous avons :

- Une augmentation du F1_score moyen de l'ensemble des points de 14.1%
- Une augmentation du F1_score moyen de la meilleure moitié (40 meilleurs points) de 8.9%
- Une augmentation du F1_score moyen du meilleur quart (20 meilleurs points) de 7.6%
- Une augmentation de la valeur du meilleur F1_score de la courbe de 7.2%

Ainsi pour chaque paramètre qui caractérise le modèle nous obtenons une amélioration et donc à fortiori de meilleures performances de détection.

Il reste cependant un point à souligner : le paramètre "Précision" est affecté par les mauvaises classifications, par exemple : si l'outil de détection se trompe de classe d'objet quand il détecte un bon objet, la valeur de ce paramètre diminue. Or dans le cas du perchage de drone, le seul point important est la bonne détection d'un objet et non sa classification. En effet, nous souhaitons que le drone se perche sur le meilleur support parmi les objets détectés. Ainsi, dans le cas où il détecte bien un objet d'intérêt mais qu'il le classifie mal, cela sera compté comme un succès. Par conséquent, les résultats obtenus pour les courbes "PR" seraient encore meilleurs si nous tenions compte de cela. Cependant, l'amélioration du modèle a été réalisé en utilisant à chaque fois le même procédé pour l'étude des performances (courbe "PR"), il n'est donc pas nécessaire de tracer d'autres courbe "PR" en prenant en compte cette remarque.

4.3.4.2 Exemples de détection avec les nouvelles classes d'objet

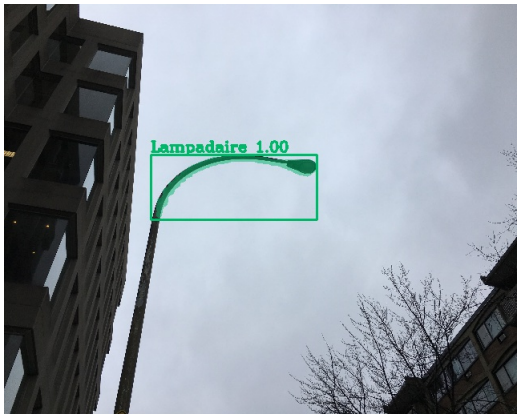
La qualité de la détection, du modèle final, a été vérifiée sur les images de l'ensemble de données d'évaluation pour chacune des classes d'objet. La figure 4.10 en dévoile un échantillon pour chacune des catégories d'objet.

4.3.5 Amélioration de la vitesse de détection du modèle

La vitesse de détection d'origine obtenue, avec l'outil Mask-RCNN [100, 98] pour des images vidéos, atteint les 3.10 fps soit 322ms par image. Afin d'atteindre une détection en temps réel nous avons cherché à améliorer cette vitesse. Tout d'abord, nous avons changé le cadre du test

(a) Classe 1 : Branche

(b) Classe 2 : Cylindre



(c) Classe 3 : Lampadaire

(d) Classe 4 : Barrière

Figure 4.10 Détection d'objets d'intérêts avec le modèle de détection final

pour venir nous placer au plus proche du cas réel de notre projet à savoir la détection réalisée par la caméra d'un drone. De ce fait, nous avons réalisé une détection directe pour un seul objet d'intérêt via une caméra avec une résolution d'image de 640×480. La nouvelle vitesse de détection pour ce test atteint les 4.22 fps soit 237ms par image. Ce résultat est supérieur à celui obtenu précédemment car d'une part nous avons des images avec des dimensions plus petites sur la hauteur et d'autre part il y a seulement 1 objet par image.

La nouvelle vitesse reste cependant inférieure au 5 fps visé. De plus, la détection des objets correspond seulement à la première partie de mon modèle complet. Par conséquent, le temps d'exécution augmentera avec l'ajout de la deuxième partie. Cette dernière traite les masques des objets détectés afin de déterminer la meilleure zone de préhension de ces derniers.

Comme énoncé brièvement auparavant, les deux paramètres suivants : "PRE_NMS_LIMIT" et "POST_NMS_INFERENCE" de Mask-RCNN influent sur la vitesse de détection. Ces derniers permettent de définir le nombre de régions d'intérêts ("ROI") que nous avons avant et après l'étape de "Suppression non-max". Cette étape permet d'enlever les ROIs qui se chevauchent et qui prédisent le même objet dans le but d'éviter la redondance. Ainsi, plus la valeur de ces paramètres est basse et moins il y aura de ROIs à traiter et à fortiori cela apportera un gain de temps de traitement. Cependant une diminution trop importante de leurs valeurs, peut entrainer une diminution de la précision de détection. Pour améliorer la vitesse de détection du modèle tout en conservant une bonne précision, nous avons établi, selon différentes valeurs pour les paramètres "PRE_NMS_LIMIT" et "POST_NMS_INFERENCE", les courbes "Précision-Rappel" et relevé la durée de traitement par image. Tous les résultats acquis ont été regroupés dans le tableau 4.5.

Tableau 4.5 Résultats des tests pour l'amélioration de la vitesse de détection

PRE_NMS_LIMIT / **POST_NMS_INFERENCE**	**6000**	**3000**	**2500**	**2000**
1000	237ms	222ms	191ms	157ms
900	211ms	N/A	N/A	N/A
800	192ms	N/A	186ms	N/A
700	171ms	N/A	169ms	N/A
600	155ms	N/A	145ms	N/A
500	132ms	N/A	129ms	N/A

En fonction des courbes "PR" obtenues, nous avons pu déterminer les couples de valeurs pour les deux paramètres qui permettaient de limiter le plus la perte de précision. L'ensemble des courbes "PR" obtenues est disponible en Annexe A dans la deuxième section.

Les résultats ont été relativement surprenant. Pour un "PRE_NMS_LIMIT" = 2500 et/ou un "POST_NMS_INFERENCE" = 800, nous obtenons de meilleures performances de détection. Tout d'abord, en terme de précision de détection, cette amélioration est notable en comparant les courbe "PR" de la figure 4.11a (obtenue pour ce couple de valeurs) et de la figure 4.8 (modèle final). Ensuite, en terme de temps de traitement, en comparant les valeurs obtenus soit un temps de 186ms par image au lieu des 237ms d'origine. Cette augmentation de précision pour les valeurs énoncées précédemment, signifie, dans ce cas, qu'une valeur trop

élevée pour ces paramètres provoque un surplus de régions d'intérêts pour un même objet. De plus, comme les classes d'objet sont moins fréquentes dans ce projet, il n'est pas nécessaire d'avoir autant de propositions de "ROI".

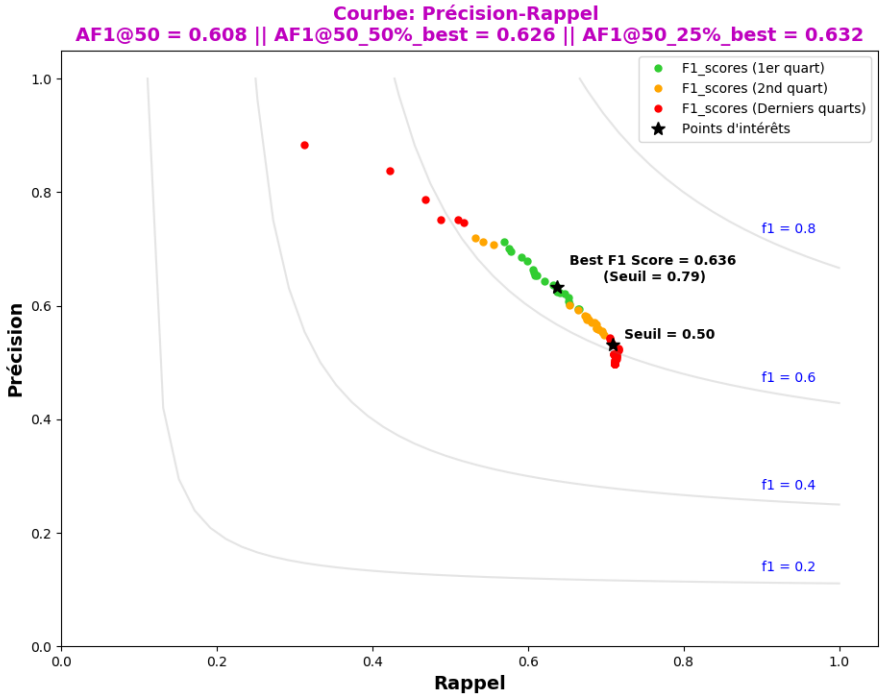


Figure 4.11 Courbe "Précision-Rappel" avec "PRE_NMS_LIMIT"=2500 et "POST_NMS_INFERENCE"=600

Afin d'avoir une meilleure vitesse de détection tout en minimisant la perte de précision nous avons sélectionné le couple de valeurs suivant :

- "PRE_NMS_LIMIT" = 2500
- "POST_NMS_INFERENCE" = 600

Par cette nouvelle configuration, nous avons une augmentation de la vitesse de détection du modèle de **39%** avec un temps de 145ms par image qui correspond à un nombre de **6.9 fps**. Pour ce qui est de la précision, par comparaison des courbes "PR" de la figure 4.11b (obtenue pour ce couple de valeurs) et de la figure 4.10 (modèle final), les résultats sont les suivants :

- Une augmentation du F1_score moyen de l'ensemble des points de 0.2%
- Une diminution du F1_score moyen de la meilleure moitié (40 meilleurs points) de 0.2%
- Une diminution du F1_score moyen du meilleur quart (20 meilleurs points) de 0.5%

- Une diminution de la valeur du meilleur F1_score de la courbe de 0.9%

Afin d'avoir un minimum de perte sur les performances du modèle, nous nous sommes limités à une configuration qui diminue au maximum de 1% la valeur des paramètres liés au F1_score.

CHAPITRE 5 ÉLABORATION DE L'ALGORITHME ÉTABLISSANT LA CONCORDANCE ENTRE LE SUPPORT ET LE PRÉHENSEUR

Le deuxième objectif spécifique de ce projet est de développer un algorithme permettant, à partir de la segmentation des objets donnée par l'outil de détection Mask-RCNN, de déterminer les dimensions des objets détectés. Le troisième objectif spécifique consiste à développer un algorithme qui, selon la caractérisation du système de préhension, renvoie le support le plus adéquat pour le perchage du drone. Cette partie du mémoire traite l'ensemble de la conception de notre algorithme, qui à partir des masques des objets détectés et des caractéristiques du préhenseur utilisé, renvoie le meilleur support pour le perchage.

Le meilleur support est défini ici, comme étant l'objet, en vue 2D, ayant des dimensions se rapprochant le plus des dimensions optimales du système de préhension du drone. Ainsi en fonction des caractéristiques du préhenseur, l'algorithme doit renvoyer un pourcentage d'association entre l'objet détecté et ce dernier. Pour cela nous avons conçu l'algorithme nommé "Algorithme CSP". "CSP" (signifiant "Concordance Support-Préhenseur") représente la variable donnant le pourcentage de concordance entre le support détecté et le préhenseur choisi. L'accès à cette donnée nous permet ainsi de déterminer le meilleur support en réalisant une comparaison sur l'ensemble des supports potentiels obtenus.

Pour obtenir les dimensions des objets et à fortiori le pourcentage de concordance objet-préhenseur souhaité, nous avons choisi de travailler sur la segmentation des objets. Les masques de chaque objet détecté, fourni en sortie de l'algorithme Mask-RCNN, constitue l'entrée de l'algorithme CSP. Une fois l'entrée acquise, notre algorithme CSP se décompose en 9 étapes successives pour aboutir, en fin d'algorithme, à l'obtention du support le plus adéquat pour le perchage du drone. Plus précisément, l'algorithme CSP nous renvoie l'image de ce support avec la variable "CSP" qui lui est associée.

La première partie de l'algorithme CSP qui correspond au cinq premières étapes a permis de déterminer les dimensions des objets détectés. En effet, à partir du masque (image faite de pixels noirs et blancs) obtenu à la suite de la détection, l'algorithme réalise un comptage des pixels représentant l'objet (pixels blancs). Cette partie a permis de valider la fonction attendue par l'objectif spécifique n°2.

La deuxième partie, correspondant aux étapes restantes, a permis de réaliser une comparaison entre les dimensions des objets détectés et les caractéristiques du préhenseur afin de déterminer le meilleur support. Cette comparaison est réalisée à partir de la fonction "Concordance" que nous avons élaboré. Cette fonction crée une relation linéaire entre : le nombre moyen de pixels d'une partie homogène de l'objet détecté et les caractéristiques du préhenseur, afin d'obtenir, en sortie, le pourcentage de concordance entre les deux. Cette deuxième partie a permis de valider l'objectif spécifique n°3 avec succès.

Chacun des choix et des étapes faites, pour atteindre ces deux objectifs spécifiques, sont détaillés dans les sections suivantes.

5.1 Processus de fonctionnement

Le processus de fonctionnement de l'algorithme se décompose en plusieurs étapes successives que seront détaillées au fur et à mesure dans cette section.

En résumé, voici le déroulement de son fonctionnement :

1. Récupération du masque d'un objet détecté.
2. Prise en compte des caractéristiques du système de préhension ainsi que celles de la caméra utilisée.
3. Conversion des paramètres du préhenseur (mm) en pixels à partir de la calibration de la caméra.
4. Calculs du nombre de pixels dans la longueur et la hauteur de l'objet.
5. Décomposition ou non du masque de l'objet en plusieurs éléments (selon l'objet).
6. Comparaison des dimensions de l'objet (ou des éléments de l'objet) avec les caractéristiques du préhenseur.
7. Détermination du score de concordance informant sur le taux d'association entre une partie d'objet et le préhenseur.
8. Renvoie, en sortie de détection, l'image de détection correspondante à la partie de l'objet ayant obtenue le meilleur pourcentage d'association.
9. Retour de l'image de détection du support idéal sur l'ensemble de la zone analysée.

5.1.1 Étape n°1 : Récupération du masque de l'objet

(a) Image de détection de Mask-RCNN (b) Masque de l'objet détecté

Figure 5.1 Détection d'un objet d'intérêt par Mask-RCNN

Lorsque l'outil de détection Mask-RCNN détecte un objet d'intérêt, il renvoie alors l'image avec la boîte englobante, le score de confiance ainsi que le masque associés à cet objet, comme présenté sur la figure 5.1a. Le masque de l'objet, présenté sur la figure 5.1b, correspond à une image avec des pixels noirs et blancs. Les pixels blancs représente la segmentation de l'objet détecté et les pixels noirs le reste de l'image (arrière-plan et autres objets). Dans le cas où une image possède plusieurs objets d'intérêts nous aurons autant d'images de masque qu'il y a d'objet et non pas une image regroupant tous les masques.

5.1.2 Étape n°2 : Paramètres de configuration

Une fois le masque de l'objet obtenu, il est nécessaire de configurer l'algorithme à propos des systèmes utilisés, à savoir : la caméra et le préhenseur.

5.1.2.1 Premier système : la caméra

La caméra va permettre de renvoyer les images des différents endroits survolés par le drone afin de les analyser et détecter les objets potentiels pour le perchage. Étant donné que nous utilisons un réseau de neurones pour la détection, la couleur correspond donc à une caractérisation des objets, par conséquent il est nécessaire de choisir une caméra de couleur. Ensuite, la résolution de la caméra dépend principalement des paramètres d'entrée sélectionnés pour l'entrainement du réseau. Ici, nous l'avons entrainé avec des images ayant une résolution de

640×512. De ce fait, il est conseillé d'utiliser une caméra ayant une résolution proche de la résolution d'apprentissage pour avoir de meilleures performances de détection. En effet, pour chaque image fournie en entrée de Mask-RCNN, celui-ci va la redimensionner en 640×512. Ainsi, pour éviter d'avoir une déformation de l'image ou une perte d'information suite au redimensionnement, le rapport dimensions images vidéos et dimensions images d'entrée d'algorithme doit être le plus proche de 1. La caméra "C922 PRO STREAM" de la marque Logitech qui a une résolution de base en 1280×720 a été choisie pour ce projet. Sa résolution a été réduite en 640×480 lors de la réalisation des tests.

5.1.2.2 Deuxième système : le préhenseur

Pour le deuxième système, étant donné qu'il est fictif dans ce projet, deux paramètres ont été pris en compte : son diamètre optimal de préhension et sa largeur de pince. Ces paramètres seront bien évidemment variables et dépendront du système de préhension utilisé. Les expériences de détections seront effectuées avec des préhenseurs de caractéristiques différentes. Pour en revenir aux paramètres, le diamètre optimal correspond au diamètre moyen de l'intervalle d'ouverture de la pince. Ainsi, le diamètre optimal correspond à la moyenne des valeurs du diamètre minimal et maximal de préhension. Quant à la largeur du préhenseur, cela correspond à la longueur minimale de l'objet requise pour que la pince puisse assurer une bonne préhension de l'objet détecté.

5.1.3 Étape n°3 : Conversion des paramètres du préhenseur (mm) en pixels à partir de la calibration de la caméra

Dans cette étape, l'algorithme doit pouvoir convertir les caractéristiques du préhenseur, données en millimètres, en nombre de pixels. Pour cela, nous avons utilisé le modèle de Sténopé qui, à partir des coordonnées spatiales d'un point dans l'espace (repère "monde"), permet d'obtenir les coordonnées de ce même point dans l'image (repère "image"). Le modèle de Sténopé (ou "Pinhole Model" en anglais) est une modélisation du procédé de la création d'images au sein d'une caméra. En général, nous nous trouvons dans un cas de projection perspective soit avec un angle de vue aléatoire de la caméra par rapport à l'objet. La relation entre les deux repères, dans ce cas, est donnée par le système suivant[103] :

$$\begin{cases} x' &= \frac{f'}{z} \times x \\ y' &= \frac{f'}{z} \times y \end{cases} \tag{5.1}$$

Avec (x,y,z) les coordonnées du point dans le repère monde (en mm), (x',y') les coordonnées du point dans le repère image (en pixels) et f' la distance focale de la caméra en pixels.

Pour ce projet de recherche et les futures tests de détection, nous avons choisi de nous placer dans un cas simple d'utilisation. Ce dernier correspond à l'utilisation d'une seule caméra, d'avoir un angle de vue fixe et parallèle aux objets ainsi qu'une distance entre la caméra et les objets également fixée pour chacune des expériences.

Lorsqu'il n'y a pas de perspective ou très peu, comme quand la caméra est parallèle aux objets, nous nous trouvons dans le cas de la projection à faible effet de perspective[103]. De ce fait, le modèle du Sténopé se simplifie :

$$\begin{cases} x' &= \frac{f'}{z_0} \times x \\ y' &= \frac{f'}{z_0} \times y \end{cases} \tag{5.2}$$

Dans ces nouvelles équations, le z a été remplacé par le z_0 représentant encore la distance entre la caméra et l'objet mais ici cette distance est la même pour l'ensemble des points images contrairement au z de l'équation 7.1. Ainsi pour connaître la relation au complet, il reste à déterminer la distance focale f' (en pixels) de la caméra. Le module "Camera Calibrator" de Matlab permet de réaliser la calibration de la caméra et ainsi obtenir la valeur de f'. Pour la calibration, il est nécessaire d'avoir un tableau témoin, figure 5.2, composé généralement de carrés blancs et noirs de dimensions connues.

Ensuite, nous réalisons un ensemble de photographies de ce tableau avec la caméra. Une fois cette étape réalisée, Matlab détermine la matrice intrinsèque, matrice 3×3, de la calibration regroupant l'ensemble des paramètres de la caméra[104] :

$$\begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}$$

Les paramètres f_x et f_y correspondent à la distance focale en pixels de la coordonnée x et y respectivement. Les paramètres c_x et c_y représentent les coordonnées, en pixels, du centre optique. Le paramètre s se définit comme la non-orthogonalité entre les lignes (axe x) et les colonnes (axe y) de pixels du capteur de la caméra. Dans le cas général où les axes x et y sont perpendiculaires entre eux, cela signifie que le paramètre s est égal à 0.

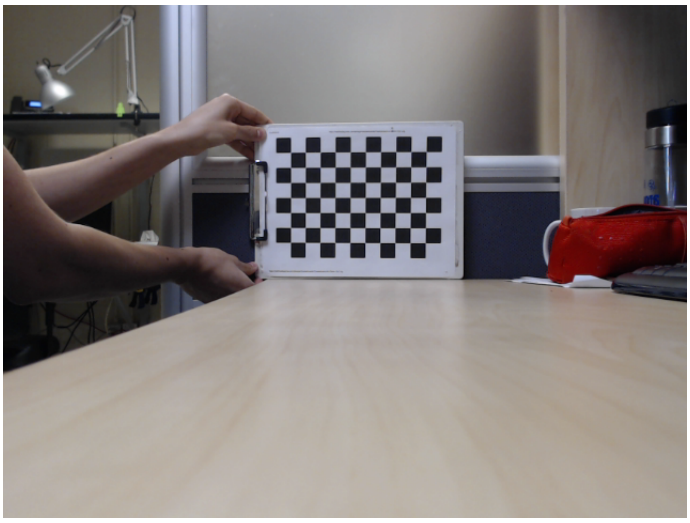


Figure 5.2 Tableau témoin de carrés blancs et noirs utilisé pour la calibration

Une fois la calibration effectuée, les caractéristiques du préhenseur peuvent être convertis en pixels. La valeur prise pour la distance focale f' correspond à la moyenne entre les distances focales obtenues pour l'axe x et y respectivement. La valeur du paramètre z_0 correspond à la distance entre la caméra et le tableau témoin prise pour réaliser la calibration de la caméra.

Le facteur nommé "FDC", introduit dans l'algorithme CSP, correspond au facteur de calibration. Il est défini comme étant le rapport entre la distance focale f' en pixels et la distance z_0 en millimètres. Le facteur nommé "FD" correspond, ici, au rapport entre la distance caméra-objets lors de la détection en mètre, nommé "DCOD", et la distance caméra-objet lors de la calibration en mètre. Par conséquent, nous arrivons sur l'établissement du facteur de conversion, nommé "FCRP", entre la taille réelle d'une mesure et sa taille associée en pixels. Ce facteur se définit comme étant le rapport des deux facteurs énoncés précédemment. Ainsi nous obtenons la relation décrite ci-dessous :

$$FCRP = \frac{FC}{FD} = \frac{\frac{f'}{z_0}}{\frac{DCOD}{z_0 \times 10^{-3}}} = \frac{f'}{z_0} \times \frac{z_0 \times 10^{-3}}{DCOD} \tag{5.3}$$

Le coefficient "FCRP" est assimilé au coefficient $\frac{f'}{z_0}$ définissant la relation entre les coordonnées d'un point entre le repère "image" et le repère "monde" mais étendu au cas de la détection. Ainsi pour une distance caméra-objet fixée lors de la détection, nous pouvons convertir les dimensions du préhenseur en pixels et à fortiori les comparer aux dimensions des objets.

5.1.4 Étape n°4 : Calculs du nombre de pixels dans la longueur et la hauteur du masque de l'objet

Les caractéristiques du système de préhension pouvant maintenant être converties en pixels grâce au coefficient "FCRP", il faut déterminer les dimensions des objets à partir de leurs masques. Le masque des objets se compose de pixels noirs et blancs. Comme on peut le voir sur la figure 5.1b, le masque des objets correspond aux pixels blancs. Il suffit alors de déterminer le nombre de pixels blancs par ligne dans la longueur puis dans la hauteur de l'image du masque afin de connaître les dimensions des objets en pixels.

Cette méthode est à la fois simple et rapide. Du fait que Mask-RCNN est relativement lent dans le traitement des images, il a donc été préférable d'opter pour une méthode qui permet de réaliser un comptage simple de pixels tout en minimisant le temps d'exécution de cette tâche.
Cette méthode contient cependant un inconvénient avec des objets se trouvant à l'oblique ou ayant une partie à l'oblique dans l'image. En effet, comme nous pouvons le voir sur la figure 5.3, si l'objet est oblique alors la distance minimale ne sera pas obtenue car la méthode consiste en un comptage à l'horizontale et à la verticale.

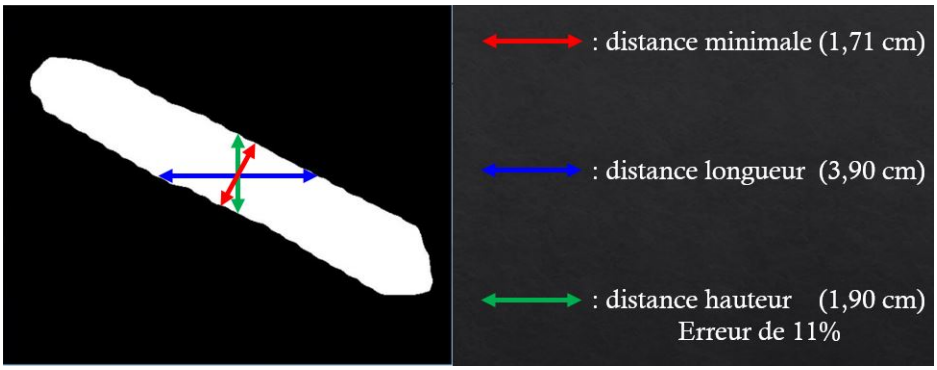


Figure 5.3 Comparaison des distances pour la méthode utilisée pour un objet oblique

Dans l'exemple donné par la figure 5.3, la distance la plus proche de la distance minimale est la distance sur la hauteur de l'objet mais il y a une erreur de 11% entre elles. Cette erreur est certes relativement faible mais elle reste présente, de plus selon la disposition de l'objet elle peut être plus ou moins conséquente. En conséquence, les objets durant les tests seront, le plus possible, préposés de manière horizontale ou verticale. En effet, selon les objets notamment ceux de la classe "Branche", ils peuvent être courbés et donc potentiellement, peu importe leur sens, avoir une partie à l'oblique.

5.1.5 Étape n°5 : Décomposition du masque de l'objet dans le cas où il possède plusieurs éléments

(a) Branche avec plusieurs ramifications

(b) Barrière avec plusieurs barres

Figure 5.4 Objets avec plusieurs parties

Un autre cas important à prendre en compte est celui des objets qui possèdent plusieurs éléments. Ici les objets de ce type seront principalement ceux des classes "Branche" et "Barrière". Comme le montre la figure 5.4, une branche, comme une barrière, peut respectivement posséder plusieurs ramifications, barres. Ainsi il est indispensable de les dissocier entre eux. Sans dissociation, le nombre de pixels par ligne ou colonne serait faussé car égale à l'addition du nombre de pixels de chacun des éléments. Autrement dit cela représente un objet d'un seul bloc au lieu de plusieurs éléments d'objets. Ainsi le but de cette étape dans la conception de l'algorithme est d'arriver au résultat obtenu donné par la figure 5.5.

Pour traiter cela, deux facteurs ont été implantés dans le but de créer une liste, nommé "liste de décomposition de masque", regroupant le nombre de pixels pour chaque élément de l'objet (si c'est le cas). Il y a d'une part, le facteur de distance entre pixels : "FDEP" et d'autre part, le facteur d'appartenance à un élément : "FAEO".
Le premier facteur correspond au nombre de pixels noirs qu'il peut y avoir entre les pixels blancs de deux éléments potentiels comme présenté en figure 5.6a. Si cette distance est supérieure à ce facteur alors le comptage se scinde en deux (ou plus selon le nombre d'éléments durant l'itération). Ainsi pour chacune des itérations, nous avons une liste regroupant le nombre de pixels blancs de chaque élément. Dans ce projet, nous avons fixé ce facteur à 50 soit une distance de 50 pixels noirs entre 2 pixels blancs. Cela représente 7.8% de la longueur et 10.4% de la hauteur de l'image. Ces pourcentages sont relativement faibles afin d'assurer une bonne décomposition de l'objet en un maximum d'éléments tout en évitant d'avoir de mauvais comptages. Pour l'exemple figure 5.6a, le point d'interrogation correspondrait ainsi à un 2 soit à un deuxième élément, pour arriver à la décomposition donnée figure 5.5.

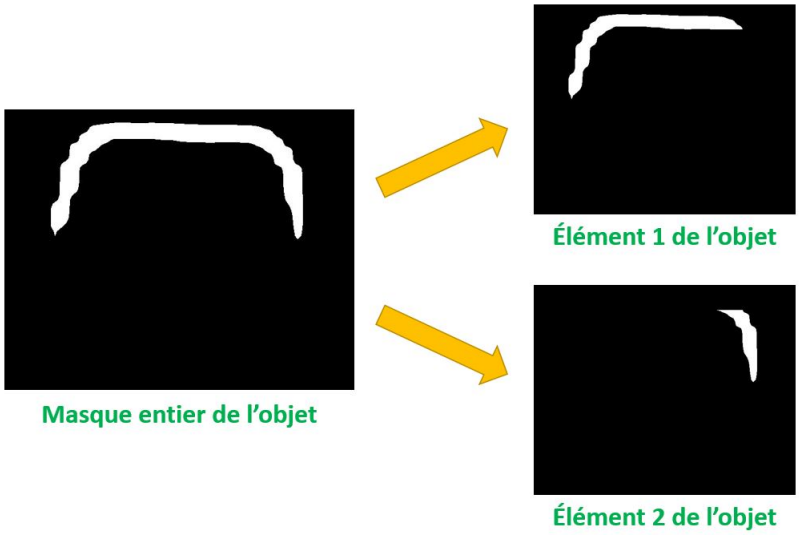
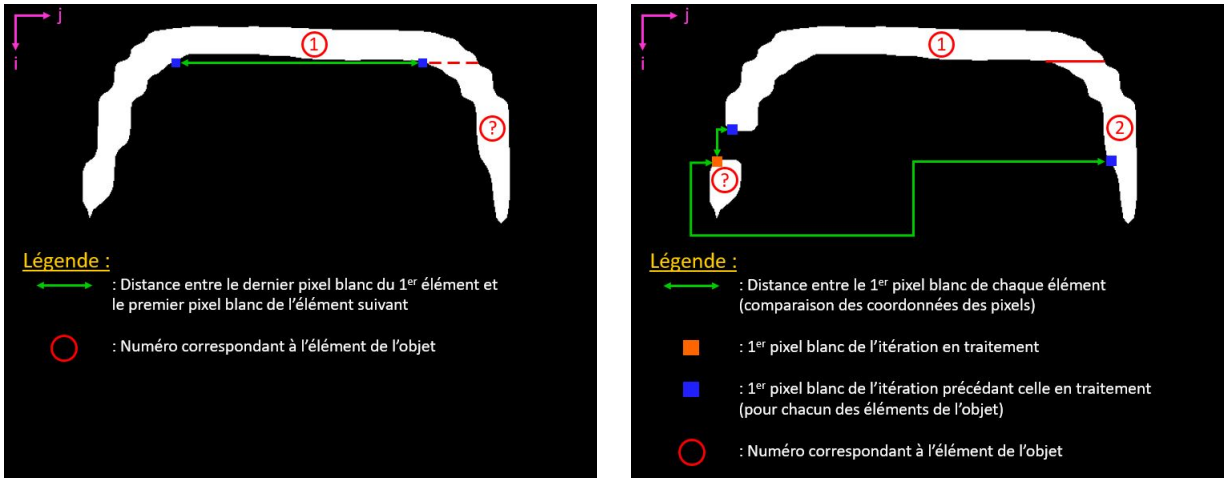


Figure 5.5 Décomposition du masque de l'objet en plusieurs éléments

À partir du moment où nous avons plusieurs éléments, il est essentiel de savoir, pour les itérations suivantes, à quel élément d'objet les pixels blancs appartiennent. C'est là qu'entre en jeu le deuxième facteur. Pour une itération donnée et sachant que nous avons plusieurs éléments d'objet, chaque donnée présent dans la "liste de décomposition de masque" doit être associer au bon élément d'objet. Si une donnée de la liste n'est affectée à aucun élément, alors un nouveau sera crée. Comme expliqué à travers la figure 5.6b, nous venons comparer la distance entre le premier pixel blanc de l'itération en cours avec le premier pixel blanc de chaque élément de l'itération précédente. Cette distance correspond à deux comparaisons : celle entre les abscisses et celle entre les ordonnées de ces premiers pixels blancs. Chaque différence obtenue est ensuite comparée au facteur défini. Ici aussi, nous avons fixé le facteur à 50. Ainsi, lorsque la différence de comparaison avec un élément (déjà existant), entre abscisses et entre ordonnées, est inférieure au facteur alors cela signifie que la donnée de la liste appartient à cet élément d'objet. Dans le cas où une donnée ne possède aucune association, alors un nouvel élément sera crée. Dans le cas de la figure 5.6b, le petit morceau de masque avec le point d'interrogation sera associé à l'élément 1 de l'objet car la distance en terme d'abscisse comme d'ordonnée est inférieure à 50 pixels.

Cependant pour ces facteurs, nous avons décidé de ne pas sélectionner des valeurs trop petites pour les cas où nous avons un masque d'objet avec des morceaux manquants comme dans la figure 5.6b. Ainsi même s'il y a des parties manquantes à certains endroits du masque mais qu'elles restent relativement petites, l'algorithme CSP pourra les associer avec un élément

d'objet et par conséquent les prendre en compte dans le comptage du nombre de pixels blancs pour ce dernier.



(a) Facteur de distance entre pixels (b) Facteur d'appartenance à un élément

Figure 5.6 Explication des 2 facteurs introduits pour la séparation des éléments d'objets (figures pour le cas du traitement à l'horizontal)

À la fin de cette étape, l'algorithme CSP regroupe dans un dictionnaire (Python), chacun des éléments de l'objet ainsi que les informations qui lui sont associées comme le nombre de pixels blancs de chacune de ses couches. De ce fait, chaque élément est bien séparé des autres et, ainsi, ils peuvent être traités un par un pour la détermination du support idéal.
Dans la suite de cette section, pour des raisons de simplicité de compréhension, l'énonciation d'élément d'objet signifiera qu'un objet possède plusieurs éléments mais traitera également le cas où un objet n'a qu'un seul élément

5.1.6 Étape n°6 : Comparaison des dimensions de l'objet (ou des éléments de l'objet) avec les caractéristiques du préhenseur

Lorsque l'ensemble du nombre de pixels blancs par couche et par élément est acquis, l'algorithme les compare directement aux caractéristiques du système de préhension. Ces dernières étant en millimètres, l'algorithme CSP les convertit en pixels grâce au coefficient "FCRP" établi dans l'étape 3. Dans le cas où un objet possède plusieurs éléments, l'algorithme CSP réalise une comparaison entre le préhenseur et chaque élément de l'objet. L'objectif général de cette maîtrise est de détecter le support idéal, cependant cette recherche ne se fait pas sur un objet complet mais **une partie d'objet**. Sur plusieurs objets, il est fort probable qu'un

objet, ayant des dimensions différentes sur sa longueur ou sa largeur, puisse avoir une partie s'associant plus avec les caractéristiques du préhenseur qu'un objet avec des dimensions constantes, comme l'explique la figure 5.7. Pour un objet possédant plusieurs éléments, nous cherchons donc l'élément s'associant le plus avec le préhenseur.

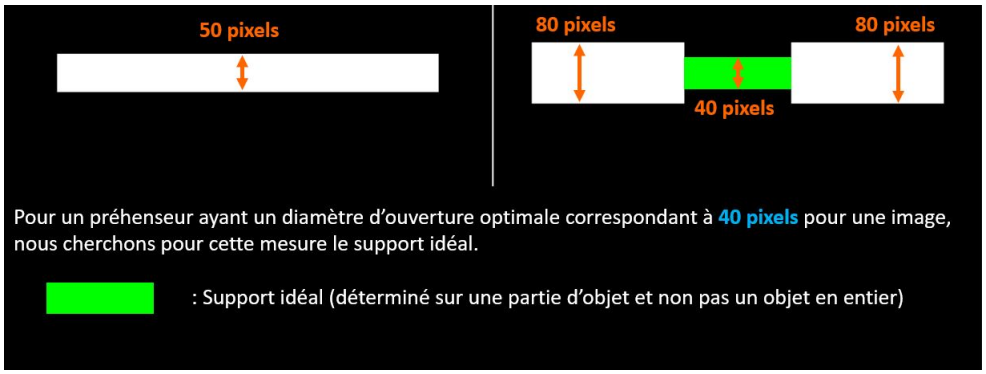


Figure 5.7 Explication de la détermination du support idéal

Pour cela, nous avons introduit 4 facteurs dans le but d'assurer une bonne comparaison entre objet et préhenseur, une sécurité pour la réussite du perchage et aussi l'assurance d'avoir une bonne qualité du masque de l'objet.

Tout d'abord, nous avons le facteur de validation qui certifie l'obtention d'une dimension relativement constante sur plusieurs itérations. Ce nombre d'itérations est défini par la largeur du préhenseur. Autrement dit, l'algorithme CSP valide ou non par ce facteur si il y a une zone dans l'objet qui est assez large pour le préhenseur afin d'assurer une bonne prise lors du perchage.
Ensuite, il y a le facteur "LSP" qui est un facteur de sécurité pour le facteur précédent. Ainsi l'utilisation de ce facteur avec le précédent permet de considérer une validation plus large de la zone de préhension du fait notamment que le drone peut bouger avec du vent ou pour prévenir des incertitudes lors du perchage.
Enfin, les deux autres facteurs permettent le traitement des excès de différences de longueur entre les couches de pixels blancs (itérations successives) et d'assurer la constance de la dimension.

L'explication des différents facteurs énoncés précédemment est fournie à travers la figure 5.8. En effet, nous pouvons voir que le facteur de validation et le facteur "LSP" correspondant ici à la zone de préhension en rouge, respectivement à la zone sécuritaire de préhension en orange. Le facteur traitant de l'excès entre les couches de pixels blancs correspond au numéro 1 de la figure 5.8 où les couches successives de pixels blancs ont une différence de longueur

importante. Ainsi en fonction de la valeur donnée à ce facteur, les deux couches en bleu et en vert dans la figure feront parties de la même, potentielle, zone de préhension ou non. Enfin, le facteur qui traite de l'homogénéité des dimensions de l'objet pour une certaine longueur, permet de sélectionner une zone constante comme celle associée au numéro 2 de la figure 5.8 et rejeter une zone non constante comme celle associée au numéro 3, dans le but d'assurer une bonne préhension du support.

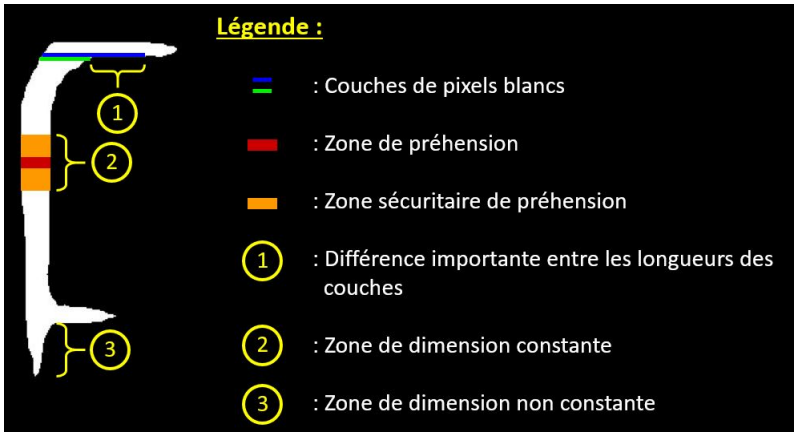


Figure 5.8 Explication des différents facteurs utilisés dans la comparaison des dimensions entre le préhenseur et l'objet

Cette étape de comparaison a pour but de valider si la partie de l'objet analysée correspond à une zone de préhension ou non. Pour cela, l'algorithme CSP sélectionne le nombre de pixels blancs par couche obtenu lors de la réalisation de l'étape n°4 puis réalise une succession d'étapes pour arriver à cette validation.
Tout d'abord, entre chaque couche (itération), le facteur d'excès permet de rejeter toutes grandes variations dans le nombre de pixels blancs entre deux couches successives.
Ensuite, s'il n'y a pas de grandes variations entre les couches successives, un calcul du nombre moyen de pixels blancs est réalisé pour chaque couche ajoutée. Si cette moyenne augmente ou diminue trop cela signifie que l'ensemble des couches analysées jusqu'à présent n'est pas assez constante donc augmente le risque d'une mauvaise préhension. Ainsi, le facteur de constance permet de rejeter ou non cette partie analysée.
Enfin, si les deux étapes précédentes sont validées sur un nombre d'itérations suffisant alors cette partie sera sélectionnée comme étant une zone de préhension potentielle pour le drone. Le nombre d'itérations suffisant correspond ici au produit du facteur de validation et du facteur de sécurité "LSP".

5.1.7 Étape n°7 : Détermination du pourcentage informant sur le taux d'association entre une partie d'objet et le préhenseur

Pour chaque partie validée dans l'étape précédente, la fonction "Concordance" de l'algorithme CSP leur attribue un score d'association. Ce dernier nommé "CSP" est un pourcentage compris entre 1% et 100%. Plus le "CSP" est élevé et plus le support est compatible et donc idéal pour le système de préhension. Nous nous sommes limité à un minimum de 1% représentant les limitations de l'intervalle d'ouverture du préhenseur. Si les dimensions des objets sont inférieures, supérieures strictement à la valeur minimale, respectivement maximale le pourcentage de concordance sera égale à 0%.

La fonction "Concordance", dont le code est présenté ci-dessous, prend en entrée les caractéristiques du préhenseur à savoir son intervalle d'ouverture, le facteur de limitation et aussi le nombre moyen de pixels blancs de la partie validée. Ainsi les facteurs "fb", "fh" correspond au diamètre d'ouverture le plus petit, le plus grand respectivement. Le facteur "fop" est défini comme étant le diamètre optimal soit la moyenne des deux extrémités de l'intervalle de prises. Cette fonction est basée sur une simple loi linéaire dont les paramètres sont définies à partir des caractéristiques du préhenseur et du facteur de limitation.

Code de la fonction "Concordance" :

```
def Concordance (x, fop, fb, fh, fl=0.01) :
    if x < fb or x > fh :
        CSP = 0
        return CSP

    elif x <= fop :
        a = (1 - fl) / (fop - fb)=
        b = 1 - a * fop

    else :
        a = - (1 - fl) / (fh - fop)
        b = 1 - a * fop
    CSP = (a * x + b) * 100

    return CSP
```

En sortie de cette fonction, nous obtenons ainsi le pourcentage d'association "CSP", entre la partie d'objet analysée et le système de préhension.

5.1.8 Étape n°8 : Renvoie, en sortie de détection, de l'image de détection correspondante à la partie de l'objet ayant obtenue le meilleur pourcentage d'association.

Une fois le premier "CSP" obtenu, la variable "Detection" est créée. Cette dernière regroupe en plus du "CSP" de la partie validée, toutes ses caractéristiques qui permettront de retravailler le masque entier de l'objet afin d'avoir en sortie seulement le masque de la partie de l'objet, qui correspond à la zone de préhension/perchage pour le drone. Pour les autres parties également validées, une comparaison est faite entre le "CSP" de la nouvelle partie et celui présent dans la variable "Detection" puis sélectionne le meilleur. Par conséquent, la variable "Detection" contiendra, à la suite du traitement de l'ensemble des parties d'objets validées comme zone de préhension potentielles, les informations de la partie de l'objet ayant obtenue le meilleur pourcentage d'association.

Les différentes informations sur la partie sont les suivantes :

- Position du début et de la fin de la partie de liste, regroupant le nombre de pixels blancs par couches, qui correspond à la partie d'objet validée
- Nombre moyen de pixels blancs de la partie
- Le "CSP"
- Coordonnées du premier pixel blanc de la première et dernière couche de l'élément traité
- Liste regroupant l'abscisse du premier pixel blanc de chacune des couches de l'élément traité

Une fois que la partie de l'objet ayant le meilleur "CSP" est identifiée, l'algorithme modifie le masque de l'objet. Partant du masque entier de l'objet et à partir des informations sur cette partie disposée dans le paramètre "Detection", nous venons le modifier pour n'avoir qu'à la fin, le masque correspondant uniquement à la zone de préhension de l'objet qui a obtenu la meilleure association avec le préhenseur. Pour cela, chacun des pixels blancs n'appartenant pas à la partie sera transformé en pixel noir.

À la fin de cette modification, pour chacun des objets détectés dans l'image d'origine, l'algorithme renvoie une image composée de l'image originale à laquelle on lui ajoute le masque de la meilleure partie et le "CSP" qui lui est associé, comme le montre la figure 5.9.

5.1.9 Étape finale : Retour de l'image de détection du support idéal sur l'ensemble de la zone analysée

L'image de détection correspond, ici, à l'image avec le masque de la meilleure zone de préhension de chaque objet présent dans cette dernière. Pour chaque image où un ou plusieurs objets sont détectés, nous avons en retour une image de détection et donc un meilleur "CSP". Comme nous utilisons une caméra dans notre cas, nous aurons un grand nombre d'image de détection. Ainsi pour chaque nouvelle image de détection, nous comparons le meilleur "CSP" de cette dernière avec le dernier à date dans le but de garder en mémoire le meilleur d'entre eux. Cette opération est alors répétée indéfiniment jusqu'à l'arrêt de la détection.
De ce fait, une fois l'arrêt de la détection, l'algorithme CSP nous retourne l'image de détection possédant le meilleur "CSP" sur l'ensemble de la zone survolée. En conclusion, nous obtenons l'objet et plus précisément la partie de l'objet qui correspond au support idéal pour le drone, comme le montre l'exemple de la figure 5.9.

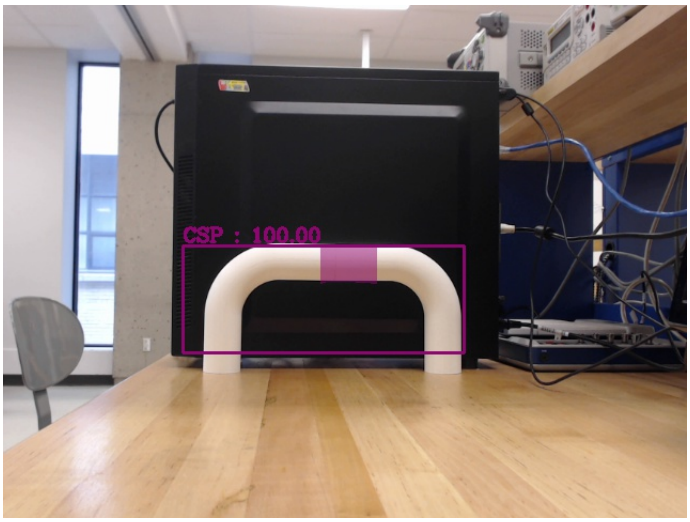


Figure 5.9 Support Idéal obtenu en fin de détection

5.2 Performances de l'algorithme CSP en terme de vitesse de détection

L'utilisation de l'algorithme CSP ajoute du temps d'exécution à l'outil de détection de Mask-RCNN. Il est donc primordial d'optimiser, grâce à certaines méthodes de traitement, cet algorithme pour réduire au maximum son temps d'exécution et ainsi limiter son apport à la durée de traitement du programme complet. De plus, il traite chaque masque d'objet présent dans l'image un à un. Par conséquent, si une image possède plusieurs objet le temps d'exécution nécessaire pour la traiter se verra multiplier par le nombre d'objets présents dans cette image. Ainsi, la durée de traitement pour une image peut vite devenir importante.

5.2.1 Performances originales

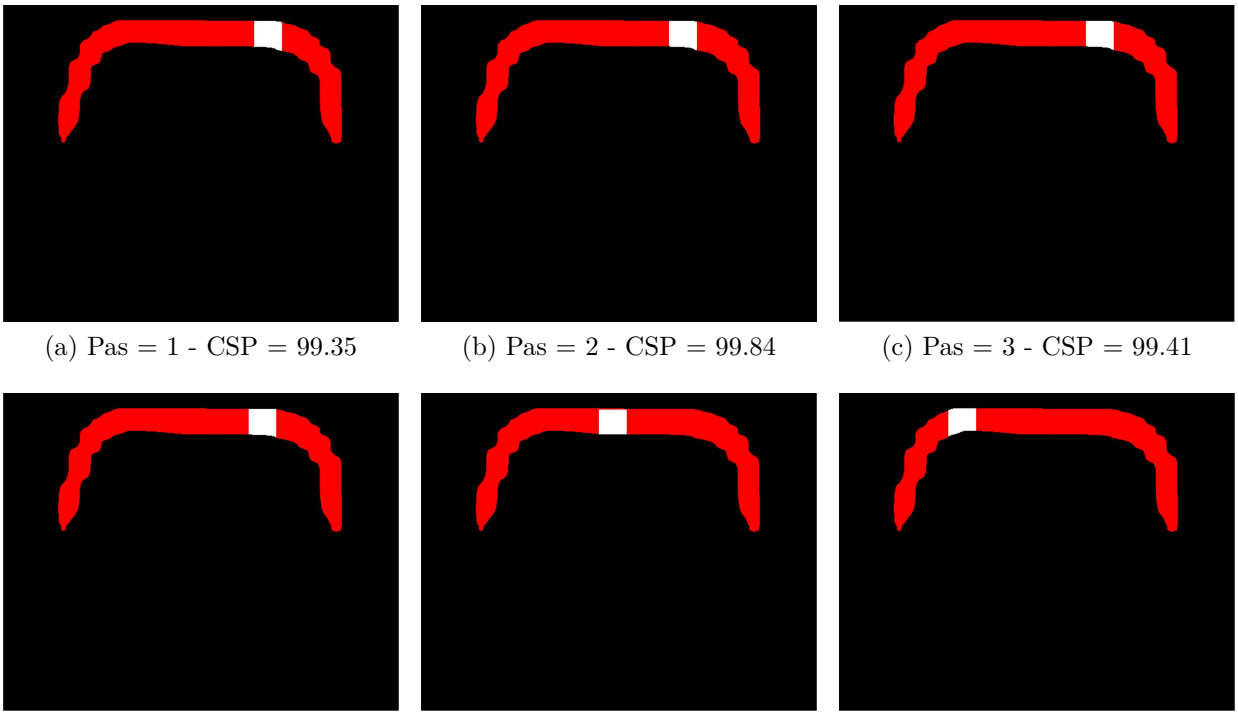
Sans améliorations, le temps de traitement obtenu pour l'algorithme CSP est de 281ms par masque d'objet. Ce temps est assez élevé surtout que comme énoncé précédemment si l'outil de détection détecte plusieurs objets, ce temps augmentera en conséquence. Pour cela diverses méthodes d'optimisation ont été apportées pour le réduire.

5.2.2 Méthodes d'amélioration

L'opération la plus couteuse dans l'algorithme CSP est le traitement du masque des objets qui correspond en deux matrices 640×480 puisque nous réalisons une étude sur la longueur puis sur la hauteur de l'objet. C'est donc sur cette étape qu'il faut sauver du temps de traitement.

Tout d'abord, nous avons défini le facteur "FSAB" qui est un facteur de sécurité d'arrêt de boucle. Autrement dit, lorsque l'algorithme rencontre le premier pixel blanc il a donc trouvé l'objet présent dans l'image 640×480. Une fois le "potentiel" dernier pixel blanc trouvé, les prochaines itérations n'en possèderont pas. Lorsqu'un certain nombre d'itérations sans détection de pixel blanc est atteint, l'algorithme arrête le traitement du masque. Ce nombre d'itération correspondant à la valeur donnée au facteur "FSAB". De ce fait, nous nous concentrons uniquement sur le traitement de l'objet présent dans la matrice (image) et évitons ainsi tous calculs inutiles. Un "FSAB" égale à 25 a été choisi afin de laisser une petite marge dans le cas où un morceau de masque est détaché du corps principal comme sur la figure 5.6b. Cette méthode d'amélioration sera plus efficace dans le cas où l'objet se situe dans la partie haute, gauche de la matrice pour le traitement sur la longueur, respectivement pour le traitement sur la hauteur.

Ensuite, utilisant une double boucle d'itérations pour le traitement du masque, nous avons défini un "pas" de boucle pour la deuxième. Cette dernière est celle qui permet le comptage des pixels blancs pour chaque couche selon le sens de traitement. Cette méthode d'amélioration offre une plus grande sensibilité pour sauver du temps d'exécution mais provoque néanmoins une perte de précision sur le comptage des pixels blancs. Par exemple pour un pas de 2, nous allons deux fois plus vite pour le traitement de la deuxième boucle mais nous risquons d'avoir un pixel blanc en moins ou en plus dans le comptage par couche. La précision pour un pas de 2 reste cependant très bonne dans cet exemple. Il faut donc trouver le bon compromis entre précision et temps de traitement pour cette méthode d'amélioration.



(a) Pas = 1 - CSP = 99.35 (b) Pas = 2 - CSP = 99.84 (c) Pas = 3 - CSP = 99.41

(d) Pas = 4 - CSP = 96.97 (e) Pas = 6 - CSP = 98.15 (f) Pas = 10 - CSP = 94.44

Figure 5.10 Masque entier de l'objet en rouge et masque de la meilleure zone de préhension en blanc selon différentes valeurs de pas

Nous remarquons plusieurs choses : Tout d'abord, pour un pas allant jusqu'à 4 la position de la zone de préhension reste relativement identique car l'approximation faite reste faible. Ensuite, pour un pas plus important, ici supérieur à 4, la zone de préhension a été déplacée et ne se trouve pas au même endroit que pour un pas de 1 (cas idéal). Cela permet de conclure que plus le pas augmente et plus il y a un risque que la zone se déplace par rapport à celle d'origine (pas = 1) témoignant d'une perte de précision. Enfin, nous observons que le CSP est inférieur, pour les pas strictement supérieurs à 3, à celui d'origine. Même si la

valeur reste relativement proche il est préférable de ne pas prendre un pas trop grand, car nous souhaitons avoir une très bonne précision de détection afin de différencier les différents supports potentiels.

5.2.3 Comparaison des performances sans et avec amélioration

5.2.3.1 Utilisation du facteur FSAB

Comme énoncé précédemment, ce facteur permet d'économiser du temps sur la double boucle pour le traitement du masque. Pour le traitement sur la longueur et sur la hauteur : plus un objet se trouvera en haut, respectivement à gauche, de l'image et plus cela sauvera du temps d'exécution. Le tableau ci-dessous compare les différent temps d'exécution de l'algorithme CSP selon la position de l'objet dans l'image. Pour l'obtention de ces résultats nous avons sélectionné un objet de la classe "Barrière" et de taille moyenne pour une distance caméra-objet d'un mètre.

Tableau 5.1 Temps de traitement par objet en fonction de la position de l'objet dans l'image

Première amélioration			
Position de l'objet dans l'image	En bas	Au centre	En haut
À gauche	0.20s	0.16s	0.11s
Au centre	0.23s	0.20s	0.16s
À droite	0.28s	0.23s	0.20s

Les résultats obtenus dans le tableau montre que dans 8 cas sur 9, nous sauvons du temps d'exécution grâce à l'utilisation du facteur "FSAB". Si nous enlevons le neuvième cas nous pouvons gagner, selon le positionnement de l'objet dans l'image, de 17.86% à 60.71% du temps d'exécution pour les mêmes résultats. En moyenne, si nous prenons en compte les 9 cas de positionnement, nous arrivons à un temps de 0.20s par image. Ainsi, en moyenne nous réduisons de 28.57% soit presque un tiers la durée de traitement par objet.

5.2.3.2 Utilisation du pas de traitement

La deuxième méthode permet de parcourir plus rapidement les pixels du masque d'objet afin d'avoir une célérité du comptage des pixels blancs accrue. Le tableau ci-dessous regroupe le temps d'exécution en fonction du pas de traitement.

Tableau 5.2 Temps de traitement par objet en fonction de la valeur du pas de traitement

Deuxième amélioration						
Pas de traitement	1	2	3	4	6	10
Temps d'exécution	0.28s	0.14s	0.09s	0.08s	0.05s	0.03s

Nous observons à travers ce dernier, que le pas de traitement permet une diminution significative du temps d'exécution. Nous avons analysé précédemment la perte de précision due à l'augmentation du pas et nous en avons conclu que ce dernier doit avoir une valeur maximale de 4. Même pour un pas égale à 4, l'algorithme économise 71.43% de sa durée de traitement pour un objet.

Le tableau 5.3 regroupe les deux méthodes d'optimisation. Afin de ne pas traiter 9 cas par rapport au positionnement de l'objet dans l'image, j'ai sélectionné le cas où l'objet est au centre de l'image et dont le temps d'exécution est proche de la moyenne obtenue pour les 9 cas.

Tableau 5.3 Temps de traitement par objet en prenant en compte les deux méthodes d'optimisation

Ensemble des deux améliorations avec objet au centre						
Pas de traitement	1	2	3	4	6	10
Temps d'exécution	0.20s	0.11s	0.06s	0.05s	0.03s	0.02s

Les masques utilisés pour obtenir ces résultats sont ceux affichés sur la figure 5.11.

(a) Masque de l'objet en haut à gauche de l'image

(b) Masque de l'objet au centre de l'image

(c) Masque de l'objet en bas à droite de l'image

Figure 5.11 Exemple de masques d'objet pour différentes positions de ce dernier dans l'image

Si nous choisissons un pas égale à 4 comme pour le deuxième tableau (5.2), nous obtenons un gain de temps de 82.14% ce qui est conséquent. Ces deux méthodes d'amélioration, par ces résultats, se sont révélées être très utiles dans l'augmentation des performances de vitesse d'exécution pour l'algorithme CSP.

CHAPITRE 6 RÉSULTATS D'EXPÉRIMENTATION DU MODÈLE DE DÉTECTION COMPLET : MASK-RCNN et ALGORITHME PSI

6.1 Paramètres choisis pour la caméra

La caméra utilisée dans ce projet de recherche est la : "C922x PRO STREAM WEBCAM" de Logitech. Cette caméra possède une résolution de 1920×1080 mais qui pour les expériences sera paramétrée pour avoir des images de résolution 640×480. Ainsi les images vidéos ont 640 pixels sur la longueur et 480 pixels sur la hauteur.

Nous avons ensuite réalisé la calibration de cette caméra à l'aide du module de Matlab. Pour une distance de calibration de 1 mètre, les distances focales obtenues, selon les axes x et y, sont les suivantes :

- Axe x : f'x = 651.3938 pixels
- Axe y : f'y = 652.3559 pixels
- Moyenne globale : f' = 651.8749 pixels

À partir de ces données, et selon la distance caméra-objet définies lors des tests, le facteur de conversion "FCRP" est maintenant connu pour cette caméra.

6.2 Paramètres choisis pour le modèle de détection au complet

L'objectif idéal, en plus de notre objectif principal, serait d'atteindre un minimum de 5 fps afin d'avoir une détection en temps réel durant la phase de perchage. Nous avons choisi judicieusement les paramètres de détection de Mask-RCNN et d'optimisation de l'algorithme PSI pour essayer de les atteindre. Les paramètres d'amélioration choisis sont les suivants :

- Pour les paramètres de détection : PRE_NMS_LIMIT = 2500 et POST_NMS_INFERENCE = 600 afin de garder une bonne précision de détection tout en améliorant la vitesse de détection
- Pour le pas de traitement pour l'algorithme PSI : pas = 3, dans le but de garder un bon traitement du masque des objets car à partir d'une valeur égale à 4, l'analyse du masque commence à se dégrader

À partir de ces paramètres, nous avons réalisé une comparaison entre le modèle au complet sans et avec les améliorations. Cette comparaison s'effectue sur la détection d'un seul objet placé au centre de l'image. Le tableau 6.1 résume les résultats obtenus.

Tableau 6.1 Comparaison du temps de traitement pour le modèle complet : Mask-RCNN + algorithme PSI, sans et avec méthodes d'amélioration

Temps de traitement	**Par image/fps associé**	**Détection Mask-RCNN**	**Algorithme PSI** (par objet)
Sans améliorations	1.49s/0.67 fps	0.42s	1.07s
Avec améliorations	0.42s/2.38 fps	0.25s	0.17s

Ces résultats montent que les 5 fps souhaités ne sont pas atteints malgré un temps de traitement **3.6 fois** plus rapide par rapport au cas sans améliorations. Ceci est principalement dû au fait que l'algorithme PSI a une durée de traitement qui est **4 fois** supérieure à celle obtenue en utilisant seulement ce dernier. La détection demandant une grande quantité de ressources pour les calculs, il reste ainsi que peu de puissance pour le traitement de l'algorithme PSI. C'est pour cela que son exécution perd en efficacité par rapport au cas où il est exécuté seul. De plus, l'utilisation du langage Python possède également ses limites : en effet, Python reste trop lent dans sa vitesse d'exécution par rapport à un langage comme C++ qui est plus orienté langage machine, et donc plus rapide. Selon les études présentées en [95, 96], Python est environ 45 fois plus lent que C++. Et enfin, Mask-RCNN étant limité à 5 fps avec l'utilisation d'une seule carte graphique (GPU), nous partons dès le début avec un temps d'exécution à la limite de la détection en temps réel. Afin d'atteindre notre objectif principal qui se base principalement sur la précision de la détection, nous avons choisi, dans le cadre de ce projet de recherche, de conserver ce résultat de vitesse de détection qui reste suffisant pour la réalisation de nos tests expérimentaux.

Néanmoins il est toujours possible d'améliorer la vitesse d'exécution du programme complet à partir de diverses solutions : d'un côté nous pouvons utiliser plusieurs GPU pour accroître la puissance de calculs et de l'autre nous pouvons utilisé une méthode de suivi d'objet ("tracking" en anglais) en parallèle de la détection afin d'alléger les ressources nécessaires pour la détection [105]. Pour la suite du projet, nous avons choisi de continuer avec ce paramétrage même si il y a un peu de latence durant les tests. Étant donné que nous nous limitons à l'utilisation d'une caméra, il est donc possible de se déplacer plus lentement durant les tests que dans le cas d'un drone, afin de réduire voire d'annuler la latence perçue. Ainsi l'objectif idéal, à savoir obtenir une vitesse de détection en temps réel, pourra être atteint dans de futurs travaux.

6.3 Tests de détection du support idéal

Cette section regroupe l'ensemble des tests qui ont été réalisés dans le but de trouver, selon les caractéristiques du préhenseur, le support idéal parmi plusieurs objets. À travers ces tests, nous allons évaluer les performances générales du modèle au complet ainsi que ses limites. Pour chacun des tests, nous avons effectué 10 répétitions d'essais afin d'assurer une bonne fidélité des résultats obtenus. Lors de chaque répétition, une fois la détection arrêtée, l'algorithme de détection nous renvoie l'image avec l'objet ayant eu le meilleur "PSI". La figure 6.1 montre un exemple du retour de fin de détection.

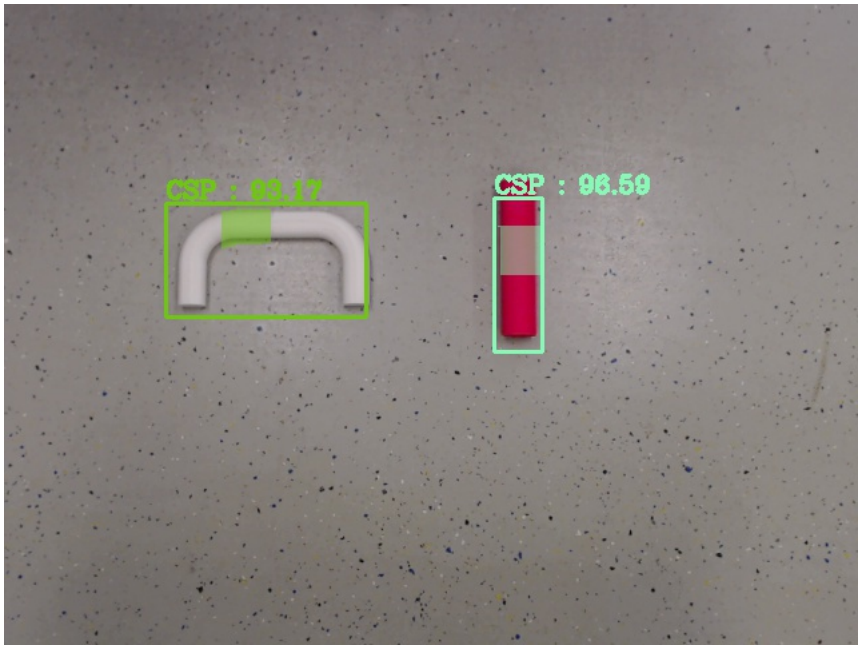


Figure 6.1 Retour du meilleur support idéal pour l'algorithme à la fin de la détection

6.3.1 Dispositif utilisé pour les essais

Les tests de détection ont été réalisés dans une salle d'intérieur avec un bon éclairage artificiel. Les objets utilisés pour les tests sont disposés sur le sol. Pour la caméra, elle est fixée en hauteur au-dessus des objets par l'intermédiaire d'un bras articulé. La figure 6.2 représente l'ensemble du dispositif utilisé.

6.3.2 Paramètres de configuration des tests

Pour déterminer la robustesse du modèle de détection, nous avons choisi de faire varier un maximum de paramètres et de conditions d'essais. Les paramètres de configuration pouvant varier sont les suivants :

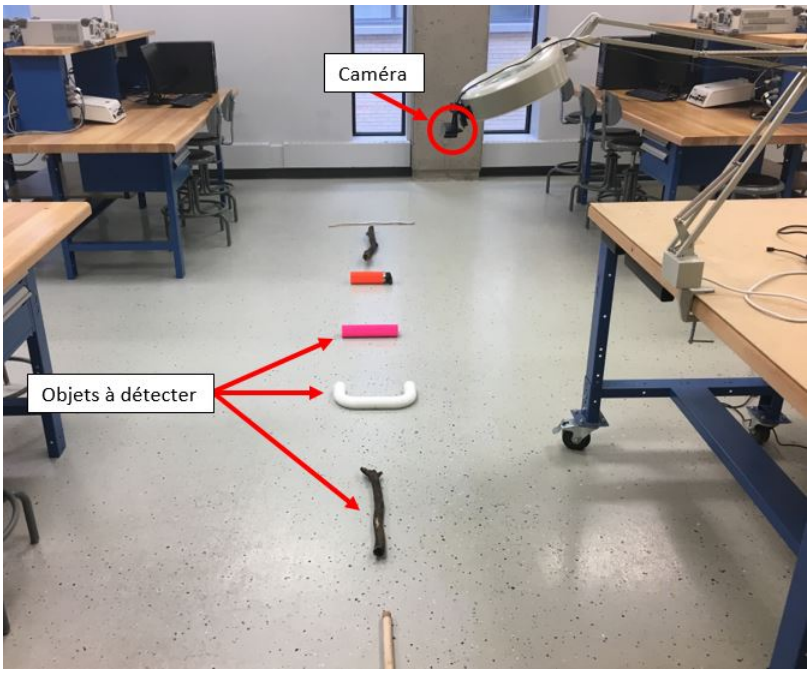


Figure 6.2 Représentation du dispositif utilisé pour les tests

- Le nombre d'objet par test
- Les caractéristiques du préhenseur
- La luminosité de la pièce
- La distance entre la caméra et les objets
- Modification de la place et de l'orientation des objets (horizontale ou verticale) entre chaque répétition de test

Les différentes valeurs prises par ces paramètres sont regroupées dans le tableau 6.2. Pour le nombre d'objet par test, plus il augmente et plus la différence de dimension entre les objets sera petite. Quand nous parlons de la différence de dimension entre les objets, énoncée dans le premier paramètre, la dimension évoquée correspond ici au diamètre de l'objet soit la dimension qui peut être comparée avec les caractéristiques du préhenseur. Dans la suite de cette partie, cette dimension sera nommée dimension préhensible afin de permettre une meilleure compréhension.

Les valeurs choisies pour chacun des paramètres sont regroupées dans le tableau 6.2.

Tableau 6.2 Valeurs des différents paramètres pour les tests de détection

Paramètres	**Valeurs de paramètre**
Nombre d'objet par test	4 / 6 / 8
Caractéristiques du préhenseur	Caractéristiques différentes (2 différentes) pour un nombre d'objets par test fixé
Distance caméra-objet (mètre)	1 m / 1.25 m / 1.5 m
Luminosité de la pièce	Normale / Faible

De ce fait, cela représente un ensemble de 36 tests avec pour chaque test 10 répétitions. Selon le nombre d'objets sélectionné nous avons :

Dans un premier temps, la différence de dimension (sur la dimension qui est préhensible) entre les objets qui varie :

- Pour 4 objets la différence de dimension entre chacun des objets sélectionnés est de 20mm minimum
- Pour 6 objets la différence de dimension entre chacun des objets sélectionnés est comprise entre 10mm et 20mm
- Pour 8 objets la différence de dimension entre chacun des objets sélectionnés est comprise entre 5mm et 10mm

Pour chaque test réalisé, les objets sélectionnés dans le cas où nous en avons 4,6 ou 8 ont des classes différentes et par conséquent des formes différentes.

Dans un deuxième temps, l'intervalle d'ouverture du préhenseur qui varie, entrainant ainsi un changement du support idéal.

Pour la luminosité de la salle, une luminosité normale correspond à l'utilisation de l'ensemble des spots lumineux (20 spots au total) et une luminosité faible correspond à l'utilisation de seulement 2 spots lumineux présent aux extrémités de la pièce.

6.3.3 Résultats obtenus pour la détection du support idéal

Chaque objet et/ou support idéal énoncé dans cette partie est renseigné en annexe B avec sa photo ainsi que son diamètre qui correspond à la dimension préhensible.

Pour les 12 premiers tests, nous avons choisi un nombre de 4 objets chacun de classes différentes et ayant une différence d'au moins 20mm entre leur dimension préhensible.

Le tableau 6.3 regroupe les caractéristiques des objets sélectionnés :

Tableau 6.3 Caractéristiques des objets pour le choix de 4 objets

Objets	**Classe d'objet**	**Dimension préhensible (mm)**
Objet n°6	Cylindre	80
Objet n°9	Lampadaire	19.1
Objet n°14	Branche	58.5 à 61.5
Objet n°3	Barrière	39.7

Pour ces 4 objets, nous avons obtenu les résultats affichés dans le tableau 6.4 :

Tableau 6.4 Résultats des 12 tests pour le choix de 4 objets

Tests	**Intervalle d'ouverture du préhenseur**	**Distance caméra-objet**	**Luminosité**	**Support idéal**	**Pourcentage de succès (10 répétitions)**
1	[5mm-85mm]	1 mètre	Normale	Objet n°3	100 %
2	[5mm-85mm]	1 mètre	Faible	Objet n°3	100 %
3	[5mm-85mm]	1.25 mètre	Normale	Objet n°3	100 %
4	[5mm-85mm]	1.25 mètre	Faible	Objet n°3	100 %
5	[5mm-85mm]	1.5 mètre	Normale	Objet n°3	100 %
6	[5mm-85mm]	1.5 mètre	Faible	Objet n°3	100 %
7	[10mm-120mm]	1 mètre	Normale	Objet n°14	100 %
8	[10mm-120mm]	1 mètre	Faible	Objet n°14	100 %
9	[10mm-120mm]	1.25 mètre	Normale	Objet n°14	100 %
10	[10mm-120mm]	1.25 mètre	Faible	Objet n°14	100 %
11	[10mm-120mm]	1.5 mètre	Normale	Objet n°14	100 %
12	[10mm-120mm]	1.5 mètre	Faible	Objet n°14	100 %

Pour les 12 tests suivants, nous avons choisi un nombre de 6 objets et ayant une différence comprise entre 10mm et 20 mm entre leur dimension préhensible.

Le tableau 6.5 regroupe les caractéristiques des objets sélectionnés :

Tableau 6.5 Caractéristiques des objets pour le choix de 6 objets

Objets	**Classe d'objet**	**Dimension préhensible (mm)**
Objet n°6	Cylindre	80
Objet n°7	Cylindre	43.6
Objet n°10	Branche	15.5 à 18.3
Objet n°12	Branche	56 à 58
Objet n°16	Branche	68.7 à 70.3
Objet n°2	Barrière	29.9

Pour ces 6 objets, nous avons obtenu les résultats affichés dans le tableau 6.6 :

Tableau 6.6 Résultats des 12 tests pour le choix de 6 objets

Tests	**Intervalle d'ouverture du préhenseur**	**Distance caméra-objet**	**Luminosité**	**Support idéal**	**Pourcentage de succès (10 répétitions)**
13	[0mm-70mm]	1 mètre	Normale	Objet n°2	100 %
14	[0mm-70mm]	1 mètre	Faible	Objet n°2	100 %
15	[0mm-70mm]	1.25 mètre	Normale	Objet n°2	70 %
16	[0mm-70mm]	1.25 mètre	Faible	Objet n°2	50 %
17	[0mm-70mm]	1.5 mètre	Normale	Objet n°2	50 %
18	[0mm-70mm]	1.5 mètre	Faible	Objet n°2	50 %
19	[10mm-114mm]	1 mètre	Normale	Objet n°12	100 %
20	[10mm-114mm]	1 mètre	Faible	Objet n°12	100 %
21	[10mm-114mm]	1.25 mètre	Normale	Objet n°12	100 %
22	[10mm-114mm]	1.25 mètre	Faible	Objet n°12	80 %
23	[10mm-114mm]	1.5 mètre	Normale	Objet n°12	90 %
24	[10mm-114mm]	1.5 mètre	Faible	Objet n°12	90 %

Pour les 12 derniers tests, nous avons choisi un nombre de 8 objets et ayant une différence comprise entre 5mm et 10 mm entre leur dimension préhensible.

Le tableau 6.7 regroupe les caractéristiques des objets sélectionnés :

Tableau 6.7 Caractéristiques des objets pour le choix de 8 objets

Objets	**Classe d'objet**	**Dimension préhensible (mm)**
Objet n°1	Cylindre	70
Objet n°5	Cylindre	62
Objet n°6	Cylindre	80
Objet n°8	Cylindre	24.2
Objet n°11	Branche	11.2 à 15
Objet n°13	Branche	38.6 à 41.5
Objet n°15	Branche	28.6 à 31
Objet n°4	Barrière	50.2

Pour ces 8 objets, nous avons obtenu les résultats affichés dans le tableau 6.8 :

Tableau 6.8 Résultats des 12 tests pour le choix de 8 objets

Tests	**Intervalle d'ouverture du préhenseur**	**Distance caméra-objet**	**Luminosité**	**Support idéal**	**Pourcentage de succès (10 répétitions)**
25	[0mm-134mm]	1 mètre	Normale	Objet n°5	80 %
26	[0mm-134mm]	1 mètre	Faible	Objet n°5	70 %
27	[0mm-134mm]	1.25 mètre	Normale	Objet n°5	90 %
28	[0mm-134mm]	1.25 mètre	Faible	Objet n°5	90 %
29	[0mm-134mm]	1.5 mètre	Normale	Objet n°5	50 %
30	[0mm-134mm]	1.5 mètre	Faible	Objet n°5	50 %
31	[10mm-80mm]	1 mètre	Normale	Objet n°13	70 %
32	[10mm-80mm]	1 mètre	Faible	Objet n°13	70 %
33	[10mm-80mm]	1.25 mètre	Normale	Objet n°13	80 %
34	[10mm-80mm]	1.25 mètre	Faible	Objet n°13	100 %
35	[10mm-80mm]	1.5 mètre	Normale	Objet n°13	50 %
36	[10mm-80mm]	1.5 mètre	Faible	Objet n°13	50 %

Nous pouvons remarquer rapidement à partir des différents tableaux obtenus (6.2, 6.4 et 6.6) que quelque soit les conditions d'environnement choisies, un test peut donner des résultats bien différents d'un autre. Ces différences sont principalement dues aux ombres des objets qui viennent perturber la bonne détermination du support idéal. Selon le test effectué, elles peuvent avoir un impact mineur ou majeur. Ainsi, pour les tests où la différence entre la dimension préhensible des objets devient petite le taux de succès risque de diminuer dû aux ombres des objets. C'est notamment pour cela que nous avons une baisse du taux de succès dans la détection du support idéal pour les tests avec une différence inférieure à 20mm. Une analyse complète et détaillée sur l'interprétation des résultats obtenus est effectuée dans le chapitre suivant.

CHAPITRE 7 INTERPRÉTATION DES RÉSULTATS OBTENUS ET VALIDATION DU MODÈLE

Ce chapitre est dédié à la discussion générale des performances obtenues du modèle au chapitre précédent. Tout d'abord, nous réalisons une interprétation des résultats fournis à partir des tableaux précédents. Ensuite, nous évoquons les différentes limitations du modèle qui ne nous permettent pas d'atteindre les 100 % de détection du support idéal, quelque soit la situation choisie. Par conséquent, la section suivante est dédiée à l'énonciation de solutions pouvant apporter une amélioration sur les résultats obtenus. Enfin, dans une dernière section, nous réalisons la validation de notre modèle de détection du support idéal pour le perchage de drones.

7.1 Interprétation des résultats

Les résultats obtenus à partir du tableau 6.4 indique que dans le cas d'objets ayant une différence d'au moins 20mm entre leur dimension préhensible et cela peu importe les paramètres utilisés et les conditions d'environnement (luminosité) : nous obtenons un succès de 100 % dans la détection du support idéal.

Pour le cas où nous avons des objets qui ont une différence comprise entre 10mm et 20mm entre leur dimension préhensible, nous obtenons un résultat de 81.67 % de succès, sur l'ensemble des 12 tests affichés dans le tableau 6.6, pour la détection du support idéal. Le support idéal correspond à l'objet ayant obtenu le score de concordance le plus élevé, selon les caractéristiques du préhenseur choisies.

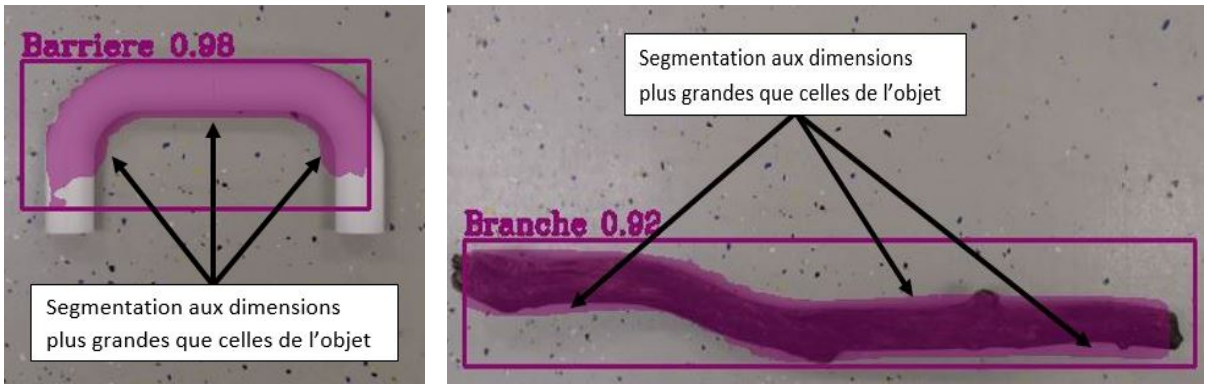
Pour le cas où nous avons des objets qui ont une différence comprise entre 5mm et 10mm entre leur dimension préhensible, nous obtenons un résultat de 70.83 % de succès, sur l'ensemble des 12 tests affichés dans le tableau 6.8, pour la détection du support idéal.

Pour conclure sur l'ensemble des tests effectués, nous avons obtenu un taux de succès global, pour la détection du support idéal et avec ce modèle complet (Mask-RCNN et l'algorithme CSP), égale à 84.17 %. Ainsi, les résultats obtenus pour l'ensemble des tests n'atteignent pas les 100 % de succès désirés. Ceci est dû à plusieurs causes que nous détaillerons dans la section suivante.

7.2 Causes de mauvaise détection du support idéal

Comme nous avons pu le voir à partir des résultats de la sous-section précédente, lorsque la différence de dimension préhensible entre les objets est inférieure à 20mm le pourcentage de succès pour la détection du support idéal est inférieur à 100 %. Cette diminution est due à plusieurs causes.

Tout d'abord, nous retrouvons la taille de la segmentation des objets. En effet, après la réalisation des premiers essais, nous avons remarqué que pour une dimension d'ouverture optimale du préhenseur égale à la dimension d'un objet nous obtenions un "CSP" inférieur à 100 %. Après avoir analyser les masques obtenus en sortie de l'algorithme de Mask-RCNN, nous avons remarqué que la segmentation des objets est toujours un peu plus grande que l'objet lui-même. Les images présentées sur la figure 7.1 viennent confirmées cette remarque. Selon les objets, la différence entre les dimensions de la segmentation et les dimensions de l'objet sera plus ou moins importante.



(a) (b)

Figure 7.1 Segmentation avec ses dimensions supérieures à celles de l'objet

Ensuite, nous avons remarqué que nous n'avions pratiquement jamais de réelle mauvaise détection du support idéal mais un conflit entre 2 objets qui obtenaient le même "CSP". La figure 7.2 montre ce conflit de "CSP" pour le support idéal.

Ceci est dû au fait que l'algorithme prend le meilleur "CSP" de chaque objet et le compare au meilleur "CSP" obtenu au cours de la détection. S'il est supérieur ou égale il sera remplacé, et lors de l'arrêt de la détection il renvoie le meilleur "CSP" obtenu sur l'ensemble de la détection sachant qu'un "CSP" égale à pu être présent plus tôt dans la détection. Si nous le remplaçons juste lorsque il est supérieur strictement cela revient à la même chose car l'algorithme renverra le meilleur "CSP" obtenu sachant qu'un "CSP" égale à pu être présent

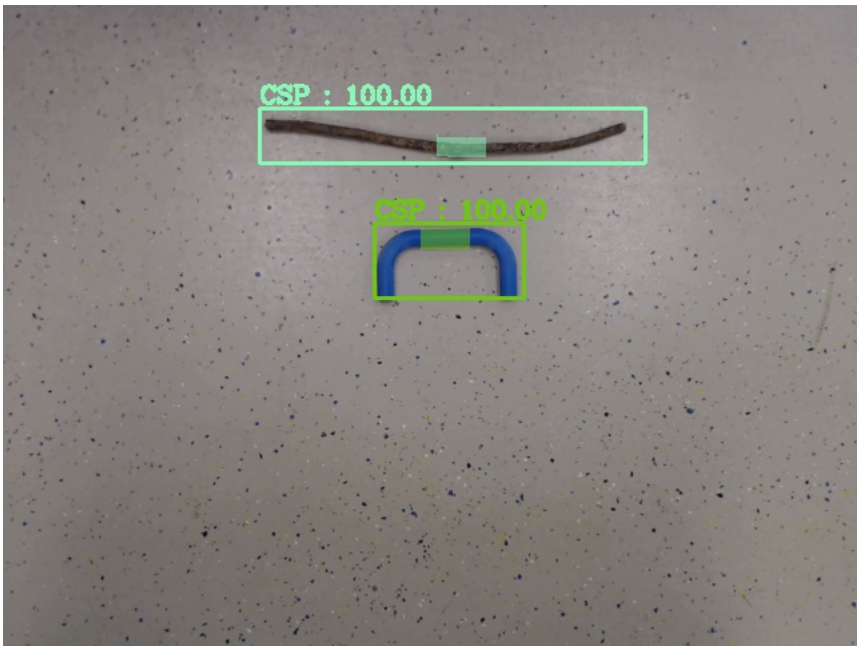


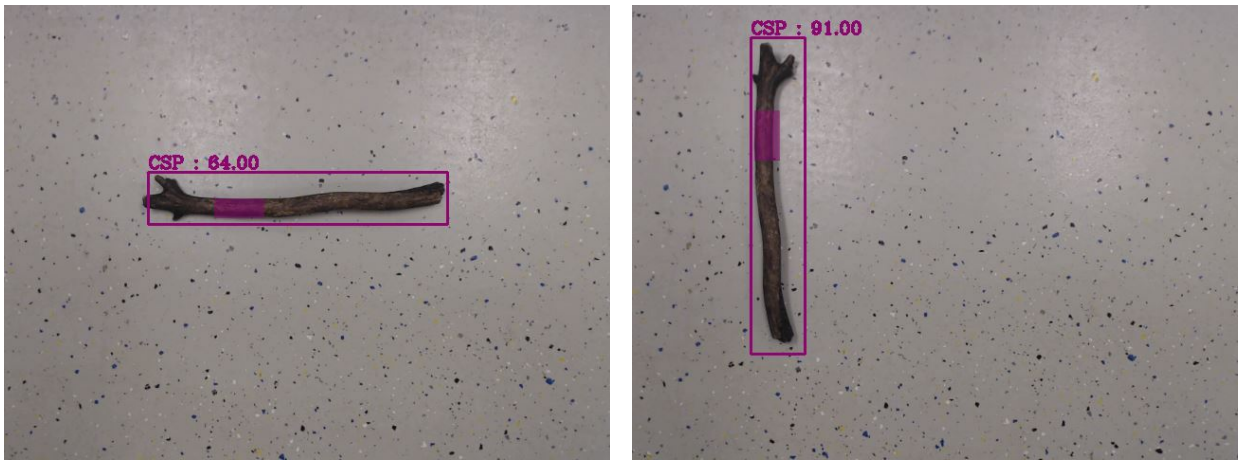
Figure 7.2 Conflit de "CSP" pour le support idéal

plus tard dans la détection. Ce conflit est donc directement lié à la diminution du taux de succès de la détection du bon support idéal.

La dernière cause, responsable notamment du conflit évoqué précédemment et provoquant une mauvaise segmentation des objets, est la présence d'ombre. En effet, nous pouvons voir clairement que la branche sur la figure 7.2 possède une dimension plus petite que la barrière mais arrive à obtenir un "CSP" égale à celui de cette dernière. L'ombre des objet est due à deux facteurs :

- Les objets sont directement posés sur le sol créant ainsi d'avantage d'ombre
- Le fait d'avoir une lumière artificielle venant de spots lumineux présents dans différents endroits de la pièce accentuant plus ou moins les ombres selon la disposition et le placement des objets dans la salle

Si nous comparons les 2 images présentes sur la figure 7.3, nous observons dans une premier temps la différence entre une bonne segmentation de l'objet en (a) et une mauvaise segmentation en (b). Dans un deuxième temps, nous remarquons que la mauvaise segmentation de l'objet entraine une sur-évaluation du "CSP". En effet, le score d'association correspondant à une bonne caractérisation des dimensions de l'objet est de 79.55 % alors que pour une mauvaise caractérisation il est de 97.73 %, soit une augmentation de 18.18 %. Pour le traduire en



(a) Pas de présence d'ombre, bonne segmentation de l'objet

(b) Présence d'ombre importante, mauvaise segmentation de l'objet

Figure 7.3 Comparaison de segmentations d'objet avec et sans présence d'ombre

taille réelle, les images obtenues en figure 7.5 correspondent à un test ayant pour intervalle d'ouverture du préhenseur : [10mm-80mm] soit une dimension optimale de 45mm pour la préhension correspondant à un "CSP" égale à 100 %. L'objet correspond à la branche n°3 présente dans le tableau 6.7 ayant un diamètre compris entre 28.6mm et 31mm. Un "CSP" égale à 79.55 % correspond, pour les caractéristiques du préhenseur choisies, à une mesure de 35.8mm soit relativement proche des 31mm. En revanche un "CSP" égale à 97.73 % correspond à une mesure de 44mm qui représente environ 8mm de plus que dans le cas où il n'y a pas d'ombre.

En résumé, selon les conditions d'environnement d'un test et la disposition des objets, un objet ayant un diamètre plus petit qu'un autre peut, à cause de son ombre, obtenir le même ou un meilleur score de concordance pour la détection du support idéal que ce dernier. Ainsi, pour les tests où la différence entre la dimension préhensible des objets devient petite le taux de succès risque de diminuer dû aux ombres des objets. C'est notamment pour cela que nous avons une baisse du taux de succès dans la détection du support idéal pour les tests avec une différence inférieure à 20mm.

7.3 Discussion générale

Pour rappel, l'objectif générale de ce projet de recherche est de développer un système de détection en temps réel qui, à partir des caractéristiques du système de préhension du drone, détermine le meilleur support pour son perchage. Lors de la détection d'un objet, il devient un support potentiel et se voit attribué un pourcentage de concordance entre ses dimensions et celles du préhenseur. Ensuite, une comparaison est faite entre l'ensemble des pourcentages de concordance obtenus. Le support le plus adéquat pour le perchage du drone correspondra à celui ayant obtenu le pourcentage le plus élevé c'est à dire celui se rapprochant le plus des 100%. Afin d'atteindre cet objectif, 4 sous-objectif ont dû être réalisés au préalable.

L'objectif spécifique n°1 est de développer un outil de détection adapté au cas du perchage de drone. Pour cela, il a été nécessaire de ré-entrainer l'algorithme de détection Mask-RCNN [100] pour de nouvelles classes d'objets. Nous avons, à la suite de plusieurs entrainements et d'évaluations de modèles, réussi à obtenir un modèle obtenant de bonnes performances que ce soit sur la précision comme sur la vitesse de détection. Pour la vitesse de détection, notre modèle atteint un nombre de 6.9 fps permettant d'avoir un outil de détection en temps réel. Par conséquent, l'objectif spécifique n°1 est validé. Cependant, dans notre cas, nous nous limitons à la détection d'un seul objet par image. Dans le cas où nous retrouvons plusieurs objets par image, le nombre de fps du modèle sera inférieur au 5 fps requis pour avoir un modèle de détection en temps réel. Afin d'obtenir de meilleures performances en terme de vitesse d'exécution dans la détection d'objets, deux solutions s'offrent à nous. D'une part, l'utilisation de la méthode de suivi d'objet pourrait diminuer le temps de traitement par image, dû au fait que cette méthode est moins coûteuse en temps de traitement que la détection [105]. La méthode de suivi d'objets permet, à partir d'une image vidéo obtenue par détection, de prévoir le mouvement des objets dans les images vidéo suivantes. D'autre part, l'apport de GPU supplémentaires augmenterait la vitesse de calculs et à fortiori la vitesse de détection du modèle. Ces améliorations permettraient ainsi d'atteindre un nombre de 5 fps tout en détectant plusieurs objets par image et nous permettrait donc d'atteindre le cas idéal de notre objectif principal.

L'objectif spécifique n°2 correspond au développement d'un algorithme permettant de déterminer les dimensions des objets détectés. Cet objectif a été réalisé à partir de la segmentation des objets fournies par l'algorithme Mask-RCNN. Les masques obtenus de chaque objet détecté, ont permis d'accéder aux dimensions des objets dans les images. Ensuite, à partir des paramètres de calibration de la caméra et de la distance entre la caméra et l'objet détecté,

nous avons pu accéder à leurs dimensions réelles. La première partie de notre algorithme CSP permet l'accomplissement de ce deuxième objectif spécifique. La limitation principale, dans la bonne détermination des dimensions réelles des objets détectés, correspond à la qualité des masques obtenus. En effet, une mauvaise segmentation des objets impacte directement la bonne détermination de leurs dimensions. Cette limitation s'applique uniquement sur l'algorithme Mask-RCNN, l'algorithme CSP réalise seulement un traitement du masque quelque soit sa qualité. Une solution contre la mauvaise segmentation des objets serait d'augmenter la précision de détection. Un entrainement fait à partir d'une base de données plus importante et/ou la recherche de paramètres d'entrainement optimaux permettraient cette augmentation.

L'objectif spécifique n°3 correspondant à la suite directe du n°2 consiste à développer un algorithme permettant de déterminer le support le plus adéquat pour le perchage du drone. Cet objectif a été réalisé avec succès par la deuxième partie de l'algorithme CSP. Pour chaque objet détecté, la fonction "Concordance" effectue une comparaison entre ses dimensions de celles du préhenseur et fournie en sortie un pourcentage. Ce pourcentage représente l'affinité entre le support et le système de préhension. Chaque pourcentage de concordance obtenu est comparé aux autres et le meilleur d'entre eux est conservé en mémoire jusqu'à l'arrêt de la détection. Le meilleur pourcentage ainsi obtenu, nous permet de connaître le meilleur support. La limitation de ce procédé est la détermination d'un seul "CSP" pour chacun des objets détectés. Ainsi, les chances d'avoir un conflit dans la décision du meilleur support, dans le cas où deux supports obtiennent le même score, peuvent être élevées. L'utilisation de coordonnées GPS permettrait de savoir à quel objet correspond tel "CSP" et ainsi calculer un "CSP" moyen par objet dans le but de supprimer le problème de conflit entre deux objets pour la détermination du support idéal.

L'objectif spécifique n°4 correspond à la validation du système de détection au complet, à savoir l'algorithme Mask-RCNN couplé avec l'algorithme CSP pour le cas d'application du perchage de drones. Pour atteindre cet objectif, nous avons réalisé un ensemble de 36 tests avec différents paramétrages et dans des conditions d'environnement différents. La validation ultime et complète de ce projet de recherche est atteinte dans le cas où nous arrivons à concevoir un système de détection en temps réel qui localise à 100 % le support le plus adéquat, et cela quelque soit les paramètres et les conditions d'environnement définis lors de la détection. Cependant de ce côté l'objectif principal idéal n'est pas atteint puisque la solution proposée ici n'atteint pas les 5 fps souhaités qui correspond au minimum à obtenir pour avoir une détection en temps réel. Pour ce qui est de la performance sur la précision

de détection, le système obtient une erreur de 15.8 % principalement due aux ombres des objets et à fortiori à une mauvaise segmentation des objets. En résumé, les erreurs sont principalement liées à la partie du système correspondant à l'algorithme Mask-RCNN qui réalise la détection et la segmentation des objets. Tous les algorithmes de détection existants possèdent un score de détection inférieur à 100 %, ainsi il est difficile d'avoir une détection et une segmentation parfaite. L'objectif spécifique n°4 est donc validé en grande partie. En effet, pour la précision du modèle, nous arrivons tout de même à obtenir un taux de succès de 100% dans le cas où les objets ont une différence de diamètre d'au moins 20mm entre eux et une bonne performance en général. Les solutions a apporté pour améliorer, notamment le temps de traitement permettant d'atteindre notre cas idéal, sont les mêmes que celles énoncés pour l'objectif spécifique n°1.

CHAPITRE 8 CONCLUSION

8.1 Synthèse des travaux

Les résultats de chacun des objectifs spécifiques ont permis d'atteindre complètement l'objectif général. Le premier sous-objectif a permis d'obtenir un détecteur adapté au cas du perchage de drones à partir d'un ré-entrainement du réseau de neurones sur de nouvelles classes d'objets. Le deuxième et le troisième sous-objectifs ont donné lieu, respectivement, à l'obtention des dimensions réelles des objets et du pourcentage de concordance entre les objets détectés et le préhenseur utilisé donnant accès à fortiori au meilleur support pour le perchage du drone. Le dernier a permis de démontrer, à partir des tests expérimentaux, qu'il est possible de concevoir un système de détection qui, à partir de l'intelligence artificielle, réussit à déterminer le support idéal pour le perchage d'un drone, même si les critères de performances n'ont pas été complètement atteints. En effet, le système de détection obtient une performance globale de 84.2 % pour la bonne localisation du support idéal sur l'ensemble des tests réalisés. Cette valeur montre que le système obtenu reste très performant puisqu'il trouvera dans plus de 4 cas sur 5, le meilleur support. De plus, il obtient un taux de succès égale à 100 % pour des objets ayant une différence de diamètre d'au moins 20mm entre eux. La performance diminue lorsque cette différence passe en dessous de cette valeur. L'exécution du modèle prend en moyenne 2.38 fps dans le cas d'un objet par image. Cette dernière comprend la détection des objets ainsi que la comparaison des dimensions des objets détectés avec celles du préhenseur. Ce nombre représente la moitié de la valeur souhaitée pour atteindre l'objectif principal idéal mais plusieurs améliorations peuvent être entreprises pour y parvenir.

Ces résultats sont importants car ils permettent d'apporter une solution viable afin de combler le problème de concordance qu'il peut y avoir entre les supports détectés et le préhenseur du drone, pouvant entrainer l'échec du perchage. En effet, le modèle complet, présenté dans ce mémoire, permet de déterminer le support le plus adéquat possible et à fortiori le meilleur taux de succès dans le perchage du drone.

8.2 Perspectives de ce travail

Les perspectives de ce travail sont d'apporter des améliorations à la fois sur la précision et la vitesse de détection. La précision de notre modèle correspond au taux de succès dans la détermination de l'objet ayant la dimension qui coïncide le plus avec celle du préhenseur du drone. Cet objet est qualifié de support idéal.

Tout d'abord, pour obtenir de meilleurs résultats sur la précision, il faut empêcher les conflits entre objets qui possèdent le même "CSP" et éviter au maximum les mauvaises segmentations dues aux ombres des objets. L'emploi d'une base de données d'images de plus grande taille, pour les classes utilisées, permettrait de réaliser un meilleur entrainement du réseau de neurones de l'algorithme Mask-RCNN et à fortiori améliorerait la précision de détection de l'algorithme ainsi que la segmentation des objets.
Il serait également judicieux de calculer un "CSP" moyen par objet. Dans notre cas, étant donné que la vitesse d'exécution de l'algorithme est de 2.38 fps, avant le passage d'un objet à un autre, l'algorithme du modèle complet obtiendra plusieurs calculs de "CSP" pour un même objet. Par conséquent, si nous calculons un "CSP" moyen à partir de l'ensemble des "CSP" obtenus pour chaque objet, le conflit entre objet deviendra très faible car chacune des moyennes sera très précise et donc facilement comparable avec les autres, assurant ainsi la détermination d'un support idéal unique. Pour réaliser cela, il est nécessaire d'utiliser une méthode permettant à l'algorithme de savoir à quel objet appartient chacun des "CSP" calculés. Une méthode possible serait d'utiliser des coordonnées GPS pour savoir où chacun des objets détectés se trouve.
L'utilisation du système de détection dans des conditions plus réelles assurerait aussi une meilleure précision. En effet, si la détection s'effectue en extérieur, d'une part les objets seront à une certaine distance du sol permettant d'avoir leurs ombres bien détachées et d'autre part la luminosité naturelle fournira potentiellement des ombres plus nettes et orientées dans une seule direction offrant de meilleures conditions de tests pour la détection et la segmentation.

Pour l'amélioration de la vitesse de détection, l'utilisation de la méthode de suivi d'objet, en parallèle de la détection, permettrait de diminuer considérablement le temps de traitement par image. L'optimisation du code actuel ainsi que la recherche d'autres méthodes d'amélioration pouvant sauver du temps d'exécution sont également à envisager. L'application de cet ensemble permettrait d'atteindre un nombre minimum de 5 fps afin d'obtenir un détecteur en temps réel.

Enfin, l'apport d'un capteur ultrason permettrait de déterminer, en temps réel et dans l'ensemble de l'espace, la distance précise entre la caméra et les objets. Ainsi le calcul du coefficient, permettant la conversion de la taille réelle des objets en pixels, s'effectuera en tout temps lors du déplacement de la caméra dans l'espace.

Le but ultime dans la suite de ce travail serait d'incorporer ce modèle de détection complet directement sur un drone, afin de réaliser des tests sur le perchage automatique de drone basé sur la vision artificielle.

RÉFÉRENCES

[1] C. Luo, L. Yu et P. Ren, "A vision-aided approach to perching a bioinspired unmanned aerial vehicle," *IEEE Transactions on Industrial Electronics*, vol. 65, n°. 5, p. 3976–3984, 2018.

[2] M. Ezgi, "Lecture notes in : R-cnn for object detection," https ://courses.cs.washington.edu/courses/cse590v/14au/cse590v_wk1_rcnn.pdf, 2014-03-10.

[3] J. Redmon, S. Divvala, R. Girshick et A. Farhadi, "You only look once : Unified, real-time object detection," dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, p. 779–788.

[4] H. Kaiming. (2017) Mask r-cnn : A perspective on equivariance. http://kaiminghe.com/iccv17tutorial/maskrcnn_iccv2017_tutorial_kaiminghe.pdf. Accessed : 2019-04-10.

[5] P. Barooah, G. E. Collins et J. P. Hespanha, "Geotrack : bio-inspired global video tracking by networks of unmanned aircraft systems," dans *Bio-Inspired/Biomimetic Sensor Technologies and Applications*, vol. 7321. International Society for Optics and Photonics, 2009, p. 73210F.

[6] M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno et V. Kumar, "System for deployment of groups of unmanned micro aerial vehicles in gps-denied environments using onboard visual relative localization," *Autonomous Robots*, vol. 41, n°. 4, p. 919–944, 2017.

[7] V. Spurnỳ, T. Báča, M. Saska, R. Pěnička, T. Krajník, J. Thomas, D. Thakur, G. Loianno et V. Kumar, "Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles," *Journal of Field Robotics*, vol. 36, n°. 1, p. 125–148, 2019.

[8] O. Araar, N. Aouf et I. Vitanov, "Vision based autonomous landing of multirotor uav on moving platform," *Journal of Intelligent & Robotic Systems*, vol. 85, n°. 2, p. 369–384, 2017.

[9] Olivier, M. Anae et R. Sylvain. (2018-11-29) Drone avec caméra. https ://www.les-drones.com/drone-avec-camera/. Accessed : 2019-04-10.

[10] Prodrone. (2019) Products. https ://www.prodrone.com/products/. Accessed : 2019-04-10.

[11] J. Thomas, G. Loianno, J. Polin, K. Sreenath et V. Kumar, “Toward autonomous avian-inspired grasping for micro aerial vehicles,” *Bioinspiration & biomimetics*, vol. 9, n°. 2, p. 025010, 2014.

[12] C. Cortes et V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, n°. 3, p. 273–297, 1995.

[13] D. Larousserie. (2018-08-14) Après soixante ans de hauts et de bas, les réseaux de neurones triomphent. https ://www.lemonde.fr/sciences/article/2018/08/14/apres-soixante-ans-de-hauts-et-de-bas-les-reseaux-de-neurones-triomphent_5342330_1650684.html. Accessed : 2019-04-10.

[14] C. Crouspeyre. (2017-07-17) Comment les réseaux de neurones à convolution fonctionnent. https ://medium.com/@CharlesCrouspeyre/comment-les-r%C3%A9seaux-de-neurones-%C3%A0-convolution-fonctionnent-b288519dbcf8. Accessed : 2019-04-10.

[15] Z. Zhao, P. Zheng, S. Xu et X. Wu, “Object detection with deep learning : A review,” *CoRR*, vol. abs/1807.05511, 2018. [En ligne]. Disponible : http: //arxiv.org/abs/1807.05511

[16] C. Bousquet-Jette, S. Achiche, D. Beaini, Y. L.-K. Cio, C. Leblond-Ménard et M. Raison, “Fast scene analysis using vision and artificial intelligence for object prehension by an assistive robot,” *Engineering Applications of Artificial Intelligence*, vol. 63, p. 33–44, 2017.

[17] G. Gaudet, M. Raison et S. Achiche, “Classification of upper limb phantom movements in transhumeral amputees using electromyographic and kinematic features,” *Engineering Applications of Artificial Intelligence*, vol. 68, p. 153–164, 2018.

[18] A. Mohebbi, L. Baron, S. Achiche et L. Birglen, “Neural network-based decision support for conceptual design of a mechatronic system using mechatronic multi-criteria profile (mmp),” dans *Proceedings of the 2014 International Conference on Innovative Design and Manufacturing (ICIDM)*. IEEE, 2014, p. 105–110.

[19] A. Mohebbi, S. Achiche et L. Baron, “Multi-criteria fuzzy decision support for conceptual evaluation in design of mechatronic systems : a quadrotor design case study,” *Research in Engineering Design*, vol. 29, n°. 3, p. 329–349, 2018.

[20] S. Achiche, M. Shlechtingen, M. Raison, L. Baron et I. F. Santos, “Adaptive neuro-fuzzy inference system models for force prediction of a mechatronic flexible structure,” *Journal of Integrated Design and Process Science*, vol. 19, n°. 3, p. 77–94, 2015.

[21] J. Sarthak. (2017-01-30) Nanonets : How to use deep learning when you have limited data. https ://medium.com/nanonets/nanonets-how-to-use-deep-learning-when-you-have-limited-data-f68c0b512cab. Accessed : 2019-04-10.

[22] W. Koehrsen. Beyond accuracy : Precision and recall. https ://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c. Accessed : 2019-04-10.

[23] C. Voskoglou. (2017-05-05) What is the best programming language for machine learning ? https ://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7. Accessed : 2019-05-13.

[24] J. Krywyk et P.-A. Jachiet. (2016-10-25) Classification d'images : les réseaux de neurones convolutifs en toute simplicité. https ://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicite/. Accessed : 2019-04-10.

[25] R. Dolbeau, "Theoretical peak flops per instruction set on modern intel cpus," 2015.

[26] Intel_Corporation. (2018) Export compliance metrics for intel microprocessors intel core processors. https ://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Core-Processors.pdf. Accessed : 2019-04-10.

[27] TechPowerUp. (2019) Gpu specs database. https ://www.techpowerup.com/gpu-specs/. Accessed : 2019-04-10.

[28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár et C. L. Zitnick, "Microsoft coco : Common objects in context," dans *European conference on computer vision*. Springer, 2014, p. 740–755.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li et L. Fei-Fei, "Imagenet : A large-scale hierarchical image database," dans *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, p. 248–255.

[30] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva et A. Torralba, "Sun database : Large-scale scene recognition from abbey to zoo," dans *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, p. 3485–3492.

[31] M. Everingham, L. Van Gool, C. K. Williams, J. Winn et A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, n°. 2, p. 303–338, 2010.

[32] Y. Zhou, H. Nejati, T.-T. Do, N.-M. Cheung et L. Cheah, "Image-based vehicle analysis using deep neural network : A systematic study," dans *2016 IEEE International Conference on Digital Signal Processing (DSP)*. IEEE, 2016, p. 276–280.

[33] R. Qian, Q. Liu, Y. Yue, F. Coenen et B. Zhang, "Road surface traffic sign detection with hybrid region proposal and fast r-cnn," dans *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2016, p. 555–559.

[34] H. Zhang, Y. Du, S. Ning, Y. Zhang, S. Yang et C. Du, "Pedestrian detection method based on faster r-cnn," dans *2017 13th International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2017, p. 427–430.

[35] X. Qifang, Y. Guoqing et L. Pin, "Aircraft detection of high-resolution remote sensing image based on faster r-cnn model and ssd model," dans *Proceedings of the 2018 International Conference on Image and Graphics Processing*. ACM, 2018, p. 133–137.

[36] R. Girshick, J. Donahue, T. Darrell et J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, n°. 1, p. 142–158, 2016.

[37] R. Girshick, "Fast r-cnn," dans *Proceedings of the IEEE international conference on computer vision*, 2015, p. 1440–1448.

[38] S. Ren, K. He, R. Girshick et J. Sun, "Faster r-cnn : Towards real-time object detection with region proposal networks," dans *Advances in neural information processing systems*, 2015, p. 91–99.

[39] J. Dai, Y. Li, K. He et J. Sun, "R-fcn : Object detection via region-based fully convolutional networks," dans *Advances in neural information processing systems*, 2016, p. 379–387.

[40] K. He, G. Gkioxari, P. Dollár et R. Girshick, "Mask r-cnn," dans *Proceedings of the IEEE international conference on computer vision*, 2017, p. 2961–2969.

[41] J. Redmon. Yolo : Real-time object detection. https ://pjreddie.com/darknet/yolo/. Accessed : 2019-04-10.

[42] J. Redmon et A. Farhadi, "Yolo9000 : better, faster, stronger," dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, p. 7263–7271.

[43] Joseph Redmon et A. Farhadi, "Yolov3 : An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [En ligne]. Disponible : http://arxiv.org/abs/1804.02767

[44] Interface-z. (2002-2019) Temps réel. https ://www.interface-z.fr/conseils/temps_reel.htm. Accessed : 2019-05-13.

[45] A. Samir, B. A. Mohamed et B. A. Abdelhakim, "A new architecture based on convolutional neural networks (cnn) for assisting the driver in fog environment," dans *Proceedings of the 3rd International Conference on Smart City Applications*. ACM, 2018, p. 85.

[46] T. Haryanto, I. Wasito et H. Suhartanto, “Convolutional neural network (cnn) for gland images classification,” dans *2017 11th International Conference on Information & Communication Technology and System (ICTS)*. IEEE, 2017, p. 55–60.

[47] S. Kido, Y. Hirano et N. Hashimoto, “Detection and classification of lung abnormalities by use of convolutional neural network (cnn) and regions with cnn features (r-cnn),” dans *2018 International Workshop on Advanced Image Technology (IWAIT)*. IEEE, 2018, p. 1–4.

[48] F.-C. Chen et M. R. Jahanshahi, “Nb-cnn : Deep learning-based crack detection using convolutional neural network and naïve bayes data fusion,” *IEEE Transactions on Industrial Electronics*, vol. 65, n°. 5, p. 4392–4400, 2018.

[49] A. Mousavian, D. Anguelov, J. Flynn et J. Kosecka, “3d bounding box estimation using deep learning and geometry,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, p. 7074–7082.

[50] A. Vialard. (2014-09-26) Segmentation et analyse d’images (partie 1). http ://dept-info.labri.fr/ vialard/Image3D/cours/cours-segmentation.pdf. Accessed : 2019-04-10.

[51] Telecom_Physique_Strasbourg. (2016) Analyse d’images : Segmentation. http ://images.icube.unistra.fr/fr/img_auth.php/archive/6/6f/20161007140608%213-Segmentation.pdf. Accessed : 2019-04-10.

[52] H. Glotin, F. Benard et C. Kermorvant. Segmentation d’images : principes. http ://glotin.univ-tln.fr/MCBIR/Segmentation_images_principes.pdf. Accessed : 2019-04-10.

[53] J. Long, E. Shelhamer et T. Darrell, “Fully convolutional networks for semantic segmentation,” dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, p. 3431–3440.

[54] R. Zhang, J. Yao, K. Zhang, C. Feng et J. Zhang, “S-cnn-based ship detection from high-resolution remote sensing images.” *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 41, 2016.

[55] A. W. Harley, K. G. Derpanis et I. Kokkinos, “Segmentation-aware convolutional networks using local attention masks,” dans *Proceedings of the IEEE International Conference on Computer Vision*, 2017, p. 5038–5047.

[56] D. Beaini, S. Achiche, A. Duperré et M. Raison, “Deep green function convolution for improving saliency in convolutional neural networks,” *arXiv preprint arXiv :1908.08331*, 2019.

[57] Q. Hou, M.-M. Cheng, X. Hu, A. Borji, Z. Tu et P. H. Torr, “Deeply supervised salient object detection with short connections,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, p. 3203–3212.

[58] J. Lecoeur et C. Barillot, “Segmentation d’images cérébrales : État de l’art,” Thèse de doctorat, INRIA, 2006.

[59] J. Dai, K. He et J. Sun, “Boxsup : Exploiting bounding boxes to supervise convolutional networks for semantic segmentation,” dans *Proceedings of the IEEE International Conference on Computer Vision*, 2015, p. 1635–1643.

[60] P. O. Pinheiro, R. Collobert et P. Dollár, “Learning to segment object candidates,” dans *Advances in Neural Information Processing Systems*, 2015, p. 1990–1998.

[61] P. O. Pinheiro, T.-Y. Lin, R. Collobert et P. Dollár, “Learning to refine object segments,” dans *European Conference on Computer Vision*. Springer, 2016, p. 75–91.

[62] J. PINTO, “Masknet : An instance segmentation algorithm,” 2017.

[63] H. Hu, S. Lan, Y. Jiang, Z. Cao et F. Sha, “Fastmask : Segment multi-scale object candidates in one shot,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, p. 991–999.

[64] Y.-T. Hu, J.-B. Huang et A. Schwing, “Maskrnn : Instance level video object segmentation,” dans *Advances in Neural Information Processing Systems*, 2017, p. 325–334.

[65] D. Pathak, Y. Shentu, D. Chen, P. Agrawal, T. Darrell, S. Levine et J. Malik, “Learning instance segmentation by interaction,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, p. 2042–2045.

[66] L.-C. Chen, A. Hermans, G. Papandreou, F. Schroff, P. Wang et H. Adam, “Masklab : Instance segmentation by refining object detection with semantic and direction features,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, p. 4013–4022.

[67] W. Fang, B. Zhong, N. Zhao, P. E. Love, H. Luo, J. Xue et S. Xu, “A deep learning-based approach for mitigating falls from height with computer vision : Convolutional neural network,” *Advanced Engineering Informatics*, vol. 39, p. 170–177, 2019.

[68] M. Liu, J. Dong, X. Dong, H. Yu et L. Qi, “Segmentation of lung nodule in ct images based on mask r-cnn,” dans *2018 9th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2018, p. 1–6.

[69] Institut_Maupertuis. (2014-04) Robotique : Les préhenseurs adaptatifs. https ://www.institutmaupertuis.fr/include/telechargement.php ?id_doc=142&fichier=1. Accessed : 2019-04-10.

[70] S. B. Backus, L. U. Odhner et A. M. Dollar, "Design of hands for aerial manipulation : Actuator number and routing for grasping and perching," dans *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, p. 34–40.

[71] J. Qi, J. Kang et X. Lu, "Design and research of uav autonomous grasping system," dans *2017 IEEE International Conference on Unmanned Systems (ICUS)*. IEEE, 2017, p. 126–131.

[72] Prodrone. (2016-09-07) Prodrone unveils the world's first dual robot arm large-format drone. https ://www.prodrone.com/archives/1420/. Accessed : 2019-04-10.

[73] U. A. Fiaz, M. Abdelkader et J. S. Shamma, "An intelligent gripper design for autonomous aerial transport with passive magnetic grasping and dual-impulsive release," dans *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2018, p. 1027–1032.

[74] C. C. Kessens, J. Thomas, J. P. Desai et V. Kumar, "Versatile aerial grasping using self-sealing suction," dans *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, p. 3249–3254.

[75] P. Xie et O. Ma, "Grasping analysis of a bio-inspired uav/mav perching mechanism," dans *ASME 2013 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 2013, p. V001T01A012–V001T01A012.

[76] C. E. Doyle, J. J. Bird, T. A. Isom, J. C. Kallman, D. F. Bareiss, D. J. Dunlop, R. J. King, J. J. Abbott et M. A. Minor, "An avian-inspired passive mechanism for quadrotor perching," *IEEE/ASME Transactions on Mechatronics*, vol. 18, n°. 2, p. 506–517, 2013.

[77] A. Torres, F. A. C. Herías, D. Mira et F. T. Medina, "Dm-uav : Dexterous manipulation unmanned aerial vehicle." dans *ICAART (1)*, 2017, p. 153–158.

[78] E. W. Hawkes, D. L. Christensen, E. V. Eason, M. A. Estrada, M. Heverly, E. Hilgemann, H. Jiang, M. T. Pope, A. Parness et M. R. Cutkosky, "Dynamic surface grasping with directional adhesion," dans *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, p. 5487–5493.

[79] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars et L. Van Eycken, "Cnn-based single image obstacle avoidance on a quadrotor," dans *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, p. 6369–6374.

[80] T. Baca, P. Stepan, V. Spurny, D. Hert, R. Penicka, M. Saska, J. Thomas, G. Loianno et V. Kumar, "Autonomous landing on a moving vehicle with an unmanned aerial vehicle," *Journal of Field Robotics*, 2017.

[81] R. Opromolla, G. Fasano et D. Accardo, “A vision-based approach to uav detection and tracking in cooperative applications,” *Sensors*, vol. 18, n°. 10, p. 3391, 2018.

[82] M. Saska, T. Baca et D. Hert, “Formations of unmanned micro aerial vehicles led by migrating virtual leader,” dans *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2016, p. 1–6.

[83] S. Gavois. (2015-12-01) Prime air : Amazon présente un nouveau drone de livraison, capable de voler à 90 km/h. https ://www.nextinpact.com/news/97478-prime-air-amazon-presente-nouveau-drone-livraison-capable-voler-a-90-kmh.htm. Accessed : 2019-04-10.

[84] C. Luo, X. Li, Y. Li et Q. Dai, “Biomimetic design for unmanned aerial vehicle safe landing in hazardous terrain,” *IEEE/ASME Transactions on Mechatronics*, vol. 21, n°. 1, p. 531–541, 2016.

[85] E. Lindsey et N. Cooper, “Monocular vision based obstacle avoidance : A literature review,” https://elindseyspace.files.wordpress.com/2017/06/midterm-paper-5.pdf, 2017-06-21.

[86] A. Al-Kaff, Q. Meng, D. Martín, A. de la Escalera et J. M. Armingol, “Monocular vision-based obstacle detection/avoidance for unmanned aerial vehicles,” dans *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, p. 92–97.

[87] D. Valencia et D. Kim, “Quadrotor obstacle detection and avoidance system using a monocular camera,” dans *2018 3rd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. IEEE, 2018, p. 78–81.

[88] L. Kovács, “Visual monocular obstacle avoidance for small unmanned vehicles,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, p. 59–66.

[89] M. Elawady, I. Sadek et H. Kidane, “Detecting and avoiding frontal obstacles from monocular camera for micro unmanned aerial vehicles,” *arXiv preprint arXiv :1603.09422*, 2016.

[90] A. J. Barry et R. Tedrake, “Pushbroom stereo for high-speed navigation in cluttered environments,” dans *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, p. 3046–3052.

[91] K. McGuire, G. De Croon, C. De Wagter, K. Tuyls et H. Kappen, “Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone,” *IEEE Robotics and Automation Letters*, vol. 2, n°. 2, p. 1070–1076, 2017.

[92] M. Zaffagni. (2017-07-18) Racerx, le drone le plus rapide du monde. https://www.futura-sciences.com/tech/actualites/drone-racerx-drone-plus-rapide-monde-68000/. Accessed : 2019-05-13.

[93] A. Kathuria, "A pytorch implementation of the yolo v3 object detection algorithm," https://github.com/ayooshkathuria/pytorch-yolo-v3, 2018.

[94] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár et C. L. Zitnick. (2015) Coco. http ://cocodataset.org/#download. Accessed : 2019-05-13.

[95] E. Jane. (2017-09-18) Java is one of the most energy-efficient languages, python among least energy efficient. https ://jaxenter.com/energy-efficient-programming-languages-137264.html. Accessed : 2019-04-10.

[96] K. Jonathan. (2018-10) A comparison of programming languages. http ://jonathankinlay.com/2018/10/comparison-programming-languages/. Accessed : 2019-04-10.

[97] J. S. Nagi, "Yolo-v3 and yolo-v2 for windows and linux," https://github.com/jaskarannagi19/yolov3, 2018.

[98] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[99] M. Jay, "Mask-rcnn-series," https://github.com/markjay4k/Mask-RCNN-series, 2018.

[100] K. He, G. Gkioxari, P. Dollar et R. Girshick, "Mask r-cnn," dans *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[101] Pxhere. https ://pxhere.com. Accessed : 2019-05-13.

[102] M. Skocik, J. Collins, C. Callahan-Flintoft, H. Bowman et B. Wyble, "I tried a bunch of things : the dangers of unexpected overfitting in classification," *BioRxiv*, p. 078816, 2016.

[103] Y. Wu, "Image Formation and Camera Calibration, Lecture Notes : Digital Image Analysis," url : http://hades.mech.northwestern.edu/images/1/17/Image_Formation_and_Camera_Calibration.pdf. Accessed : 2019-10-13.

[104] mathworks. (2019) cameraparameters. https://www.mathworks.com/help/vision/ref/cameraparameters.html. Accessed : 2019-05-13.

[105] L. Masson, "Suivi temps-réel d'objets 3d pour la réalité augmentée," Thèse de doctorat, 2005.

[106] B. Jason. (2019-01-23) How to configure the learning rate when training deep learning neural networks. https ://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/. Accessed : 2019-04-10.

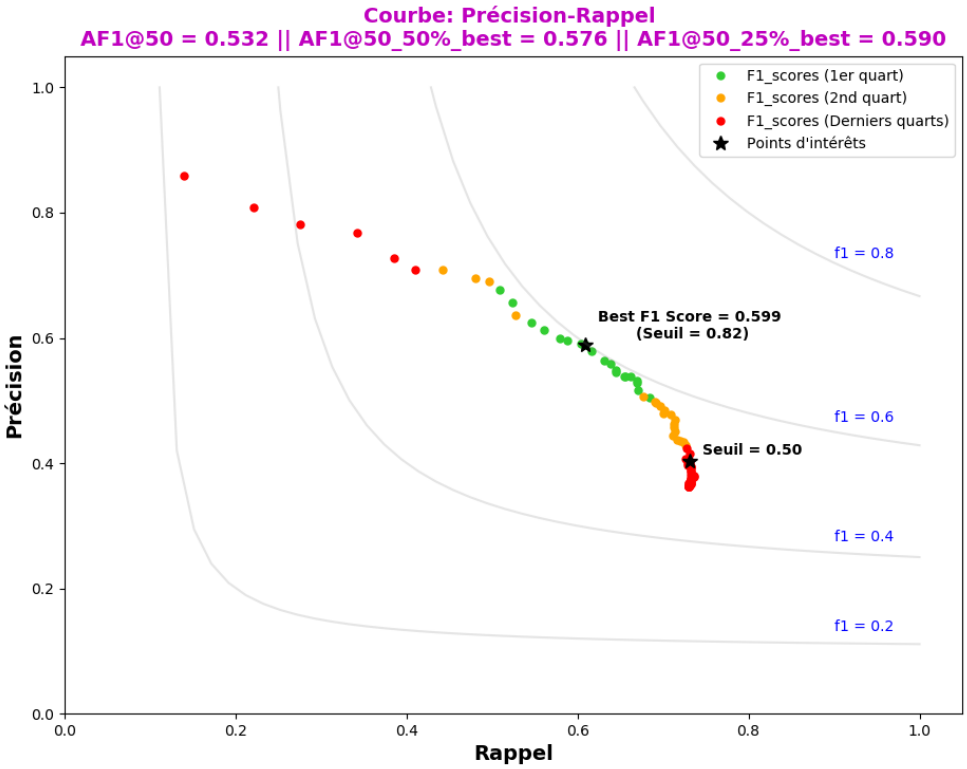
ANNEXE A ENSEMBLE DES COURBES PRÉCISION-RAPPEL QUI TRADUISENT L'ÉVALUATION DES DIFFÉRENTS MODÈLES DE DÉTECTION OBTENUS

Courbes Précision-Rappel selon la configuration d'entrainement sélectionnée

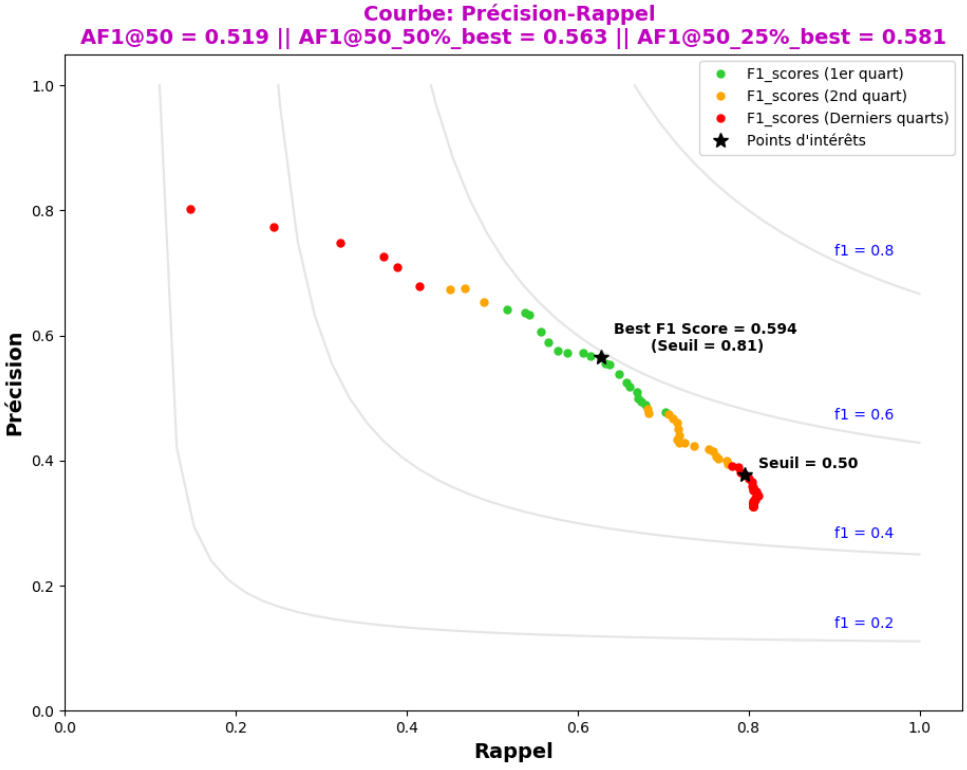
Dans cette section est regroupé l'ensemble des courbes "Précision-Rappel" pour chacun des paramétrages d'entrainement sélectionné. Ces courbes nous informent sur la précision de détection du modèle choisi. La structure d'entrainement du réseau de neurones de Mask-RCNN se décompose comme suit :

- Allant de la configurations de base à la configuration n°20 (inclue) :
    1 Entrainement des premières couches ("heads layers") pour "X" epochs avec un "Learning_rate" = 0.001
    2 Entrainement de toutes les couches ("all layers") pour 30 epochs avec un "Learning_rate" = 0.001
    3 Entrainement de toutes les couches ("all layers") pour 15 epochs avec un "Learning_rate" = 0.0001
- Allant de la configurations n°21 à la configuration n°35 (inclue) :
    1 Entrainement des premières couches ("heads layers") pour "X" epochs avec un "Learning_rate" = 0.001
    2 Entrainement de toutes les couches ("all layers") pour 10 epochs avec un "Learning_rate" = 0.001
    3 Entrainement de toutes les couches ("all layers") pour 35 epochs avec un "Learning_rate" = 0.0001

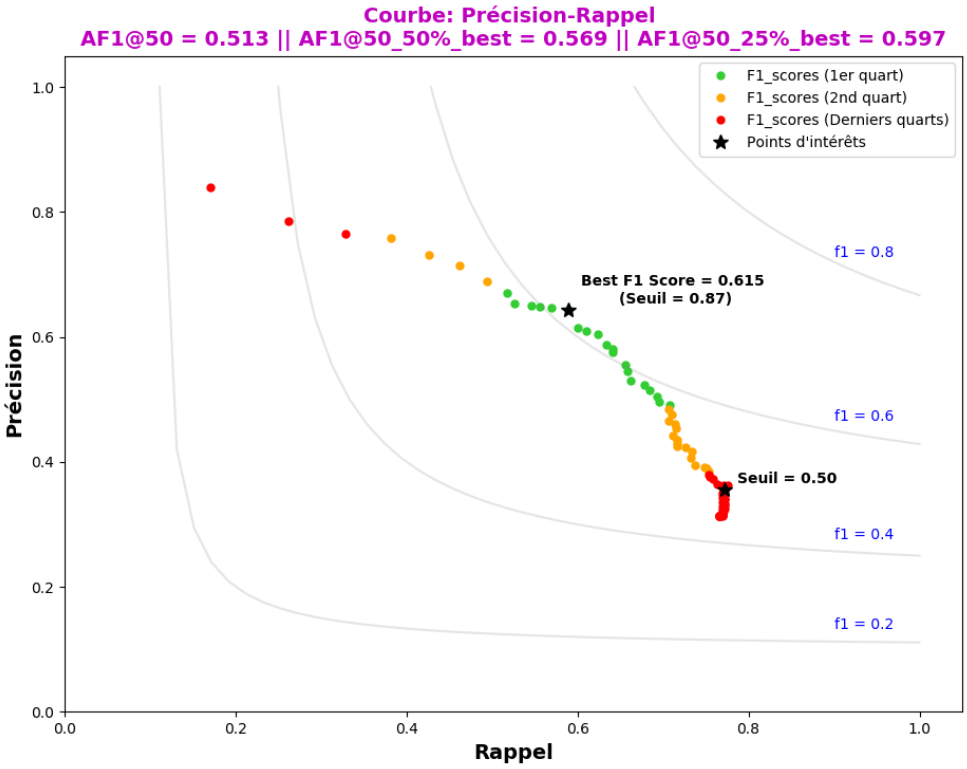
Le paramètre "X" correspond à la valeur du paramètre "EPOCHS".
Le paramètre "Learning_rate" contrôle la rapidité d'apprentissage du réseau. Ce paramètre est directement lié au contrôle de la variation de l'évolution des poids des neurones. Ainsi, plus ce dernier est petit et plus le réseau évoluera lentement dans son apprentissage. Les détails et les explications précises de ce paramètre sont présentés en [106].



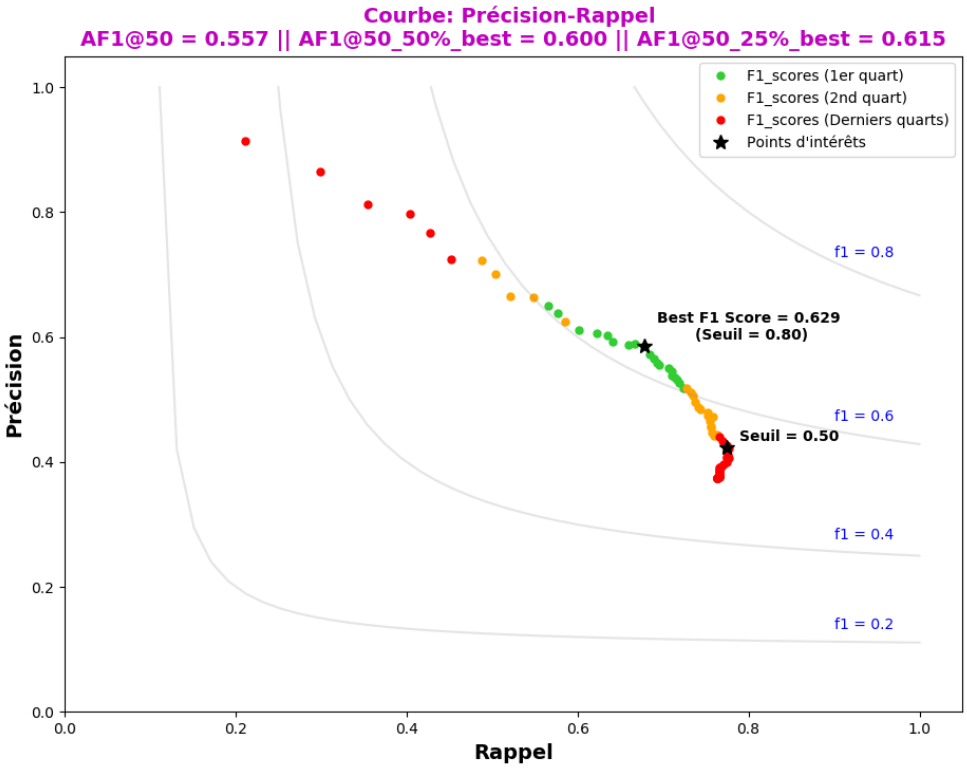
Configuration n°0



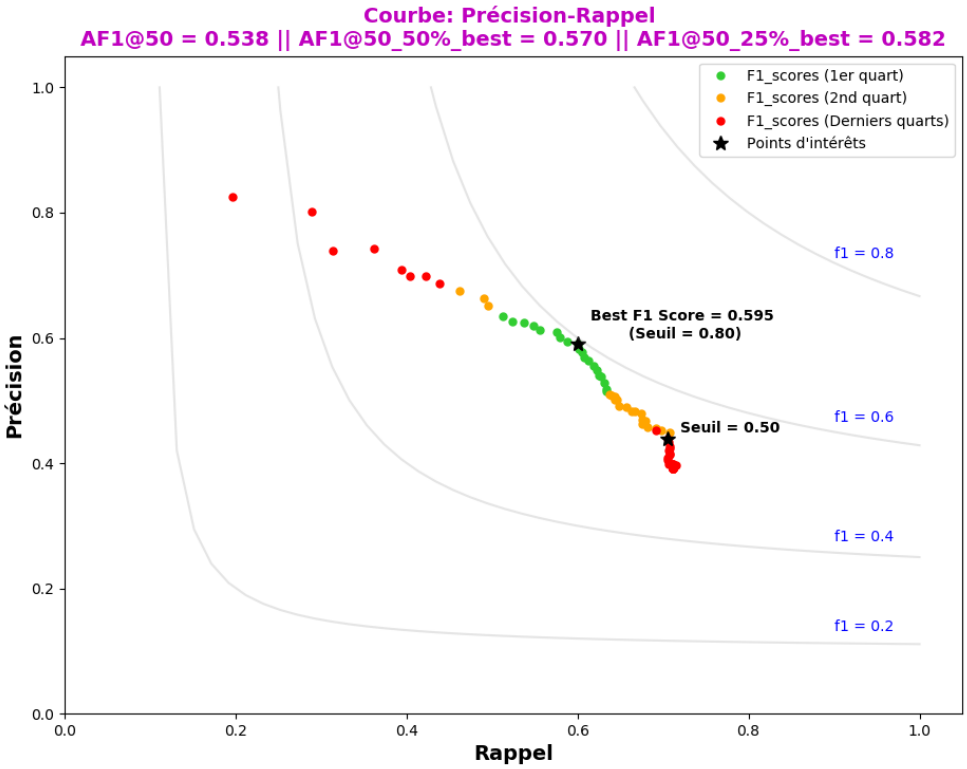
Configuration n°1



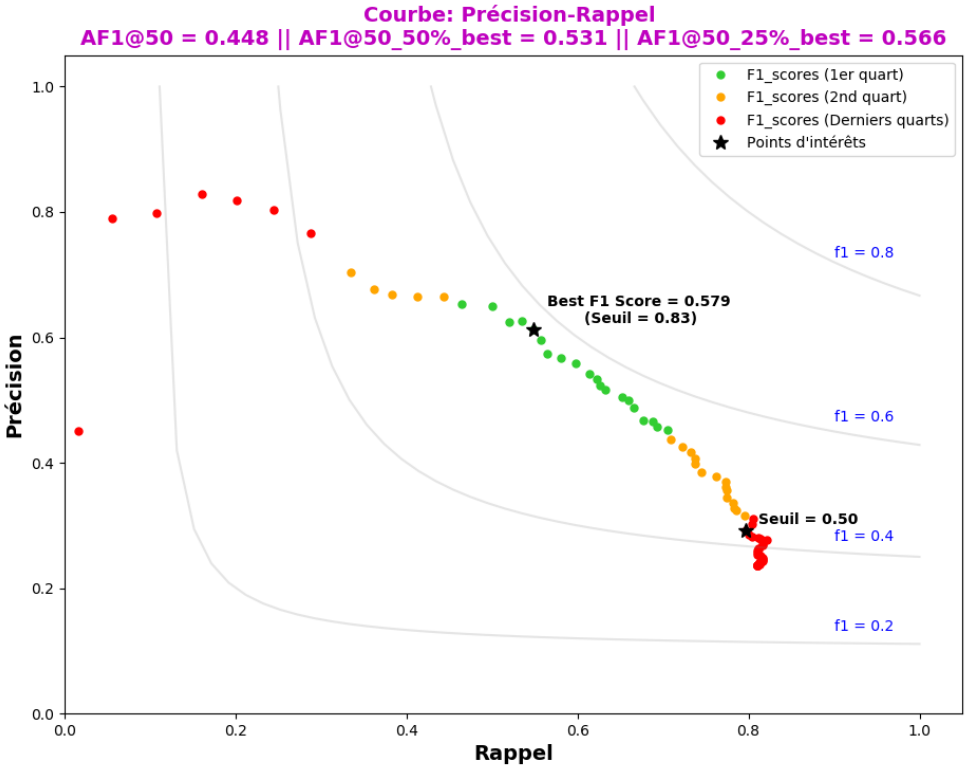
Configuration n°2



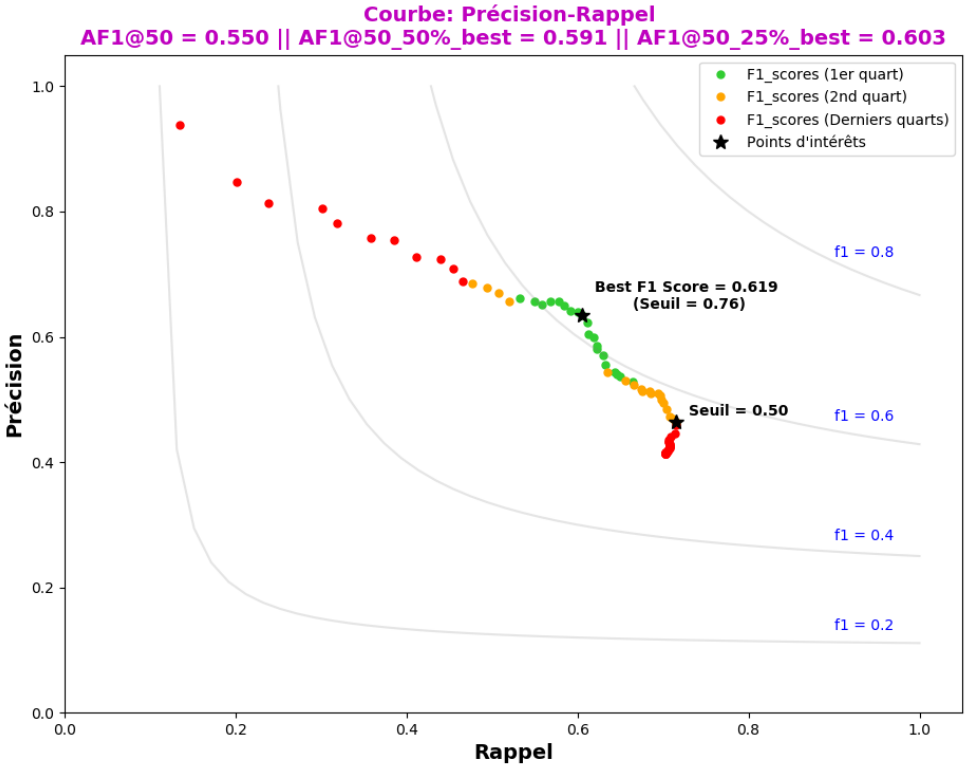
Configuration n°3



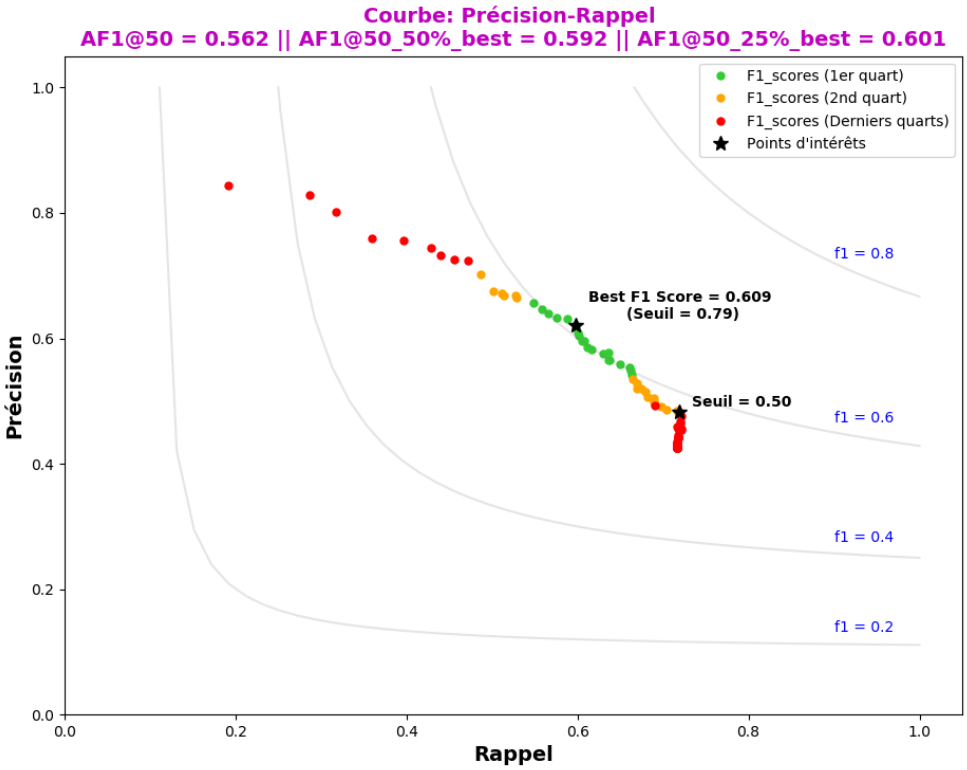
Configuration n°4



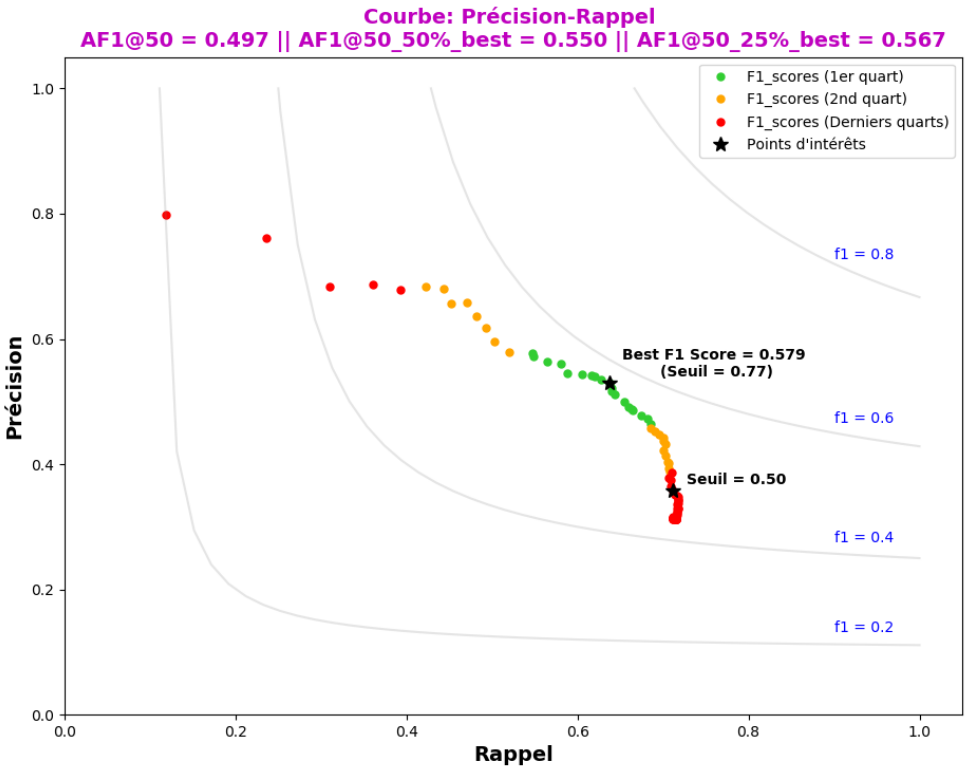
Configuration n°5



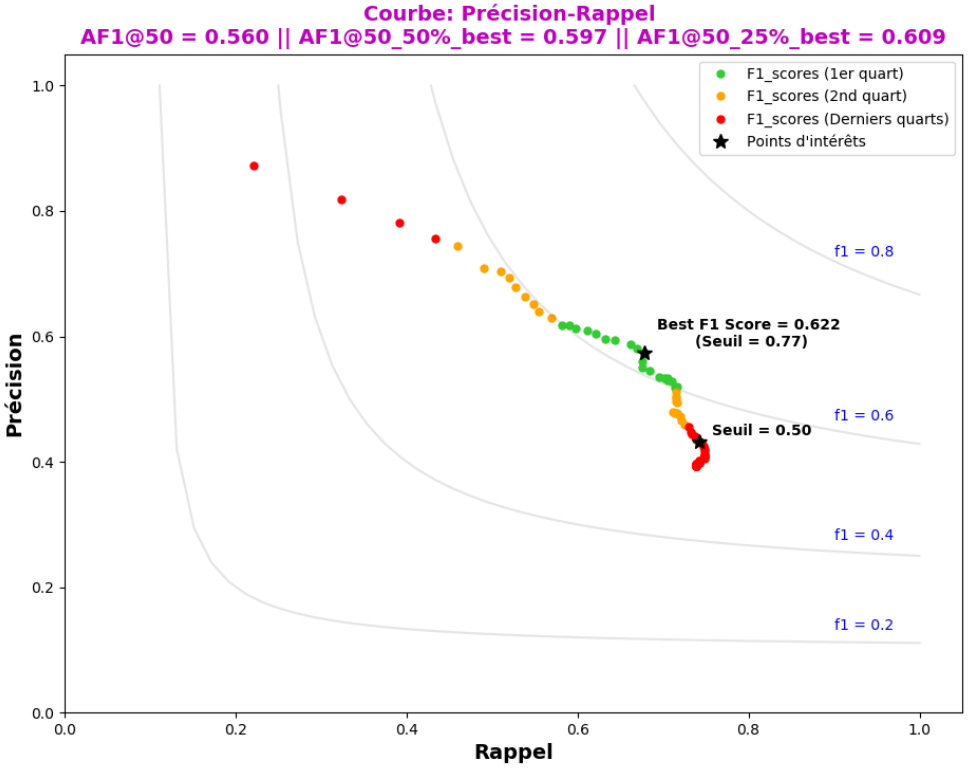
Configuration n°6



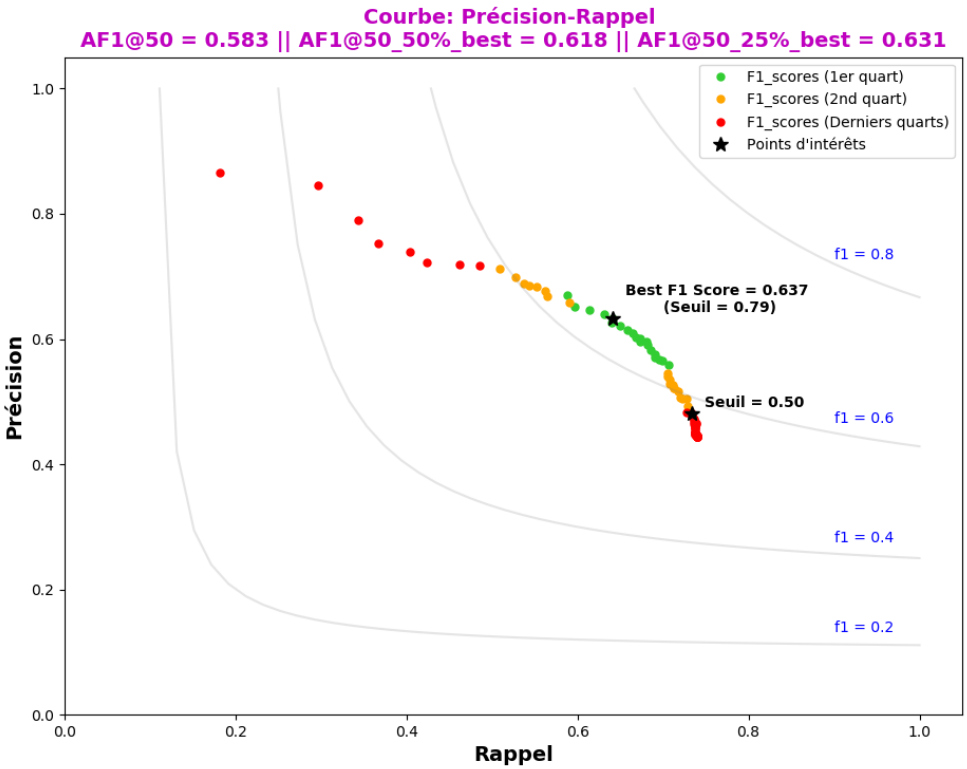
Configuration n°7



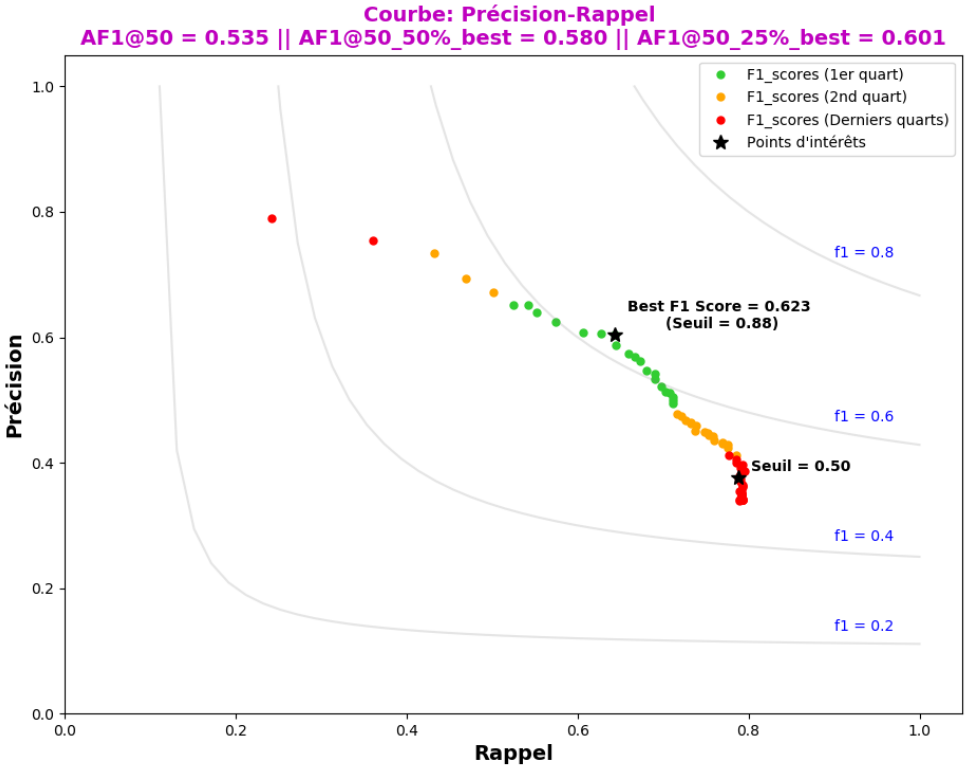
Configuration n°8



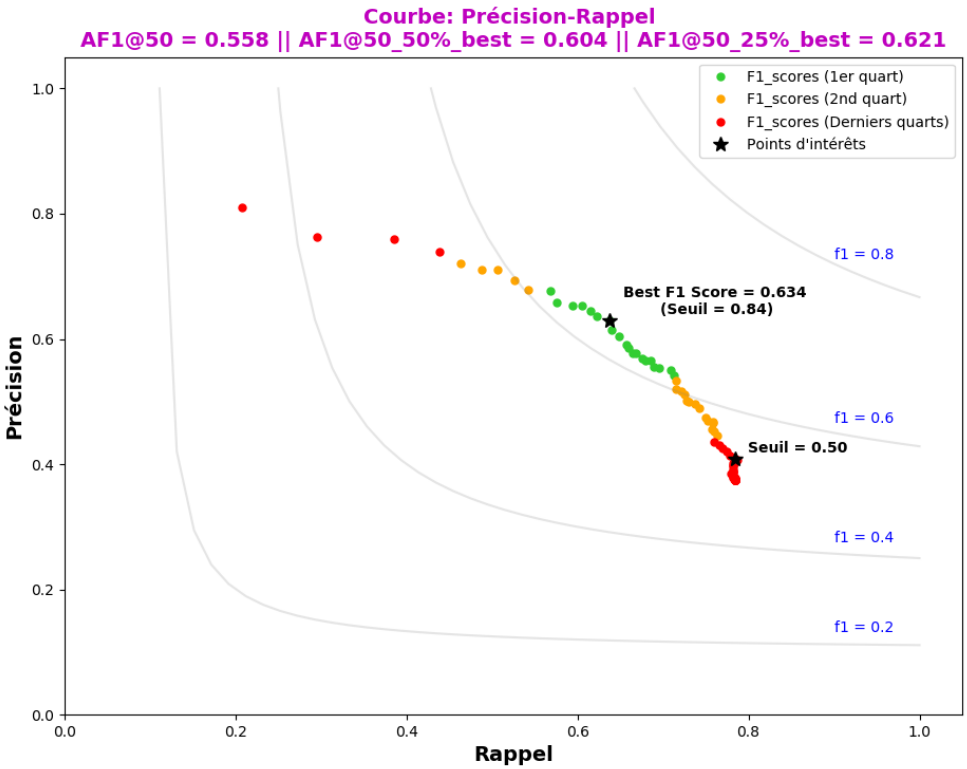
Configuration n°9



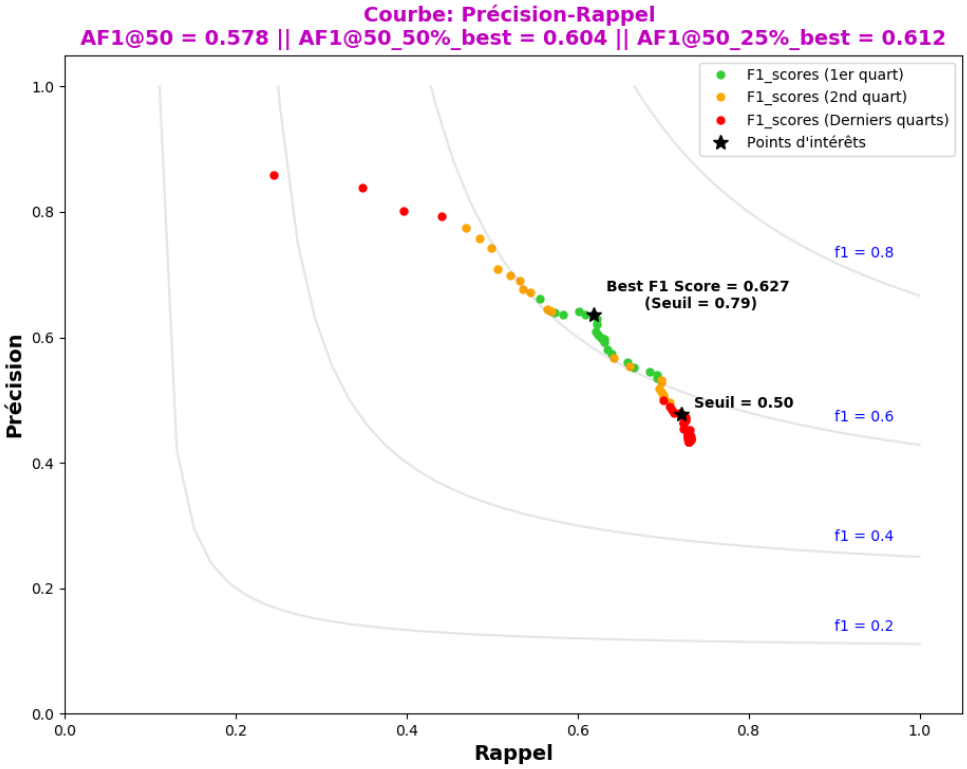
Configuration n°10



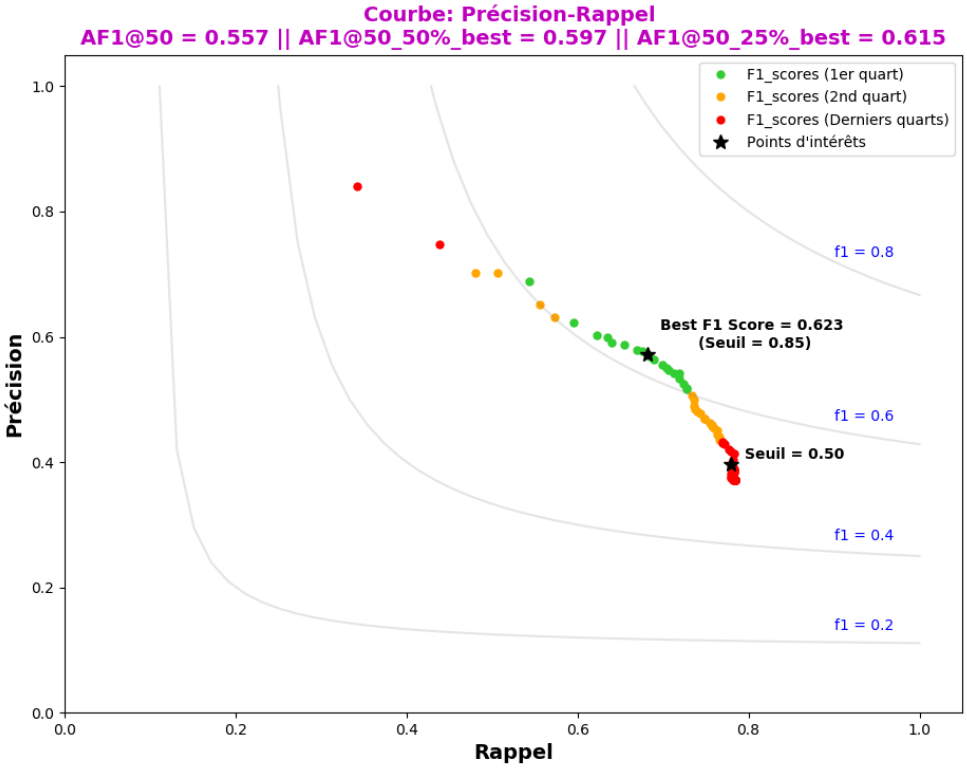
Configuration n°11



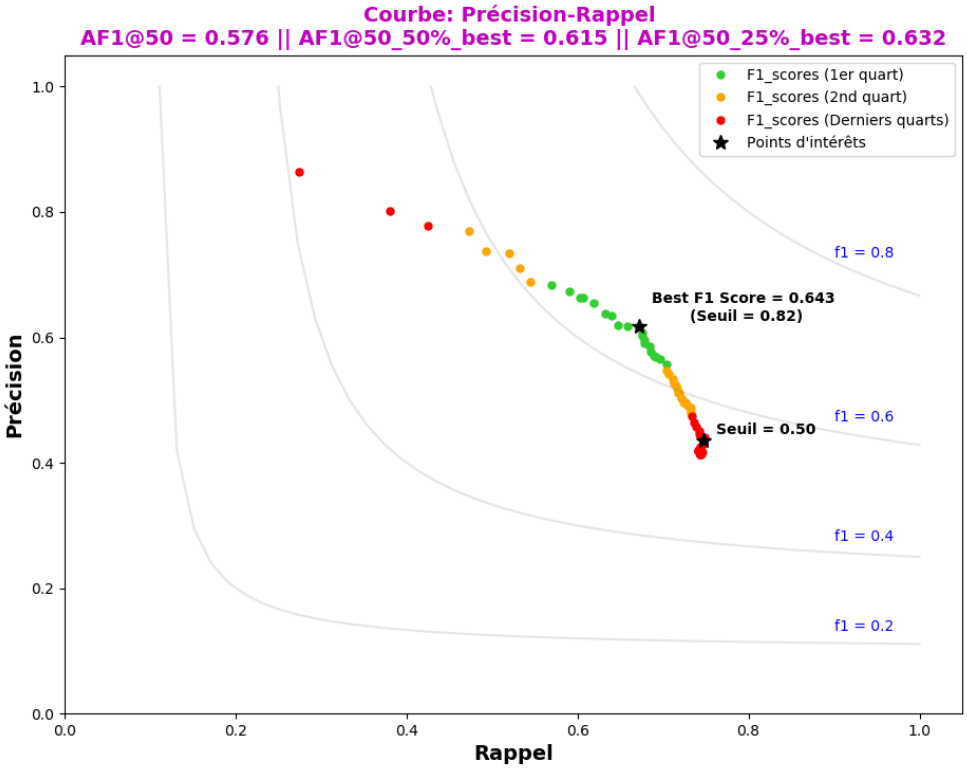
Configuration n°12



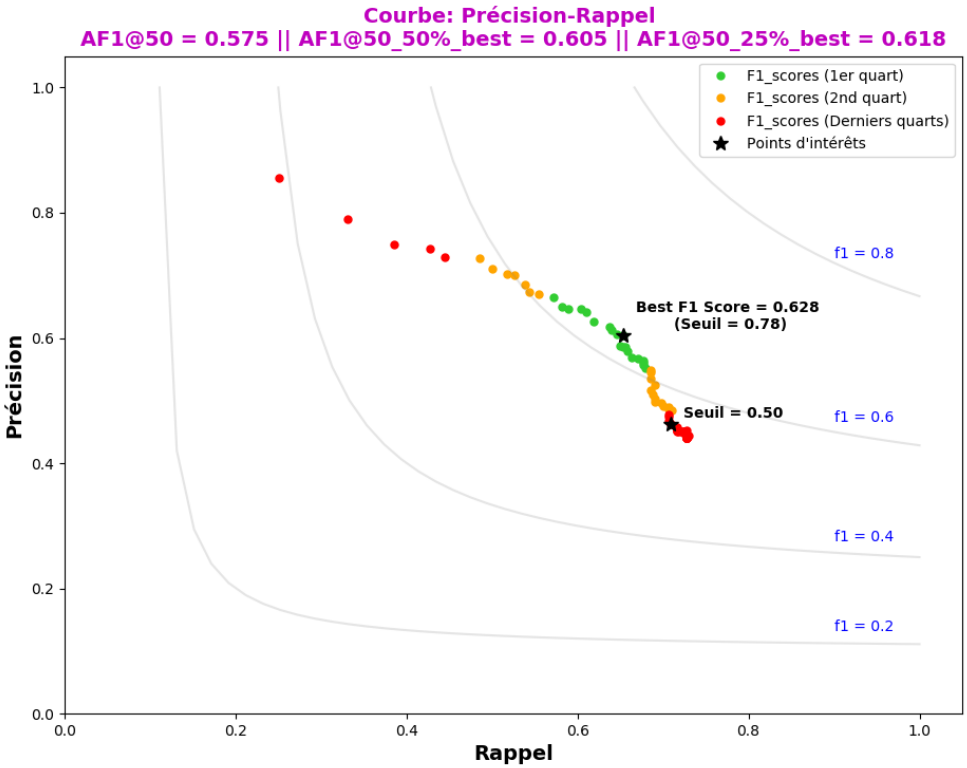
Configuration n°13



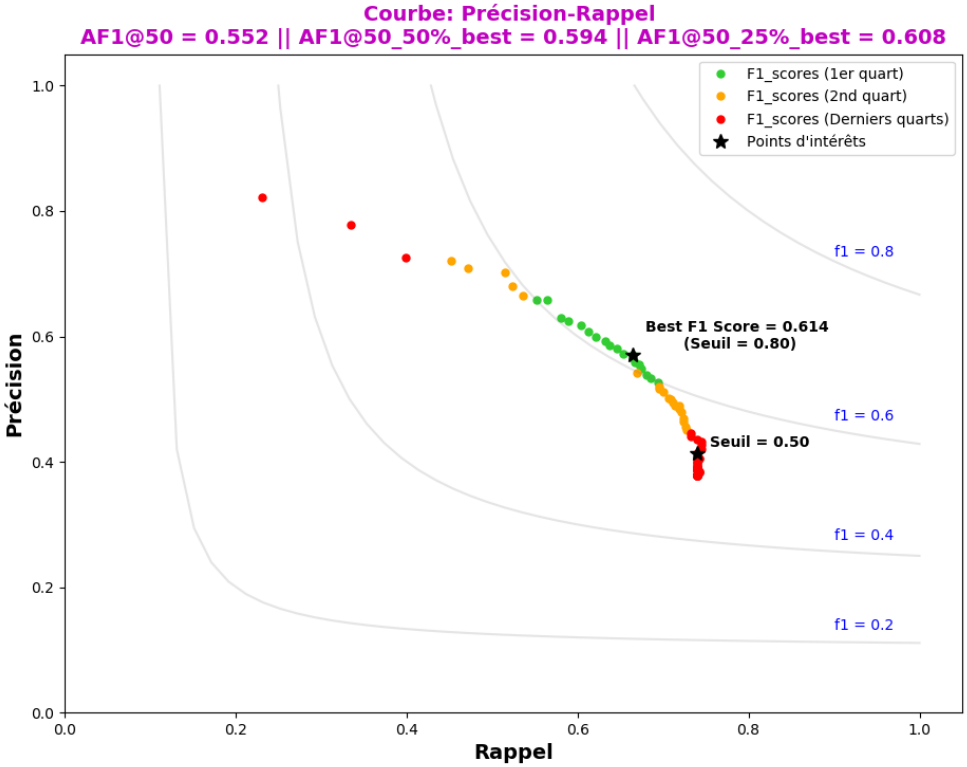
Configuration n°14



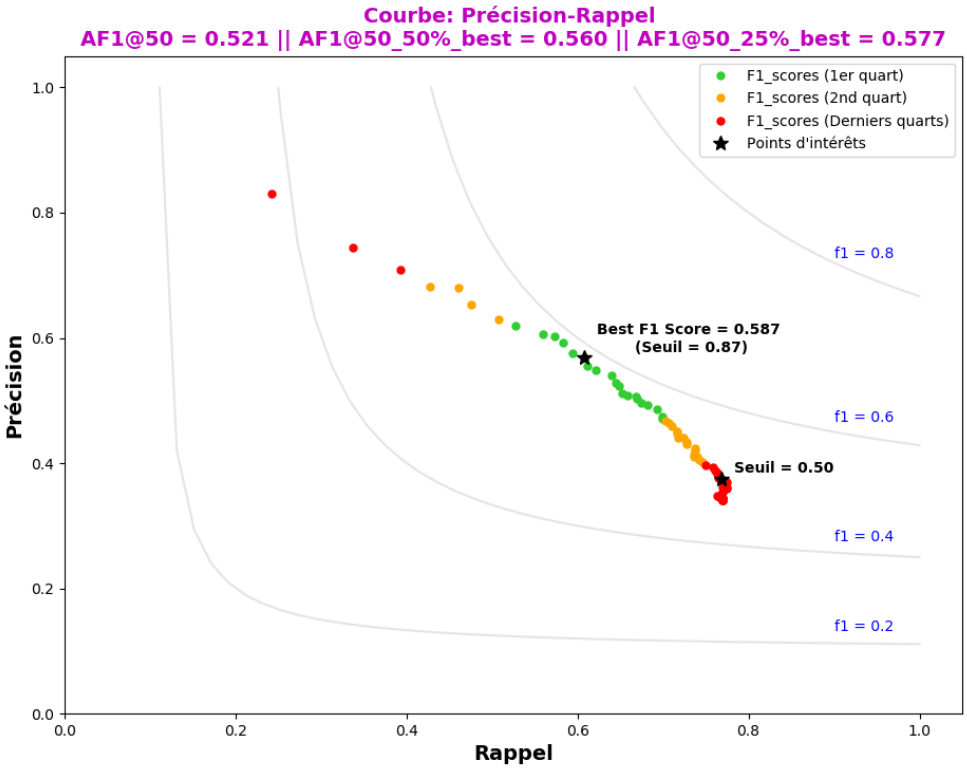
Configuration n°15



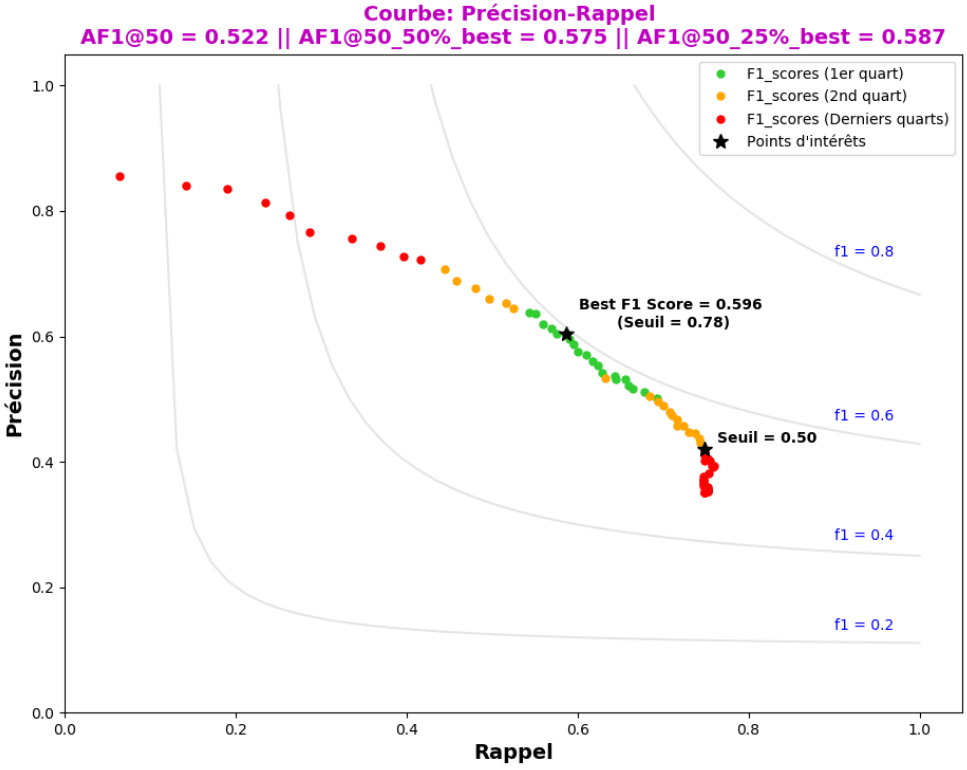
Configuration n°16



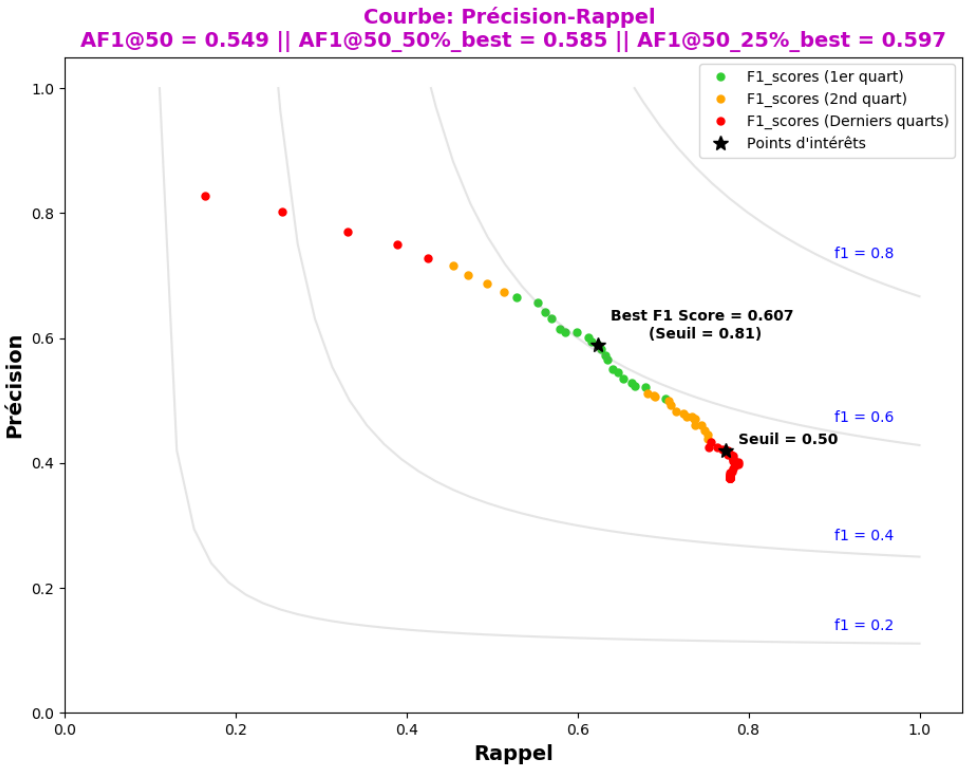
Configuration n°17



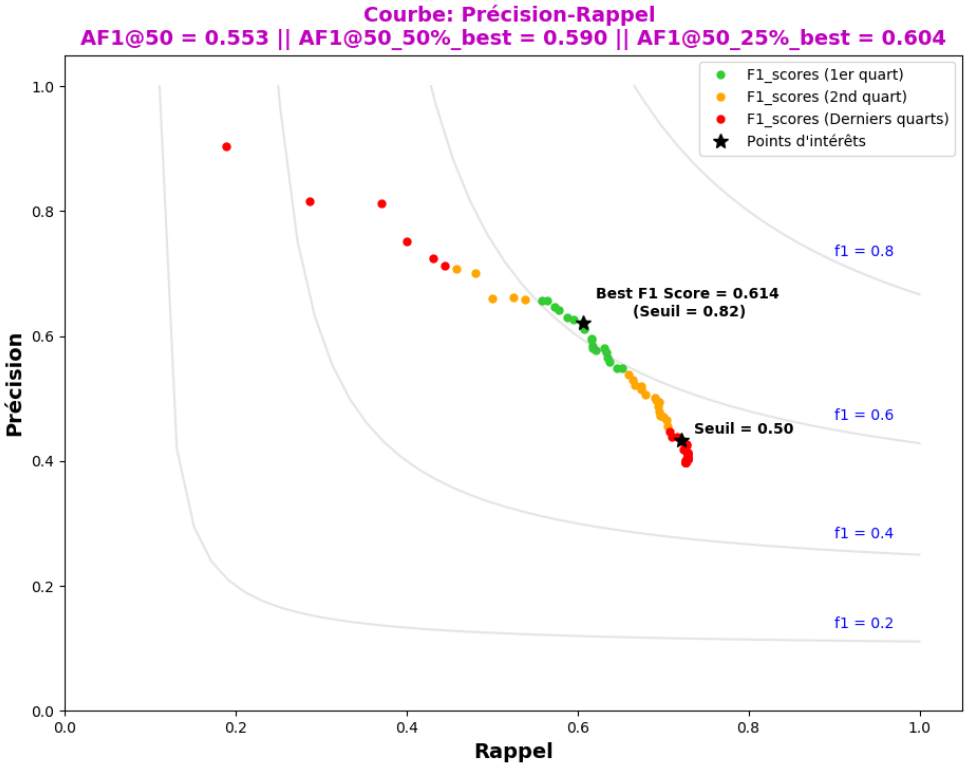
Configuration n°18



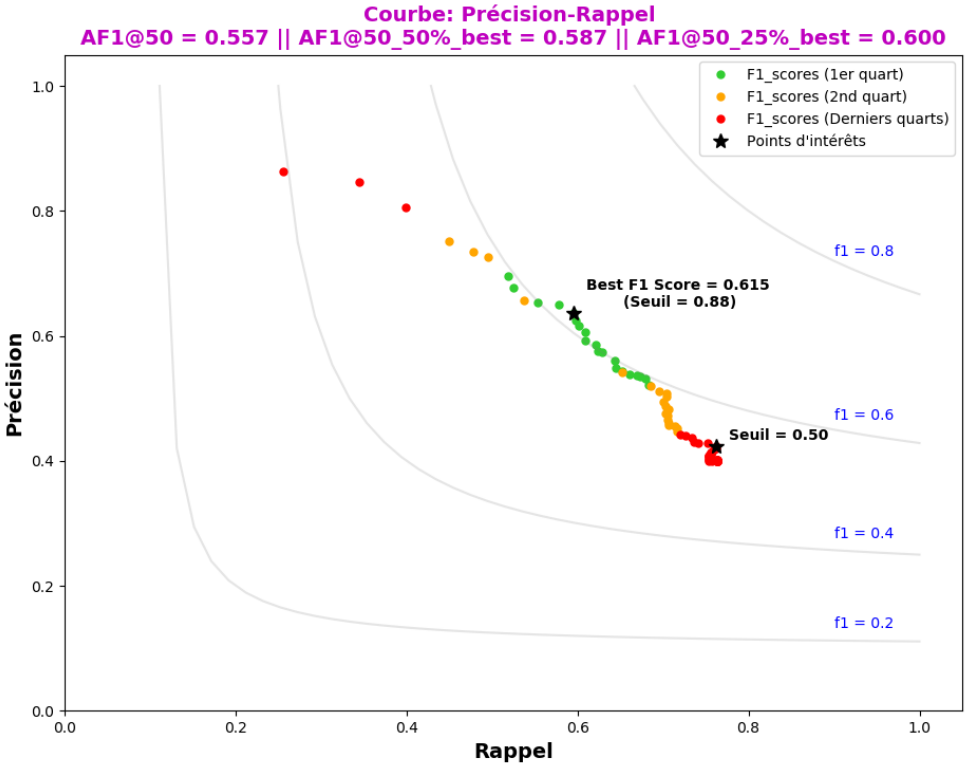
Configuration n°19



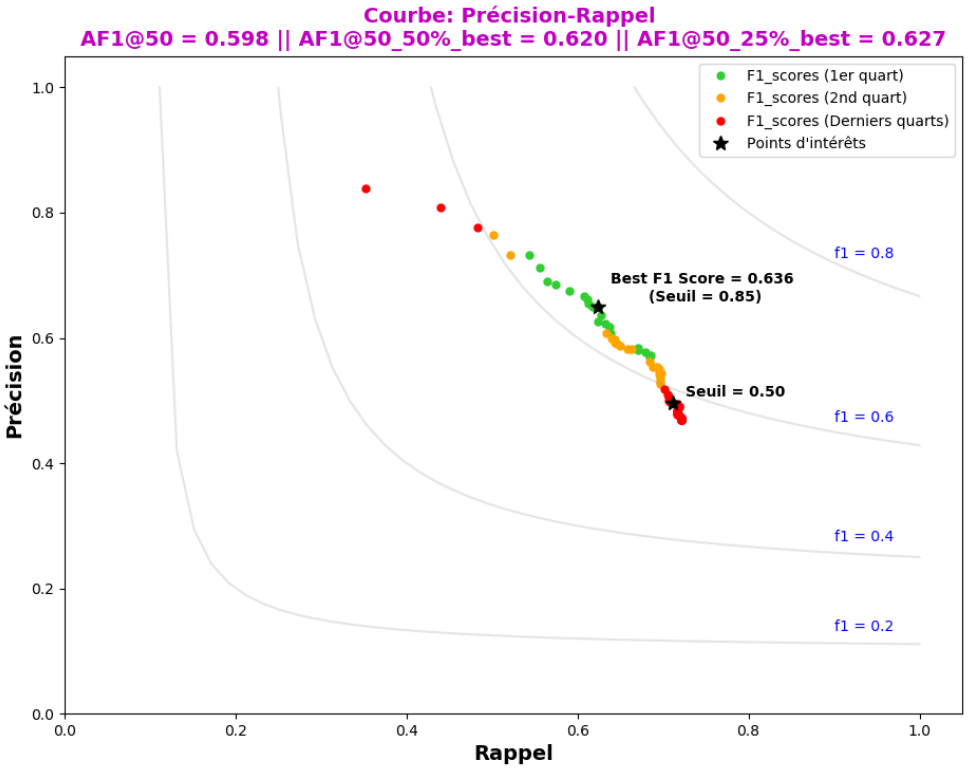
Configuration n°20



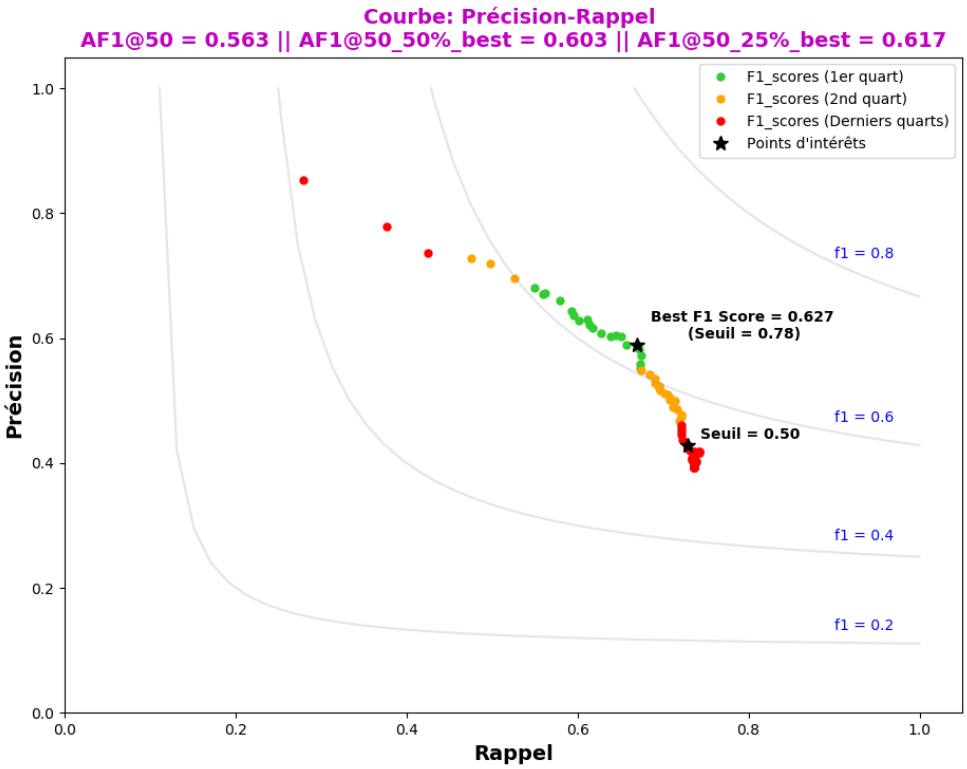
Configuration n°21



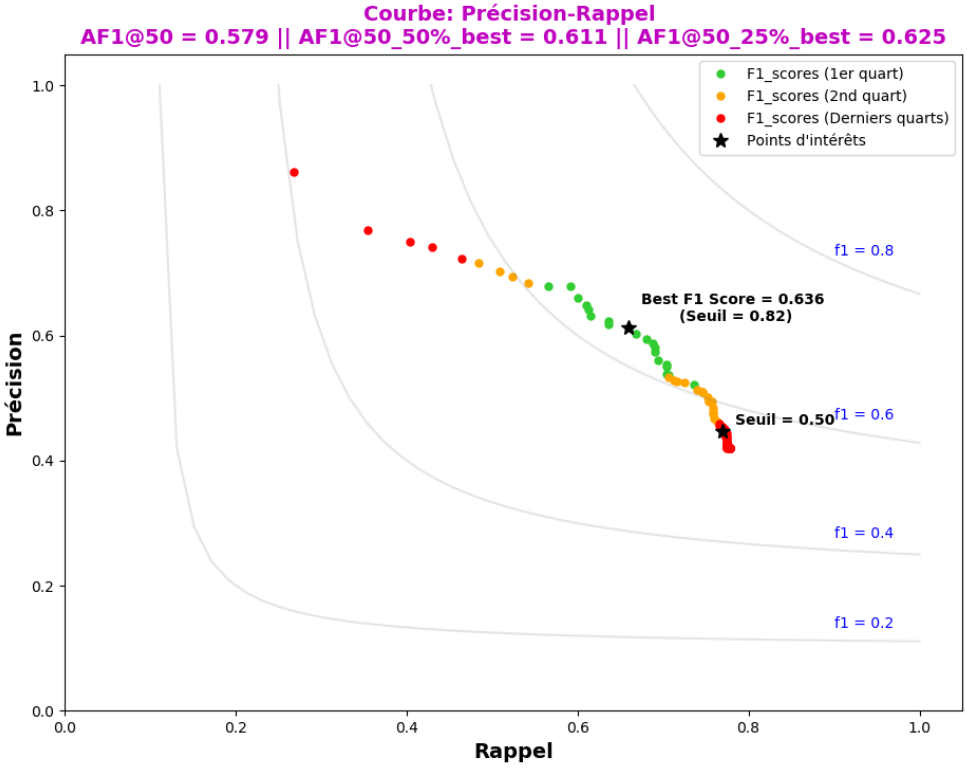
Configuration n°22



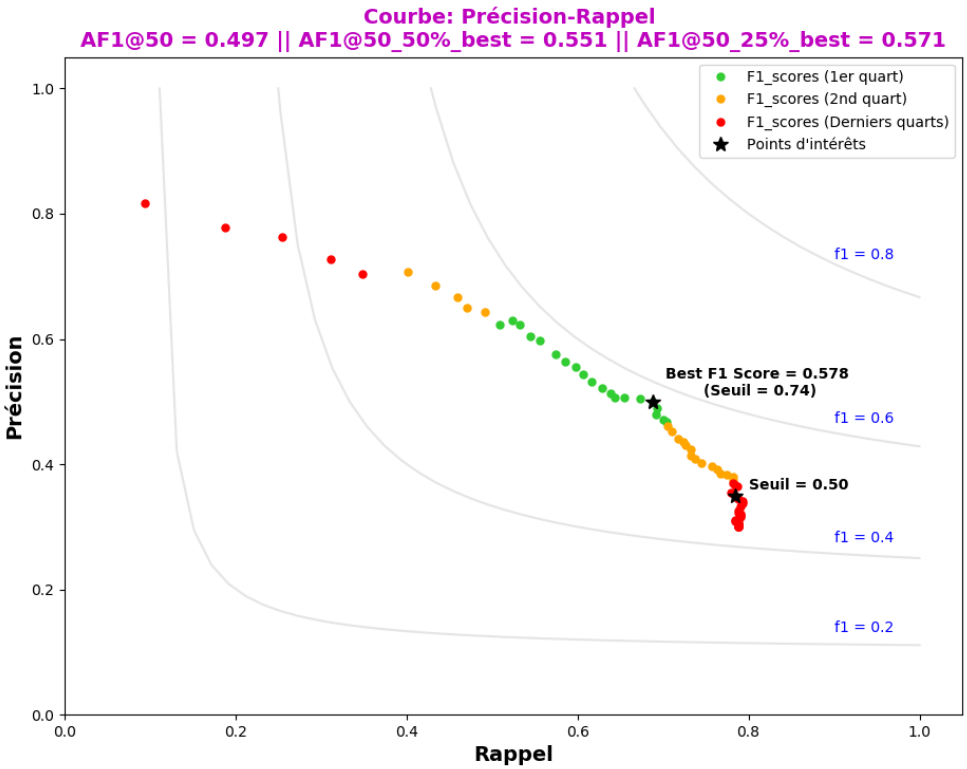
Configuration n°23



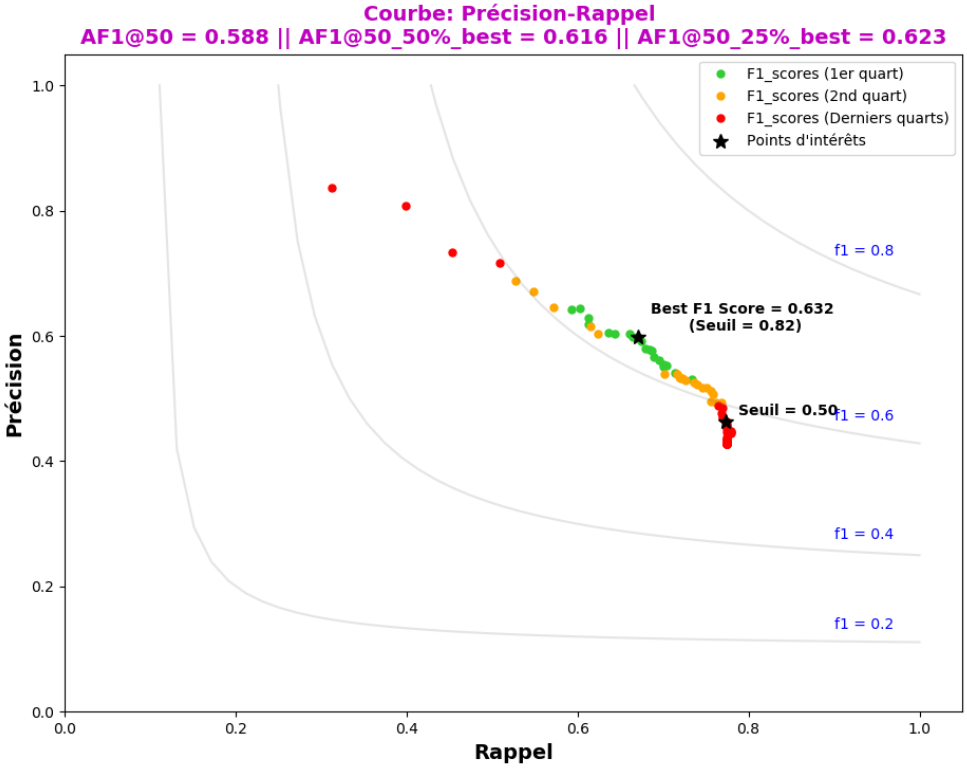
Configuration n°24



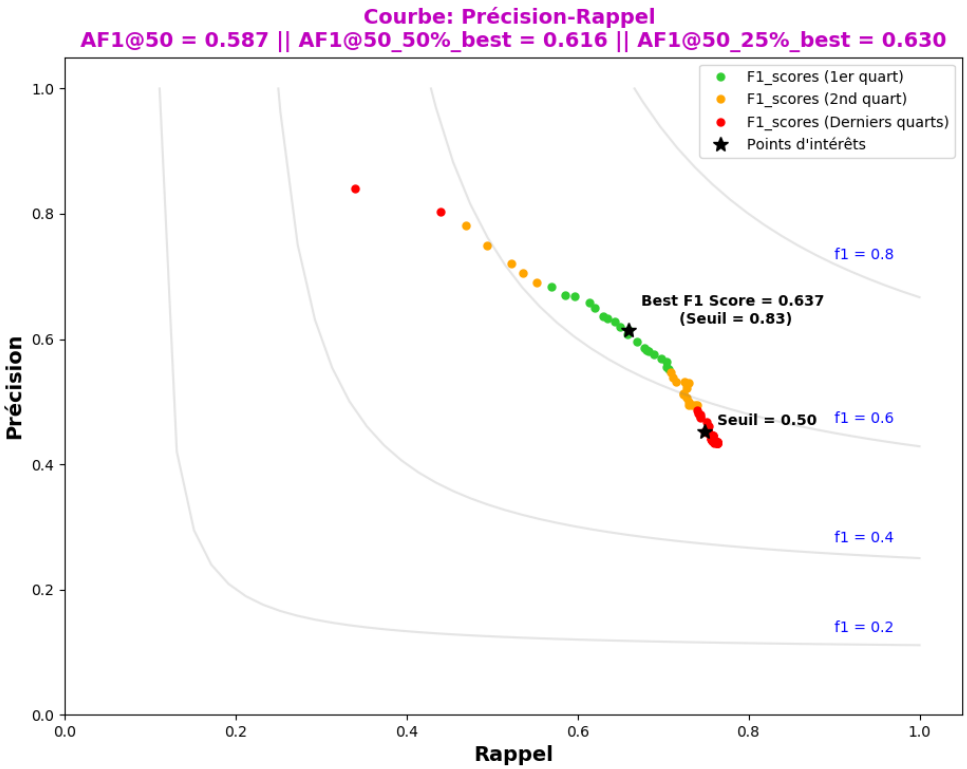
Configuration n°25



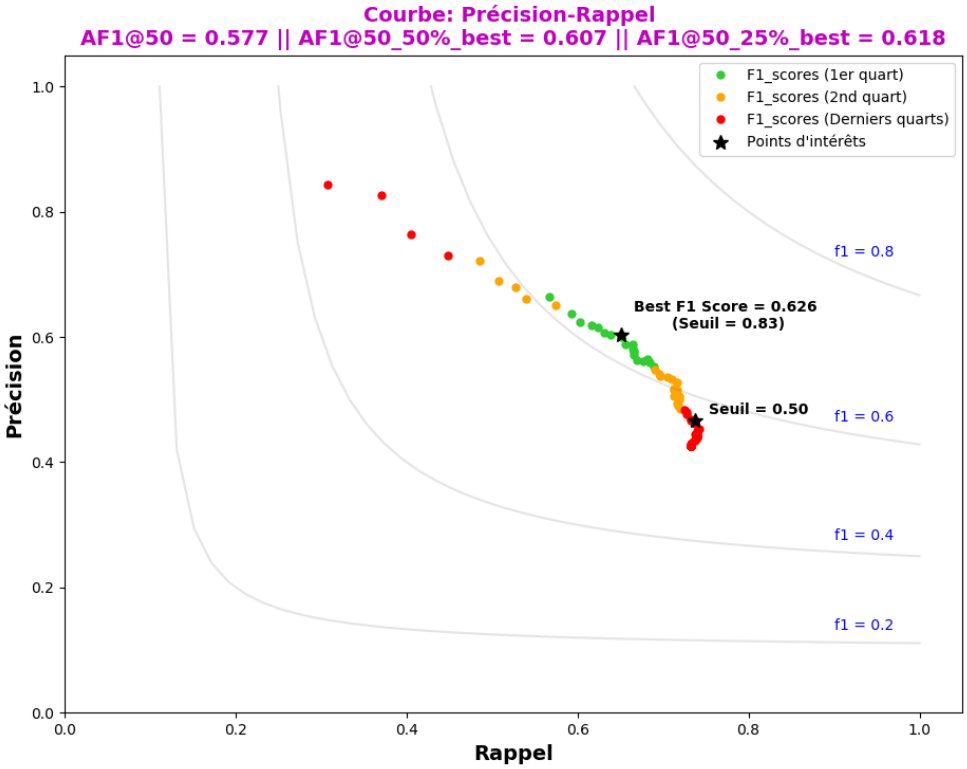
Configuration n°26



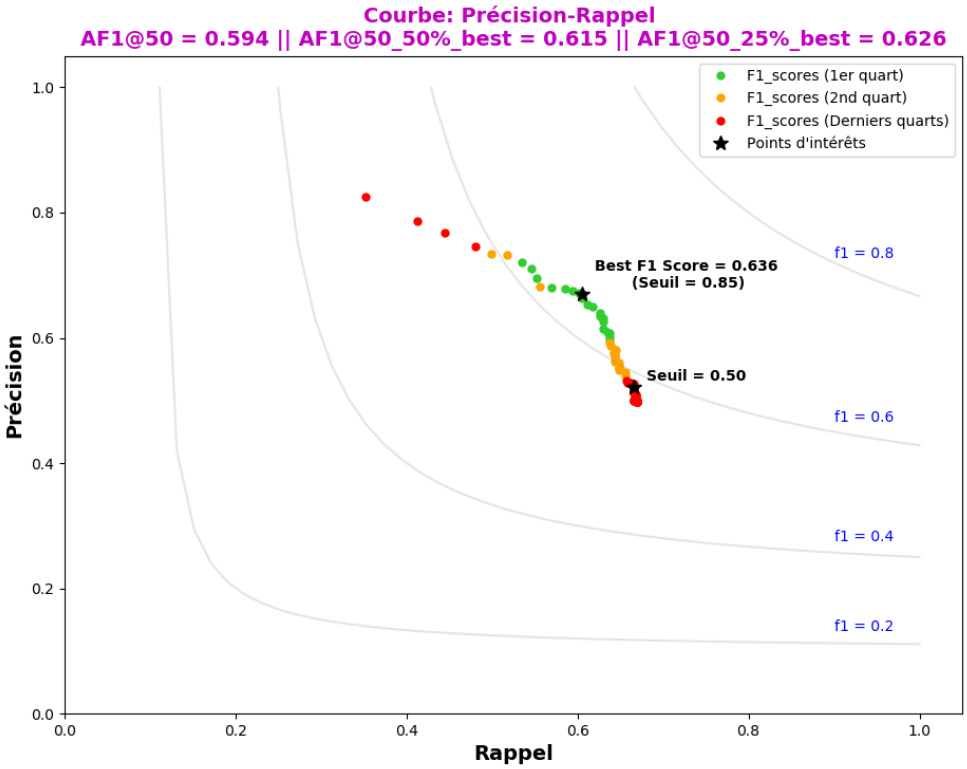
Configuration n°27



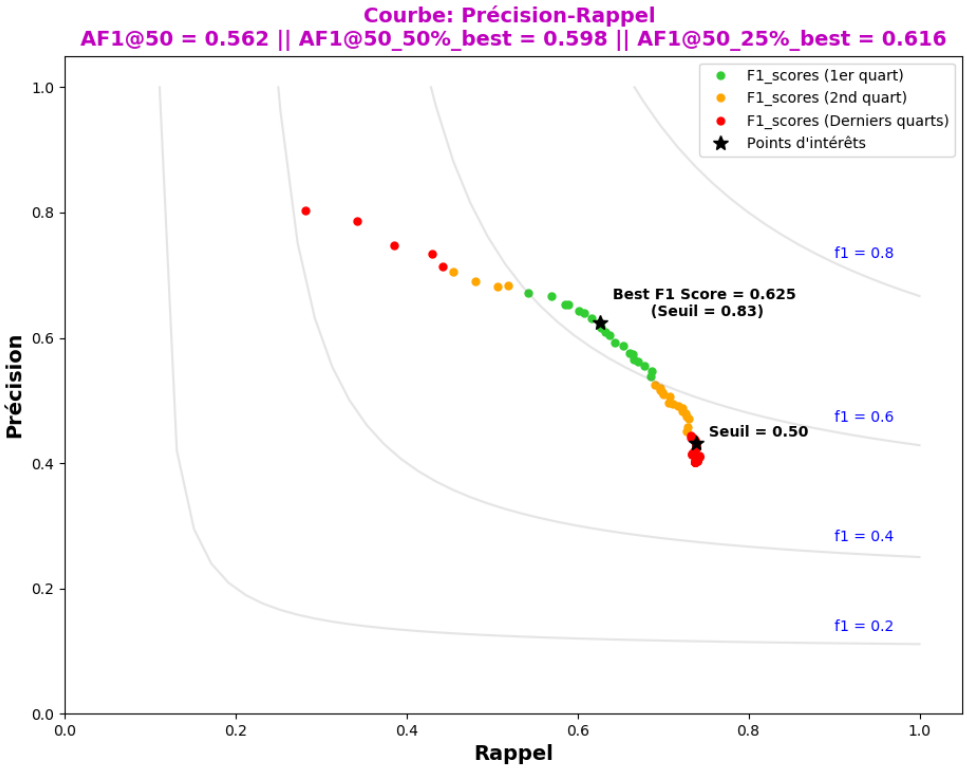
Configuration n°28



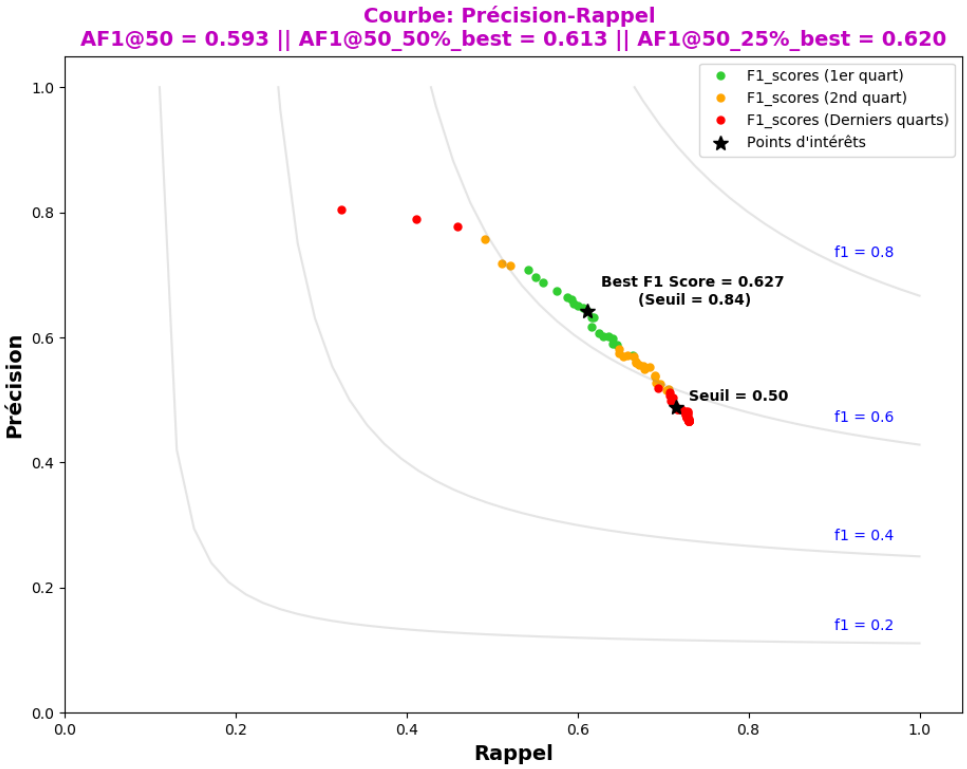
Configuration n°29



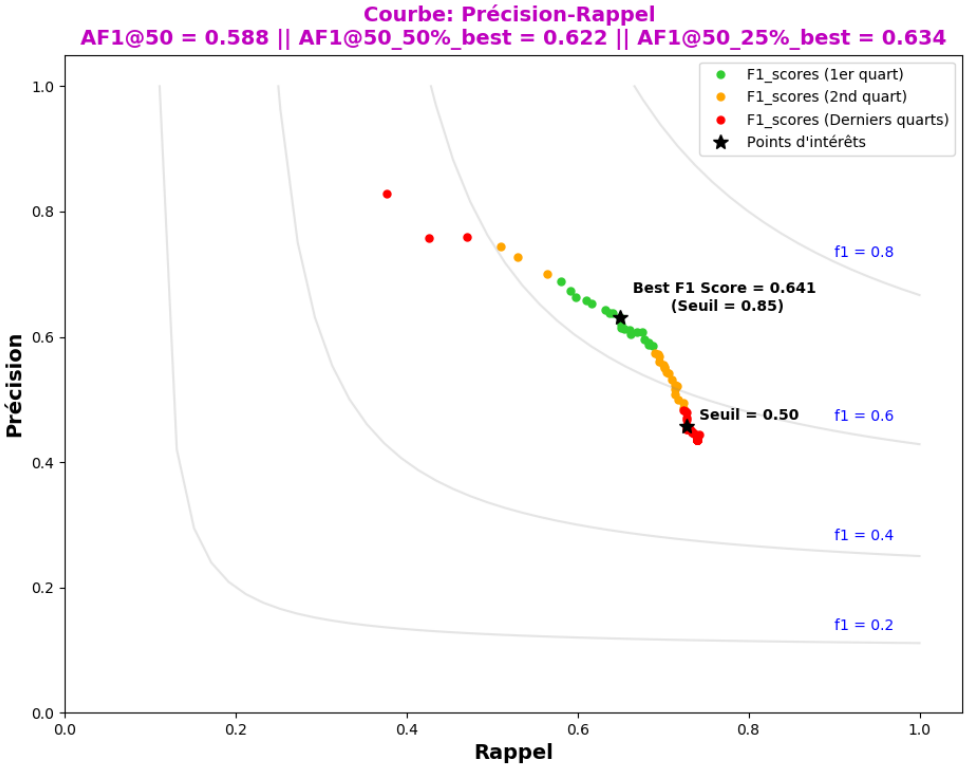
Configuration n°30



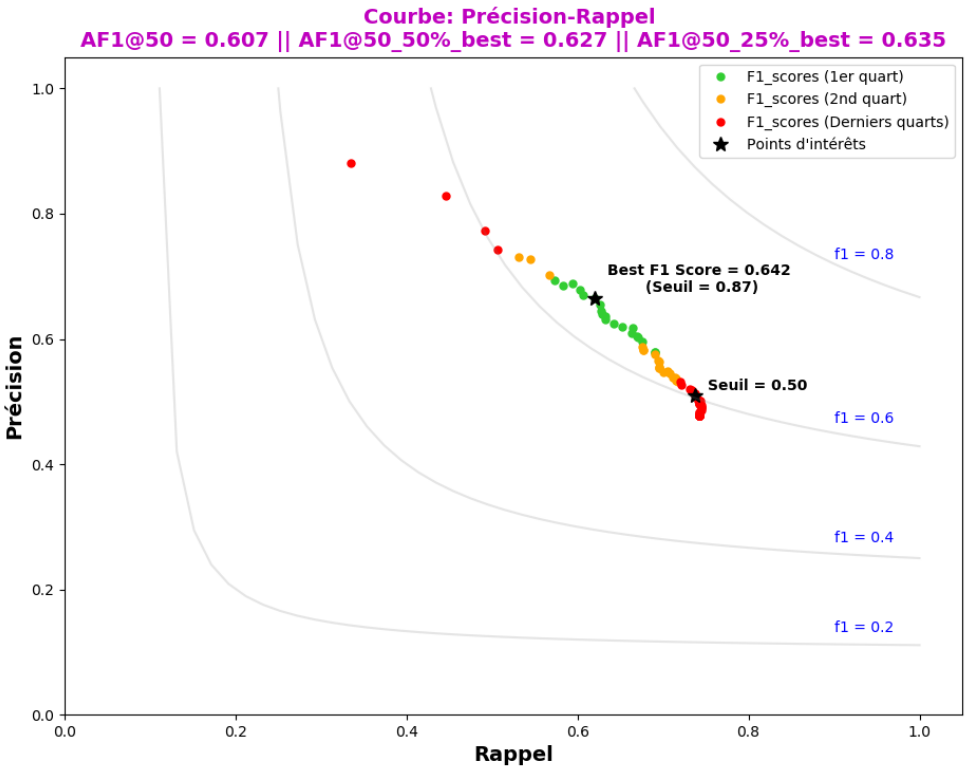
Configuration n°31



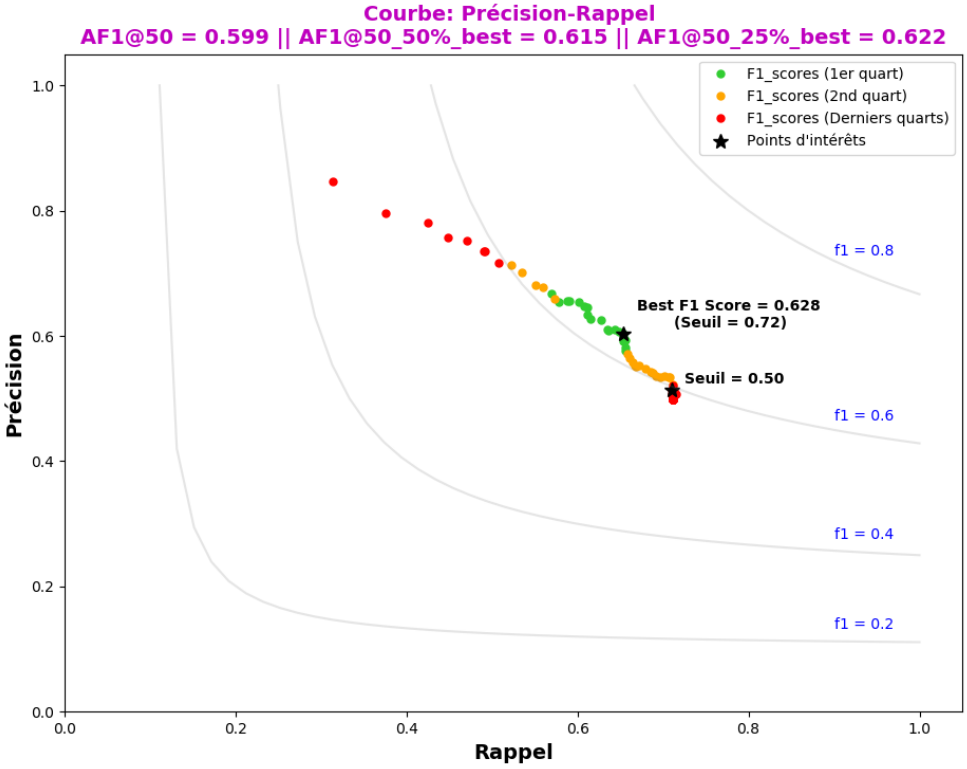
Configuration n°32



Configuration n°33



Configuration n°34



Configuration n°35

Afin d'alléger la notation dans les tableaux suivants les variables correspondant aux différentes colonnes ont été renommées.

- SPE = STEPS_PER_EPOCHS
- VS = VALIDATION_STEPS
- RTAPI = RPN_TRAIN_ANCHORS_PER_IMAGE
- TRPI = TRAIN_ROIS_PER_IMAGE
- RNT = RPN_NMS_THRESHOLD

Tableau A.1 Valeurs des différents paramètres selon la configuration choisie

N°	**EPOCHS**	**SPE**	**VS**	**RTAPI**	**TRPI**	**RNT**
0	15	800	50	256	200	0.70
1	10	800	50	256	200	0.70
2	20	800	50	256	200	0.70
3	25	800	50	256	200	0.70
4	30	800	50	256	200	0.70
5	15	800	50	256	200	0.50
6	15	800	50	256	200	0.80
7	15	800	50	256	200	0.90
8	15	800	50	128	200	0.70
9	15	800	50	384	200	0.70
10	15	800	50	512	200	0.70
11	15	800	50	640	200	0.70
12	15	800	50	768	200	0.70
13	15	800	50	1024	200	0.70
14	15	800	50	256	25	0.70
15	15	800	50	256	50	0.70
16	15	800	50	256	75	0.70
17	15	800	50	256	100	0.70
18	15	800	50	256	150	0.70
19	15	800	50	256	400	0.70
20	15	800	100	256	200	0.70

N°	EPOCHS	SPE	VS	RTAPI	TRPI	RNT
21	15	800	50	256	200	0.70
22	15	800	100	512	50	0.90
23	15	800	50	512	50	0.70
24	15	800	50	512	75	0.70
25	15	800	50	512	100	0.70
26	15	800	50	512	400	0.70
27	15	800	50	512	50	0.60
28	15	800	50	512	50	0.65
29	15	800	50	512	50	0.75
30	15	800	50	512	50	0.80
31	15	800	50	512	50	0.90
32	15	800	100	512	50	0.70
33	15	1000	100	512	50	0.70
34	15	1200	100	512	50	0.70
35	15	1400	100	512	50	0.70

Le tableau A.2, ci-dessous, établit une comparaison globale des performances des différents modèles d'entrainement. C'est par cette comparaison que nous avons pu obtenir le modèle final d'entrainement.

Tableau A.2 Comparaison des performances des différents modèles d'entrainement de Mask-RCNN adapté au cas du perchage de drone

Configurations	F1_score moyen de l'ensemble des points	F1_score moyen de la meilleure moitié	F1_score moyen du meilleur quart	Meilleur F1_score de la courbe
Configuration n°0	0.532	0.576	0.590	0.599
Configuration n°1	-0.013	-0.013	-0.009	-0.006
Configuration n°2	-0.019	-0.007	+0.007	+0.016
Configuration n°3	+0.025	+0.024	+0.025	+0.030
Configuration n°4	+0.006	-0.006	-0.008	-0.004
Configuration n°5	-0.084	-0.045	-0.024	-0.020
Configuration n°6	+0.018	+0.015	+0.013	+0.020
Configuration n°7	+0.030	+0.016	+0.011	+0.010
Configuration n°8	-0.035	-0.026	-0.023	-0.020
Configuration n°9	+0.028	+0.021	+0.019	+0.023
Configuration n°10	+0.051	+0.042	+0.041	+0.038

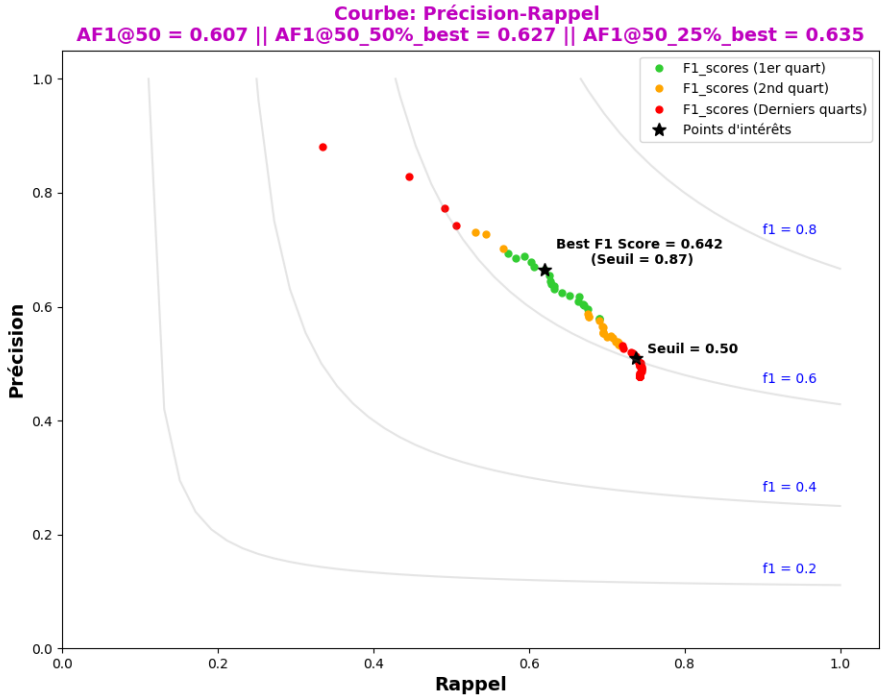
Configurations	F1_score moyen de l'ensemble des points	F1_score moyen de la meilleure moitié	F1_score moyen du meilleur quart	Meilleur F1_score de la courbe
Configuration n°0	0.532	0.576	0.590	0.599
Configuration n°11	+0.003	+0.004	+0.011	+0.024
Configuration n°12	+0.026	+0.028	+0.031	+0.035
Configuration n°13	+0.046	+0.028	+0.022	+0.028
Configuration n°14	+0.025	+0.021	+0.025	+0.024
Configuration n°15	+0.044	+0.039	+0.042	+0.044
Configuration n°16	+0.043	+0.029	+0.028	+0.029
Configuration n°17	+0.020	+0.018	+0.018	+0.015
Configuration n°18	-0.011	-0.016	-0.013	-0.012
Configuration n°19	-0.010	-0.001	-0.003	-0.003
Configuration n°20	+0.017	+0.009	+0.007	+0.008
Configuration n°21	+0.021	+0.014	+0.014	+0.015
Configuration n°22	+0.025	+0.011	+0.010	+0.016
Configuration n°23	+0.066	+0.044	+0.037	+0.037

Configurations	F1_score moyen de l'ensemble des points	F1_score moyen de la meilleure moitié	F1_score moyen du meilleur quart	Meilleur F1_score de la courbe
Configuration n°0	0.532	0.576	0.590	0.599
Configuration n°24	+0.031	+0.027	+0.027	+0.028
Configuration n°25	+0.047	+0.035	+0.035	+0.037
Configuration n°26	-0.035	-0.025	-0.019	-0.021
Configuration n°27	+0.056	+0.040	+0.033	+0.033
Configuration n°28	+0.055	+0.040	+0.040	+0.038
Configuration n°29	+0.045	+0.031	+0.028	+0.027
Configuration n°30	+0.062	+0.039	+0.036	+0.037
Configuration n°31	+0.030	+0.022	+0.026	+0.026
Configuration n°32	+0.061	+0.037	+0.030	+0.028
Configuration n°33	+0.056	+0.046	+0.044	+0.042
Configuration n°34	+0.075	+0.051	+0.045	+0.043
Configuration n°35	+0.067	+0.039	+0.032	+0.029

Les résultats indiqués en vert correspondent aux meilleurs résultats du tableau. Les résultats de comparaisons fournis par le tableau A.2 indiquent que la meilleure configuration d'entrainement est la Configuration n°34. Cette dernière possède donc la meilleure configuration d'entrainement parmi celles sélectionnées dans ce projet de recherche.

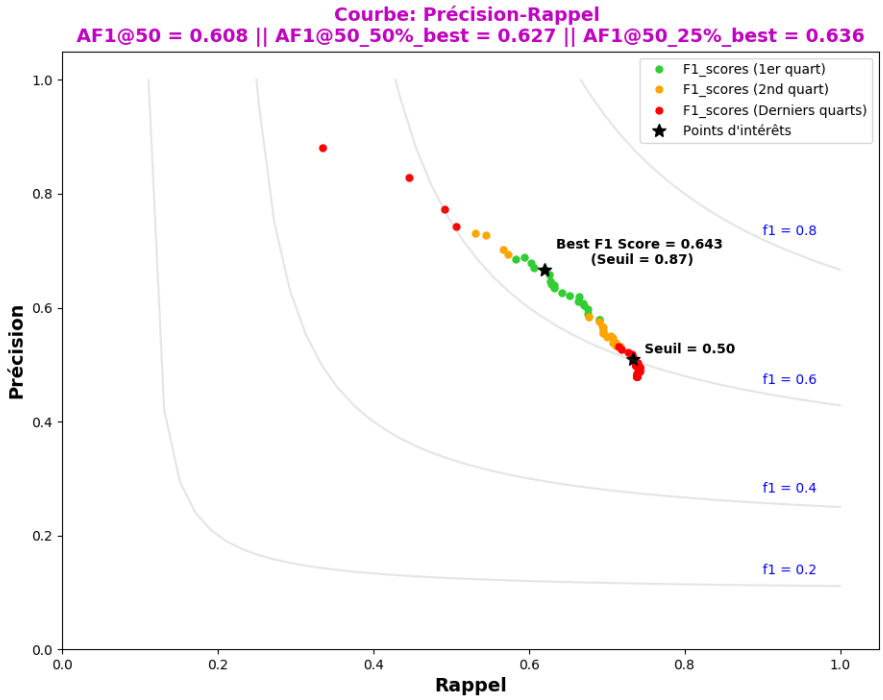
Courbes Précision-Rappel selon les paramètres "PRE_NMS_LIMIT" et "POST_NMS_INFERENCE" choisis

Cette deuxième section regroupe l'ensemble des courbes "Précision-Rappel" selon la configuration des paramètres "PRE_NMS_LIMIT" et "POST_NMS_INFERENCE".



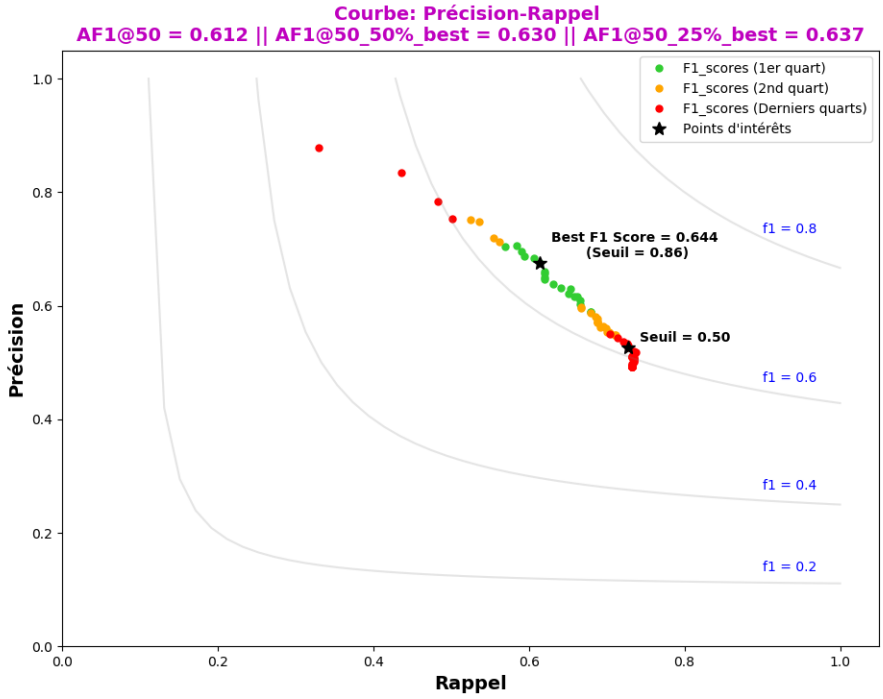
Configuration n°0

- PRE_NMS_LIMIT = 6000
- POST_NMS_INFERENCE = 1000



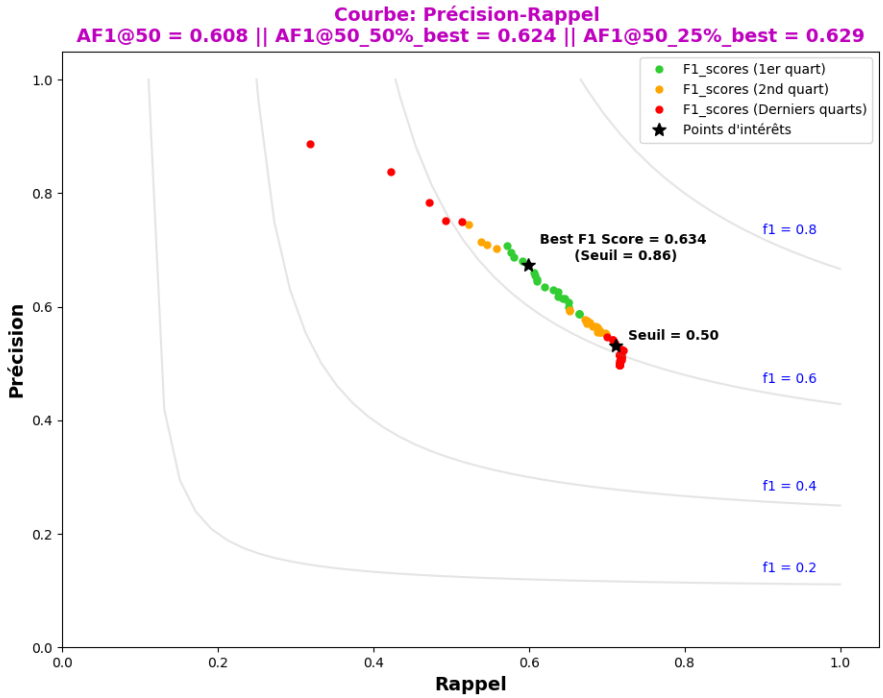
Configuration n°1

- PRE_NMS_LIMIT = 3000
- POST_NMS_INFERENCE = 1000



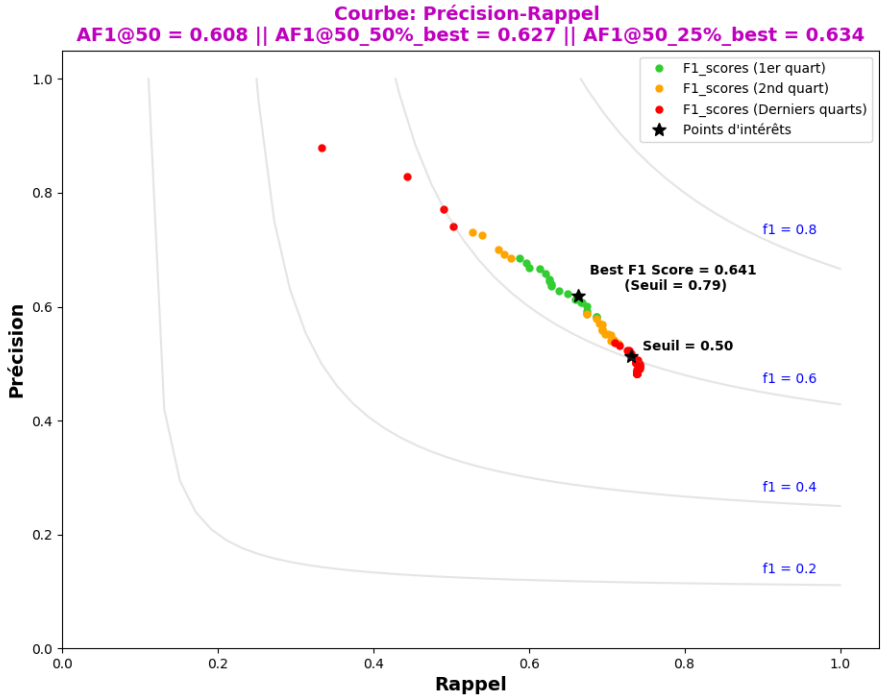
Configuration n°2

- PRE_NMS_LIMIT = 2500
- POST_NMS_INFERENCE = 1000



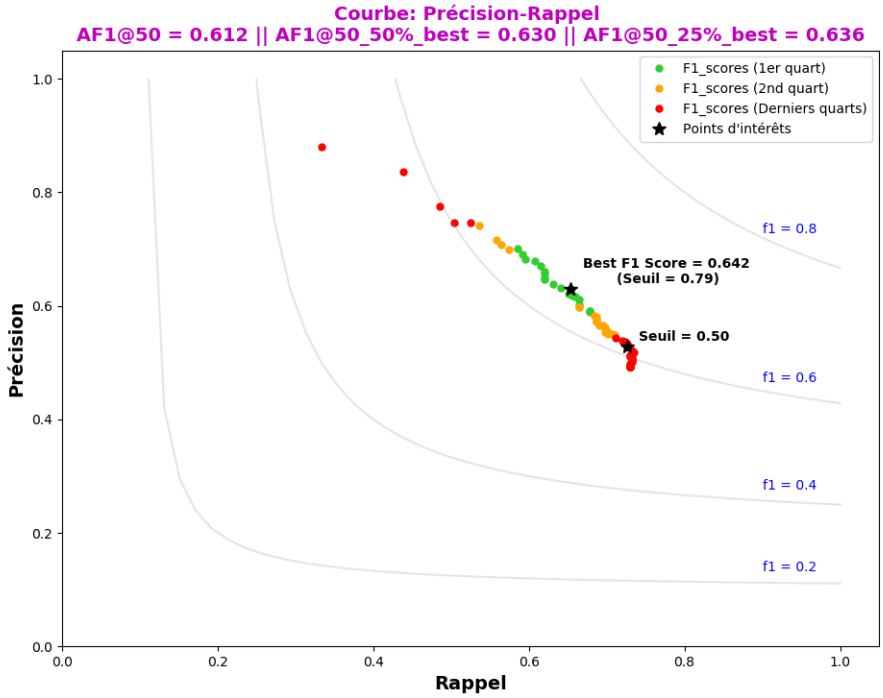
Configuration n°3

- PRE_NMS_LIMIT = 2000
- POST_NMS_INFERENCE = 1000



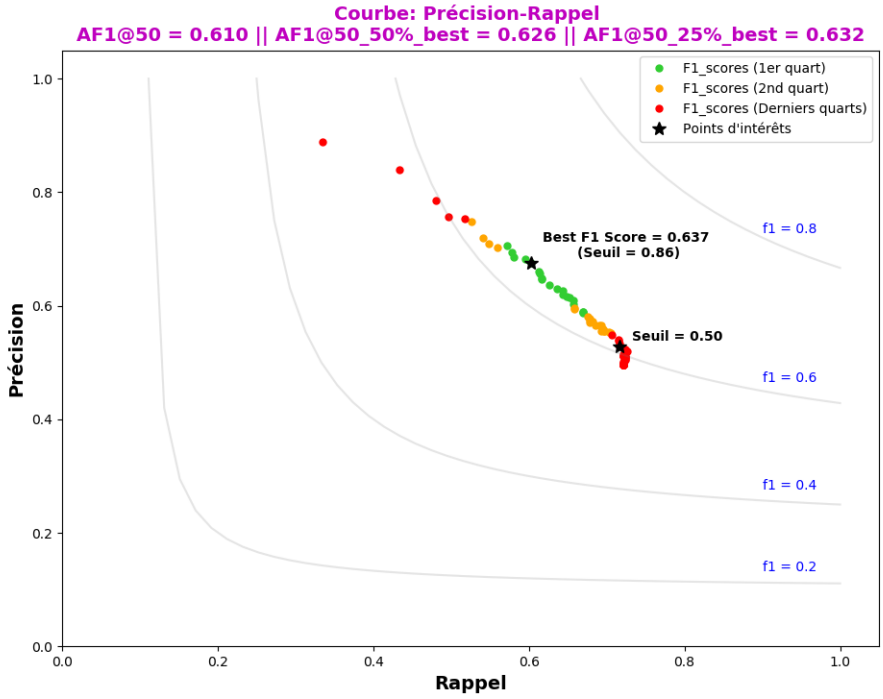
Configuration n°4

- PRE_NMS_LIMIT = 6000
- POST_NMS_INFERENCE = 900



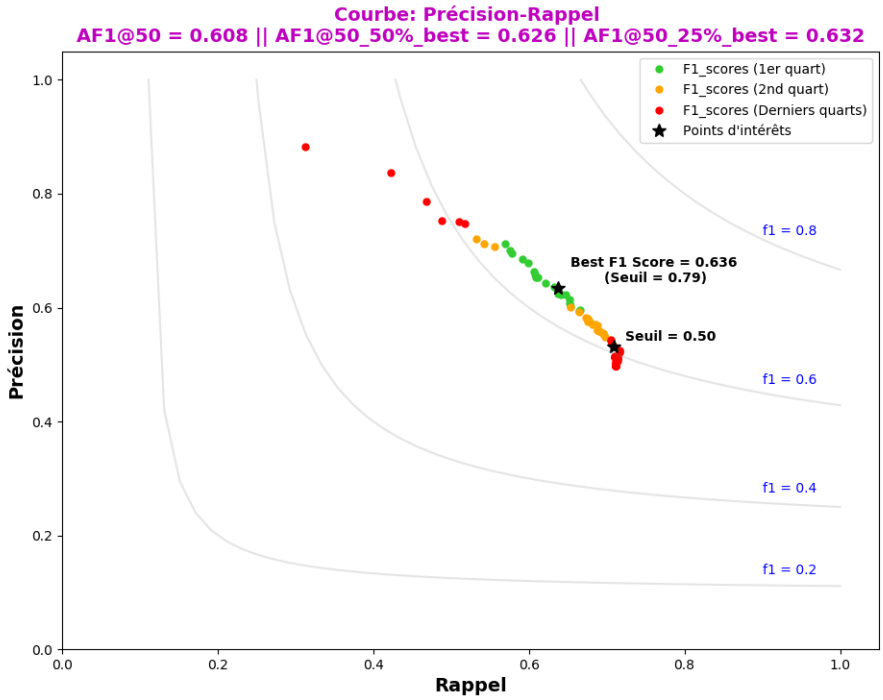
Configuration n°5

- PRE_NMS_LIMIT = 6000
- POST_NMS_INFERENCE = 800



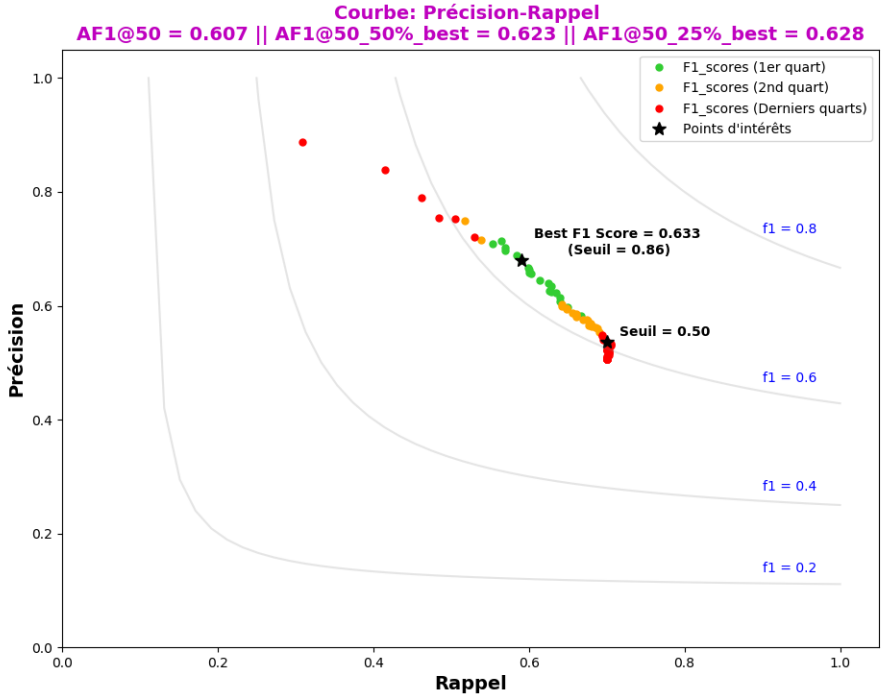
Configuration n°6

- PRE_NMS_LIMIT = 6000
- POST_NMS_INFERENCE = 700



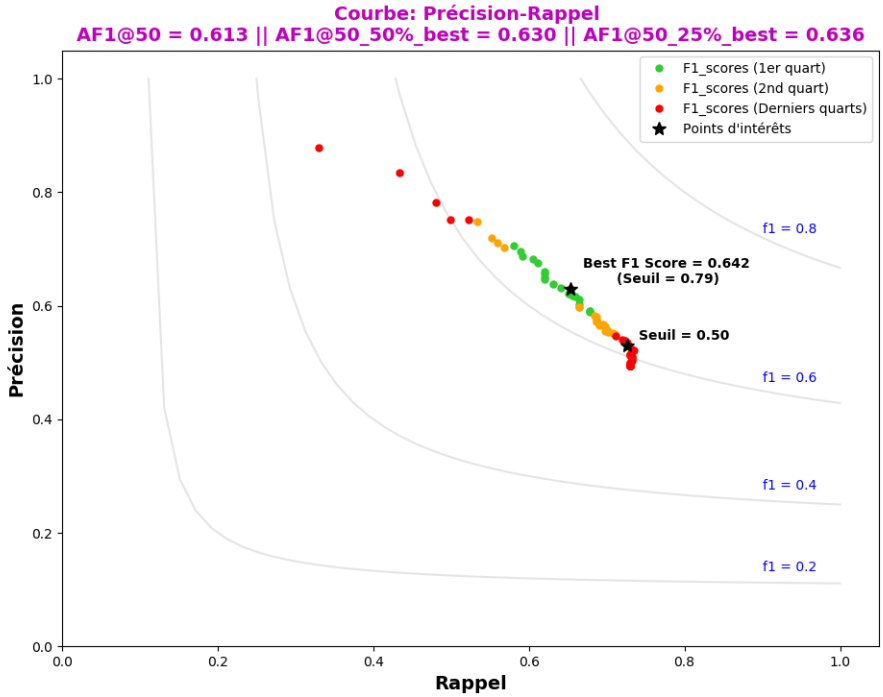
Configuration n°7

- PRE_NMS_LIMIT = 6000
- POST_NMS_INFERENCE = 600



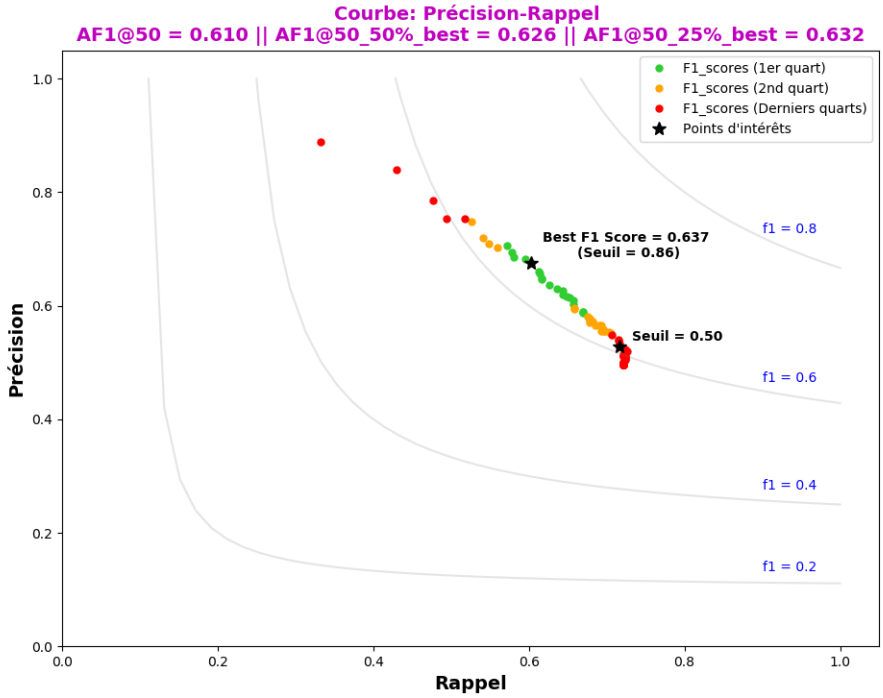
Configuration n°8

- PRE_NMS_LIMIT = 6000
- POST_NMS_INFERENCE = 500



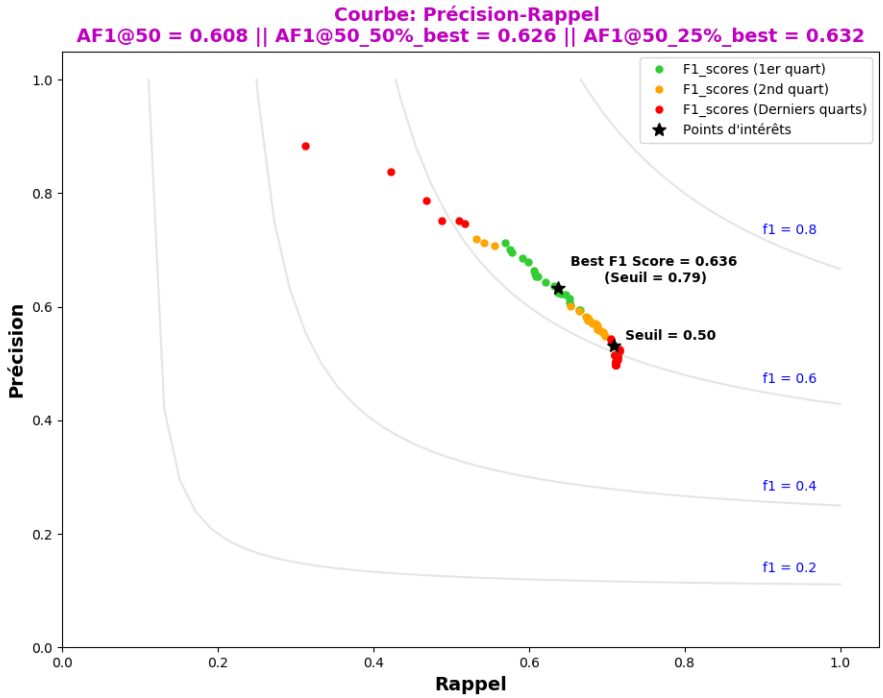
Configuration n°9

- PRE_NMS_LIMIT = 2500
- POST_NMS_INFERENCE = 800



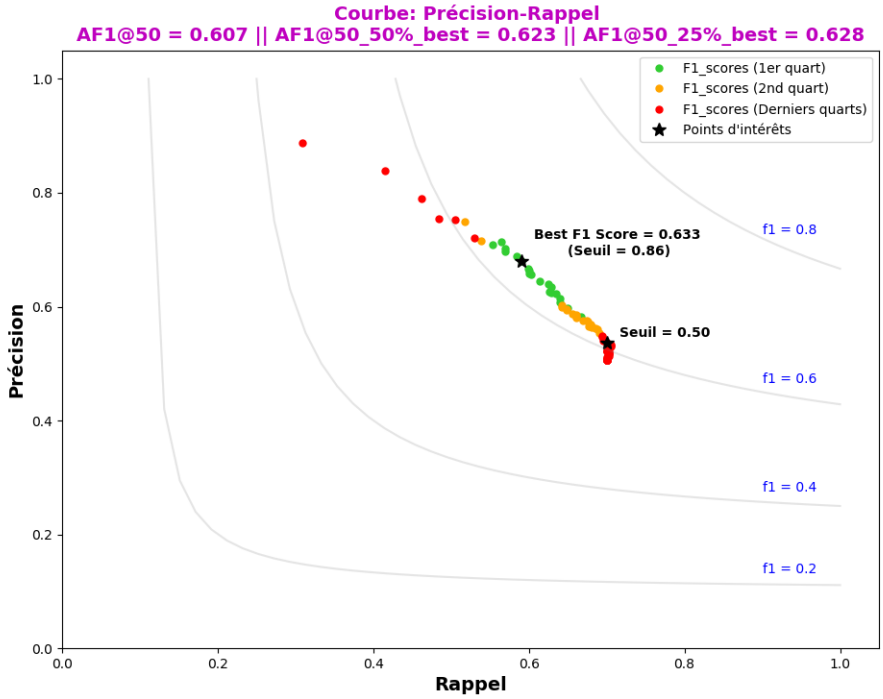
Configuration n°10

- PRE_NMS_LIMIT = 2500
- POST_NMS_INFERENCE = 700



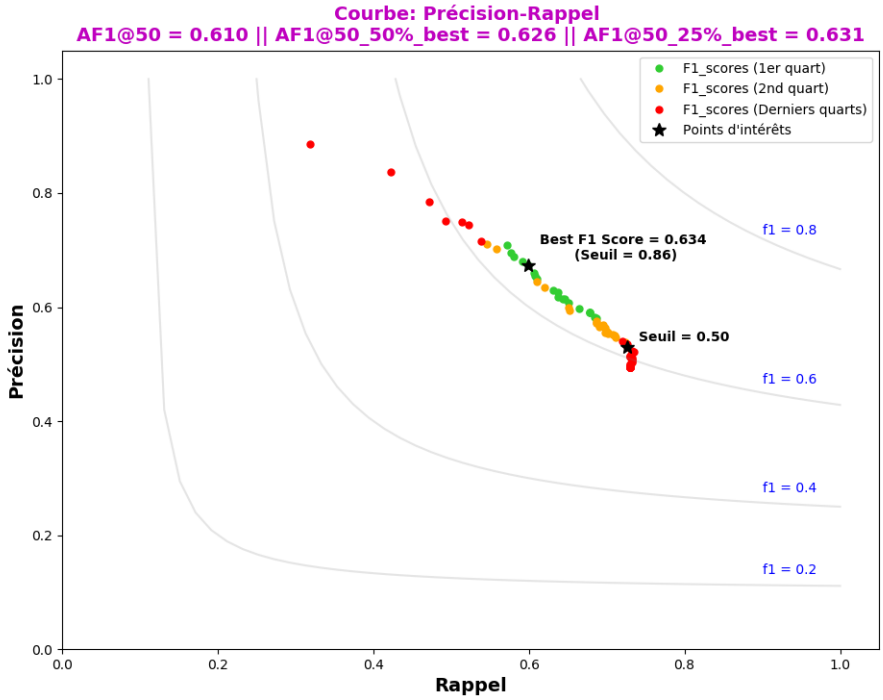
Configuration n°11

- PRE_NMS_LIMIT = 2500
- POST_NMS_INFERENCE = 600



Configuration n°12

- PRE_NMS_LIMIT = 2500
- POST_NMS_INFERENCE = 500



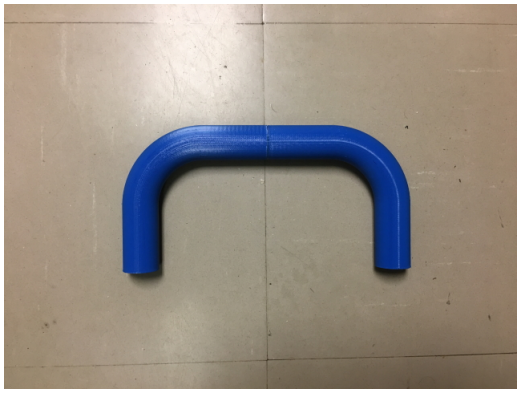
Configuration n°13

- PRE_NMS_LIMIT = 2000
- POST_NMS_INFERENCE = 800

ANNEXE B ENSEMBLE DES SUPPORTS UTILISÉS POUR LES TESTS EXPÉRIMENTAUX

Images des objets utilisés pour les tests expérimentaux

(a) Objet n°1 : Gourde
Diamètre = 70mm



(b) Objet n°2 : Barrière bleue
Diamètre = 29.9mm

(c) Objet n°3 : Barrière blanche 1
Diamètre = 39.7mm

(d) Objet n°4 : Barrière blanche 2
Diamètre = 50.2mm

(a) Objet n°5 : Cylindre rose 1
Diamètre = 62mm

(b) Objet n°6 : Cylindre rose 2
Diamètre = 80mm

(c) Objet n°7 : Cylindre en carton
Diamètre = 43.6mm

(d) Objet n°8 : Cylindre en bois
Diamètre = 24.2mm

(e) Objet n°9 : Lampe de chevet
Diamètre = 19.1mm

(f) Objet n°10 : Branche 1
Diamètre = 15.5mm - 18.3mm

(a) Objet n°11 : Branche 2
Diamètre = 11.2mm - 15mm

(b) Objet n°12 : Branche 3
Diamètre = 56mm - 58mm

(c) Objet n°13 : Branche 4
Diamètre = 38.6mm - 41.5mm

(d) Objet n°14 : Branche 5
Diamètre = 58.5mm - 61.5mm

(e) Objet n°15 : Branche 6
Diamètre = 28.6mm - 31mm

(f) Objet n°16 : Branche 7
Diamètre = 68.7mm - 70.3mm