

Ajout de résultats de machine learning aux drones agricoles en utilisant un « Companion Computer »



Informatique de gestion

Thèse de Bachelor

Écrit par : Olivier Arbella

Professeur en charge : Dominique Genoud

Travail rendu le : 30.07.2019

RÉSUMÉ

L'écriture de ce travail fut réalisée dans le contexte du Bachelor en informatique de gestion à l'HES-SO Valais. L'objectif principal de ce travail de Bachelor est d'établir la faisabilité d'un système automatisé de machine learning embarqué sur un Companion Computer (CC).

Cette thèse réalisée avec l'aide de l'entreprise de drone agricole Areo41 est constituée de recherche de sur les différents OS (Operating System) embarqués et leur communication avec un Flight Controller Unit (FCU). Ensuite un prototype devra être réalisé afin de réaliser de l'analyse d'image via une caméra en temps réel sur un microordinateur, puis implémenter un comportement au drone en cas d'arbre. Des tests seront ensuite réalisés sur en drone réel.

Enfin, nous citerons et expliquerons les différentes limitations techniques relevées. Si des solutions ont été trouvées, elles seront détaillées.

Mots-clés : drone, machine learning, reconnaissance d'image, automatisation, réseau de neurones

AVANT-PROPOS

Dans notre monde moderne, un grand nombre de processus ont réussi à être automatisés grâce à l'informatique et la robotique. Le monde de l'agriculture n'y fait pas exception. Il reste cependant certains processus qui sont encore difficiles à automatiser comme le sulfatage précis de vigne dans la pente. Ces opérations sont encore souvent réalisées à dos d'homme ou par des hélicoptères. Il est cependant, possible d'imaginer de remplacer certaines de ces opérations par des drones d'une certaine envergure plus précise et plus facile d'utilisation.

Ces drones ne sont que de simples appareils pilotés à distance. L'entreprise Aero41 aimerait proposer un moyen de rendre plus sécurisés leurs propres appareils. Il serait capable d'introduire de la reconnaissance d'image via du machine learning, à bord de l'appareil via un Companion Computer.

Ce thème de thèse de Bachelor propose d'utiliser un drone et de lui permettre d'éviter des obstacles grâce à de la reconnaissance d'image effectuée sur un Companion Computer. Ceux-ci interagissent directement avec le contrôleur de vol du drone pour lui donner des ordres d'urgence (changer de direction, monter ou s'arrêter par exemple). La reconnaissance d'image sera effectuée en utilisant des algorithmes de machine learning.

REMERCIEMENTS

Je tenais grandement à remercier toutes les personnes suivantes qui m'ont aidé, apporté leur soutien à la réalisation de cette thèse :

Dr Dominique Genoud pour avoir été mon professeur durant la réalisation de ce projet. Il s'est montré très compréhensif et a toujours été disponible en cas de besoin. Il m'a donné de nombreux conseils et fait preuve d'une extrême gentillesse.

M. Jérôme Treboux, M. Calixte Mayoraz, et Justin Capik de l'équipe de la HES-SO, qui m'ont guidé dans la réalisation de la partie machine learning et reconnaissance d'image, et accepté dans leur bureau.

L'équipe d'Aero41 qui m'a accueilli durant la réalisation de ce projet et répondu à mes questions.

Mme Coralie Maret, Mme Aurélie Glassey, M. Jonathan Shnyder qui m'ont apporté leur soutien moral et leur temps.

Mme Alexandra Hugo, qui m'a permis de me guider et conseiller moralement le long de cette thèse.

Et enfin, ma correctrice personnelle d'orthographe, confidente, qui a renoncé à ses vacances pour me soutenir moralement, ma maman.

TABLE DES MATIÈRES

Table des matières.....	iv
Table des illustrations.....	viii
Liste des abréviations.....	x
Glossaire.....	xi
1. Introduction.....	1
1.1. Problématique et contexte.....	1
1.2. Méthodologie de travail.....	1
1.2.1. Étude.....	3
1.2.2. État de l'art.....	3
1.2.3. Choix technologique.....	3
1.2.4. Récolte de données.....	3
1.2.5. Reconnaissance d'image.....	4
1.2.6. Implémentation.....	4
1.2.7. Validation.....	4
2. Drone.....	5
2.1. Qu'est-ce qu'un drone.....	5
3. Quels sont les composants d'un drone.....	6
3.1. Le châssis.....	6
3.2. Le système de propulsion.....	7
3.2.1. Les moteurs.....	7
3.2.2. Les hélices.....	7
3.2.3. Electronic Speed Control Circuits (ESCs).....	8
3.3. Système d'alimentation.....	8
3.3.1. Power Distribution Board.....	8
3.3.2. La batterie.....	8
3.4. Système de communication.....	9
3.4.1. Radiocommande (TX).....	9
3.4.2. Récepteur RX.....	10
3.4.3. Ground Station.....	11

3.4.4.	Protocole MavLink	11
3.5.	Flight Control Unit	12
3.6.	Companion Computer	Erreur ! Signet non défini.
3.6.1.	Système d'exploitation du Companion Computer	14
3.6.2.	Couche ROS	14
3.6.3.	MavROS	14
4.	État de l'art - OS pour Companion Computer	16
4.1.	Analyse	17
4.2.	Maverick	18
4.2.1.	Installation	18
4.2.2.	Configuration	18
4.2.3.	Fonctionnement	19
4.2.4.	Problèmes	20
4.2.5.	Test en simulateur	20
4.2.6.	Durabilité	20
4.2.7.	Licence	20
4.3.	Dronecode/ArduPilot	21
4.3.1.	Installation	21
4.3.2.	Fonctionnement	21
4.3.3.	Configuration	22
4.3.4.	Fonctionnalité	22
4.3.5.	Test en simulateur	22
4.3.6.	Test en temps réel	22
4.3.7.	Durabilité	22
4.3.8.	Licence	22
4.3.9.	Résumé	23
4.4.	FlytOS	24
4.4.1.	Installation	24
4.4.2.	Fonctionnement	25
4.4.3.	Fonctionnalités	26

4.4.4.	Test en simulateur.....	26
4.4.5.	Test en temps réel	26
4.4.6.	FlytPod	27
4.4.7.	Durabilité	27
4.4.8.	License	27
5.	Choix technologique	28
5.1.	Restrictions matérielles	28
5.2.	Simulation	28
5.3.	Problèmes rencontrés.....	31
5.4.	Tableau de comparaison.....	32
6.	Etat de l'art - Machine Learning embarqué.....	34
6.1.	Solutions observer.....	34
6.1.1.	Iris on board	34
6.1.2.	MobileEYE.....	35
6.1.3.	Ainstein	37
6.1.4.	Px4 Obstacle Avoidance	38
7.	Implémentation	39
7.1.	Business Understanding + récolte	39
7.2.	Data understanding	42
7.3.	Data preparation	43
7.4.	Modélisation	44
7.4.1.	Clarifai	44
7.4.2.	Darknet.....	44
7.4.3.	Réseau de neurones.....	46
7.5.	Data preparation (2)	47
7.6.	Modélisation	50
7.6.1.	Paramétrages.....	50
7.7.	Evaluation.....	52
7.7.1.	Code de reconnaissance d'image.....	54
7.8.	Déploiement.....	58

7.8.1.	MavProxy DroneKit	59
7.8.2.	Code de changement de mode	60
7.8.3.	Code de reconnaissance d'image puis changement de mode	63
7.8.4.	Tests effectués	64
8.	Conclusion	68
8.1.	Script et contrôle sur le FCU	68
8.1.1.	Bilan technique	68
8.1.2.	Problèmes rencontrés	68
8.1.3.	Améliorations futures et recommandations	69
8.2.	Reconnaissance d'image	69
8.2.1.	Bilan technique	69
8.2.2.	Problèmes rencontrés	70
8.2.3.	Améliorations futures et recommandations	70
8.3.	Bilan général	70
8.4.	Conclusion personnelle	71
	Références	72
	Annexes	76
	Déclaration de l'auteur	108

TABLE DES ILLUSTRATIONS

Figure 1: CRISP-DM schéma, https://www.kdnuggets.com/wp-content/uploads/crisp-dm-4-problems-fig1.png	2
Figure 2 : Drone utilisé pour test, image de l'auteur.....	6
Figure 3: Sens de rotation des hélices, récupérée sur https://d2otfaypcda2dg.cloudfront.net/blog/wp-content/uploads/2017/08/Eachine-Wizard-X220-setup-propeller-installation2.jpg	7
Figure 4: Schéma du système de propulsion, schéma réalisé par l'auteur.....	8
Figure 5: Schéma du système d'alimentation, schéma réalisé par l'auteur	9
Figure 6 : différent axe de contrôle, récupéré sur	10
Figure 7: Schéma du système de communication, schéma réalisé par l'auteur	11
Figure 8: common.xml, capture d'écran de l'auteur.....	12
Figure 9: Schéma des flux d'énergie avec FCU, Schéma réalisé par l'auteur.....	13
Figure 10: flux d'énergie et de données d'un drone récapitulé, schéma réalisé par l'auteur.....	15
Figure 11 : Drone DJI MG-1 récupéré sur : https://www4.djicdn.com/cms_uploads/ckeditor/pictures/1179/content_MG-1S_b.jpg	17
Figure 12 : Architecture Maverick, retrouvé sur : https://goodrobots.github.io/maverick/current/	19
Figure 13: architecture dronecode, schéma de l'auteur	21
Figure 14 : Ensemble FlytOS, récupéré sur https://flytbase.com	24
Figure 15 Architecture FlytOS, récupéré sur https://flytbase.com	25
Figure 16: Schéma de simulation, récupéré sur : https://dev.px4.io/v1.9.0/en/simulation/	29
Figure 17: schéma de simulation HITL, récupéré sur https://dev.px4.io/v1.9.0/en/simulation/hitl.html	29
Figure 18: prototype pour tester les connexions, image de l'auteur	30
Figure 19: Schéma de fonctionnement, schéma réalisé par l'auteur	31
Figure 20: problème d'installation geolib, capture d'écran de l'auteur	31
Figure 21: système Casia, récupéré sur : http://www.irisonboard.com/casia/	34
Figure 22: vision de la caméra Casia, capture d'écran de l'auteur, prise sur https://www.youtube.com/watch?time_continue=75&v=d4oM7n4mb00	35
Figure 23 Installation MobileYE Series, capture d'écran de l'auteur, prise sur https://www.mobileye.com/us/fleets/products/mobileye-6-collision-avoidance-system/	36
Figure 24 Vision de la caméra MobileEYE, capture d'écran de la vidéo de promotion, prise sur https://www.mobileye.com/us/fleets/technology/	37
Figure 25: CRISP-DM schéma, https://www.kdnuggets.com/wp-content/uploads/crisp-dm-4-problems-fig1.png	39
Figure 26: drone équipé d'un Raspberry, image de l'auteur	40
Figure 27: image floue, image de l'auteur	41

Figure 28: prise avec vibration et net, image de l'auteur	42
Figure 29 : image du dataset d'entraînement, prise par Mr Jérôme Treboux	43
Figure 30: couche de neurones, récupéré sur	47
Figure 31: Arbre provenant de récolte de données et .txt correspondant, capture d'écran de l'auteur	48
Figure 32: Annotation tool, capture d'écran de l'auteur	49
Figure 33: structure du fichier obj.data, capture d'écran de l'auteur	50
Figure 34: entraînement du réseau en cours, capture de l'écran de l'auteur.	51
Figure 35: output de Darknet, image de l'auteur.	52
Figure 36: output de Darknet avec mAP, image de l'auteur	53
Figure 37: code de reconnaissance, capture de l'écran de l'auteur	54
Figure 38: code de reconnaissance, capture de l'écran de l'auteur	54
Figure 39: code de reconnaissance, capture de l'écran de l'auteur	55
Figure 40: code de reconnaissance, capture de l'écran de l'auteur	55
Figure 41: code de reconnaissance, capture de l'écran de l'auteur	56
Figure 42: code de reconnaissance, capture de l'écran de l'auteur	56
Figure 43: code de reconnaissance, capture de l'écran de l'auteur	57
Figure 44: code de reconnaissance, capture de l'écran de l'auteur	57
Figure 45: reconnaissance d'image, capture d'écran de l'auteur	58
Figure 46: structure du Raspberry, schéma de l'auteur	58
Figure 47: connexion au FCU via mavProxy, capture d'écran de l'auteur	59
Figure 48: structure du Raspberry Version 2, schéma de l'auteur.....	60
Figure 49: Schéma de séquence du code, schéma de l'auteur	60
Figure 50: Code de connexion au FCU, capture d'écran de l'auteur	61
Figure 51: listener de la channel 12, capture d'écran de l'auteur.....	62
Figure 52: logique du changement de mode, capture d'écran de l'auteur	62
Figure 53: activation de la reconnaissance d'image, capture d'écran de l'auteur.....	63
Figure 54: Code du calcul du ratio, capture d'écran de l'auteur	63
Figure 55: matériel utilisé pour test réel, photo prise par l'auteur,	64
Figure 56: logs du FC, image de l'auteur	65
Figure 57: image de stop, de l'auteur	66
Figure 58: reconnaissance d'objet avec grand nombre de faux positif, image de l'auteur.	67
Figure 59 Logo de darknet, récupéré sur : https://github.com/pjreddie/darknet	76

LISTE DES ABRÉVIATIONS

Unmanned Aerial Vehicle	UAV
Electronic Speed Control	ESC
Global Positioning System	GPS
Flight Control Unit	FCU
Operating System	OS
GigaByte	GB
Companion Computer	CC
Robotic Operating System	ROS
Machine Learning	ML
Application Programming Interface	API
Representational State Transfer	REST
Extensible Markup Language	XML
You Only Look Once	YOLO
Secure Shell	SSH
Secure Digital	SD

GLOSSAIRE

Machine Learning :	Ou apprentissage automatique. Champ d'étude pour conférer à une machine la possibilité d'apprendre d'elle-même.
Companion Computer :	Microordinateur monté sur un drone capable de communiquer avec un FCU.
Operation Systeme :	Ou système d'exploitation. Ensemble de programmes utilisant les ressources physiques d'un ordinateur.
Open source :	Programme ou ensemble de programme où le code-source est soumis à une License permettant aux utilisateurs de le modifier, l'étudier et le distribuer pour n'importe quelle utilisation.
Onboard :	A bord d'un appareil.
Firmware :	Programme bas niveau permettant d'utiliser un hardware spécifique à un appareil.
Hardware :	Composant physique d'un ordinateur.
Autopilot:	Synonyme de FCU.
From scratch :	A partir de rien.
Framerate :	Fréquence d'image apparaissant.
Bias :	Fonction statistique représentant l'erreur expérimentale.
Mean Average Precision :	Ou précision moyenne. Elle permet de mesurer la précision d'une détection d'image.
Docker :	Ensemble d'application permettant d'isoler un programme sous forme de container.
API :	Ensemble de d'outil, protocole de communication permettant de construire des applications.

REST :	Architecture logiciel permettant de créer des applications Web.
Channel Swapping :	Changement de chargement des couleur, OpenCV charge les couleurs de BGR à la place de RGB.
Librairie :	Ou Bibliothèque logicielle. Collection de routines déjà compilées capable d'être exécuté par des programmes.
Script :	Suite de commande simple représenté sous forme de programme chargé d'exécuter une action.
Framework :	Ensemble d'outil et / ou d'application permettant de construire des programmes informatiques.
Retrofit :	Action d'ajouter un composant à quelque chose qui n'était pas équipé à l'origine.
Product Backlog :	Liste de fonctionnalité d'un produit représentée sous forme d'user story, qui doit être réalisé.
CUDA :	Technologie développée par la firme NVIDIA permettant d'utiliser une carte graphique afin de procéder à des calculs, en général pour de l'apprentissage automatique. Une carte graphique est plus performante qu'un processeur pour faire du calcul matriciel.
Builds :	Compilément d'une application.
Channel :	Chaîne de communication.
K-Means :	Algorithme permettant de regrouper des données.
Switch :	Intérupteur.
Make:	Commande permettant de compiler un programme.
Multicopter :	Appareil compose de plusieurs rotors.

Failsafe :	Procédure d'urgence programmée pour un drone en cas d'incident ou problème.
Loiter :	Mode permettant de s'arrêter et tenir une position et altitude en cas de relâchement des commandes.
Hold :	Mode permettant de réaliser du surplace.
Listener :	Concept de programmation, où l'on attend un évènement.
OpenCV :	Librairie de fonctions permettant de réaliser du traitement d'images.
Dataset:	Ensemble de données.
Agile :	Méthodologie de travail par itérations.

1. Introduction

1.1. Problématique et contexte

Les évolutions techniques et la miniaturisation des technologies ont amenés de prodigieux instruments de travail. Il s'agit d'aéronef sans humain, plus communément nommé drones ou UAV (Unmanned Aerial Vehicle). Leurs utilisations sont multiples et variées, en passant du plus simple jouet au drone de transport logistique. Un drone lambda est en général contrôlé à distance via une télécommande envoyant les commandes au drone. Cependant, lorsque la taille du drone est conséquente, la sécurité doit être optimale, car un accident est très vite arrivé.

Afin d'améliorer la sécurité de ces appareils, il est fréquent de trouver des senseurs permettant de prévenir un obstacle. Mais pour reconnaître des obstacles, un FCU est trop lent et ne devrait s'occuper que de gérer la stabilisation de l'appareil. Un CC permet de réaliser des calculs et du machine learning, bien plus rapidement, et possède moins de limitations techniques.

Sur ces microordinateurs, il est commun de trouver un système d'exploitation Linux. Cependant, dû aux nombres très élevés de version du noyau UNIX, il convient de définir un système sur lequel le développement pourra être effectué, et réaliser du machine learning sur une caméra en direct et communiquer avec un autopilote. Pour reconnaître des obstacles, le Companion Computer devra utiliser des modèles entraînés d'algorithmes d'apprentissage automatique tel qu'un réseau de neurones. Étant donné que ces algorithmes sont en général entraînés sur un ordinateur bien plus puissant, il faudra trouver une solution pour réaliser la reconnaissance Onboard.

Ces différents points nous amènent donc à la question « Comment ajouter des résultats de machine learning aux drones agricoles en utilisant un Companion Computer »

Cette thèse permettra de proposer un moyen de pouvoir faire du machine learning sur un drone avec une caméra embarquée, établir le comportement à adopter en cas d'obstacle, puis tester les résultats sur un drone réel lors d'essai en extérieur.

1.2. Méthodologie de travail

Aucune méthodologie ne nous a été imposée, car nous devons choisir celle qui correspond aux ressources et délais impartis. Un projet comme celui-ci mêlant plusieurs domaines différents, demande cependant un cadre rigoureux afin de ne pas perdre du temps. L'étudiant, ayant une formation dans ses cours sur la méthode agile, a estimé que son application dans un projet seul ne pourrait pas s'appliquer tel quel.

Nous utiliserons alors une variation d'agile afin de régler l'orientation du projet au fur et à mesure des semaines de travail. Nous nous sommes fixé un rendez-vous chaque semaine avec M. Dominique Genoud, afin d'évaluer la quantité de travail restant. De cette manière, nous avons réalisé des itérations d'une seule semaine, suivi d'une réunion pour fixer le cadre de l'avancement de la thèse. Nous avons rédigé dans un premier temps et suivant la consigne, un « product backlog » sur excel, afin de fixer un peu mieux le cadre de début de projet. Nous avons converti ce tableau en un journal de bord, permettant d'évaluer la semaine et le travail restant.

Ce journal de bord est disponible en annexe. Nous avons rédigé un rapport court et concis de chaque rendez-vous, afin de garder une trace écrite dans la case commentaire de la semaine.

Cette méthodologie agile sera complétée lors de la réalisation de la partie machine learning, par la méthode CRISP-DM. Celle-ci, étant plus adaptée pour des projets de machine learning, permet de changer d'étape plus rapidement et accorde plus de liberté entre les étapes.

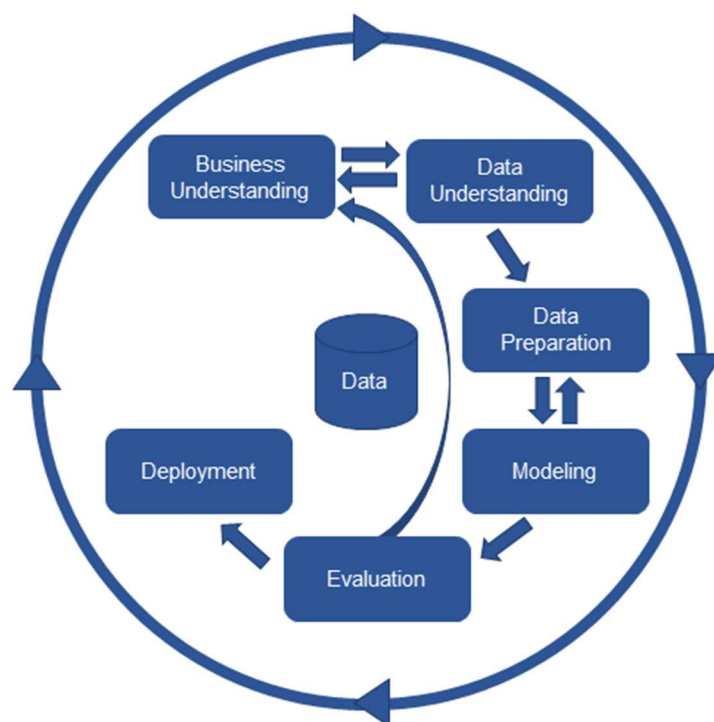


Figure 1: CRISP-DM schéma, <https://www.kdnuggets.com/wp-content/uploads/crisp-dm-4-problems-fig1.png>

De plus, nous avons fixé des étapes plus générales dans la réalisation de ce projet basée sur la donnée de la thèse.

1.2.1. Étude

Dans un premier temps, l'étudiant étant un novice en UAV, il se documentera sur le fonctionnement général d'un drone, ainsi que les connexions entre un contrôleur et un appareil, ainsi que les flux d'énergie à son bord.

1.2.2. État de l'art

Nous séparons l'état de l'art en deux catégories, celle concernant le système d'exploitation d'un Companion Computer et ensuite de machine learning embarquée.

Le but de cette thèse dans un premier temps, est d'établir les différentes solutions de système d'exploitation de Companion Computer déjà existant sur le marché, afin de manipuler un drone. L'étudiant devra rechercher les différentes solutions de système d'exploitation compatible avec un FCU sur le marché et les comparer selon différents critères. Cette étape peut se réaliser en parallèle de la phase d'étude. Afin d'expliquer ces comparaisons, l'étudiant devra écrire un état de l'art des technologies actuelles.

Par la suite, l'étudiant réalisera un état de l'art des technologies actuelles en matière de reconnaissance d'image, il analysera ensuite certains frameworks permettant de réaliser un entraînement de modèle afin de reconnaître une classe d'objets définie au préalable.

1.2.3. Choix technologique

Nous devons essayer les différents systèmes d'exploitation en simulation. Il est possible de réaliser certains tests sous un simulateur. L'installation et la configuration doivent se réaliser sur un système d'exploitation Linux.

Nous réaliserons aussi quelques tests en extérieur, afin que l'étudiant puisse comprendre l'organisation nécessaire et les difficultés rencontrées, lors de phase en extérieur.

Le résultat de ces tests et de notre analyse permettra de décider sur quel système nous allons procéder à l'implémentation d'un prototype de reconnaissance d'image.

1.2.4. Récolte de données

Afin de pouvoir réaliser de la reconnaissance d'image, l'étudiant ira sur le terrain avec son professeur afin de récolter des images d'arbres qui serviront d'image d'entraînement.

1.2.5. Reconnaissance d'image

L'étudiant étant novice en matière de reconnaissance d'image, il devra se former au préalable sur la manipulation d'image, puis sur les différents algorithmes permettant de réaliser de la détection d'objet.

L'étudiant tentera d'entraîner son propre réseau de neurones sur les images récoltées afin de détecter des arbres. Il essaiera d'implémenter la reconnaissance sur un Raspberry en temps réel.

1.2.6. Implémentation

Dernière étape pratique, la reconnaissance d'image devra permettre de modifier le comportement d'un drone automatiquement. De ce fait, le résultat de la détection d'arbre arrêtera le drone grâce à un prototype proposé par l'étudiant

1.2.7. Validation

Par la suite, ce comportement sera testé en extérieur. La mission que le prototype devrait être capable de réaliser pour validation, est simple : reconnaître un arbre en temps réel, puis envoyer une commande au FCU pour se stopper.

2. Drone

Avant d'expliquer comment nous pouvons ajouter du machine learning sur un Companion Computer, il est nécessaire de comprendre le fonctionnement d'un drone, ainsi que de comprendre comment les différents composants d'un drone communiquent ensemble. Nous focaliserons l'état de l'art sur les systèmes d'exploitation et solution déjà disponible.

2.1. Qu'est-ce qu'un drone

La définition même d'un drone se différencie selon la langue. En anglais, il signifie un aéronef robotisé et inhabité plus souvent appelé Unmanned Aerial Vehicle (UAV). Pour le français, le mot caractérise tout type de véhicule aérien, terrestre de surface ou sous-marins, dépourvus de passager ou pilote à son bord. Il est contrôlé à distance par un être humain ou peut se déplacer de manière autonome.

Étant donné le passé militaire des drones, ils sont souvent catégorisés en deux : les drones militaires, et les civils. La catégorie qui nous intéresse est la partie civile, plus spécifiquement, les multicopters commerciaux.

3. Quels sont les composants d'un drone

Comme un drone est composé de plusieurs parties électroniques, nous allons décortiquer ensemble son squelette. Nous survolerons plusieurs aspects, par leur complexité afin d'avoir une vue d'ensemble du fonctionnement d'un drone.



Figure 2 : Drone utilisé pour test, image de l'auteur.

3.1. Le châssis

Afin d'assembler un drone, les composants électroniques d'un drone reposent sur un châssis. Il est différent selon le nombre de bras, est peut ainsi avoir différentes formes et caractéristiques selon son utilisation. Les matériaux utilisés afin de constituer le châssis ont aussi leur importance. De manière générale, il s'agit de bois, plastique, métaux/alliages ou fibre de carbone. Selon l'emploi, il requiert flexibilité ou rigidité. Bien entendu, le prix est aussi à prendre en considération, car la fibre de carbone est souvent plus chère que les autres matériaux.

3.2. Le système de propulsion

Le système de propulsion est l'ensemble des composants qui permettent à l'appareil de se propulser dans les airs.

3.2.1. Les moteurs

Il s'agit des moteurs électriques qui font tourner les hélices. Selon le comportement du drone souhaité, il est nécessaire d'avoir le bon type de moteur : un drone stable aura des moteurs qui tourne à 8000 tours par minute, un polyvalent tournerons à 10000 tours/minutes, et 12000 tours/minutes pour un drone nerveux.

3.2.2. Les hélices

Les hélices viennent se visser sur les moteurs, qui en tournant extrêmement rapidement, créent une portance en déplaçant l'air vers le bas et permettant de soulever le drone dans les airs. Elles peuvent être de tailles variables selon l'utilisation, et ont un sens horaire ou contre horaire permettant de les placer à l'endroit adéquat sur le drone, tout en respectant le schéma de montage en fonction du nombre de bras du multicopter.



Figure 3: Sens de rotation des hélices, récupérée sur <https://d2otfaypcda2dg.cloudfront.net/blog/wp-content/uploads/2017/08/Eachine-Wizard-X220-setup-propeller-installation2.jpg>

3.2.3. Electronic Speed Control Circuits (ESCs)

Chaque moteur possède son propre ESC. Il s'agit d'un variateur, qui s'occupe de la gestion électrique du moteur. En fonction de la quantité de courant, ce circuit va s'occuper de faire tourner les moteurs avec la même tension afin de garder le même rapport tours par minutes. Il s'occupe aussi de gérer le frein magnétique.

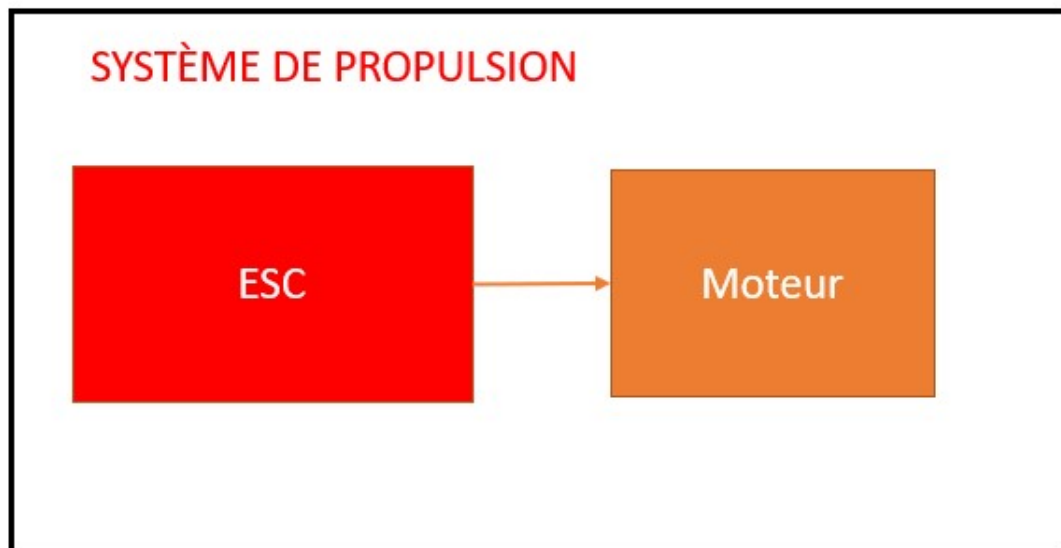


Figure 4: Schéma du système de propulsion, schéma réalisé par l'auteur

3.3. Système d'alimentation

Le système d'alimentation fournit l'électricité à tous les autres systèmes du drone. Il est bien évidemment composé d'une batterie de plusieurs cellules, ainsi que d'une Power Distribution Board.

3.3.1. Power Distribution Board

Cet élément est très simple. Il permet d'établir le courant entre la batterie, et les autres éléments du drone via des pads. Il est possible de souder les câbles du courant positif et négatif.

3.3.2. La batterie

Il s'agit de la source d'alimentation. Le nombre de cellules, ou accumulateur varie selon la batterie. Elle peut être constituée de divers matériaux, mais la tendance est à la technologie lipo (Lithium polymère). Le poids réduit de ce type de batterie est un avantage non négligeable sur un

drone. Elles sont dénué d'effet de mémoire. L'effet de mémoire d'une batterie faite à base de nickel, oblige l'utilisateur à la décharger complètement avant de la recharger. Différents critères permettent de calculer la puissance et l'autonomie d'un drone :

- La tension des batteries, qui dépend du nombre de cellules exprimées en Volts
- La capacité de l'accumulateur, il s'agit de la quantité d'énergie délivrée à charge complète, exprimée en mAh (milliampères par heure).
- La décharge ou l'intensité de courant délivrée est représentée par la variable C. Il s'agit du coefficient multiplicateur de l'ampérage restitué pendant l'utilisation. Le calcul se fait ainsi
 - Si l'on possède une batterie de 2000 mAh 75C, elle est théoriquement capable de restituer $2 \times 75 = 150$ ampères
- Le poids s'exprime comme toute les masses en gramme. Une batterie légère permet de réduire la force de poussée des moteurs, et donc la consommation d'énergie mais possède moins de lithium et donc moins de capacité

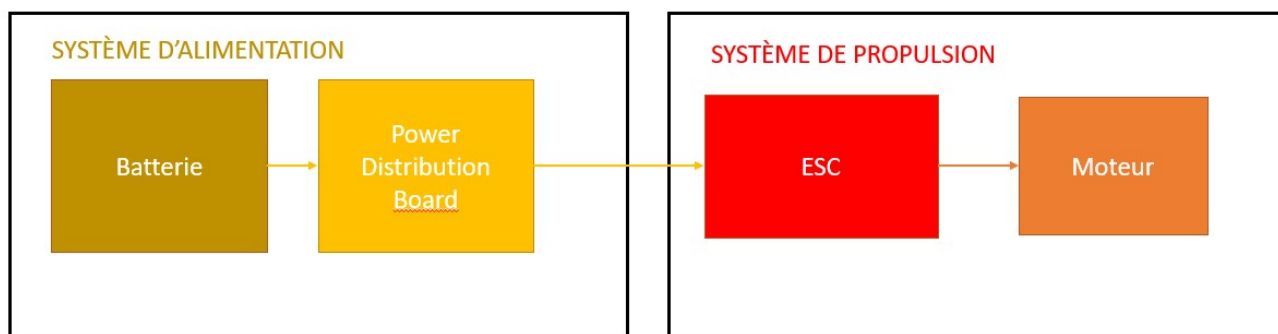


Figure 5: Schéma du système d'alimentation, schéma réalisé par l'auteur

3.4. Système de communication

Le système de communication est un ensemble d'échange de messages entre les différents systèmes d'un drone. Il s'agit de toutes les interactions qui impacteront les commandes et donc le comportement du drone.

3.4.1. Radiocommande (TX)

Afin de pouvoir envoyer des instructions à distance à un drone, un pilote utilise une télécommande utilisant les technologies radio, afin de communiquer les actions. Une télécommande utilise 2

joysticks qui commanderont les actions principales de déplacement du drone. Ces actions sont les suivantes :

- « Throttle » ou accélération : en augmentant la vitesse de rotation des hélices cela créer plus de portance et augmente l'altitude du drone
- « Roll » ou roulis : il s'agit de pivoter sur l'axe longitudinal de l'avant à l'arrière du drone
- « Pitch » : il s'agit de la rotation sur l'axe transversale, du côté droit au côté gauche
- Yaw : il s'agit de la rotation sur l'axe vertical, soit le centre

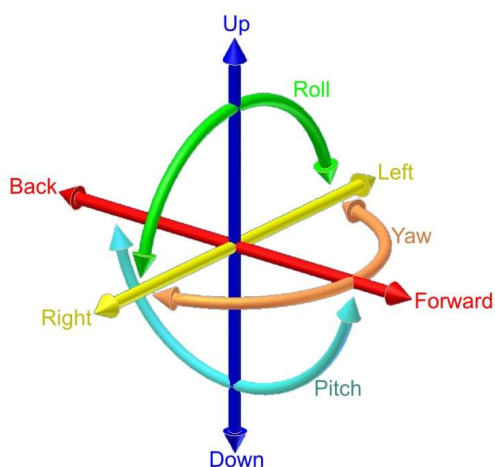


Figure 6 : différent axe de contrôle, récupéré sur

https://upload.wikimedia.org/wikipedia/commons/thumb/f/fa/6DOF_en.jpg/330px-6DOF_en.jpg

Un ensemble de commutateurs peut être assigné à des fonctionnalités diverses comme failsafe (procédure en cas de perte de contrôle).

Toutes ces communications se font via différents canaux (canaux en français), car un canal doit transporter des informations séparées. Chaque action des joysticks possède son propre canal.

En fonction des différentes marques de radiocommande, certains fabricants ont implémenté leur propre protocole de communication. Durant notre étude nous avons remarqué un vaste ensemble de protocoles proposés. Nous nous sommes concentrés sur le protocole de la radiocommande disponible pour la réalisation de cette étude.

3.4.2. Récepteur RX

Le récepteur RX, comme son nom l'indique, récupère les informations transmises par la radiocommande puis les redirige vers le FCU, qui s'occupera de traiter les paquets de données au destinataire.

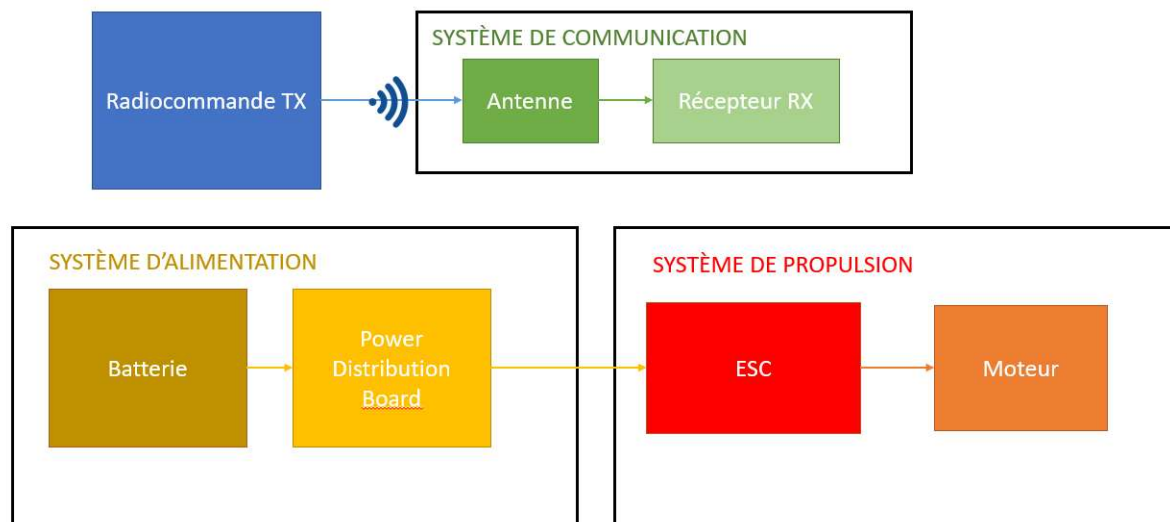


Figure 7: Schéma du système de communication, schéma réalisé par l'auteur

3.4.3. Ground Station

Une ground station est un software communiquant avec le drone par télémétrie. Elle permet de récolter les informations en direct telles que tension des moteurs, accélération, etc. Elle permet de simuler l'apparence d'un cockpit. Il est aussi possible de donner des instructions au drone, comme fixer des points de passage ou de prendre le contrôle manuellement. Elle communique via le FCU à grâce au protocole MavLink.

3.4.4. Protocole MavLink

MavLink est un protocole de communication léger, permettant d'échanger des informations de manières standardisées. Il est ainsi possible de communiquer entre des composant entre eux, ou avec une ground station, via un système hybride de publication/souscription et point to point. Il s'agit de messages définis dans des fichiers XML sur la ground station et le Companion Computer, où chaque fichier définit le message pris en charge par un système MavLink. L'implémentation de ces messages est définie via un seul fichier fichier common.xml. La majorité des outils utilisant MavLink, génèrent des bibliothèques pour chaque langage de programmation.

```
<?xml version="1.0"?>
<mavlink>
  <version>3</version>
  <dialect>0</dialect>
  <enums>
    <enum name="MAV_AUTOPILOT">
      <description>Micro air vehicle / autopilot classes. This identifies the individual model.</description>
      <entry value="0" name="MAV_AUTOPILOT_GENERIC">
        <description>Generic autopilot, full support for everything</description>
      </entry>
      <entry value="1" name="MAV_AUTOPILOT_RESERVED">
        <description>Reserved for future use.</description>
      </entry>
      <entry value="2" name="MAV_AUTOPILOT_SLUGS">
        <description>SLUGS autopilot, http://slugsuav.soe.ucsc.edu</description>
      </entry>
      <entry value="3" name="MAV_AUTOPILOT_ARDUPILOTMEGA">
        <description>ArduPilot - Plane/Copter/Rover/Sub/Tracker, http://ardupilot.org</description>
      </entry>
      <entry value="4" name="MAV_AUTOPILOT_OPENPILOT">
        <description>OpenPilot, http://openpilot.org</description>
      </entry>
      <entry value="5" name="MAV_AUTOPILOT_GENERIC_WAYPOINTS_ONLY">
        <description>Generic autopilot only supporting simple waypoints</description>
      </entry>
    </enum>
  </enums>
</mavlink>
```

Figure 8: common.xml, capture d'écran de l'auteur

Les messages transmis ne pèsent que 8 bytes, voir 14 pour la version 2. Ce qui en fait un système de transmission très efficace. Nous utiliserons MavLink afin de communiquer du CC au FCU.

3.5. Flight Control Unit

Il s'agit de l'ordinateur de bord. Il possède différents capteurs de base comme un accéléromètre, baromètre ou GPS. Il s'occupe de stabiliser le drone en l'air. Il peut aussi éviter les dérives d'un drone lorsque suffisamment de satellites sont disponibles afin de déterminer la position exacte du drone. Ce genre d'outil n'est en général pas doté de processeur, et mémoire vive très faible. Une antenne GPS peut être branché sur celui-ci afin d'améliorer la précision du surplace.

Sur certains FCU, il est possible de trouver des capteurs permettant d'éviter les collisions. Cependant, le prix de ce genre d'outil est très élevé et n'est pas open source. Afin de pallier ces inconvénients, il est possible de connecter un microordinateur comme un Raspberry afin de réduire les couts et augmenter l'efficacité. Il s'agit alors d'un compagnon Computer

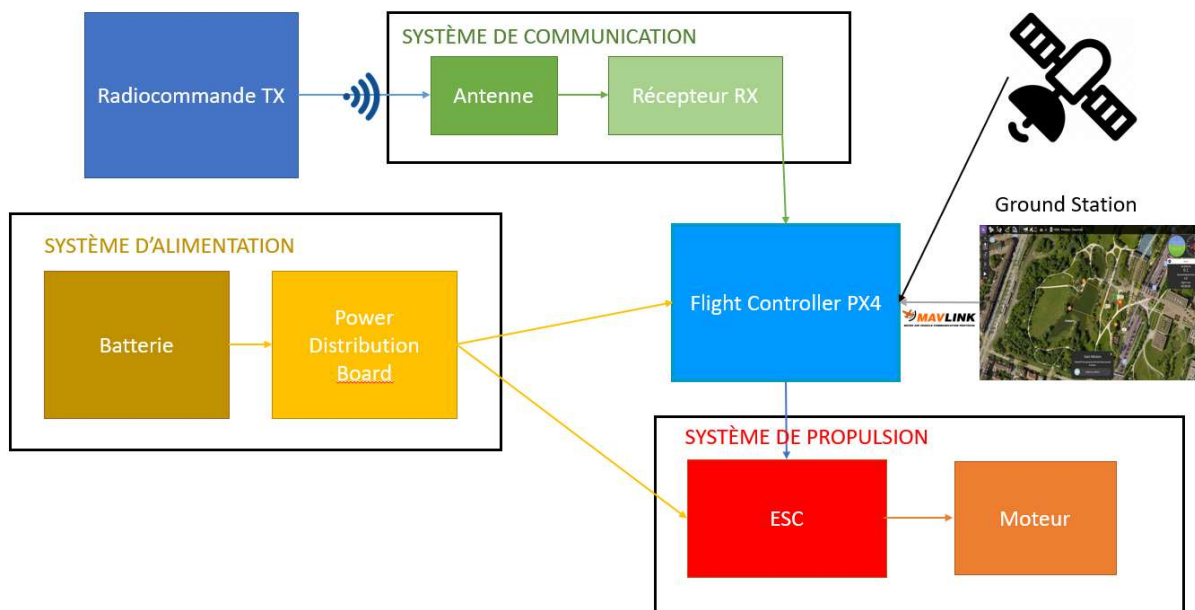


Figure 9: Schéma des flux d'énergie avec FCU, Schéma réalisé par l'auteur

3.6. Companion Computer

Comme expliqué dans le paragraphe précédent, il s'agit de microordinateur peu cher, doté d'une mémoire vive d'en général de 1GB et d'un processeur 64 bits. Étant donné les performances bien plus élevées que peut fournir ce genre d'appareil, il est possible de lancer des programmes avec des calculs complexes qui seront réalisés bien plus rapidement. Il doit être capable d'être intégré à un drone et fonctionner sur une prise 2V qui sera branchée à la batterie. Il est bien évidemment nécessaire de le faire communiquer avec le flight controller. Une option sur le FCU intitulée Offboard, qui permet au CC de prendre le contrôle du drone lorsque celle-ci est activée. Le microordinateur doit être capable de pouvoir supporter la connexion à une caméra, ou différentes sortes de capteur afin de récolter les données de ceux-ci.

Un Companion Computer s'occupera ainsi de réaliser la reconnaissance d'image d'une caméra, et ainsi reconnaîtra les obstacles en direct. Puis selon la procédure liée à l'obstacle, il enverra les commandes. Tout cela se réalisera ainsi sur un drone avec le CC en embarqué afin de pallier les problèmes éventuels du réseau tel que lag, perte de réseau ou faible vitesse de communications que peut avoir un système avec un ordinateur au sol.

Il existe une multitude de microordinateurs, les plus connus provenant de la marque Raspberry comme le RPI 3 ou RPI Zero. NVIDIA propose aussi des microordinateurs ayant des caractéristiques

très intéressantes, mais un prix plutôt élevé. L'entreprise Intel propose un kit complet pour les drones nommé Intel Aero.

3.6.1. Système d'exploitation du Companion Computer

Un Companion Computer a besoin d'un système d'exploitation pour pouvoir utiliser l'ensemble des ressources physique et programme fourni par le drone et le microordinateur. Ces OS sont en général des noyaux Unix, modifiés pour travailler avec le FCU. Dans l'idéal, le système doit être le plus ouvert possible, afin de proposer un maximum de liberté au développeur

3.6.2. Couche ROS

Les systèmes d'exploitation embarqués dans ce genre d'appareil peuvent utiliser une couche de middleware que l'on nomme Robotic Operating System (ROS). Par exemple, un robot est constitué de plusieurs microordinateurs qui doivent parfois communiquer entre eux.

Il ne s'agit pas là d'un système d'exploitation, mais d'un framework léger qui fournit tous les outils et bibliothèques nécessaires à la création de robots et l'implémentation de comportements désirés dans ceux-ci. La communication peut aussi être codé en plusieurs langages, ROS étant basé principalement sur du C++ et Python. Dans notre cas, la couche ROS permettra de communiquer de manière indépendante sur le flight Controller. Il est toutefois à noter que ROS est un système Open source avec une licence BSD, qui permet l'utilisation et modification à des fins commerciales.

Nous n'avons pas poussé les recherches plus loin pour une éventuelle alternative à une couche ROS, car l'étudiant ne possède pas les compétences d'ingénierie nécessaire dans les temps impartis.

3.6.3. MavROS

MavROS est un package ROS qui permet l'échange de communication entre un Flight Controller utilisant le protocole MavLink et un CC ayant ROS installé dessus.

Nous pouvons ainsi résumer l'ensemble des énergies de cette manière :

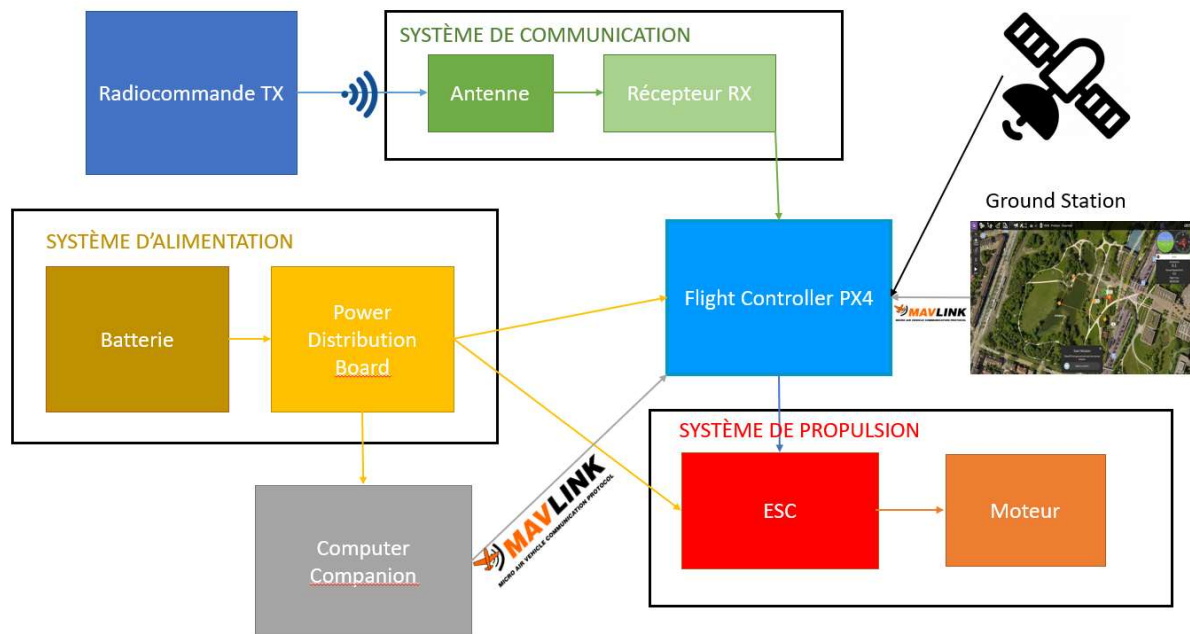


Figure 10: flux d'énergie et de données d'un drone récapitulé, schéma réalisé par l'auteur

4. État de l'art - OS pour Companion Computer

Avant d'établir une description des systèmes et un comparatif, nous voulons nous focaliser sur les drones commerciaux déjà existants qui proposent de la reconnaissance d'image et des solutions de drones agricoles.

Il s'agit plus d'un descriptif de ce qui se fait sur le marché, car ces drones ne sont que très difficilement modifiables et où le logiciel reste fermé et très peu paramétrable. De plus, Aero 41 cherche surtout à réaliser leur propre version de reconnaissance d'image, en utilisant le drone déjà en production et en respectant un budget limité, ce qui restreint grandement le panorama des solutions existantes.

Il existe actuellement beaucoup de drones équipant des systèmes permettant d'éviter des obstacles grâce à des capteurs ultrasons, mais il s'agit de solutions fermées et disponibles uniquement sur de drone de plus petite envergure telle que DJI Mavic Pro ou Phantom. Sur une grande majorité de drones de plus grosse envergure, il s'agit de drones déjà construits, mais dépourvu de caméra ou de capteur.

Nous pouvons cependant noter que la marque mondialement connue DJI propose déjà des solutions de drone d'agriculture pour le marché chinois et américain sans pour autant intégrer de solutions de reconnaissance d'images. Il embarque un radar, permettant de se tenir à une distance constante des cultures, en cas de variation de terrain. Il s'agit d'un drone équipé de tous les composants de base, plus un réservoir de produit sanitaire et d'un système de sulfatage.



Figure 11 : Drone DJI MG-1 récupéré sur : https://www4.djicdn.com/cms_uploads/ckeditor/pictures/1179/content_MG-1S_b.jpg

Il existe aussi différentes entreprises développant aussi leur propre drone d'agriculture. Cependant ceux-ci n'étant pas suisse il serait difficile de les importer, car les restrictions en matière de drone en Suisse dépassant les 30 kg obligent une homologation par l'Office Fédéral de l'Aviation Civile (OFAC), comme cité dans cet article de loi (art. 14a règle de l'air) :

« Les règles de l'air suivantes s'appliquent aux aéronefs sans occupants d'un poids supérieur à 30 kg, sauf en ce qui concerne les prescriptions sur les hauteurs minimales de vol:

- a. en premier lieu les règles figurant dans le règlement d'exécution (UE) n° 923/2012;
- b. à titre complémentaire, les règles figurant dans la présente ordonnance. »¹

Maintenant que nous comprenons mieux le fonctionnement d'un drone et ses modes de communications, nous comprenons que la répartition des tâches doit être claire pour chaque composant.

4.1. Analyse

Nous avons choisi un ensemble de trois solutions que nous allons analyser. Il s'agit tous de système compatible avec un Raspberry pi 3 qui est notre Companion Computer choisi pour des raisons économiques.

¹ <https://www.admin.ch/opc/fr/classified-compilation/19940351/index.html> consulté le 27.05.2012

Afin de simplifier les recherches dans un premier temps, nous avons décidé de nous orienter sur les solutions publiques trouvables sur Github. Par la suite, nous les comparerons avec une solution commerciale fournie pour la réalisation de ce travail. Nous nous sommes limités à 3 systèmes pour la durée de réalisation de ce travail, pour des raisons évidentes de temps. Notre analyse se fera via différents aspects :

L'installation, qui peut se faire via une carte SDD flashée, ou un OS et ajouter des composants comme ROS, OpenCV ...

La configuration sera l'une des parties de notre analyse qui aura le plus de points. Étant donné la limitation temporelle de notre analyse, une configuration trop longue et technique à établir bloque les étapes suivantes de notre projet. Quand nous parlons de configuration, il s'agit principalement de configurer le système comme l'accessibilité à la couche ROS, l'utilisation d'une caméra avec la possibilité d'utiliser OpenCV qui sera compilé sur le Raspberry.

Nous analyserons les avantages et désavantages de chaque système. Pour cela nous avons dressé un barème, ainsi que des étapes de test à réaliser sur simulateur ainsi qu'en temps réel.

L'OS open source que nous allons analyser dans un premier temps s'appelle Maverick. Il s'agit d'un OS embarqué dont le but premier est d'interfacer le CC avec le protocole Mavlink le plus efficacement possible.

4.2. Maverick

Maverick est une solution open source qui a pour but de faciliter le développement d'UAV en permettant d'utiliser le protocole MavLink.

4.2.1. Installation

L'installation avec un Raspberry est simple. Il s'agit d'une image Raspbian à flasher sur une carte SDD, puis allumer le microordinateur. Il est aussi possible de partir d'une installation récente de Debians et installer au fur et à mesure les composants de Maverick.

4.2.2. Configuration

La configuration est cependant plus difficile, car les commande pour régler le réseau proposé par Maverick, provoque l'arrêt de celui-ci. La configuration se fait par les commandes de Maverick, qui ajouterons les bonnes configurations du git. Durant notre installation, nous avons remarqué que les configurations du réseau dans les interfaces, etc/network/interface étaient réécrites, ce qui provoqué un arrêt du service réseau du Raspberry.

De plus pendant la configuration et l'utilisation du Raspberry, la chaleur du processeur montait très vite. Nous pensons qu'il s'agit d'une mauvaise gestion des ressources due à l'OS installé. La configuration du réseau restant impossible dû à la configuration de la commande maverick-configure, nous avons abandonné l'utilisation de ce système d'exploitation. Ensuite, pour configurer le reste des fonctionnalités comme l'accès à la couche ros, il est nécessaire de l'installer par différent module.

4.2.3. Fonctionnement

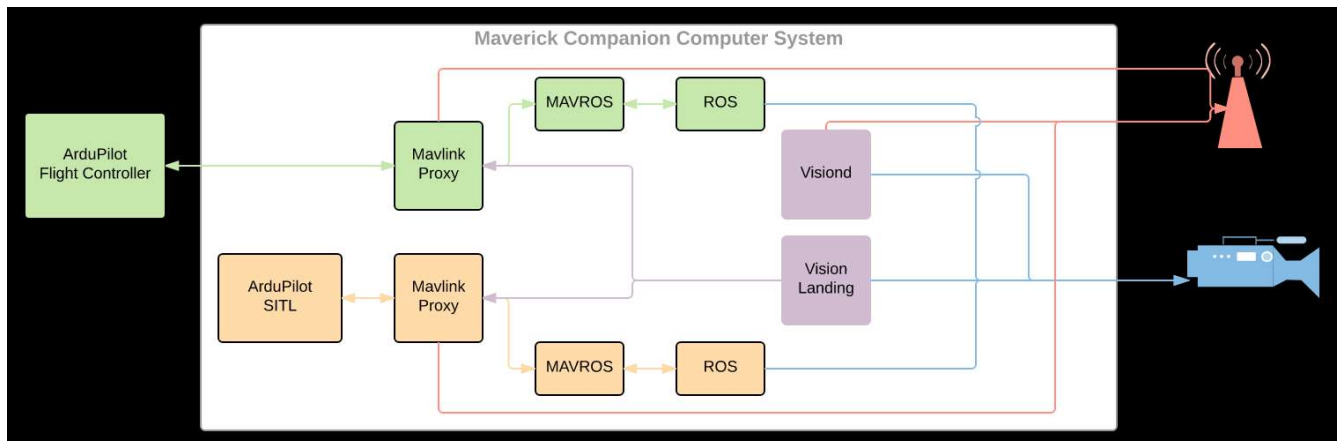


Figure 12 : Architecture Maverick, retrouvé sur : <https://goodrobots.github.io/maverick/current/>

Maverick est une couche qui vient se rajouter sur un système d'exploitation Unix. Le développement a été réalisé afin de rendre disponible Maverick sur les microordinateurs suivants : Raspeberry Pi, ODRROID, Intel Joule, et BeagleBone Black.

La couche ROS étant aussi intégrée à Maverick, il est possible de réaliser du développement sur la couche ros puis d'envoyer les instructions à l'autopilot.

La solution est développée de manière à automatiser le maximum possible les configurations, en se basant sur le système de management de configuration : Puppet.

Comme expliqué dans la documentation :

« Even though Maverick enables a rich set of functionalities out of the box, the main goal of the project is actually to provide a framework for automated system configuration, rather than

the features themselves. The actual functionality of the platform is just beginning and is at a very early stage. »²

Ainsi les fonctionnalités tel qu'elles sont développées permettent plutôt des bases afin de développer son système personnalisé, en établissant des configurations qui seront automatisées lors d'une plus grosse production. N'ayant pas pu réaliser une configuration complète de l'appareil, nous n'avons pas pu établir correctement les points positifs de ce système.

4.2.4. Problèmes

Durant notre installation, la configuration de cet OS a été compliquée. Ayant été développé sur plusieurs versions de Debian, il fallait configurer plusieurs fichiers qui doivent communiquer entre eux. Le problème étant que lors de la configuration de Maverick, celui-ci remodifie la configuration en écrasant ses propres fichiers.

L'ensemble de la documentation disponible pour Maverick est une page statique de présentation Github. Elle permet de comprendre relativement facilement l'ensemble de la solution, mais documente les premières versions. Mais cela n'apporte pas de réel aide en cas de problème et il faut ouvrir une issue sur Github.

Il n'y pas de compatibilité supportée avec le Flight Controller PX4.

4.2.5. Test en simulateur

Dû à l'incapacité d'installer cet OS, le test du simulateur et en temps réel, n'ont pas pu être accomplis.

4.2.6. Durabilité

Il est difficile d'estimer une certaine durabilité, quant à un projet en pleine croissance. Le fait est que l'équipe de développeurs derrière Maverick est assez petite et la majorité des push n'est réalisée que par un seul développeur, ne nous conforte pas vraiment sur l'avenir de ce projet.

4.2.7. Licence

En ce qui concerne la licence de Maverick, il s'agit d'un système open source sous licence du MIT. Elle est donc gratuite et permet de réaliser une utilisation commerciale.

² <https://goodrobots.github.io/maverick/current/#/> consulté le 13.05.2019

4.3. Dronecode/ArduPilot

Il ne s'agit pas d'une solution complète, mais un ensemble de propositions de systèmes à paramétrer, afin d'en faire une base stable de « bricolage ». Ce système est un ensemble d'application open source, et développé sans réelle garantie de fonctionnement optimal. De ce fait, beaucoup de configurations sont à faire afin de faire fonctionner l'ensemble.

Elle propose en l'occurrence d'installer uniquement une couche ROS et interagir via des scripts ou via une application externe. Il est possible de télécharger une image Ubuntu MATE déjà configurée avec ROS Kinetic installé, afin de simplifier la tâche de l'installation, proposée par UbiquityRobotics.

4.3.1. Installation

L'installation peut être réalisée facilement grâce à l'image téléchargeable sur Ubiquity Robotics³.

Il s'agit d'une image Ubuntu 16.0.4 adaptées pour Raspberry contenant déjà ROS Kinetic.

4.3.2. Fonctionnement

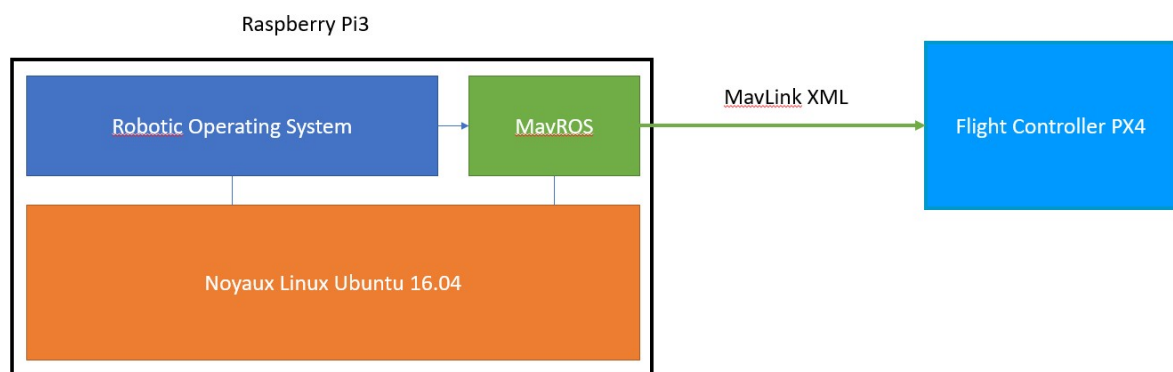


Figure 13: architecture dronecode, schéma de l'auteur

Le fonctionnement de cet ensemble repose sur un noyau Ubuntu de version 16.04, optimisé pour Raspberry, sur lequel vient se rajouter une couche ROS version Kinetic. Pour interagir avec le Flight

³ <https://downloads.ubiquityrobotics.com/pi.html>

Controller, nous devons rajouter un composant qui se nomme MavROS qui échangera des informations via le protocole MavLink.

4.3.3. Configuration

La principale difficulté de la configuration est de pouvoir configurer toutes les dépendances de manière à installer MAVROS.

Nous n'avons que très peu de connaissances sur les commandes make de Linux, qui permettent de déployer et compiler un programme. Dû à ces lacunes, les premiers make se sont soldés par des échecs, et ont coûté du temps. Lors de la compilation de MavROS, nous avons dû réaliser une partition swap, afin de pouvoir le compiler, faute de mémoire vive sur le Raspberry

4.3.4. Fonctionnalité

L'ensemble Dronecode n'a pas de réelle fonctionnalité, car il demande à l'utilisateur de pouvoir se débrouiller et bricoler des solutions ensemble. Le maître mot de ce système s'apparenterait à liberté d'utilisation.

4.3.5. Test en simulateur

Lors de la réalisation de test en simulateur, nous avons pu voir que l'appareil restait stable quand il passait en mode Offboard, et aucun arrêt de moteur n'a été signalé. Nous avons aussi pu lancer des scripts afin de pouvoir bouger et décoller le drone.

4.3.6. Test en temps réel

Nous n'avons pas réalisé de d'essais en extérieur.

4.3.7. Durabilité

La communauté derrière Dronecode est active. La quantité d'informations en ligne permet de pouvoir maintenir la solution longtemps, mais demande une attention extrême en cas de mise à jour. Une dépendance qui risque d'être mise à jour et c'est tout le système qui est corrompu. Cependant, sur le long terme, il est possible de devoir retoucher très souvent la configuration du système.

4.3.8. Licence

Il s'agit d'une licence Open-Source et permet une utilisation commerciale. Il n'y a donc pas de frais additionnel, si ce n'est les heures de configurations.

4.3.9. Résumé

L'ensemble Dronecode peut être considéré comme une base de bricolage sans frais. Il regroupe tous les outils pour pouvoir réaliser un prototype, et possiblement le faire entrer en production. En effet, l'ensemble est très ouvert et permet de travailler dessus sur toute les couches. Il adopte une politique vraiment orientée open source et la communauté derrière permet de se sortir de beaucoup de problèmes de configuration et de développement. De plus, la documentation de Dronecode permet de réaliser son propre simulateur.

4.4. FlytOS

FlytOS est proposé par la compagnie FlytBase. Cette entreprise propose une plateforme de développement compatible avec les principaux acteurs du marché de drone comme DJI, ainsi que les fabricants de microordinateurs tels que Intel, NVIDIA ou Raspberry. Elle est composée de 3 composantes principales :

- FlytOS : Système d'exploitation embarqué tournant sur un noyau Linux
- FlytCloud : L'ensemble des connexions possibles entre les drones et interfaces
- DevTools : différents outils de développement proposés.

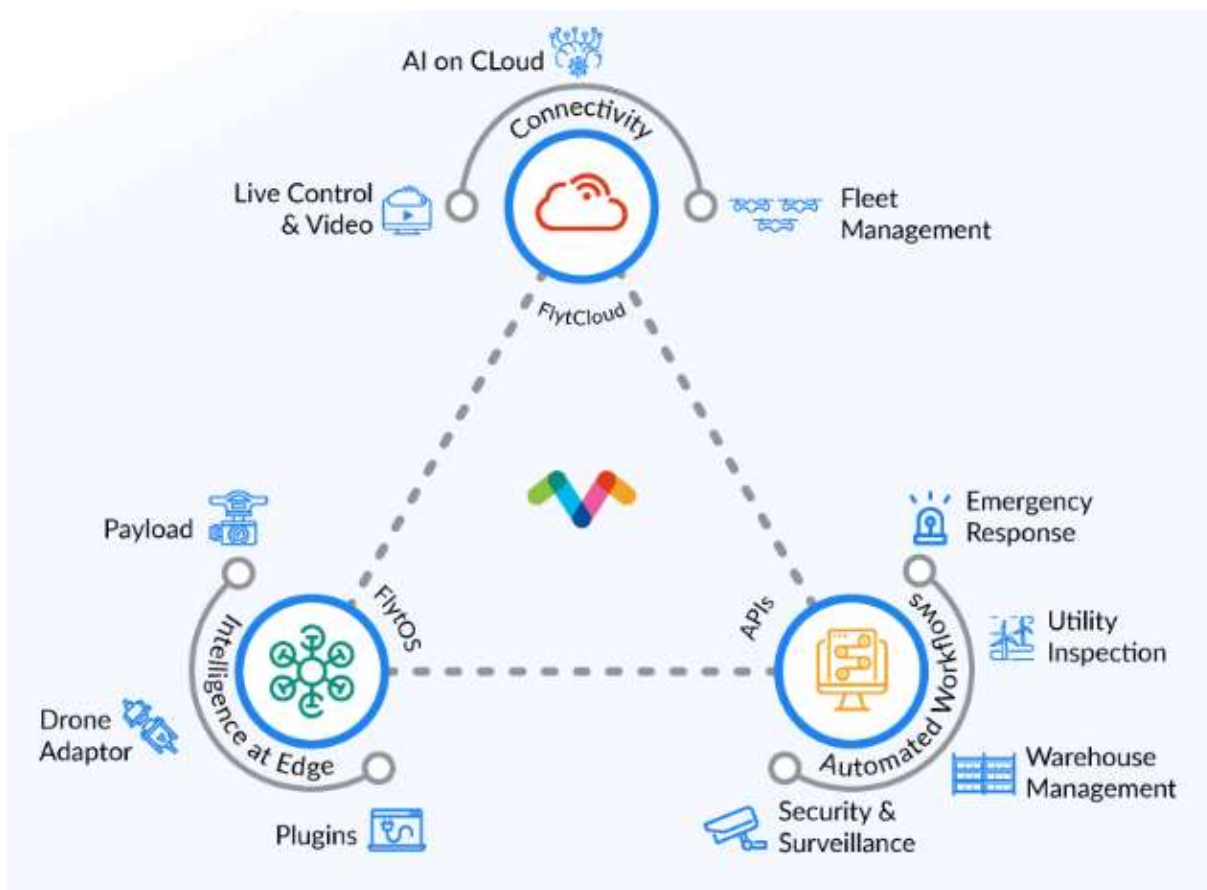


Figure 14 : Ensemble FlytOS, récupéré sur <https://flytbase.com>

4.4.1. Installation

Les étapes d'installation de FlytOS sont très simples et la documentation fournie est très claire. Une fois le compte créé, il suffit de flasher une carte SD pour installer l'OS dessus et brancher un

câble Ethernet. Il ne manque que la validation de la licence, réalisable sur FlytConsole, en se rendant sur l'adresse IP de la machine.

4.4.2. Fonctionnement

FlytOS est composé de plusieurs couches, comme démontré dans le schéma ci-dessous. La première couche est un système Linux de base. Dans ce cas, il s'agit de Raspbian, l'OS prévu pour être installé sur les Raspberrys. Par-dessus est installée une couche ROS, afin de communiquer avec les composants du drone. La couche FlytOS, qui nous intéresse, contient différentes API ROS, C++, Python, REST et Websocket.

Ces différentes API permettent de développer facilement des applications Web ou des scripts plus haut niveau. De plus, des modules ROS/Linux peuvent être intégrés très facilement

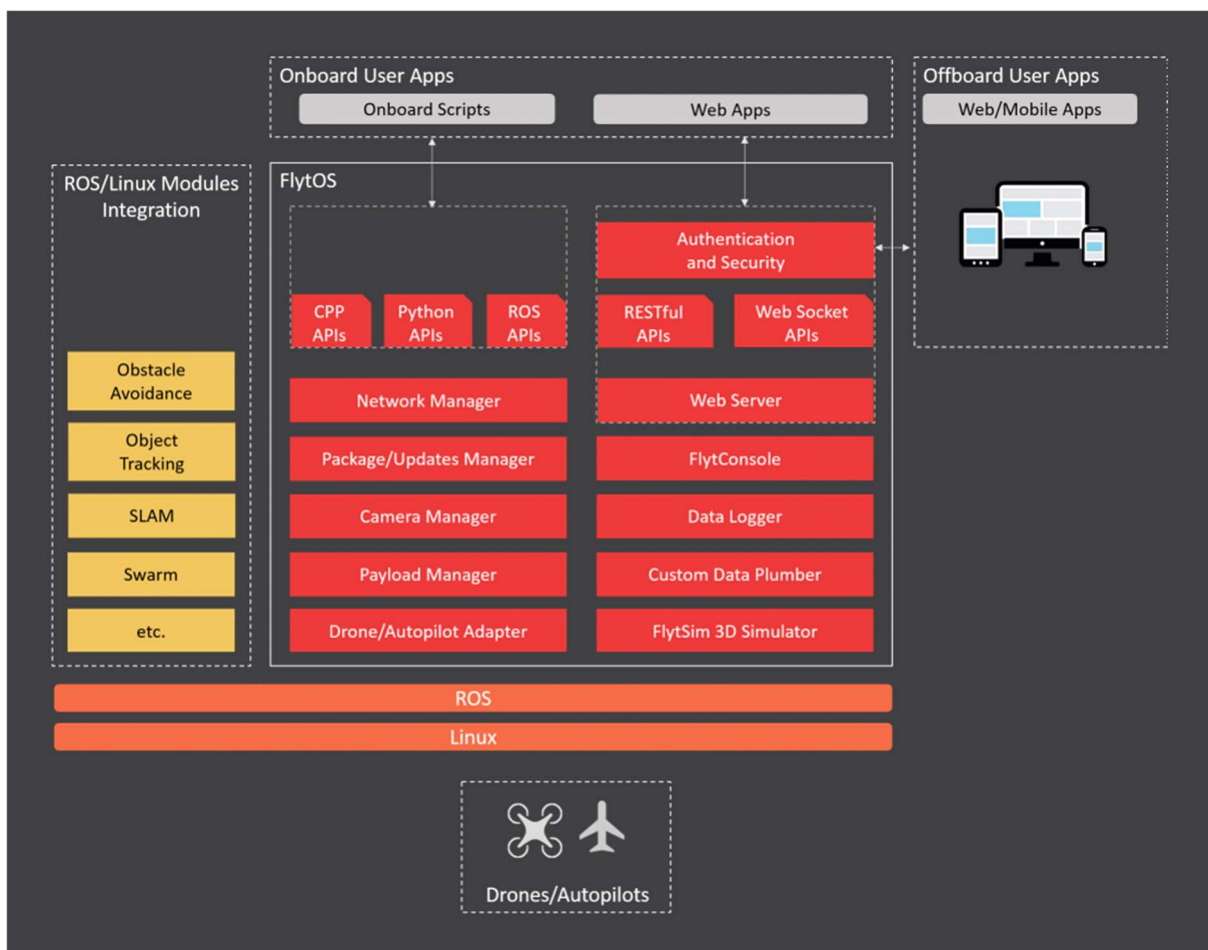


Figure 15 Architecture FlytOS, récupéré sur <https://flytbase.com>

4.4.3. Fonctionnalités

L'ensemble des fonctionnalités de FlytOS permet de faciliter le développement. L'environnement de développement intégré (IDE), utilisé avec FlytCloud rend le déploiement d'application d'une facilité déconcertante.

FlytOS permet au développeur de programmer ses propres applications, via la couche FlytOS, ou même Ros. Il s'agit d'un très gros point, car accéder à la couche ROS permet de communiquer directement avec un composant. Nous avons pu remarquer que communiquer via la couche FlytOS apporte l'avantage de pouvoir coder des applications en python directement et d'utiliser les bibliothèques permettant d'accéder au contrôle d'un drone. Dans notre cas, en cas de détection d'un objet, stopper le drone serait autant facile qu'une simple ligne de code.

Parmi les fonctionnalités existantes, FlytOS propose déjà un système de reconnaissance d'image qui permet de fixer un objet et se déplacer vers lui. Il est aussi très important de noter que le développement d'application peut être réalisé directement sur le CC via éditeur cloud disponible si l'on a accès à la fonctionnalité.

4.4.4. Test en simulateur

L'ensemble FlytBase propose un simulateur sur une image Docker permettant de simuler le comportement d'un drone. Il propose en plus un simulateur 3d tournant sur gazebo. Il n'est cependant pas très stable et le nombre d'images par secondes est beaucoup trop faible, cependant l'ensemble des communications à la plateforme web est fluide.

Lors de notre analyse, nous avons réalisé un script python basé sur les applications de démo proposé par l'entreprise. Le temps de réponse, presque immédiat, nous conforte à établir une possibilité d'utilisation de la couche FlytOS.

4.4.5. Test en temps réel

Lors des tests réalisés avec un drone réel, la première étape fut d'observer le comportement du drone quand le système prend le contrôle sur l'autopilote (mode Offboard). En l'occurrence, le mode actuel du drone pour réaliser du survol gardait le contrôle sur le comportement du drone, en plus d'écrire des logs dans le Raspberry. Malheureusement, l'horloge interne du Raspberry était incorrecte et de ce fait, la récolte des logs nous est apparue faussée.

4.4.6. FlytPod

Durant nos recherches sur le système FlytOS, nous avons remarqué un système embarqué nommé FlytPod, proposé par les développeurs de FlytOS. Il s'agit d'un microordinateur déjà configuré, qui embarque une technologie hybride et intègre un FCU dans l'appareil.

4.4.7. Durabilité

Le projet ayant pris beaucoup d'envergure depuis son développement permanent en 2016, avec des forums, une communauté, et une équipe présente sur divers forums de passionnés, FlytOS est une solution stable. Dans les prochaines années, il est fort probable que le système évolue, mais sans toutefois changer la connexion à la couche ROS

4.4.8. License

Le coût de la licence de production de FlytOS est de 299 \$ pour une machine. Il s'agit là du point le plus désavantageux du système, car les autres systèmes, étant des solutions open sources ne sont pas coûteuses et possèdent des licences open sources qui permettrait un développement commercial. Il s'agit là d'un point non négligeable, car il en résulte des coûts financiers bien supérieurs, pour avoir un ensemble de fonctionnalité qui est réalisable par les autres, mais qui demande un certain temps de développement.

5. Choix technologique

5.1. Restrictions matérielles

Lors de la réalisation de ce travail ainsi que les essais, certaines contraintes techniques et matérielles étaient imposées.

Companion Computer : Raspberry pi (version 3, puis 4)

Flight Controller : Pixhawk 4 ou 5

Firmware du FCU : PX4

Caméra : basée sur de l'USB, variable, dépendant des caméras disponibles compatibles avec le Raspberry

De plus, nous avons installé une machine virtuelle Ubuntu 18.10, et utilisé un Intel NUC, pour pouvoir tester les OS sur simulateur Gazebo qui n'est disponible que sur Linux.

5.2. Simulation

Afin de pouvoir tester au mieux les systèmes, nous avons décidé de choisir différents simulateurs : Gazebo et FlytSim. Le problème étant que ces simulateurs ne peuvent tourner uniquement sur des systèmes UNIX. Nous avons configuré une machine virtuelle Ubuntu afin de réaliser des tests dans le simulateur Gazebo, pour pouvoir utiliser ROS. Il est d'une part nécessaire de réaliser des tests sur simulateur, car un essai sur terrain peut être synonyme de crash et donc casse de matériel. De plus, un drone reste une machine dangereuse et les accidents sont une éventualité à ne pas négliger. Les simulations sont réalisées de manière différente :

Software In The Loop : Il s'agit de simuler l'ensemble des composants du drone à l'intérieur du simulateur. Ces essais peuvent être utiles afin de réaliser des tests de comportement.

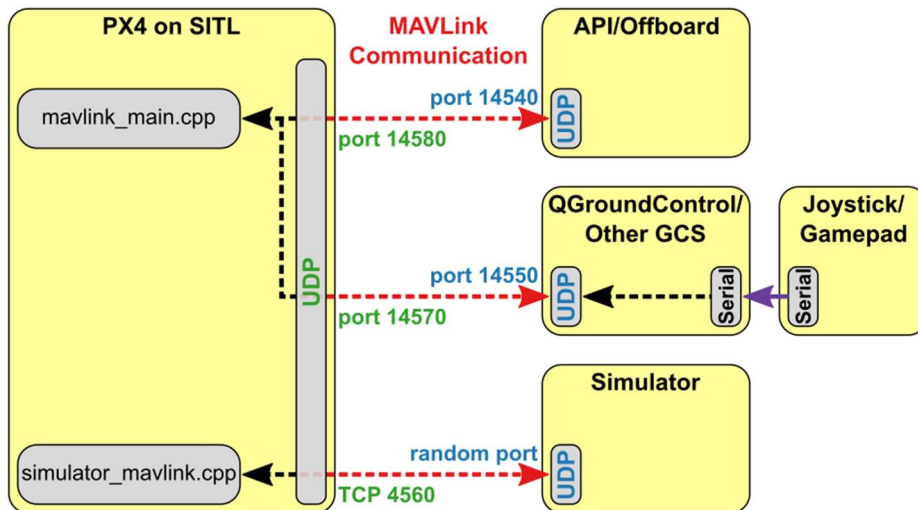


Figure 16: Schéma de simulation, récupéré sur : <https://dev.px4.io/v1.9.0/en/simulation/>

Hardware In The Loop : Il s'agit d'utiliser des composants physiques et externes à un seul système, comme un FCU et CC réel et de les immerger dans un monde virtuel afin de d'expérimenter ces composants et leurs programmes. Il n'est cependant pas possible de réaliser des tests sur un FCU connecté à un Raspberry. Une solution pour simuler ce type de connexion, serait compiler le firmware de l'autopilot sur le Raspberry qui sera branché en Ethernet et UDP. Il sera ainsi possible d'interagir via des scripts directement sur le firmware. Nous n'avons cependant pas tenté cette alternative.

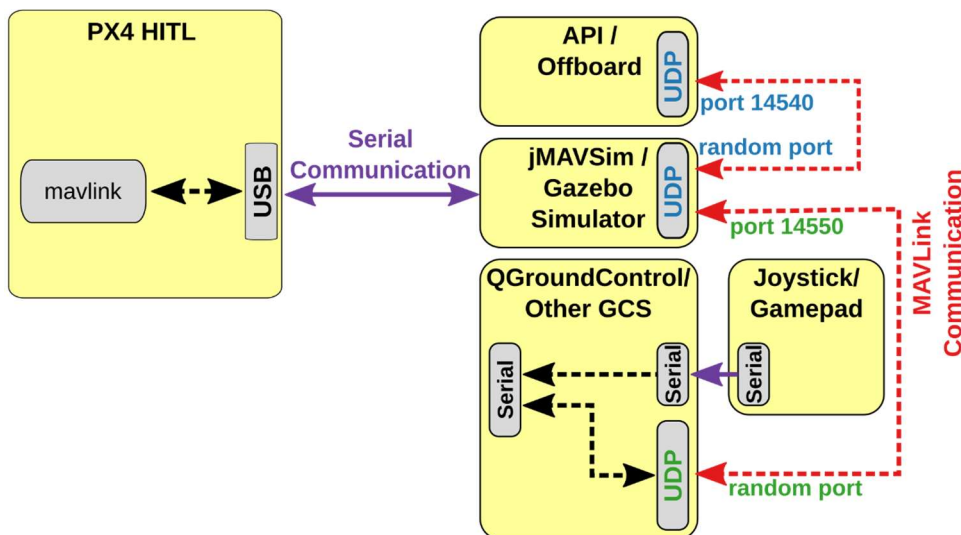


Figure 17: schéma de simulation HITL, récupéré sur <https://dev.px4.io/v1.9.0/en/simulation/hitl.html>

Cependant, les essais en extérieurs restent indispensables. Un simulateur ne peut pas reproduire à 100% la réalité. Les effets de lumière sur une caméra sont un bon exemple, car difficilement

reproductible sur un simulateur. Nous n'avons malheureusement pas pu réaliser beaucoup des tests réels, car ils demandent un temps considérable.

Une séquence de test lorsque l'appareil est allumé se déroule ainsi :

Nous nous connectons sur le FCU via un Ground Control, puis lançons une commande permettant d'armer les moteurs de l'appareil manuellement ou via un script. Nous lançons la commande de décollage, et vérifions que tous les indicateurs sont valides. En cas de problème, nous initialisons une séquence d'atterrissage. En vol, lorsque nous voulons laisser le Raspberry envoyer certaines commandes, nous passons en mode Offboard. Lorsque la mission est réalisée, nous faisons atterrir l'appareil.

Enfin, nous avons reçu un petit système permettant de tester les connexions du Raspberry sur le FCU.



Figure 18: prototype pour tester les connexions, image de l'auteur

L'ensemble de la configuration de ce petit appareil représentant un drone sans système de propulsion peut-être schématisé de la manière suivante.

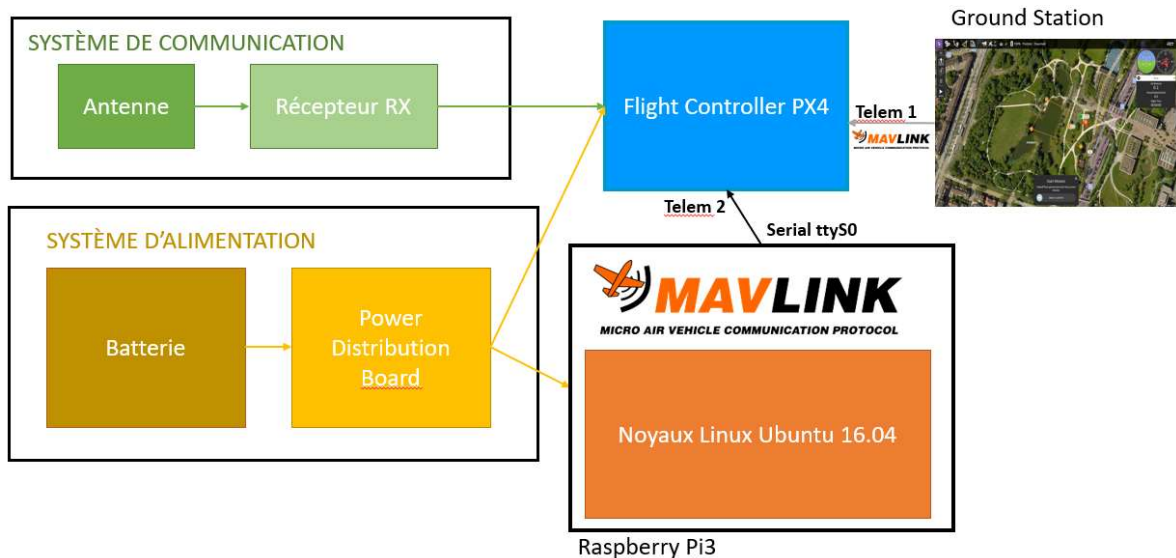


Figure 19: Schéma de fonctionnement, schéma réalisé par l'auteur

5.3. Problèmes rencontrés

Le problème de la configuration d'un simulateur est que certaines dépendances ne correspondaient pas à celle proposée par le script d'installation proposé pour chaque version d'Ubuntu, avec en plus des versions précises de ROS différents. De ce fait, nous avons dû réaliser une installation complètement différente et fastidieuse sur la machine virtuelle.

En essayant le simulateur fraîchement installé sur la version 18.10 d'Ubuntu, le simulateur se lançait, mais restait sur un écran noir.

Nous nous sommes aperçus que plus tard que cette version n'était pas supportée par Gazebo et ROS. Nous avons donc réalisé une installation sur la version 18.04. La procédure d'installation proposée sur Dronecode n'installait pas correctement les librairies geolib.

```
2019-07-10 17:36:16 (94.7 MB/s) - written to stdout [1018/1018]
bash: line 26: hash: geographiclib-get-geoids: not found
bash: line 30: hash: geographiclib-datasets-download: not found
OS not supported! Check GeographicLib page for supported OS and lib versions.
./ubuntu_sim_ros_melodic.sh: line 99: catkin: command not found
./ubuntu_sim_ros_melodic.sh: line 104: /home/oli/catkin_ws/devel/setup.bash: No such file or directory
./ubuntu_sim_ros_melodic.sh: line 109: cd: /home/oli/src/Firmware: No such file or directory
```

Figure 20: problème d'installation geolib, capture d'écran de l'auteur

Nous avons dû donc procéder à une installation « from scratch ». Les étapes furent donc les suivantes : installation de ROS, puis Gazebo avec les composants de compatibilité ROS et enfin les composants pour réaliser des tests SITL. Lors des tests, les images par secondes étaient beaucoup trop faibles, et le simulateur trop gourmand en mémoire graphique. Après plusieurs recherches, nous avons essayé de réaliser un « PCI passthrough » virtuel, qui permettrait d'utiliser la carte graphique de notre ordinateur, et de la dédier à la machine virtuelle. Cependant, cette manière de dédier un processeur graphique ne fonctionne que si l'hôte est un système UNIX. Dans notre cas, il est question d'un hôte Windows 10. Cette solution n'a donc pas fonctionné.

Pour résoudre ces problèmes de simulateur, nous avons proposé d'utiliser un Intel NUC avec Ubuntu 18.04 installé dessus.

Au total, la configuration des simulateurs a demandé 1 semaine et demie à plein temps. Il s'agissait malheureusement d'une étape nécessaire, sans quoi aucun test ne pouvait être réalisé en toute sécurité.

En plus de la machine virtuelle, nous avons dû configurer un docker FlytOS contenant un OS embarqué, et un système de simulateur portable (Gazebo). Durant la configuration, il nous était impossible de réaliser des tests sur gazebo, car soit le système ne démarrait pas, soit le simulateur plantait. Le reste du système restait cependant opérationnel et il était possible de créer des missions sur un drone.

5.4. Tableau de comparaison

Nous avons remarqué dans cette analyse que le panorama des Companions Computers de drone manque de développement collaboratif comme les FCU. Lorsque l'on parle de solution, voire de software pour un companions computer, il s'agit très souvent d'image de système d'exploitation à graver sur une carte SDD, et réaliser du développement dessus. Concevoir un développement sur ce genre d'appareil est difficile, car il demande des connaissances spécifiques au développement logiciel sur système Unix, ainsi que maîtriser la compilation d'application, et il est très difficile de mettre à jour le système sans que le tout « s'effondre ». L'exemple des dépendances python, qui parfois varient dans une mise à jour, entraîne la chute de tout le système qui a pu être mis en place. Lorsque de nouvelles fonctionnalités sont ajoutées, un utilisateur doit retélécharger la nouvelle image et recommencer une importante partie de développement.

Nous avons rédigé un tableau comparatif afin de pouvoir mieux visualiser les différents avantages et désavantages de ces systèmes.

Tableau 1: tableau comparatif des systèmes analysés, de l'auteur

Theme	FlytBase	Not	Maverick	Not	Dronocode/ ROS	Not	Pondération
Documentation	Présence de développeur sur différent forum, forum dédié à l'entreprise disponible avec un compte, documentation clair et abondante, plusieurs projet github avec readme et explication sur le code	5	page github.io comme doc / gitter pour poser des questions , réponse assez rapide	3	Ensemble de documentation, forum	3	2
OS-based	Debian, noyau Linux	x	Debian, noyau Linux	x	Ubuntu, noyau Linux	x	
Facilité de configuration	Image à flash sur une carte sd, réseau local activé, simple à configurer	5	Image raspbian fournie pour téléchargement. Configuration réseau très difficile: dès que la commande maverick-configure, le réseau ne fonctionne plus	5	Ensemble de dépendances à configurer, installation de mavlink et mavros qui ne s'installait pas correctement au départ	2	1
Fonctionnalité ?		5	Graphana , le reste marche pas, pas de URL pour trouver le lien des api	3	Noyau linux et couche ros, Fonctionnalité à rajouter par nous-même	2	2
Compatibilité simulateur	Simulateur gazebo ram beaucoup trop, par contre simuler une mission marche bien, possibilité de faire des scripts python pour réaliser une mission et visualisable sur la map dans le navigateur	3	Impossible configurer	0	SITL fonctionne avec ubuntu	4	1
Compatibilité FCU	Px4	5	compatible ardupilot, incompatible PX4, à la prochaine mise à jour e	0	Compatible avec px4	5	2
Facilité d'industrialisation	Deploiement d'application, normalement facile, n'a pas pu être réalisé par manque de temps	4	Pas de mise à jour récente, déconseillé vivement	0	Très Ouvert, demande juste des connaissance suffisante C++	5	1
License ? Et verification pour la prod	Oui, 299 par appareil	1	Open-source	5	Open-source	5	2
Durabilité	Dépendant des mise à jour du produit, communauté derrière et dev travaillant toujours sur le projet	4	Par mise à jour de l'os, dépens de la maintenance du projet --> dernier push date de 1 ans et pas vraiment d'information sur le continuité du projet	2	Dependance ros nécessaire, demande une veille technologique permanente, grosse communauté derrière ROS/px4 qui est un enorme point à prendre en considération	3	1
		4		2,4167		3,66667	

Comme nous pouvons le voir sur ce tableau comparatif, FlytOS devance bien les deux autres. Nous avons remarqué que la solution FlyOS offrait un ensemble de fonctionnalité et une stabilité devançant les autres, mais handicapés par un prix de licence beaucoup trop élevé. Les fonctionnalités proposées ne seraient, de plus, pas toutes utilisées.

Maverick aurait été une solution intéressante si le développement avait permis une utilisation compatible avec notre Flight Controller durant la réalisation de ce travail. La configuration réseau effectuée pendant la configuration ne nous permettait pas non plus d'utiliser le Raspberry à distance et demande d'être câblé en Ethernet pour pouvoir être utilisable. Le projet Maverick est prometteur, car il s'agit là du seul projet réellement open source de Companion Computer. Il est malheureusement assez incomplet comme cité précédemment. Il est annoncé que pour les prochaines versions, certaines fonctionnalités seraient développées, par exemple la compatibilité avec le PX4. Cependant, pour la réalisation de ce travail, les délais d'attente étant trop courts, Maverick ne sera pas choisi afin de réaliser un prototype.

Notre choix technologique se portera sur l'ensemble Dronocode, afin d'être plus libre lors du développement d'application. Étant donné que les ressources financières doivent être prises en considération, nous ne pouvons pas nous orienter sur FlytOS. L'ensemble des fonctionnalités aurait été très intéressant à utiliser, mais ne correspondrait qu'à quelques types de scénarios.

6. Etat de l'art - Machine Learning embarqué

Maintenant que nous avons choisi notre système sur lequel sera orienté notre développement de reconnaissance d'image et évitement d'obstacle, nous allons pointer déjà les solutions existantes.

En matière d'évitement d'obstacle, beaucoup d'entreprises proposent déjà des solutions afin de permettre à un appareil d'éviter des objets. Nous allons présenter rapidement ces projets déjà existants avant d'entrer un peu plus en détail sur le fonctionnement de ce genre de système

6.1. Solutions observées

6.1.1. Iris on board

Il existe une compagnie spécialisée dans la reconnaissance aérienne nommée Iris automation. Cette entreprise américaine a créé un système embarqué permettant de détecter et d'éviter des avions de petite envergure via une caméra branchée. Ce système nommé Casia, utilise des algorithmes de machine learning pour arriver à détecter un objet à une distance assez impressionnante, tout en les classifiant. Cela permet de gérer un nombre plus élevé de menaces dans les airs. Nous avons remarqué qu'il s'agissait principalement de drone volant à une altitude élevée. Dans notre cas plus spécifiquement, un drone ne dépasserait que très rarement la dizaine de mètres au-dessus du sol et les avions ne sont pas des obstacles que nous devrions prendre en considération.

L'entreprise ne propose pas réellement de prix quant à leur produit, mais demande plutôt un contact directement avec eux, afin de personnaliser le système et établir une offre.



Figure 21: système Casia, récupéré sur : [//www.irisonboard.com/casia/](http://www.irisonboard.com/casia/)

Le site web n'explique pas plus en profondeur en quoi est composé le système, si ce n'est les qualités de celui-ci. Comme cite dans l'article (Dukowitz, 2019) :

« Casia scans the environment once every 200ms, creating a real-time map and looking for any potential intruders, all onboard. Once it detects another aircraft, it tracks it, and uses machine learning to classify it, estimating the risk profile, sending an alert to the ground station, and triggering an automated maneuver to avoid the collision. »⁴

Nous comprenons un peu le système et sa manière de fonctionner en utilisant une caméra industrielle utilisée par le système. Le système est impressionnant par sa capacité à reconnaître des objets à très longue distance. Il aurait été intéressant de connaître le poids de ce système mais nous ne l'avons pas trouvé.

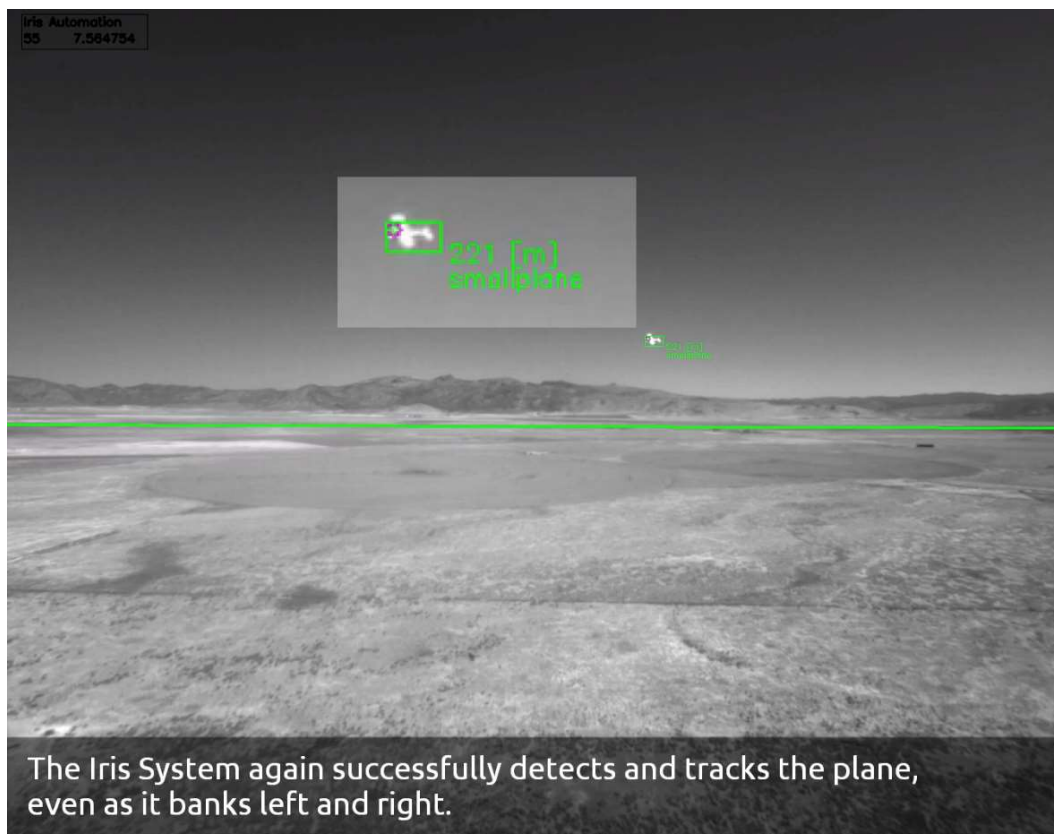


Figure 22: vision de la caméra Casia, capture d'écran de l'auteur, prise sur https://www.youtube.com/watch?time_continue=75&v=d4oM7n4mb00

6.1.2. MobileEYE

⁴ <https://uavcoach.com/iris-automation/> consulté le 26.06.2019

MobilEye est une compagnie spécialisée dans la reconnaissance d'image, associée à Intel. Elle produit des solutions anticollisions adaptables sur des automobiles récentes et de différentes dimensions. Sa gamme est composée de 2 produits : MobilEye Series et MobilEye shield +.

La version Series est adaptée pour analyser l'environnement dans le sens de la route et reconnaître des piétons, voitures et cycliste. Il permet d'alerter le conducteur en cas de risque de collisions avec un véhicule, de changement de ligne avec obstacle dans l'angle mort, avertit la présence de piétons ou cyclistes en journée et affiche la limitation de vitesse courante. Tout ce système se fait via une caméra installée de manière retrofit sur le rétroviseur, et d'ordinateur connecté à celle-ci. L'avertissement s'affiche sur un petit écran rond placé sur le tableau de bord.



Figure 23 Installation MobilEYE Series, capture d'écran de l'auteur, prise sur <https://www.mobileye.com/us/fleets/products/mobileye-6-collision-avoidance-system/>

La version Shield, est faite de la même manière, mais avec des caméras situées sur les angles des véhicules bien plus grands comme bus ou camion. Elle permet d'avertir au maximum le conducteur de présence d'obstacles dans les angles morts bien plus grands que sur un véhicule de tourisme.

Les procédés d'analyse de font bien entendu avec du machine learning et vision par ordinateur. La vision de la machine arrive à détecter le bitume, les feux de signalisation et panneau, l'arrière/avant et côté de voitures de manière séparée et bien entendu, les piétons et vélos.



Figure 24 Vision de la caméra MobileEYE, capture d'écran de la vidéo de promotion, prise sur <https://www.mobileye.com/us/fleets/technology/>

Il s'agit du leader en termes de reconnaissance et de système d'avertissement d'obstacle, mais fournit uniquement les véhicules terrestres. Il est cependant très intéressant de noter la vision que donne l'analyse d'image. De plus, nous pouvons remarquer que le système embarqué est assez petit.

6.1.3. Ainstein

Le système que nous allons analyser provient de la compagnie nommée Ainstein. Ils proposent un système de reconnaissance basé sur des radars et capteur plutôt que sur une caméra. Cette entreprise propose des systèmes adaptables sur différentes plateformes qu'elle soit aérienne ou terrestre. En utilisant du machine learning et de l'intelligence artificielle associée aux technologies radar et capteur, il est possible de reconnaître des obstacles, et permettre d'éviter ceux-ci. La compagnie s'est lancée dans 3 voies différentes : les drones, les véhicules autonomes et une gamme d'instruments Internet of Things.

Cette compagnie propose des modules radar et capteurs de tout type, sans aucun type de caméra. Chaque module peut être installé indépendamment des autres.

L'entreprise propose même une version adaptée du système sur des drones agricoles et permet le vol en terrain même accidenté. Selon le site web, le système envoie des ondes radar à haute fréquence directement vers le sol afin de détecter la disposition du terrain.

6.1.4. PX4 Obstacle Avoidance

Nous ne pouvons pas passer à côté de ce système, car il s'agit d'une solution open source, disponible sur Github par le développeur du PX4. Il s'agit d'un système pour PX4, capable de programmer un nouveau chemin en cas d'obstacle, en utilisant une caméra Intel RealSense. Il est composé d'un système optique permettant de comprendre la notion de profondeur. Ce système nécessite cependant un investissement financier particulier, car demande beaucoup de ressources de calcul. Il faut alors un micro-ordinateur beaucoup plus puissant, comme un Intel NUC ou NVIDIA Jetson TX2. Ce genre d'appareil est alors plus lourd qu'un Raspberry et beaucoup plus gourmand en énergie.

Nous pensons que ce système aurait été particulièrement intéressant à utiliser, mais demande des connaissances ROS et surtout le matériel nécessaire coûte particulièrement cher. Nous aimerions pouvoir réaliser un système à bas coût et plus léger.

7. Implémentation

Maintenant que nous avons pu voir les solutions déjà existantes dans le monde de l'automobile ou des drones, nous allons parler de notre méthodologie de travail concernant la partie machine learning. Elle diffère de notre méthodologie de travail pour cette thèse, car nous utiliserons Cross Industry Standard Process for Data Mining (ou CRISP-DM), qui est la référence dans les domaines de l'analyse de données.

Cette méthodologie permet de décrire les approches sur un projet d'analyse de données en le séparant en 6 phases majeures, qui peuvent être réalisées de manière itérative et revenir d'étape en étape. Nous allons réaliser notre thèse en accord avec ce schéma.

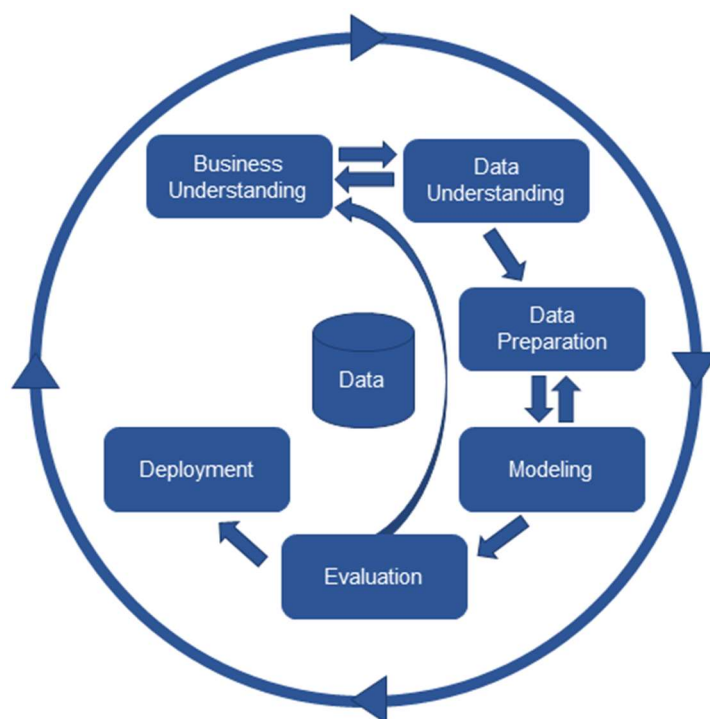


Figure 25: CRISP-DM schéma, <https://www.kdnuggets.com/wp-content/uploads/crisp-dm-4-problems-fig1.png>

7.1. Business Understanding + récolte

La réalisation des compréhensions des besoins métiers étant faite dans la problématique, nous allons passer à la récolte de données. Dans les étapes de compréhension, nous devons aussi établir les étapes de reconnaissance d'image, puis établir un processus accordé à cette reconnaissance. Nous devons soit éviter l'obstacle, soit nous arrêterons le drone et enverrons un signal au pilote afin qu'il puisse gérer la manière la plus appropriée d'agir en fonction de l'obstacle.

Nous avons défini le scénario suivant un faux positif, donc un objet qui n'est pas un arbre mais reconnu en tant que tel. Il devrait s'arrêter afin de ne pas risquer un crash.

Pour la réalisation de travaux de machine learning, nous avons besoin de données sous un format d'images. Afin de les récolter, nous devons nous rendre sur un terrain composé de vignes et d'arbres et qui puisse représenter les conditions les plus réelles en production d'une machine. Les arbres seront ainsi désignés comme les obstacles qu'un drone peut reconnaître et éviter. Bien évidemment, nous utiliserons un UAV afin de récolter les images.

Avant tout chose, nous avons dû définir le lieu de vol. L'environnement doit bien entendu contenir des arbres qui seront pris en photo et de la vigne. Il doit être suffisamment dégagé afin de pouvoir voler en toute sécurité. Avant de pouvoir voler, nous devons prendre en compte tous les éléments légaux qui incombent à un pilote de drone comme les zones protégées et les restrictions de l'espace aérien. Dans notre cas, nous récoltions des données en vol sur une zone restreinte aux modèles réduits, s'ils dépassaient les 150m au-dessus du sol. Notre drone ne dépassant pas les 15 mètres au-dessus du sol, nous n'avons pas besoin de faire de demande particulière. Enfin, nous prenons bien évidemment en compte les conditions météorologiques, afin d'éviter de voler en cas de mauvais temps ce qui peut être dangereux.

Lors du vol, nous avons utilisé un modèle de drone construit par le professeur Dominique Genoud, équipé d'un autopilot PX4, connecté à un Raspberry avec la caméra USB expliquée précédemment.



Figure 26: drone équipé d'un Raspberry, image de l'auteur

Sur ce drone, nous avons installé un Raspberry pi 3b utilisant un système Raspbian pour les premiers puis FlytOS, sur lequel nous avons installé une caméra à focus automatique. La caméra utilisée dans

un premier temps, est un appareil USB un autofocus et un capteur 5 Megapixel. Étant donné que sur un Raspberry, il n'y a pas de logiciel de capture d'image installée de base dessus, nous avons dû réaliser un script afin de pouvoir filmer avec la caméra.

Nous avons commencé par réaliser un script qui enregistrerait une vidéo en 1080p sur un Raspberry sur lequel FlytOS était installé. Nous pensions modifier l'API et le lien de monitoring, afin de pouvoir filmer en même temps que nous pouvions avoir une vision live de la caméra. Cependant, nous avons remarqué qu'il n'était pas possible d'accéder à une caméra avec 2 processus en même temps. Il est donc impossible de voir en live sur un navigateur et d'enregistrer en même temps. Afin de pallier ce problème, nous avons utilisé une image Raspbian et installé les modules pour gérer une connexion USB. Par la suite, nous avons eu quelques problèmes avec l'enregistrement, car la résolution étant trop élevée pour le logiciel FFMEG, les framerates ne dépassaient pas les 5 images par seconde, ce qui n'était pas viable. Nous avons alors baissé la résolution à 720p puis 480p. Il est possible de trouver le script versions 1 en annexe.

Lors de la première prise de données sur le terrain, nous avons réalisé que les vibrations provoquées par le drone faisaient trembler la caméra. Les vidéos enregistrées étaient de trop faible qualité et les vibrations ne donnaient que des images floues. Il était aussi difficile de récolter des images nettes d'obstacle. Le vent étant aussi très fort en après-midi, nous avons déplacé les récoltes aux matinées lors de vent moins fort.



Figure 27: image floue, image de l'auteur

Nous avons ainsi décidé qu'il serait plus facile de récolter des images, plutôt que des vidéos, afin de laisser plus de temps et frames au capteur afin de stabiliser l'image et le contraste. Nous avons ainsi revu notre script afin de pouvoir prendre des images en rafales, en essayant de minimiser au maximum l'écart entre les frames. Ce script a d'abord été en codé en script bash, puis remplacé par une version python permettant de réaliser des modifications avec OpenCV. Le principal problème de ce script est qu'il ne permettait pas de prendre des rafales à plus d'une image par seconde, car le processus d'écriture et d'accès à la caméra, combiné à l'utilisation de FFMPEG, bloque l'accès à la caméra.

Étant donné que les tâches de prise d'image dépassent le cadre de cette thèse et de nos connaissances, nous nous sommes accordés avec notre professeur, qu'il réaliserait un prototype afin d'accélérer la capture d'image et une automatisation de celle-ci lors de pression sur un bouton.

Nous sommes arrivés ainsi à des images de relativement bonne qualité d'arbres, cependant un grand nombre d'entre elles restaient trop « saccadées » pour servir d'image d'entraînement.



Figure 28: prise avec vibration et net, image de l'auteur

Enfin, Mr Jérôme Treboux nous a fourni des vidéos prises grâce à un drone DJI Mavic Pro d'arbres longeant des vignes, nous avons rédigé un script python trouvable en annexe afin d'extraire un frame chaque 10 secondes des films fournis.

7.2. Data understanding

L'étape de récolte de données étant terminée, nous avons analysé les images recueillies. Lors de consultation des images, nous nous sommes aperçus que la somme d'images résultantes des récoltes n'était pas suffisante. Cependant prendre des données sur terrain demande un temps considérable. Nous avons donc cherché des datasets gratuit d'arbre sur le web afin de pouvoir entraîner le mieux possible notre modèle.

Notre total d'image obtenu est de 273. Dans ce dataset, nous avons ainsi des arbres près de vignes, des arbres seuls dans un champ, des haies d'arbres ... Certaines de ces images étaient en différents formats (png, JPG, jpeg), qui seront modifiées dans la préparation des données

Les arbres trouvables dans notre cas pratique sont en général de couleur plus foncée que les vignes. Mais certaines séquences de pixel peuvent être difficiles à différencier l'arbre d'un plant de vigne. Cela va causer une dégradation de la qualité des prédictions d'un modèle.



Figure 29 : image du dataset d'entraînement, prise par Mr Jérôme Treboux

Comme nous pouvons le remarquer avec cette image, il est difficile de reconnaître une branche d'arbre, car les séquences de couleur des pixels encadrées en rouges ne varient pas suffisamment.

Le type d'arbre fourni va avoir une certaine importance sur la qualité d'une prédiction. Il faudrait ainsi fournir un grand nombre de types d'arbres comme des palmiers en passant par des sapins. Ces plantes possèdent des formes complètement différentes, mais qui doivent être considérées dans la même classe. De plus, il ne faut pas oublier que les couleurs et formes varient en fonction des saisons.

7.3. Data preparation

Nous avons traité et appliqué des filtres afin que l'étudiant puisse comprendre comment un algorithme de reconnaissance d'image aller traiter une image. Nous avons passé ainsi quelques jours sur l'utilisation et la compréhension d'OpenCV en utilisant les images récoltées.

7.4. Modelisation

Durant la réalisation de la modélisation, nous avons d'abord cherché sur le web l'état des connaissances actuelles sur l'analyse d'image, puis choisi 2 frameworks qui faciliteront le procédé d'entraînement, et analyserons l'implémentation d'un des modèles sur un Raspberry. Le problème principal est de pouvoir avoir un modèle léger qui puisse être intégrable à une reconnaissance en temps réel sur un Raspberry.

Nous avons analysé 2 frameworks permettant d'appliquer de la reconnaissance d'image.

7.4.1. Clarifai

Le premier que nous avons analysé s'appelle Clarifai. C'est un framework payant permettant de faciliter la reconnaissance d'image au moyen de divers outils. Des modèles déjà préentraînés sur différentes données sont disponibles sur la plateforme et accessibles avec une requête à leur API. Les requêtes possibles à l'api sont diverses et proposent un panorama très complet selon l'utilisation nécessaire. Par exemple ?

Leurs produits sont très intéressants à utiliser, mais le nombre d'appels est limité à 5000 opérations par mois. À partir de ces 5000 appels, il faudra déboursier un peu selon l'utilisation ou le choix du package.

Nous n'avons cependant pas intégré cette solution sur Raspberry, car l'accès au réseau est nécessaire pour pouvoir accéder à l'API de Clarifai. Étant donné que nous avons besoin de liberté dans l'utilisation du modèle, nous n'emploierons pas ces outils.

7.4.2. Darknet

Le second framework que nous avons décidé d'utiliser s'appelle Darknet. Il s'agit d'un framework permettant d'entraîner et utiliser un réseau de neurones convolutif (Convolutional Neuronal Network) nommé YOLO (You Only Look Once). Cet ensemble a été développé par Joseph Redmond en C et utilisant les technologies CUDA, permettant de procéder à l'entraînement et générer la reconnaissance en utilisant une carte graphique. Ces technologies sont très utiles afin de réaliser de la reconnaissance d'image en temps réel, une carte graphique étant plus efficace en calcul matriciel, comme réalisé sur une image.

Yolo est un état de l'art de réseau de neurones de convolution (ou CNN) populaire dans la reconnaissance d'image et plus précisément la détection d'objet. Le fonctionnement de ce réseau est explicite grâce à son nom, il observe une image une seule fois pour le test, ce qui permet de créer une prédiction via un contexte global de l'image et passe par un seul réseau pour réaliser une

prédiction. Cela permet d'augmenter considérablement la vitesse de détection d'un objet, mais augmente le risque d'erreur sur de petites cibles. Dans sa dernière version, Yolov3 utilise la méthode de clustering K-means afin de choisir le meilleur nombre de classe initiale.

Une version de ce réseau nous intéressé plus particulièrement. Il s'agit de yolov3-tiny, qui est une version très légère du réseau yolov3 de base. Étant donné les capacités limitées du Raspberry, ce réseau permet de réaliser de la reconnaissance en temps réel à environ à plus de 0.5 image par seconde, grâce à un réseau déjà préentraîner sur un dataset COCO.

Nous nous sommes basés sur ce tableau comparatif disponible sur le site web de Joseph Redmon, pour faire notre choix :

Tableau 2: tableau comparatif des performances de YOLO sur le dataset COCO, récupéré sur : <https://pjreddie.com/darknet/yolo/>

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Comme nous pouvons remarquer, le réseau de neurones étant le plus rapide est Tiny YOLO.

De plus, l'usage de Darknet et YOLO sont open source, et permettent aussi une utilisation à des fins commerciales.

L'utilisation de Darknet requiert cependant de connaître la manière de réaliser des builds, et n'est accessible uniquement que sur Linux à l'origine. Il existe cependant un fork Github permettant de build le projet sur Windows. Il faut au préalable installer et compiler OpenCV 4. De plus, afin d'utiliser toutes les performances du réseau et de Darknet, il est vivement conseillé d'installer CUDA. Un guide d'installation de Darknet, OpenCV et CUDA est disponible en annexe. Ce guide permet d'éviter les erreurs de compilation, qui peuvent prendre un temps considérable.

7.4.3. Réseau de neurones

Comme nous l'avons expliqué, nous utiliserons un certain type d'algorithmes de Deep Learning nommé réseaux de neurone. Nous expliquerons les notions de base de ce type d'algorithmes, mais n'irons pas plus en profondeur, car le niveau de théorisation de ce type de recherche dépasse le cadre de cette thèse de bachelor.

Un réseau de neurones artificiel (ANN) essaie d'imiter les connexions nerveuses d'un cerveau. Un cerveau est composé de milliards de neurones, interconnectés entre plusieurs millions d'entre eux. L'idée est d'appliquer ce système de connexion sur des neurones virtuels à des méthodes statistiques, afin de réaliser de l'apprentissage automatique.

Un neurone est un point de réseau. Il possède des informations entrantes (input) et une sortante (output), qui sont transmises via des connexions nommées weight (poids). Un neurone s'active lorsqu'il reçoit de l'information qui correspond à son activation, qui sera ensuite transformée en intensité pour le prochain neurone. Cette intensité est représentée sous forme de nombre réel normalisé. Un neurone va faire la somme des entrées reçues et ajouter son bias (biais), afin de savoir de quelle intensité il doit s'activer, ou non.

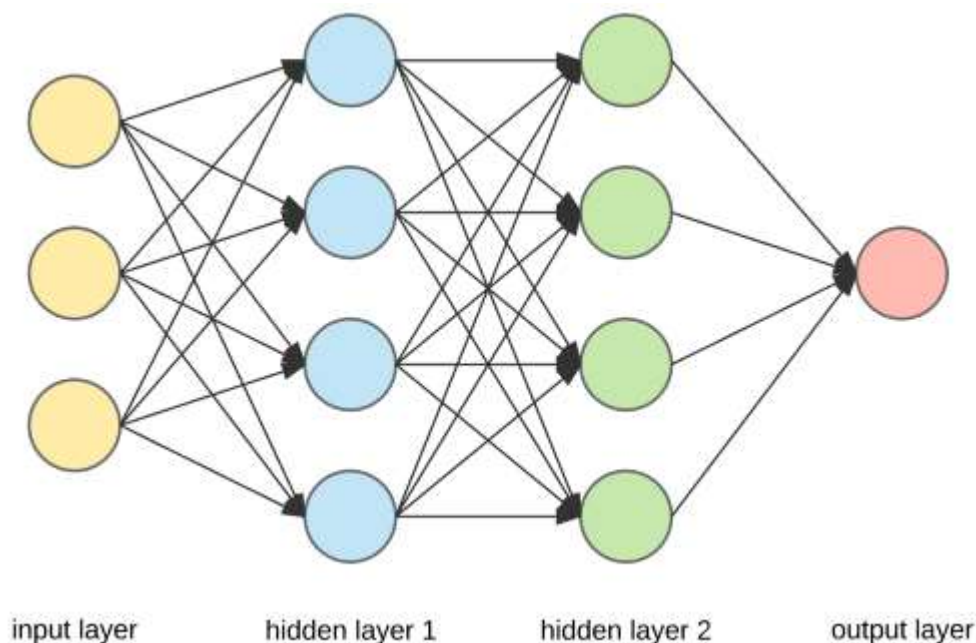


Figure 30: couche de neurones, récupéré sur

https://miro.medium.com/max/500/1*3fA77_mLNiJTSgZFhYnU0Q.png

Une couche (layer) de neurone est ensemble de neurones réalisant des opérations. On distingue plusieurs types de couches de neurones. La couche jaune ou Input layer est celle qui contient les données brutes, comme les couleurs d'un pixel. Les couches bleues et vertes, sont des couches cachées (hidden layers) et réalisent des opérations de calcul. La couche rouge ou Output layer est celle qui va produire le résultat de tous les inputs. Si le but d'utilisation du réseau est de la classification, il produira une prédiction de classe en output.

Un réseau de neurones est alors un ensemble couche de neurone interconnectés qui permet de fournir une opération de calcul. On peut prendre un réseau de neurones comme une fonction mathématique très complexe.

Dans notre cas, nous utiliserons un réseau de neurones convolutif. Celui-ci utilise une matrice de convolution, afin de traduire différents blocs de pixel d'une image. Cette matrice changera en fonction des couches de couleur. Cela permet de décomposer une image et de la réduire en petit bloc.

7.5. Data preparation (2)

Maintenant que notre framework a été choisi, nous avons dû préparer les images et leurs métadonnées. L'entraînement du réseau de neurones demande pour une donnée : en entrée, une image en JPG et un fichier texte de même nom que l'image, tous placés dans un seul dossier. Cela

permet de fournir la localisation de l'objet à apprendre, sa taille en float ainsi que la classe qui lui est attribué sous forme d'entier.

Format: <object-class> <x> <y> <width> <height>

Object-class représente la ligne de la classe d'objets, donnée dans un fichier nommé obj.names. Dans notre cas, nous avons qu'une seule classe dans ce fichier, tree qui est à la première ligne, donc 0. X, y, width et height sont des floats relatifs à la taille de l'image. La position est le centre du rectangle et non pas le coin supérieur gauche. Cela donne la possibilité de ne pas avoir besoin de redimensionner toutes les images à une taille prédéfinie, Darknet s'occupant de gérer les dimensions entrantes.



Figure 31: Arbre provenant de récolte de données et .txt correspondant, capture d'écran de l'auteur

Le calcul se fait ainsi $\langle x \rangle = \langle \text{absolute_x} \rangle / \langle \text{image_width} \rangle$ ou alors $\langle \text{height} \rangle = \langle \text{absolute_height} \rangle / \langle \text{image_height} \rangle$. Afin de ne pas se compliquer la tâche, il existe des petits programmes permettant de réaliser les annotations en cliquant sur le sommet supérieur de l'écran.

Nous avons annoté ainsi les 273 images. Sur certaines de ces images, nous avons décidé de prendre en compte uniquement les arbres les plus visibles et qui se démarque le plus dans une image. Le fait est que dans l'image qui suit, nous pouvons voir sur la gauche que des feuilles d'un autre arbre sont visibles. Il s'agit d'un réel problème, car il signifie la présence d'un obstacle, mais ne sera pas remarqué dû au manque de structure qui caractérise un arbre. Dans notre cas, nous avons décidé de prendre ce cas comme une anomalie, et annoté uniquement l'arbre principal. Ce choix s'explique en raison du manque de temps à disposition, car demanderait un effort supplémentaire dans l'annotation.

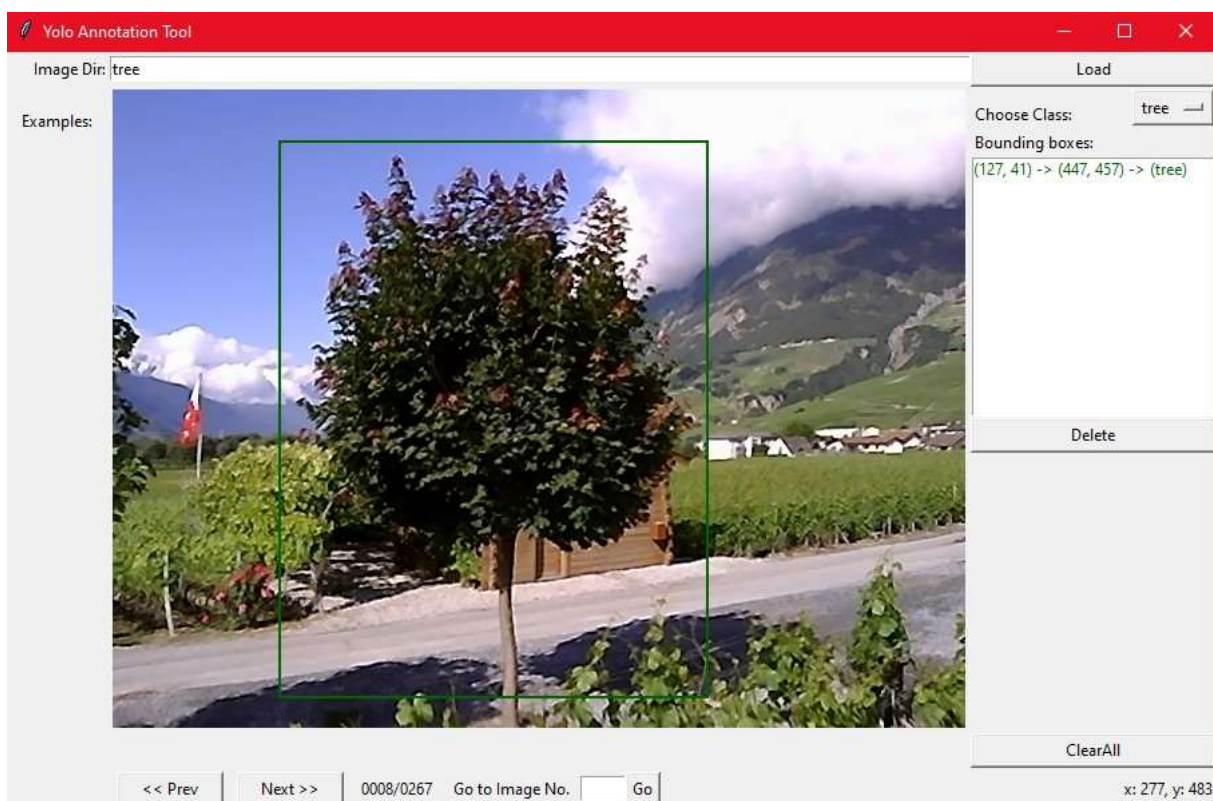
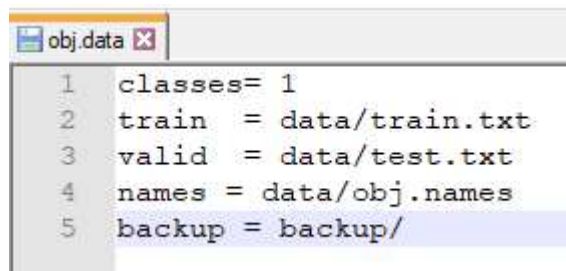


Figure 32: Annotation tool, capture d'écran de l'auteur

Comme chaque procédure de machine learning demande de séparer les données en entraînement et données de test, nous les avons séparées de manière aléatoire grâce un script python disponible en annexe, provenant d'un Github permettant d'apprendre à utiliser OpenCV. Le chemin système d'une image sera ainsi noté aléatoirement dans fichiers nommés train.txt ou test.txt.

Afin de pouvoir spécifier tous les chemins absolus des données d'entraînement et de test, le répertoire de sauvegarde du réseau pendant l'entraînement, et le nombre de classes et leurs noms, nous devons créer un fichier `obj.data` qui répertorie les variables. Voici la forme de ce fichier :



```
obj.data x
1 classes= 1
2 train = data/train.txt
3 valid = data/test.txt
4 names = data/obj.names
5 backup = backup/
```

Figure 33: structure du fichier `obj.data`, capture d'écran de l'auteur

7.6. Modélisation

Maintenant que nos données ont été annotées, et séparées, nous devons générer notre modèle avec Darknet. Nous utiliserons un réseau déjà préentraîné, qui ne contient pas notre classe d'objets. Ce procédé appelé transfert learning, permet d'augmenter la vitesse d'apprentissage du réseau, plutôt que de partir de zéro. Nous utiliserons ainsi un modèle préentraîné de `yolov3-tiny` disponible sur le site de Joseph Redmon⁵. Nous allons l'adapter pour pouvoir reconnaître qu'une seule classe d'arbre, afin de le rendre plus efficace. Nous utiliserons le même nombre de couche YOLO et de convolution.

Nous devons paramétrer le fichier de configuration du réseau. Il s'agit du fichier correspondant au réseau pré entraîné, suivi de l'extension `.cfg`. Il s'agit d'une étape très importante, car elle permet de régler l'apprentissage du réseau. Nous allons voir ensemble quelles sont les valeurs à appliquer, et pourquoi nous avons choisi ces valeurs.

7.6.1. Paramétrages

Dans la première section `[net]`, nous devons régler le batch et les subdivisions. Le batch indique le nombre d'images utilisées dans une itération d'apprentissage des weights du réseau. Le processus d'entraînement va mettre à jour à chaque itération les weights en se basant sur les erreurs faites sur les données d'entraînement. Prendre toutes les images pour entraîner à chaque itération est très

⁵ <https://pjreddie.com/media/files/yolov3-tiny.weights> récupéré le 08.07.2019

gourmand en ressource et n'est pas vraiment nécessaire. Dans notre cas, nous utiliserons un batch de 64 images.

Les subdivisions sont réglées de cette manière, car elles permettent de fractionner le batch afin de procéder à l'entraînement sans erreur de manque de mémoire. Dans notre cas, la machine entraînant le réseau de neurones n'ayant pas une carte graphique de plus de 1GB, nous nous sommes limités à 8 subdivisions.

Il est possible de régler en plus la taille de redimensionnement de l'image, mais nous l'avons laissé à 416 pixels, car en cas d'utilisation d'une image plus grande, nous avons une erreur de manque de mémoire. Nous avons laissé le reste des valeurs par défaut. Nous avons changé le nombre maximum de batch, et réglé à 2000, comme préciser sur le Github de la version Windows de Darknet. Nous avons aussi dû régler les steps au 80% et 90% du nombre de max_batches.

Il nous a fallu régler le nombre de classes à reconnaître dans les couches [yolo] à 1, puis changer les filtres des couches [convolutional] à 18, ce qui correspond à classes+5*3.

La dernière chose à faire était de placer tous les fichier et dossier de données à l'intérieur du dossier de Release de Darknet et l'entraîner lancer le script d'entraînement du réseau

```

920: 1.889689, 1.834875 avg loss, 0.000716 rate, 0.697000 seconds, 58880 images
Resizing
480 x 480
try to allocate additional workspace_size = 52.43 MB
CUDA allocate done!
loaded: 0.000000 seconds
v2 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.755891, GIOU: 0.752247), Class: 0.994655, Obj: 0.404378, No Obj: 0.002586, .SR: 1.000000, .75R: 0.400000, count: 10
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 23 Avg (IOU: 0.468767, GIOU: 0.373369), Class: 0.989332, Obj: 0.345886, No Obj: 0.003777, .SR: 0.400000, .75R: 0.400000, count: 10
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.764891, GIOU: 0.754522), Class: 0.992188, Obj: 0.562760, No Obj: 0.003183, .SR: 1.000000, .75R: 0.375000, count: 8
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 23 Avg (IOU: 0.640345, GIOU: 0.605298), Class: 0.983396, Obj: 0.386596, No Obj: 0.006590, .SR: 0.928571, .75R: 0.357143, count: 14
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.631709, GIOU: 0.672432), Class: 0.981096, Obj: 0.400135, No Obj: 0.003935, .SR: 0.908091, .75R: 0.227273, count: 11
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 23 Avg (IOU: 0.610904, GIOU: 0.597689), Class: 0.984729, Obj: 0.412395, No Obj: 0.008675, .SR: 0.833333, .75R: 0.166667, count: 12
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.706532, GIOU: 0.685214), Class: 0.993008, Obj: 0.458937, No Obj: 0.003227, .SR: 0.875000, .75R: 0.375000, count: 8
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 23 Avg (IOU: 0.574893, GIOU: 0.529655), Class: 0.984127, Obj: 0.416557, No Obj: 0.000782, .SR: 0.636364, .75R: 0.181818, count: 22
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.644769, GIOU: 0.617027), Class: 0.984186, Obj: 0.432118, No Obj: 0.002973, .SR: 0.800000, .75R: 0.380000, count: 10
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 23 Avg (IOU: 0.563711, GIOU: 0.518259), Class: 0.982419, Obj: 0.358825, No Obj: 0.000673, .SR: 0.687500, .75R: 0.187500, count: 16
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.675809, GIOU: 0.651134), Class: 0.995683, Obj: 0.647743, No Obj: 0.003366, .SR: 0.875000, .75R: 0.250000, count: 8
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 23 Avg (IOU: 0.541727, GIOU: 0.498947), Class: 0.987088, Obj: 0.311469, No Obj: 0.001013, .SR: 0.642857, .75R: 0.178571, count: 28

```

Figure 34: entraînement du réseau en cours, capture de l'écran de l'auteur.

Nous avons paramétré ce réseau de neurones et entraîné deux fois. Une fois en lançant le script normalement, puis une deuxième fois en calculant la Mean Average Precision via le flag -map de Darknet. Ce qui dans un cas concret devrait être supérieur afin de pouvoir affiner la précision du modèle en fournissant plus de données au fur et à mesure des itérations CRISP pour la modélisation. Nous ne disposons pas de suffisamment de temps pour entraîner plusieurs fois le réseau et affiner les paramètres. De ce fait, nous sommes passé directement à la partie de l'évaluation du modèle.

7.7. Evaluation

L'output donné par l'entraînement est un chart permettant de voir la loss du réseau, ainsi que 4 versions entrainées du neurone. Ces versions sont des sauvegardes du réseau à un certain nombre d'itérations, à la fin des 2000 itérations que nous avons spécifiées dans le max_batch, et la version avec la plus faible loss. Ces sauvegardes peuvent être comparées avec le chart afin d'établir quelle version utiliser.

La logique voudrait d'utiliser la version avec la plus faible moyenne de loss.

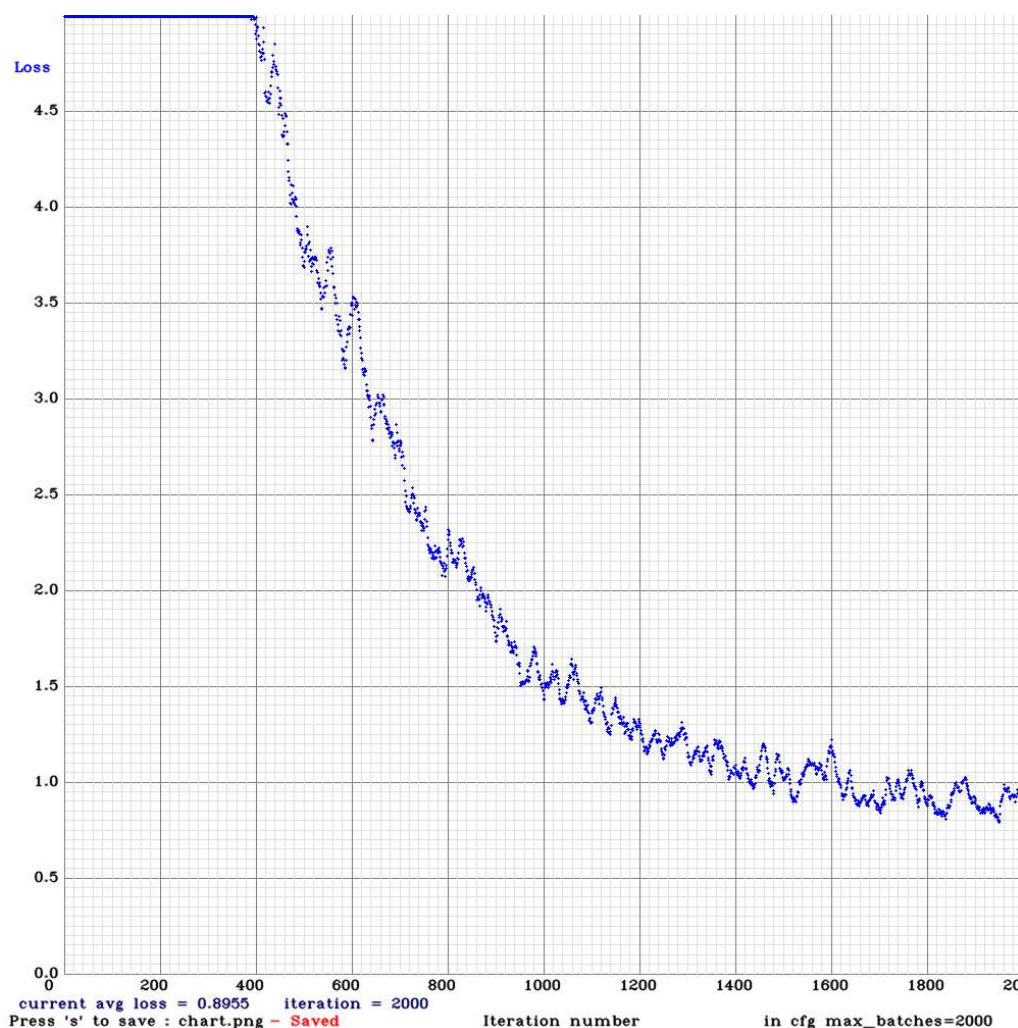


Figure 35: output de Darknet, image de l'auteur.

Nous pouvons remarquer dans ce graphique des faibles variations au fur et à mesure des itérations, chose typique d'un réseau de neurones dans une courbe de loss. La moyenne de loss, est de 0.8955 après 2000 itérations.

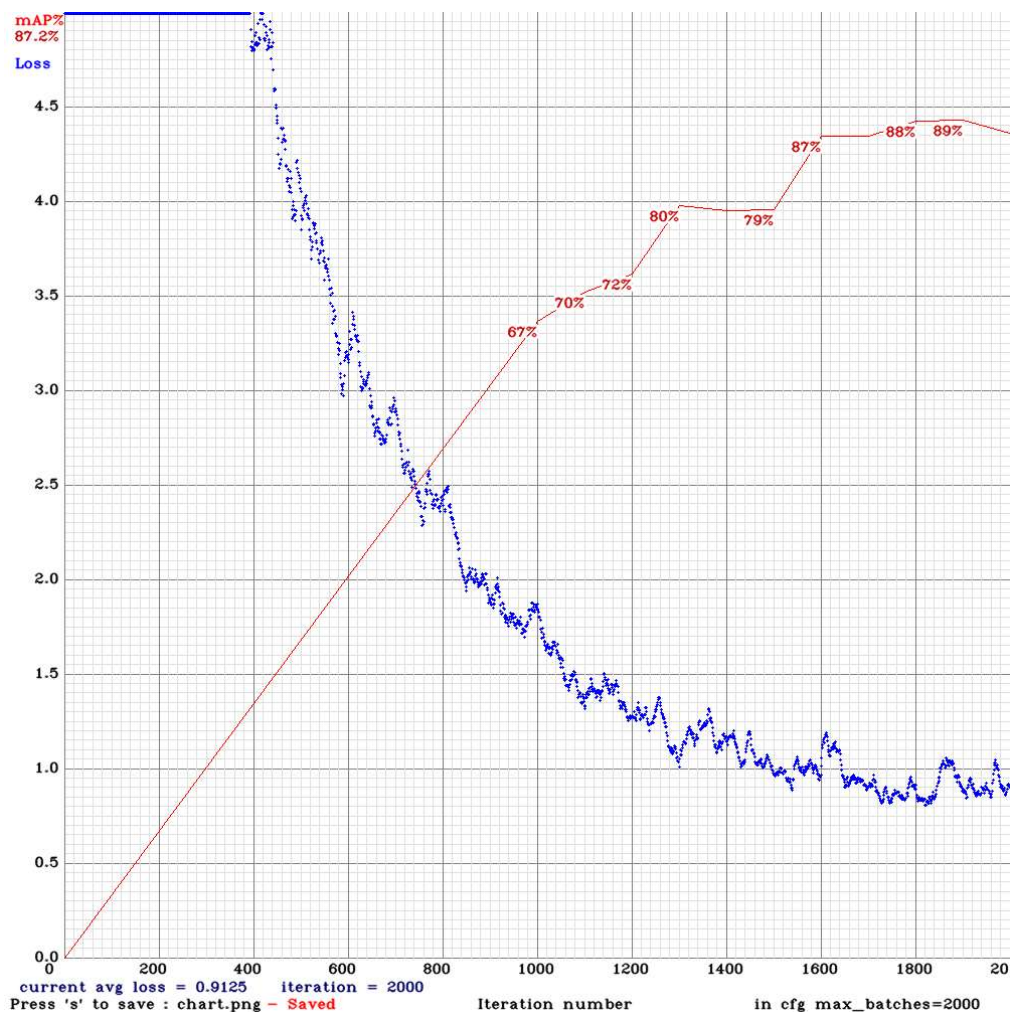


Figure 36: output de Darknet avec mAP, image de l'auteur

Dans ce deuxième entraînement, en ajoutant le paramètre `-map`, nous pouvons voir la courbe de Mean Average Precision augmentant jusqu'à 89 %.

Ensuite, afin de pouvoir tester la reconnaissance, nous avons utilisé ce code en python disponible sur le site `pyimagesearch`. Cette implémentation permet d'utiliser la version 4 d'OpenCV, ce qui nous offre la possibilité d'utiliser la librairie `dnn` et lire un réseau provenant de Darknet via la fonction `cv2.dnn.readNetFromDarknet`. Cette fonction n'est disponible qu'à partir de la version 3.4.2 d'OpenCV. De plus, il est pratique de pouvoir tester directement les différentes versions en spécifiant le dossier contenant le réseau et paramètre. N'ayant pas suffisamment de connaissance et de temps à disposition, nous avons gardé en grande partie ce code afin de tester notre réseau de neurones.

7.7.1. Code de reconnaissance d'image

Nous ne nous attarderons pas sur la première partie du code, car il s'agit simplement d'ajouter des arguments à la commande qui lance le script, puis de créer les chemins vers le réseau, sa configuration et les labels de class. Nous prenons une photo depuis la caméra, en ayant fixé les dimensions de celle-ci au préalable.

```
print("[INFO] loading YOLO from disk...")  
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
```

Figure 37: code de reconnaissance, capture de l'écran de l'auteur

Comme nous l'avons expliqué précédemment, nous utilisons les nouvelles options d'OpenCV et chargeons le réseau en mémoire.

```
while True:  
    # load our input image and grab its spatial dimensions  
    frame = get_image()  
    cv2.imwrite("frame.jpg", frame)  
    sleep(0.1)  
  
    image = cv2.imread("frame.jpg")  
    (H, W) = image.shape[:2]  
  
    # determine only the 'output' layer names that we need from YOLO  
    ln = net.getLayerNames()  
    ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```

Figure 38: code de reconnaissance, capture de l'écran de l'auteur

Dans une boucle infinie, nous écrivons sur le disque une image. Ensuite nous prenons l'image et récupérons les dimensions de celle-ci. Nous récupérons toutes les couches du réseau, et enlevons les couches au fur et à mesure pour reprendre uniquement celle d'output.

```
# construct a blob from the input image and then perform a forward
# associated probabilities
blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
                             swapRB=True, crop=False)
net.setInput(blob)
start = time.time()

layerOutputs = net.forward(ln)
end = time.time()
# show timing information on YOLO
print("[INFO] YOLO took {:.6f} seconds".format(end - start))
```

Figure 39: code de reconnaissance, capture de l'écran de l'auteur

Nous utilisons ensuite la fonction `blobFromImage` afin de retourner une même image après moyenne des couleurs, soustraction, normalisation et changement de couleurs. Les arguments à passer en paramètre sont les suivants : Image, mean, taille. Puis nous définissons d'autres valeurs par défaut.

Nous initialisons les tableaux d'objet détecté.

```
# loop over each of the layer outputs
for output in layerOutputs:
    # loop over each of the detections
    for detection in output:
        # extract the class ID and confidence (i.e., probability) of
        # the current object detection
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
```

Figure 40: code de reconnaissance, capture de l'écran de l'auteur

Nous récupérons la confiance en cherchant le score maximum dans la détection de toutes les couches d'output. Puis, dans cette même boucle for:


```
# filter out weak predictions by ensuring the detected
# probability is greater than the minimum probability
if confidence > args["confidence"]:
    # scale the bounding box coordinates back relative to the
    # size of the image, keeping in mind that YOLO actually
    # returns the center (x, y)-coordinates of the bounding
    # box followed by the boxes' width and height
    box = detection[0:4] * np.array([W, H, W, H])
    (centerX, centerY, width, height) = box.astype("int")
    # use the center (x, y)-coordinates to derive the top and
    # and left corner of the bounding box
    x = int(centerX - (width / 2))
    y = int(centerY - (height / 2))
    # update our list of bounding box coordinates, confidences,
    # and class IDs
    boxes.append([x, y, int(width), int(height)])
    confidences.append(float(confidence))
    classIDs.append(classID)
```

Figure 41: code de reconnaissance, capture de l'écran de l'auteur

Nous vérifions que la confiance de la détection est supérieure au seuil passé en paramètre. Puis récupérons les coordonnées du centre de la boîte de l'objet détecté. Nous calculons la dérivée de la position x y du sommet supérieur gauche pour la dessiner par après. Nous ajoutons les autres objets à notre liste d'objet détecté.

```
# apply non-maxima suppression to suppress weak, overlapping bounding
# boxes
idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
                        args["threshold"])
```

Figure 42: code de reconnaissance, capture de l'écran de l'auteur

Nous appliquons un algorithme de non-maxima suppression afin d'éviter que des boites se superpose et de récupérer un nombre d'objets limités.

```
# ensure at least one detection exists
if len(idxs) > 0:
    # loop over the indexes we are keeping
    for i in idxs.flatten():
        # extract the bounding box coordinates
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        # draw a bounding box rectangle and label on the image
        color = [int(c) for c in COLORS[classIDs[i]]]
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
        cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, color, 2)
```

Figure 43: code de reconnaissance, capture de l'écran de l'auteur

Dans cette avant-dernière partie du code, nous bouclons sur tous les objets détectés s'il en existe, puis dessinons les rectangles sur cette image.

```
# show the output image
cv2.imwrite("prediction.png", image)
cv2.imshow('pred', image)
key = cv2.waitKey(5)
```

Figure 44: code de reconnaissance, capture de l'écran de l'auteur

Enfin, nous sauvegardons l'image de prédiction et l'affichons à l'écran.

Nous avons utilisé ce même code pour réaliser des tests en « temps réel » sur Raspberry. Il ne s'agit pas effectivement de temps réel, car, nous affichons une seule frame capturée à la fois. La fréquence des images affichées après traitement sur Raspberry est d'une image chaque 1,9 seconde environ. Ce qui est un résultat plutôt convenable contenu de l'environnement hardware restreint du microordinateur, et des connexions simultanées avec le FCU ralentissant le système.

```
(yolo) ubuntu@ubiquityrobot:~/yolo/darknet-nnpack$ sudo python3 reco.py -y yolo-custom
[INFO] loading YOLO from disk...
[INFO] YOLO took 2.375612 seconds
[INFO] YOLO took 1.870391 seconds
[INFO] YOLO took 1.821895 seconds
[INFO] YOLO took 1.929254 seconds
[INFO] YOLO took 1.929497 seconds
[INFO] YOLO took 1.840114 seconds
[INFO] YOLO took 1.928347 seconds
[INFO] YOLO took 1.834250 seconds
[INFO] YOLO took 1.914022 seconds
[INFO] YOLO took 1.870688 seconds
```

Figure 45: reconnaissance d'image, capture d'écran de l'auteur

Nous avons décidé avec les partenaires du projet, que ce résultat suffisait afin de continuer vers l'étape de déploiement.

7.8. Déploiement

La phase de déploiement de notre cas consiste à implémenter la reconnaissance d'un arbre à un script permettant de contrôler le drone depuis le Raspberry. Pour ce faire nous avons utilisé l'environnement décidé dans la partie choix technologique qui sont les recommandations Dronecode avec une couche ROS.

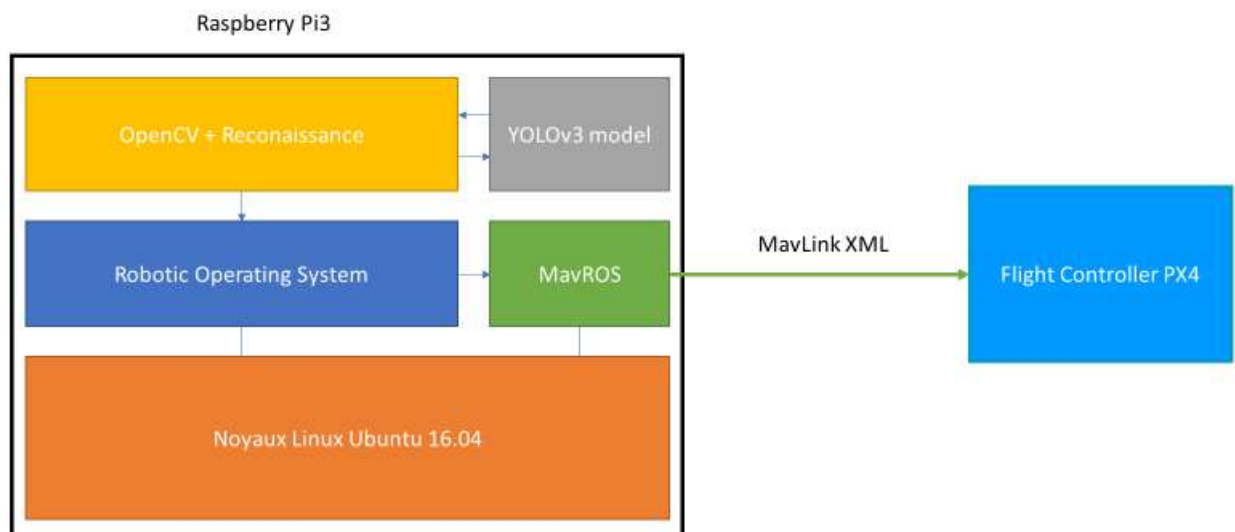


Figure 46: structure du Raspberry, schéma de l'auteur

Dans un premier temps, il était convenu que lorsque le mode Offboard était activé, nous lancions un script permettant au drone d'avancer en direction de l'arbre de manière automatisée. Lorsque le rectangle de reconnaissance sorti par la fonction `dnn.readfromdarknet` et `layer.output`, avait une aire supérieure à 20 % de l'aire de l'image, le drone devait s'arrêter et faire du surplace.

Nous avons commencé à réaliser le développement avec la couche ros, et utilisons MavROS pour communiquer avec le FCU. Nous avons revu les tutoriels ROS, puis modifié des tutoriels pris sur internet. Nous n'avons cependant pas anticipé la complexité de coder en C++, ce qui est nécessaire pour réaliser les scripts ROS. Après une journée à essayer de réaliser un seul script permettant de déplacer sur différents points, nous n'avons pas réussi à le compiler. Nous avons donc tenté de chercher une alternative à l'utilisation de ROS, car nos connaissances ne sont pas assez poussées dans ce langage.

7.8.1. MavProxy DroneKit

Nous avons trouvé une alternative pour communiquer via le protocole MavLink en utilisant les bibliothèques python MavProxy et DroneKit. MavProxy est un software permettant de se connecter au FCU facilement.

```
ubuntu@ubiquityrobot:~$ sudo mavproxy.py --master=/dev/ttyAMA0 --baudrate=921600
Connect /dev/ttyAMA0 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from /dev/ttyAMA0
MAV> online system 1
MANUAL> Mode MANUAL
fence breach
Received 749 parameters
Saved 750 parameters to mav.parm
MAV>
```

Figure 47: connexion au FCU via mavProxy, capture d'écran de l'auteur

Combiné à Dronekit, une bibliothèque python permettant de développer des scripts python passant par le protocole MavLink, il est possible de réaliser facilement des scripts. Cette API est beaucoup plus simple à coder, et grâce au listener de channel permet de programmer rapidement des applications et scripts fonctionnels.

Après recherche, nous nous sommes aperçus que la couche ROS n'était pas forcément nécessaire, et nous n'avons besoin que du protocole MavLink associé aux bibliothèques. De cette manière, le schéma ne serait que très légèrement modifié et remplacerait Ros et MavROS.

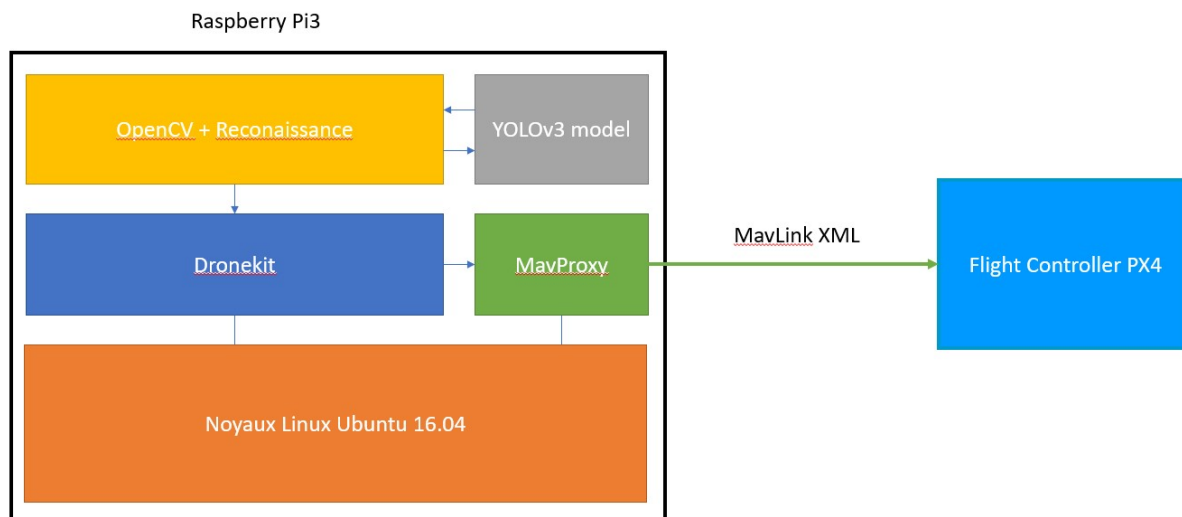


Figure 48: structure du Raspberry Version 2, schéma de l'auteur

De cette manière nous avons écrit un script permettant de lancer la reconnaissance d'image en écoutant une channel programmée à cet effet et passer en mode Loiter en cas d'arbre trop proche. La séquence est ainsi la suivante.

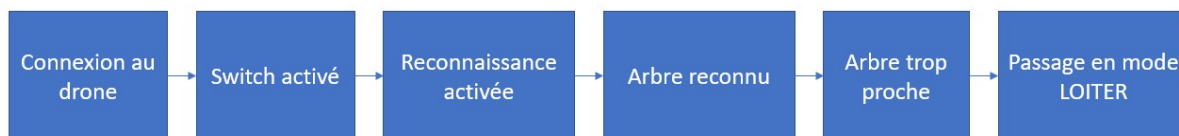


Figure 49: Schéma de séquence du code, schéma de l'auteur

Au lancement du script, nous allons nous connecter au drone, via un port serial. Quand un switch est activé, nous lançons la reconnaissance d'image. Lorsqu'un arbre est reconnu, nous vérifions sa taille par rapport à l'image. Si la taille est trop élevée, cela signifie qu'un arbre est trop proche et nous passons ainsi le mode Loiter pour faire du surplace. Le pilote reprend les commandes et évite l'arbre.

Nous allons parler de quelques points intéressants du code développé.

7.8.2. Code de changement de mode

Dans un premier temps, nous avons d'abord commencé par écrire un code écoutant le changement de switch, qui ensuite changera le mode en vol. Le code est assez simple. La première partie permet de se connecter en serial sur le FCU via MavProxy.

```
from dronekit import connect, Command, LocationGlobal, VehicleMode
from pymavlink import mavutil
import time, sys, argparse, math

GO_SWITCH = False

try:
    # Connect to the Vehicle
    print("Connecting to vehicle on: ttyS0 ")
    # Connect to the Vehicle (in this case a UDP endpoint)
    vehicle = connect('/dev/ttyS0', wait_ready=True, baud=921600)
    # Bad TCP connection
except socket.error:
    print("No vehicle exists!")

    # Bad TTY connection
except exceptions.OSError as e:
    print("No serial exists!")

    # API Error
except dronekit.APIException:
    print("Timeout!")

    # Other error
except:
    print("Some other error!")
```

Figure 50: Code de connexion au FCU, capture d'écran de l'auteur

Nous essayons de nous connecter à l'adresse du drone, en l'occurrence le port serial ttyS0 du Raspberry avec une vitesse de 921600 bits par seconde. En cas de problème et d'exceptions, nous les affichons sur la console. Nous affichons ensuite des informations basiques, comme le type de véhicule, le statut système et l'affichage du GPS.

```
@vehicle.on_message('RC_CHANNELS')
def listener(vehicle, name, message):
    global GO_SWITCH
    if message.chan12_raw > 1500:
        GO_SWITCH = True
    else:
        GO_SWITCH = False
```

Figure 51: listener de la channel 12, capture d'écran de l'auteur.

Le listener va écouter toute les channels radio, puis lors d'un changement dépassant la valeur 1500, il changera la variable globale GO_SWITCH.

```
while True:
    print("wait for switch")
    while not GO_SWITCH:
        print(".")
        time.sleep(1)

    print("LOITER switch detected")

    vehicle.mode = VehicleMode("LOITER")
    print("Loiter mode success")
    time.sleep(20)
    GO_SWITCH = False
    print("quit...")
```

Figure 52: logique du changement de mode, capture d'écran de l'auteur

Dans la dernière partie du code, nous attendons que la variable GO_SWITCH soit passée à True, puis passons le véhicule en mode Loiter. Certains modes ne sont pas compatibles avec le PX4, comme Auto_loiter.

Pour donner suite au test sur notre drone de simulation, nous avons constaté que le mode était effectivement changé sur notre ground control, et passe en mode HOLD, équivalent de Loiter. Aucune erreur ne donnait lieu à un certain changement dans le code. Nous avons ensuite réalisé un test sur un drone physique en vol, et aucun arrêt de moteur ou exception n'a eu lieu. Le mode Loiter, a bien entendu était activé en vol.

7.8.3. Code de reconnaissance d'image puis changement de mode

À la suite de la réussite de notre script, nous l'avons modifié légèrement pour intégrer un stop lorsqu'un arbre reconnu dépasse un ratio de la taille de l'image. Ainsi lors de l'exécution du script, et que le switch de la channel 12 est activée, nous lançons la reconnaissance d'image. Toutes les prédictions d'objet sont sauvegardées dans un dossier nommé « pred ».

```
print("Switch detected, begin recognition")

if treeRecognition.activateRecognitionFilm("volo-custom", 0.08, 0.2):
    vehicle.mode = VehicleMode("LOITER")
    print("Loiter mode success")
    time.sleep(10)
```

Figure 53: activation de la reconnaissance d'image, capture d'écran de l'auteur

La fonction `activateRecognitionFilm` de `treeRecognition` retourne un boolean. Nous passons en paramètre le seuil d'acceptation de reconnaissance d'un arbre. Les images en vol étant souvent floues, nous avons volontairement baissé très bas le seuil. Le cas d'un stop en cas de faux positif étant moins grave, nous préférons que le drone s'arrête s'il reconnaît quelque chose qui n'est pas un arbre. Le deuxième étant le non-maxima suppression comme expliquée précédemment. Lorsqu'un arbre est reconnu et est trop grand, nous recevons un boolean qui permet de rentrer dans l'if. Le mode est ainsi changé et attendons 10 secondes.

```
# calculate ratio of box
r = (w*h) / (H*W)

print("w: " + str(w) + " h: " + str(h) + " R : " + str(r))

cv2.imwrite("./pred/prediction"+str(r)+".png", image)

if r > 0.2:
    print("SOMETHING FOUND STOP")
    # show the output image
    cv2.imwrite("predictionSTOP.png", image)
    return True
```

Figure 54: Code du calcul du ratio, capture d'écran de l'auteur

La principale différence avec le code que nous avons expliqué pour la reconnaissance est cette partie placée dans la vérification qu'un objet a été détecté. Nous calculons le ratio du box de détection, en fonction de la taille de l'image donnée en output :

$$\text{Soit } (\text{widthBox} * \text{heightBox}) / (\text{HeightImage} * \text{WidthImage})$$

Nous sauvegardons ensuite la prédiction, puis vérifions que la box ne dépasse pas la ration de 20% de l'image, sinon nous enregistrons l'image qui déclenche le stop et retournons un True.

Le script se termine dès cette étape de reconnaissance passée.

7.8.4. Tests effectués

Afin de valider le prototype, nous nous sommes rendus dans un vignoble à proximité de Chamoson où une série de 3 arbres ont été utilisés pour la récolte de données. De cette manière, nous voulions valider la reconnaissance et le changement de mode en vol. Pour se faire, nous avons donc un drone équipé d'un Raspberry branché à une caméra, un pilote s'occupant de faire voler le drone et d'un ordinateur qui se connecte via SSH au Raspberry et se connectant au PX4 Via QGroundControl.



Figure 55: matériel utilisé pour test réel, photo prise par l'auteur,

Nous avons réalisé plusieurs fois la même séquence afin de vérifier le bon déroulement du script. Nous lançons le script via SSH alors que le drone est en vol, en mode position. Lorsque le switch de la channel 12 est activé, la reconnaissance se lance et le pilote dirige le drone vers un arbre gentiment. Les tests se sont déroulés avec succès, le drone s'étant effectivement arrêté et passé en mode Hold sur QgroundControl. Nous pouvons voir dans les logs du FCU, que le mode Loiter a bien été activé durant le test de 17h.

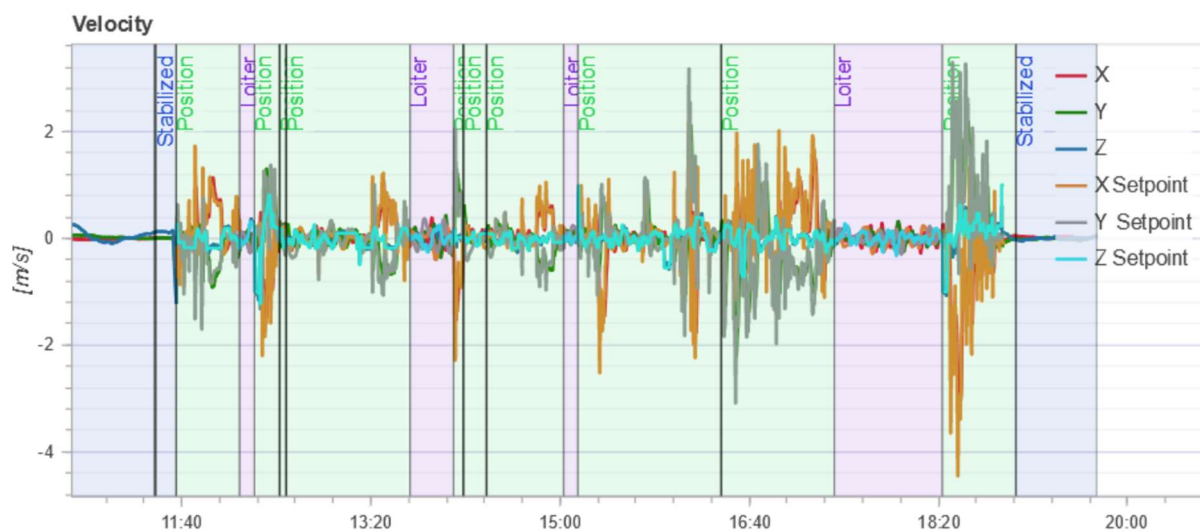


Figure 56: logs du FC, image de l'auteur

Nous avons extrait l'image du test final, qui a permis le changement de mode.

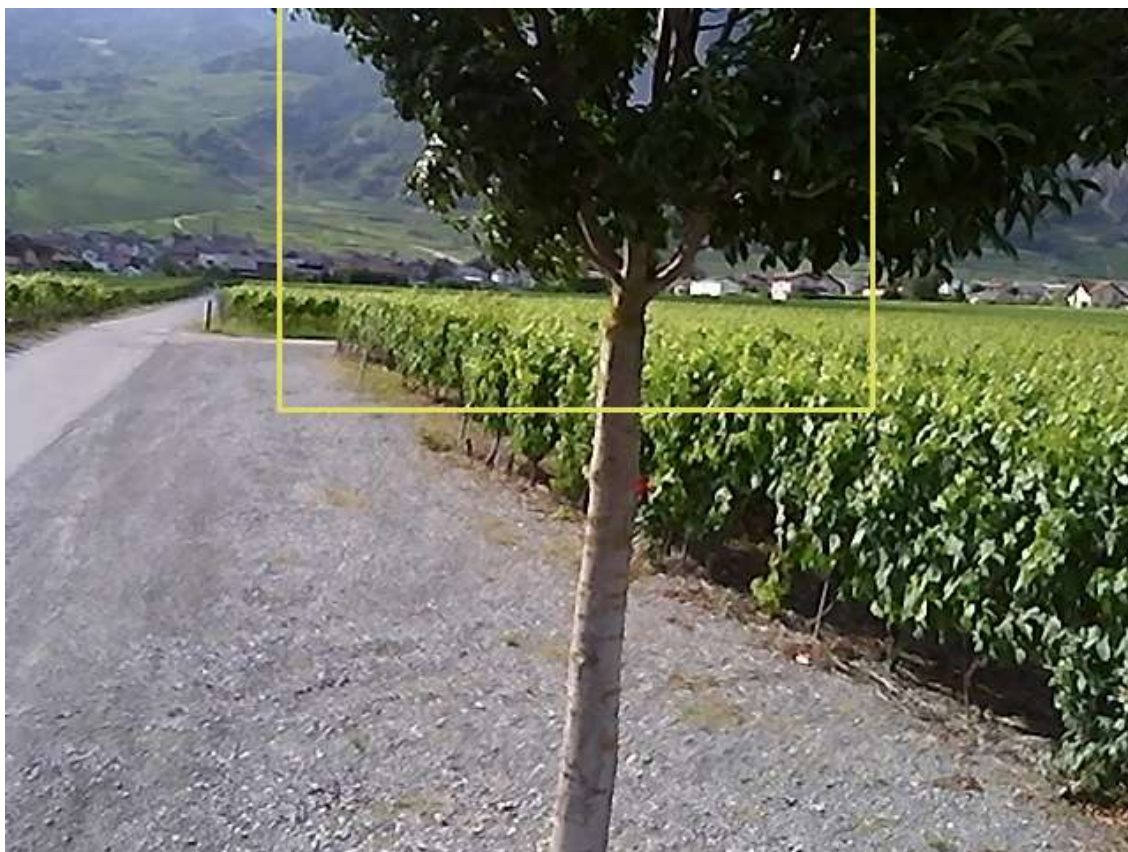


Figure 57: image de stop, de l'auteur

Comme nous pouvons le constater, le modèle a prédit la présence d'un arbre, mais n'a pas su reconnaître la totalité de celui-ci, comme le tronc. Nous expliquons cela par les images d'entraînement fourni au modèle. En effet, dans le dataset, très peu d'arbres possèdent une forme pareille, où le tronc est fin et allongé.

Dans les images fournies par la prédiction, nous avons remarqué que l'algorithme prenait certains points dans l'image pour des arbres. Cela s'explique par le seuil de confiance que nous avons appliqué dans le code. Nous ne changerons pas ce seuil, car en l'occurrence, ces faux positifs sont de taille réduite et ne change pas le fonctionnement du code, qui se base sur la taille du rectangle de précision.



Figure 58: reconnaissance d'objet avec grand nombre de faux positif, image de l'auteur.

8. Conclusion

Nous allons séparer cette conclusion en plusieurs parties, car nous voulons approfondir au mieux le bilan technique, les problèmes rencontrés et les recommandations de chaque composante de notre prototype

8.1. Script et contrôle sur le FCU

8.1.1. Bilan technique

Le code rendu à la fin de cette thèse est un prototype réalisé afin de prouver le potentiel d'utilisation de ce genre de technologie embarquée sur un drone. Il est ainsi loin d'être parfait, car ne gère pas suffisamment d'exceptions, comme en cas de perte de connexion. Le script requiert une connexion via SSH pour pouvoir être lancé, et n'est pas automatisé au démarrage du Raspberry. Lorsque la reconnaissance se termine, il devrait être possible de redémarrer celle-ci, lorsque le switch de la télécommande est de nouveau activé. La détection d'objet est réalisée dans une boucle while, ce qui n'est pas très optimisé, mais fonctionnel de notre cas. De plus, nous aurions voulu automatiser l'accélération du drone, afin de le faire avancer gentiment vers un arbre.

Nous sommes cependant assez satisfaits de la tournure du script, car il reste fonctionnel, malgré sa simplicité. Il remplit ainsi une grande partie des exigences qui étaient demandés.

8.1.2. Problèmes rencontrés

La documentation pour ROS est assez complète, mais les tutoriels sont extrêmement difficiles à comprendre et contre-intuitifs. De manière générale, essayer de coder un nœud Ros par soi-même demande des connaissances avancées en C++ et même les packages pour utiliser PyROS manque de clarté. Étant en grande partie profane de ces langages et cette manière de fonctionner, nous avons quand même tenté de réaliser un nœud, sans succès, car nous n'avons pas réussi à le compiler à cause de problèmes de path. Il s'agit d'un des problèmes, qui arrive très souvent et qui nous a valu des heures d'erreur. Il faut s'assurer que toutes les bibliothèques soient correctement installées, avec les bonnes variables d'environnement et les bonnes versions. Nous avons remarqué une inconsistance entre les différentes documentations, entre celle de Gazebo, ROS, ArduPilot, PX4 et même.

Nous avons trouvé difficile de se retrouver dans la documentation de DroneKit, malgré le fait que l'utilisation de cette bibliothèque soit beaucoup plus simple. De plus, il semblerait que le développement soit passé dans les mains de la communauté, et ne soit plus développé par 3DR

8.1.3. Améliorations futures et recommandations

L'automatisation du script en vol, et de sa réutilisabilité, est l'une des principales pistes d'amélioration. De plus, il serait possible de réviser la structure afin de réaliser plusieurs threads, et ayant la reconnaissance d'image pouvant tourner en simultané de l'écoute du switch.

Nous pensons que l'api MavSDK pourrait être une très bonne alternative à DroneKit, mais la documentation python n'existe pas encore. Il est possible de se baser sur celle en C++ pour appeler l'api, mais nous avons manqué de temps pour essayer de réaliser cette transition. Le projet est encore en développement et serait intéressant de l'essayer. Il reste important d'établir le poids d'une contrainte tel que choisir une technologie encore en développement, et manquant de documentation, avant de la proposer dans une solution commerciale.

8.2. Reconnaissance d'image

8.2.1. Bilan technique

Notre modèle entraîné étant très léger, il reste cependant assez imprécis et ne détecte pas forcément les arbres environnants qui ne possèdent pas une forme équivalente à celle de nos arbres du datasets d'entraînement.

Nous avons entraîné notre réseau de neurones avec un total de 273 images. Il s'agit d'un nombre faible de données, et demanderait un plus gros apport. Il s'agit d'une tâche considérable, car la prise d'images doit être en accord avec les besoins métier, et demanderait des arbres plus spécifiques et proche de vignes. L'annotation d'arbres prend un temps considérable et demande une certaine réflexion. En effet, il faut prendre en considération qu'un buisson dépassant une vigne peut aussi être considéré comme un arbre.

L'utilisation d'OpenCV nous a paru nécessaire durant la réalisation de cette thèse. Les dernières versions de celui-ci permettent d'utiliser des réseaux de neurones facilement et clairement. Cette librairie ayant une place très importante dans le traitement d'image, est de surcroît open source. Elle permet d'utiliser un grand nombre d'algorithmes de ML et une utilisation simple. De plus, une très grande communauté de développeur l'utilise quotidiennement et la documentation est de bonne qualité.

Nous avons tenté d'installer CUDA sur notre machine, afin d'accélérer l'entraînement du réseau de neurones. L'installation est un peu fastidieuse pour aussi permettre à OpenCV d'utiliser cette technologie. Notre machine étant équipé d'une carte graphique de relativement bonne qualité (1GB de mémoire graphique) l'entraînement c'est déroulé très rapidement en moins de 20 minutes.

8.2.2. Problèmes rencontrés

Lors de la compilation du framework, nous avons eu quelques soucis sur le complément d'OpenCV sur Windows. De plus celui-ci étant relativement lourd, une seule erreur peut coûter un certain temps en build.

8.2.3. Améliorations futures et recommandations

L'entraînement plus complet du réseau de neurones nous paraît nécessaire afin de réaliser des prédictions de meilleure qualité, et essayer de régler un peu mieux la configuration de celui-ci. Il serait aussi possible de réaliser des manipulations sur les images, comme les retourner. Afin de compléter notre datasets, un arbre inversé reste un arbre après tout. Nous pensons que le réseau tiny-yolo soit un peu trop imprécis, mais reste un bon compromis entre rapidité et précision.

Les limitations techniques du Raspberry apportent un délai trop élevé de reconnaissance en temps réel pour être utilisées en production. Il serait possible de pallier ce problème par l'utilisation de microordinateur plus puissant, mais malheureusement plus cher. Un bon compromis serait le Jetson Nano de la firme NVIDIA, bien plus puissante qu'un Raspberry pour un prix légèrement plus élevé de 99\$. Il utilise les technologies Cuda, qui permettent d'utiliser plusieurs réseaux de neurones ensemble. Il serait ainsi possible d'utiliser des versions plus efficaces de YOLO que la version tiny.

Nous recommandons l'utilisation de CUDA, si possession d'une carte graphique de bonne qualité, car il accélère l'entraînement du réseau.

8.3. Bilan général

Ce projet s'est donc terminé avec un prototype fonctionnel qui répond donc aux demandes établies du projet. Nous avons remarqué que les systèmes disponibles sur le marché sont soit des solutions payantes et qui demande une adaptation au besoin avant de réellement fonctionner, soit des solutions open sources redoutables de complexité à installer avant fonctionnement. Cette thèse comprenait aussi de nombreux pièges, car les deux principales parties sont réellement complexes, autant en développement qu'en analyse. Ce qui en vient à la principale difficulté qui était de ne pas se perdre.

En effet, nous avons passé un temps considérable à rechercher des informations autant sur la reconnaissance d'image, que sur la manipulation de drone via python ou ROS depuis un Companion Computer. La somme d'informations à rechercher est réellement impressionnante et demande un temps considérable d'appréhension.

Nous estimons cependant avoir prouvé la possibilité d'arriver à implémenter des résultats de machine learning sur un drone d'agriculture. En vue des résultats, il nous reste une question à se

poser en envisageant une suite de ce travail qui est « Comment optimiser la reconnaissance d'image sur un drone d'agriculture ? ».

Nous avons bon espoir que ce projet permettra le développement de solutions plus automatisées pour l'entreprise d'Aero41.

8.4. Conclusion personnelle

Cette thèse de Bachelor était un vrai défi personnel. Ayant déjà suivi des cours sur le pilotage d'un drone, je ne possédais cependant pas énormément de connaissance sur son fonctionnement et je pris un temps considérable à comprendre ce que je ne maîtrisais pas autant pour la partie drone que la partie machine learning.

Lors de l'analyse de cette thèse, je pris conscience de toutes les difficultés que peut rencontrer le monde de l'open source, où la simple utilisation d'un outil peut prendre 2 voire 3 jours à paramétrer. Ce fut par exemple le cas avec OpenCV, qui est un outil puissant, mais complexe à installer sur Windows. En parlant de Windows, le monde de la robotique que j'ai rencontré avec ROS n'est pas fait pour celui-ci. Tous les programmes utilisés pour faciliter le développement ne fonctionnent que sur un noyau Linux, voire même plus précisément, Ubuntu. De plus, les différentes versions d'un système Ubuntu ne sont pas compatibles avec certaines versions de ROS et Gazebo. Il est extrêmement facile de se perdre avec toutes ces différentes versions. Je dois cependant admettre que l'utilisation de Linux, la création de scripts, et l'utilisation de python m'ont permis de développer considérablement mes connaissances sur ce système d'exploitation, ainsi que la gestion des variables environnement. De plus, je me sens aussi plus à l'aise dans un projet de machine learning.

RÉFÉRENCES

Drone. (s. d.). Consulté le 8 mai 2019, à l'adresse <https://www.futura-sciences.com/sciences/definitions/aeronautique-drone-6174/>

Rouse, M., Earls, A., Shea, S., & Wigmore, I. (2018, October 1). *What is drone (unmanned aerial vehicle, UAV)? - Definition from Wha...* Consulté le 8 mai 2019, à l'adresse <https://internetofthingsagenda.techtarget.com/definition/drone>

De quoi est composé un drone? (s. d.). Consulté 8 mai 2019, à l'adresse <https://www.studiosport.fr/guides/drones/de-quoi-est-compose-un-drone.html>

Corrigan, F. (2018, August 24). *How A Quadcopter Works With Propellers And Motors Explained.* Consulté le 8 mai 2019, à l'adresse <https://www.dronezon.com/learn-about-drones-quadcopters/how-a-quadcopter-works-with-propellers-and-motors-direction-design-explained/>

ROS.org | About ROS. (s. d.). Consulté le 14 mai 2019, à l'adresse <http://www.ros.org/about-ros/>

Batteries de drones : types et performances. (s. d.). Consulté le 15 mai 2019, à l'adresse <https://www.studiosport.fr/guides/drones/batteries-de-drones-types-et-performances.html>

Le guide des batteries Lipo. (2018, mai 7). [Blog post]. Consulté le 15 mai 2019, à l'adresse <https://www.rcteam.fr/fr/blog/le-guide-des-batteries-lipo-n5>

Lesage, A. (2018, janvier 10). *Guide des composants | Comprendre les LiPo - Culture FPV* [Blog post]. Consulté le 15 mai 2019, à l'adresse <https://culturefpv.fr/comprendre-les-lipo-20171007/>

Mazzari, V. (2016, juillet 26). *ROS - Robot Operating System.* Consulté le 20 mai 2019, à l'adresse <https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/>

Maverick documentation. (s. d.). Consulté 13 mai 2019, à l'adresse <http://goodrobots.github.io/maverick/>

What is Pitch, Roll and Yaw? - Emissary Drones. (2018, mars 19). [Blog post]. Consulté le 22 mai 2019, à l'adresse <https://emissarydrones.com/what-is-roll-pitch-and-yaw>

Ubiquity Robotics Downloads. (s. d.). Consulté 13 juin 2019, à l'adresse <https://downloads.ubiquityrobotics.com/pi.html>

- Murugavel, M. (2019, mars 8). *How to train YOLOv2 to detect custom objects*. Consulté 4 juillet 2019, à l'adresse https://medium.com/@manivannan_data/how-to-train-yolov2-to-detect-custom-objects-9010df784f36
- Redmon, J. (s. d.). *YOLO: Real-Time Object Detection*. Consulté 7 juillet 2019, à l'adresse <https://pjreddie.com/darknet/yolo/>
- Murugavel, M. (2018, juin 23). *How to train YOLOv3 to detect custom objects* [Blog post]. Consulté 7 juillet 2019, à l'adresse https://medium.com/@manivannan_data/how-to-train-yolov3-to-detect-custom-objects-ccbcafeb13d2
- Yolo Annotation Tool-New*. (2019, mars 5). [Blog post]. Consulté 7 juillet 2019, à l'adresse https://medium.com/@manivannan_data/yolo-annotation-tool-new-18c7847a2186
- Maj, M. (s. d.). *Object Detection and Image Classification with YOLO*. Consulté 24 juillet 2019, à l'adresse <https://www.kdnuggets.com/2018/09/object-detection-image-classification-yolo.html>
- Mellid, J. (2017, octobre 6). *Old Habits Die Hard - Open Source UAV API, DroneKit-Python and Geopy*. Consulté 21 juillet 2019, à l'adresse <https://javiermunhoz.com/blog/2017/10/06/open-source-uav-api-dronekit-python-and-geopy.html>
- BaudRate - MATLAB*. (s. d.). Consulté 24 juillet 2019, à l'adresse https://www.mathworks.com/help/matlab/matlab_external/baudrate.html
- AlexeyAB/darknet*. (s. d.). Consulté le 2 juillet 2019, à l'adresse <https://github.com/AlexeyAB/darknet>
- IrisAutomation. (2019, juin 11). *#HelloCasia*. Consulté 3 juillet 2019, à l'adresse <https://www.irisonboard.com/2019/06/11/hellocasia/>
- Mobileye. (s. d.). *Real-Time Collision Avoidance System for Fleets | Mobileye*. Consulté 4 juillet 2019, à l'adresse <https://www.mobileye.com/us/fleets/>
- Ainstein. (2019, mars 6). *Ainstein News - Ainstein*. Consulté 6 juillet 2019, à l'adresse <https://ainstein.ai/ainstein-news/>
- Rosebreck, A. (2018, novembre 12). *YOLO object detection with OpenCV - PyImageSearch* [Blog post]. Consulté 4 juillet 2019, à l'adresse <https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>

- Quadcopter Communication Protocols - Overview • LearningRC.* (2017, avril 6). [Blog post]. Consulté 3 juin 2019, à l'adresse <http://learningrc.com/rc-communication-protocols/>
- Les réseaux de neurones | Intelligence artificielle 41.* (2018, septembre 17). [YouTube]. Consulté 11 juillet 2019, à l'adresse <https://www.youtube.com/watch?v=8qL2ISQd9L8>
- Redmon, J., A, Farhadi (2018, avril 8). *YOLOv3: An Incremental Improvement.* Consulté 15 juillet 2019, à l'adresse <https://arxiv.org/abs/1804.02767>
- Unsupervised Feature Learning and Deep Learning Tutorial.* (s. d.). Consulté 19 juillet 2019, à l'adresse <http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>
- Menegaz, M. (2018, mars 16). *Understanding YOLO.* Consulté 15 juillet 2019, à l'adresse <https://hackernoon.com/understanding-yolo-f5a74bbc7967>
- Zhang, J. (2019, February 13). *An improved tiny-yolov3 pedestrian detection algorithm.* Consulté 16 juillet 2019, à l'adresse <https://www.sciencedirect.com/science/article/pii/S003040261930155X>
- Josh. (2016, juin 26). *Expose GPU from Windows 10 host to Ubuntu 16.* Consulté 10 juin 2019, à l'adresse <https://superuser.com/questions/1093795/expose-gpu-from-windows-10-host-to-ubuntu-16>
- Dukowitz, Z. (2019, mai 23). *Iris Automation, The Future of BVLOS and Drone Automation* [Blog post]. Consulté 26 juin 2019, à l'adresse <https://uavcoach.com/iris-automation/>
- ManivannanMurugavel/Yolo-Annotation-Tool-New-.* (s. d.). Consulté 29 juillet 2019, à l'adresse <https://github.com/ManivannanMurugavel/Yolo-Annotation-Tool-New->
- Obstacle Avoidance · PX4 User Guide.* (s. d.). [Documentation]. Consulté 2 juillet 2019, à l'adresse https://docs.px4.io/master/en/computer_vision/obstacle_avoidance.html
- cs213n. (s. d.). *CS213n Convolutional Neural Networks for Visual Recognition.* Consulté 12 juillet 2019, à l'adresse <http://cs231n.github.io/convolutional-networks/>
- Souza, A. M. F., & Soares, F. M. (2016). *Neural Network Programming with Java.* Packt Publishing.
- Département fédéral de l'environnement, des transports, de l'énergie et de la communication. (1994, novembre 24). RS 748.941 *Ordonnance du DETEC du 24 novembre 1994 sur les aéronefs de*

catégories spéciales (OACS). Consulté 30 juillet 2019, à l'adresse <https://www.admin.ch/opc/fr/classified-compilation/19940351/index.html>

ognjanovski.gavril. (2019, février 07). *Everything you need to know about Neural Networks and Backpropagation - Machine Learning Easy and Fun*. Consulté 11 juillet 2019, à l'adresse <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3aa>

Zhang E., Zhang Y. (2009) *Average Precision*. In: LIU L., ÖZSU M.T. (eds) *Encyclopedia of Database Systems*. Springer, Boston, MA. DOI : <https://doi.org/10.1007/978-0-387-39940-9>

Bowley, J. (2019, juin 21). *Accelerating OpenCV 4.1.0 - build with CUDA, Intel MKL + TBB and python bindings*. Consulté 30 juillet 2019, à l'adresse <https://jamesbowley.co.uk/accelerating-opencv-4-build-with-cuda-intel-mkl-tbb-and-python-bindings/>

ANNEXES

ANNEXE I

Guide d'installation de Darknet

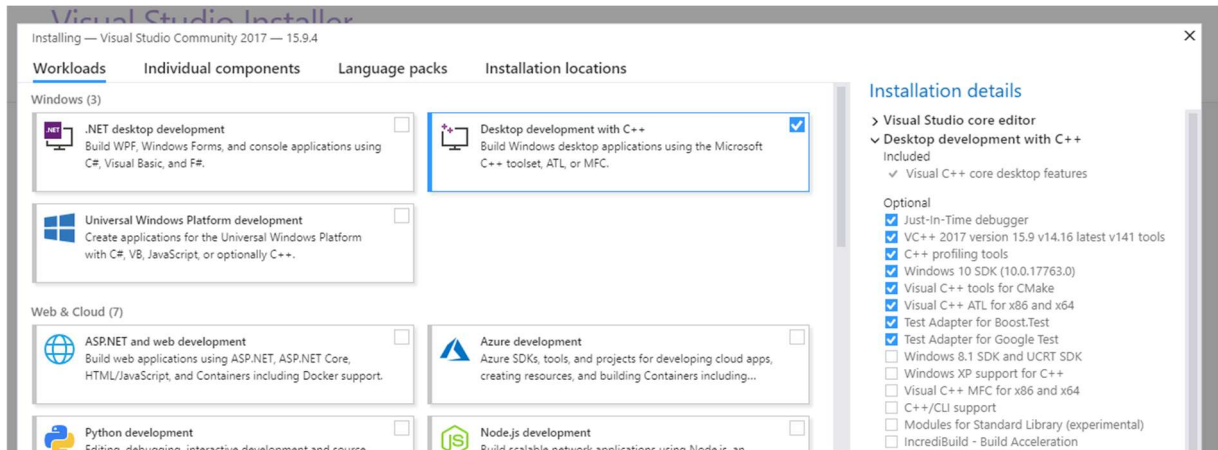


Figure 59 Logo de darknet, récupéré sur : <https://github.com/pjreddie/darknet>

Bienvenue dans ce guide d'installation. Darknet est un framework permettant d'entraîner et utiliser un réseau de neurone nommé YOLO tout en utilisant une carte graphique. Pour pouvoir le compiler sur Windows, nous devons au préalable installer correctement OpenCV 4.1 et en utilisant CUDA.

Requirement :

Lors de cette installation, nous avons utilisé Visual Studio en version 2017 15 sur windows 10. Attention de bien ajouter l'environnement de développement en c++



Afin d'accélérer le compilé d'openCV, nous utiliserons un soft de build nommé Ninja. Pour s'installer, nous l'avons fait grâce à chocolatey, un manager de package que l'on peut installer sur windows. Il s'utilise de la même manière que pip ou apt-get.

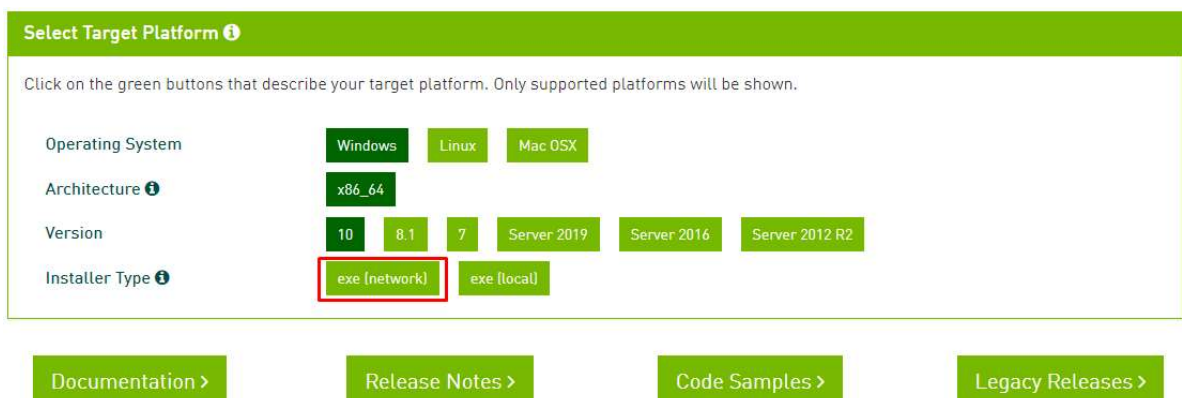
Nous utiliserons aussi Cmake, logiciel de compilation sur Windows, de manière graphique

Dans un premier temps, il nous faut installer CUDA sur notre pc. Etant donné que l'installation selon le système d'exploitation diffère, nous ne nous focaliserons uniquement sur celle sur Windows 10

Afin de le setup l'ensemble, il faut au préalable se renseigner sur la carte graphique équipée, et vérifié qu'elle soit compatible avec CUDA.

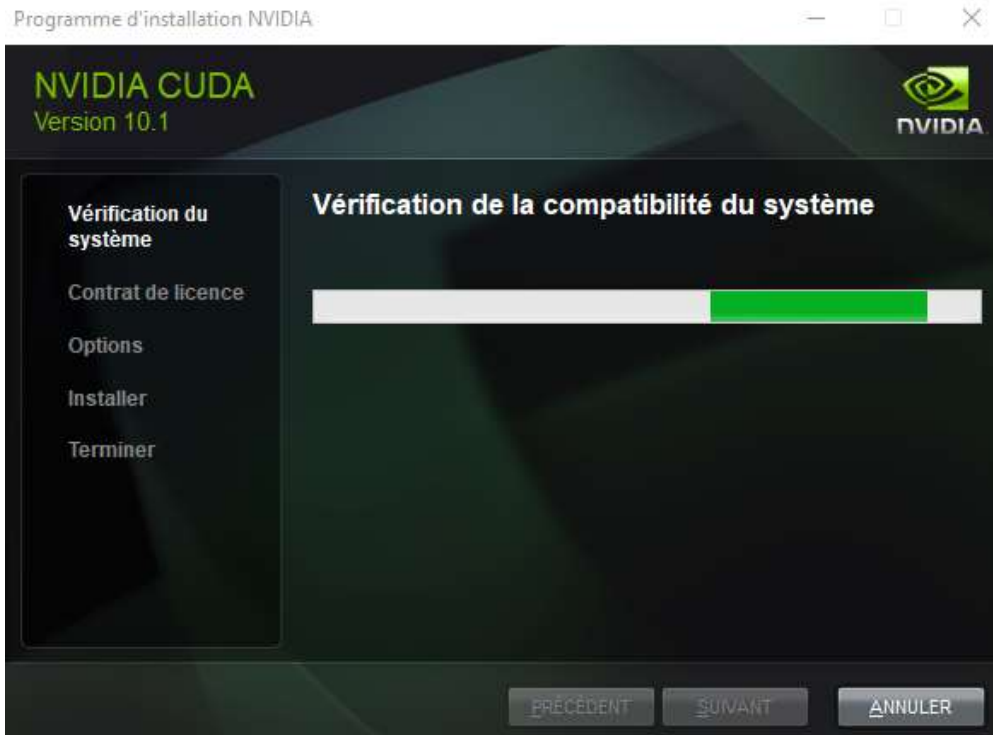
Télécharger le toolkit NVIDIA CUDA : <https://developer.nvidia.com/cuda-downloads>

Nous avons utilisé la version network.

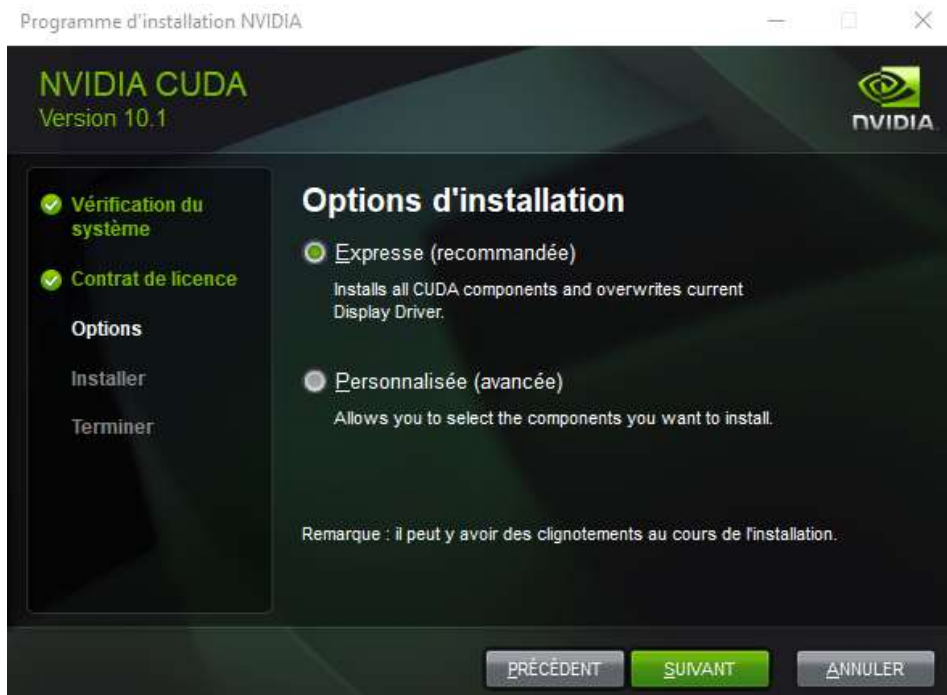


Il faut ensuite lancer l'exé puis, puis choisir le dossier où sera installé temporairement le toolkit.

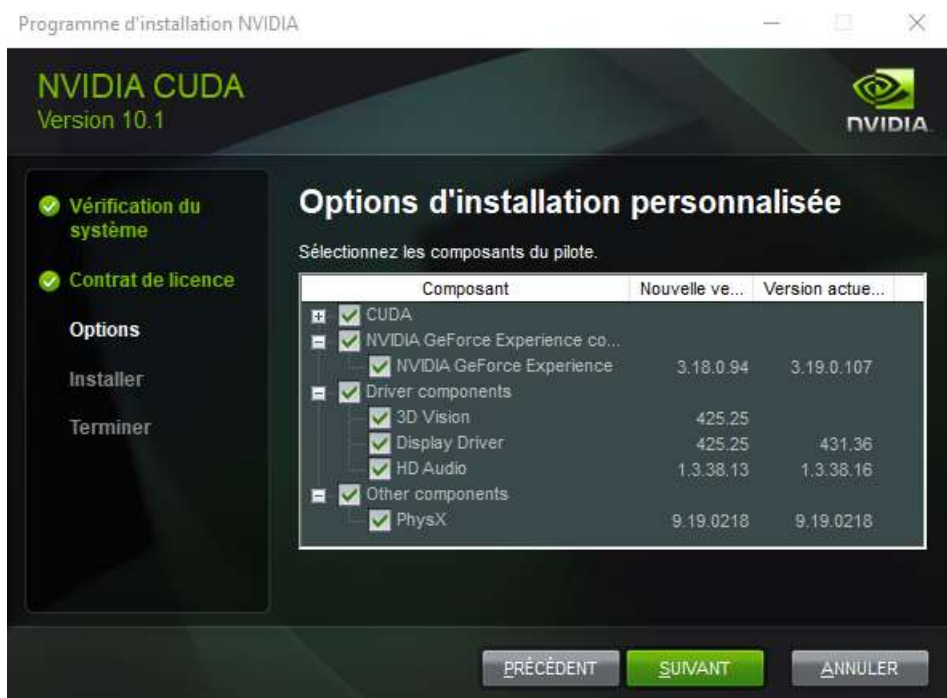
Cuda va vérifier la compatibilité du système, puis il faudra accepter le contrat de licence NVIDIA



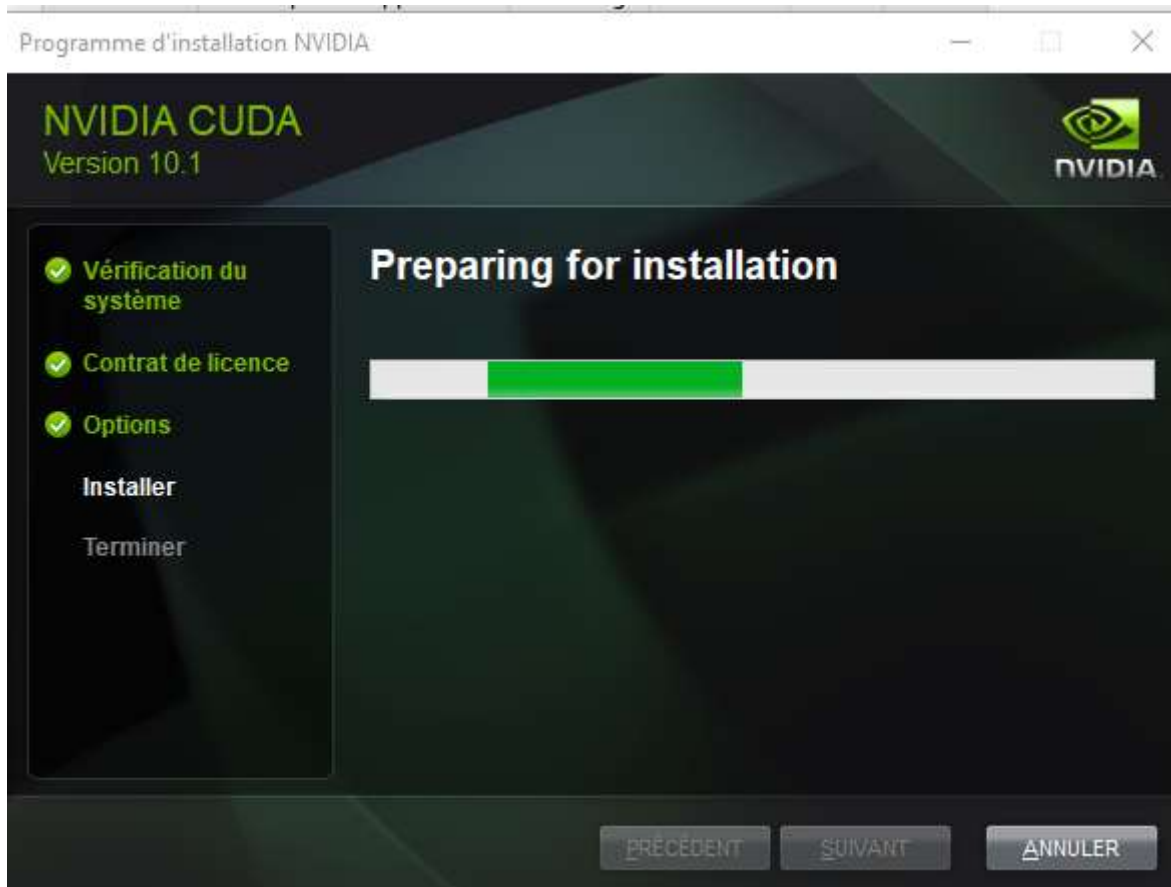
Il faudra ensuite choisir le type d'installation, nous avons choisi Express. Il est possible de réaliser une installation personnalisée, et installer que le nécessaire.



En cas s'installation choisir que le nécessaire :



Il faut ensuite attendre quelque instant que le programme s'installe puis cliquer sur terminer.



Il est ensuite important de vérifier l'utilisations du bon model de driver. En effet il existe 2 modele de driver disponible sur windows 10. Le premier, WDDM est utilisé pour de l'affichage. Le second, [Tesla Compute Cluster \(TCC\)](#), est un mode de NVIDIA. Il faut être vigilant quant à ces modes, car si le mode TCC est activé pour une carte graphique précise, celle-ci ne peut pas être utilisé pour de l'affichage.

L'installation devrait s'être correctement effectué. Cependant, il e est possible de vérifier que tout c'est bien dérouler. Tout d'abord en exécutant la commande `nvcc -v` afin de vérifier la version de cuda

```
C:\WINDOWS\system32>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Apr_24_19:11:20_Pacific_Daylight_Time_2019
Cuda compilation tools, release 10.1, V10.1.168
```

Nous avons ensuite installé cuDNN. Il faut s'inscrire dans le programme de développeur de NVIDIA, puis télécharger la version correspondante à CUDA 10.1.

[Home](#)

cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

[Download cuDNN v7.6.2 \(July 22, 2019\), for CUDA 10.1](#)

[Download cuDNN v7.6.2 \(July 22, 2019\), for CUDA 10.0](#)

[Download cuDNN v7.6.2 \(July 22, 2019\), for CUDA 9.2](#)

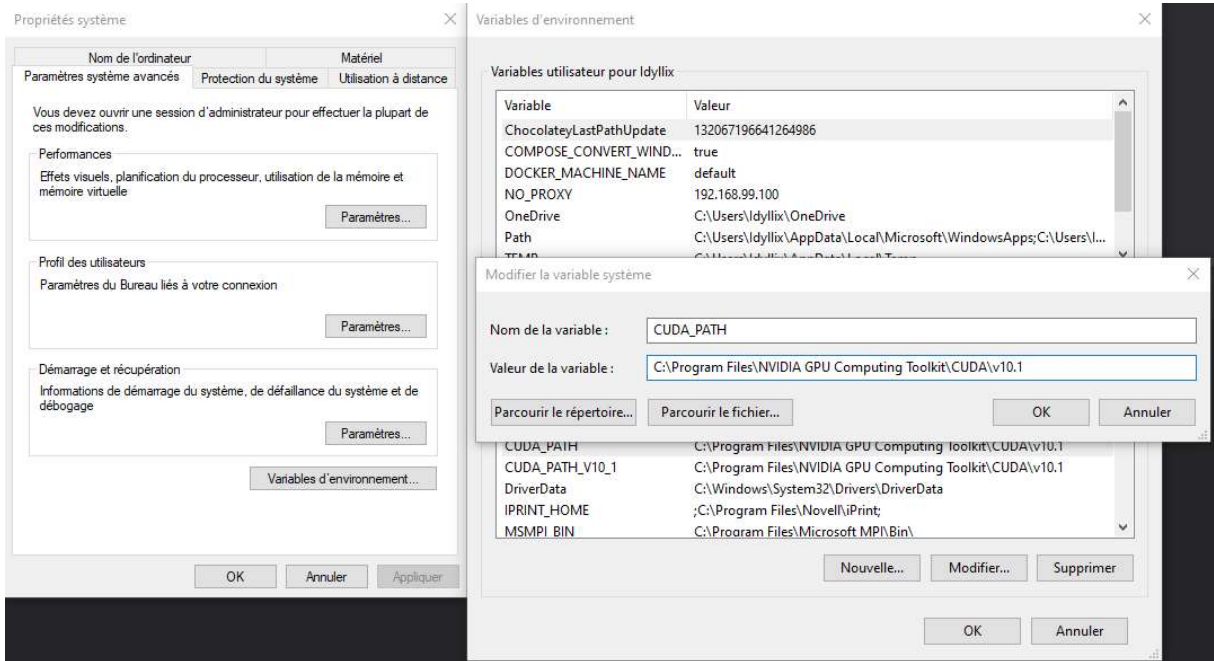
[Download cuDNN v7.6.2 \(July 22, 2019\), for CUDA 9.0](#)

[Archived cuDNN Releases](#)

Aller jusque dans votre dossier de téléchargement, puis copiez-collez chacun des fichiers suivants dans :

- `<installpath>\cuda\bin\cuda64_7.dll` vers C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin.
- `<installpath>\cuda\include\cuda.h` vers C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\include.
- `<installpath>\cuda\lib\x64\cuda.lib` vers C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\lib\x64.

Vérifiez ensuite que la variable environnement pour CUDA est bien existante :



Nous pouvons maintenant télécharger Opencv.

Pour cela, il est possible d'utiliser git, afin de récupérer la dernière version. Ou alors voici les liens des archives

<https://github.com/opencv/opencv/archive/4.1.0.zip>

https://github.com/opencv/opencv_contrib/archive/4.1.0.zip

!/ Attention de bien utiliser les versions correspondante, opencv4.1 avec opencv_contrib4.1 !/

Ouvrir une commande prompt, puis entrer les variables suivantes. Attention à bien entrer les chemin système pour Ninja, opencv et opencv_contrib :

```
"C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build\vcvars64.bat"
set "ninjaPath=PATH_TO_NINJA"
set path=%ninjaPath%;%path%
set "openCvSource=PATH_TO_OPENCV_SOURCE"
set "openCVExtraModules=PATH_TO_OPENCV_CONTRIB_MODULES"
set "openCvBuild=%openCvSource\build"
set "buildType=Release"
set "generator=Ninja"
```

Dans notre cas, car nous avons installé ninja via chocolatey. Voici notre commande :

```
"C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\VC\Auxiliary\Build\vcvars64.bat"
set "ninjaPath=C:\ProgramData\chocolatey\lib\ninja\tools"
set path=%ninjaPath%;%path%
set "openCvSource=C:\opencv"
set "openCVExtraModules=C:\opencv_contrib\modules"
set "openCvBuild=%openCvSource%\build"
set "buildType=Release"
set "generator=Ninja"
```

Puis lancez cette commande, elle permettra de configurer les fichiers de builds de Ninja.

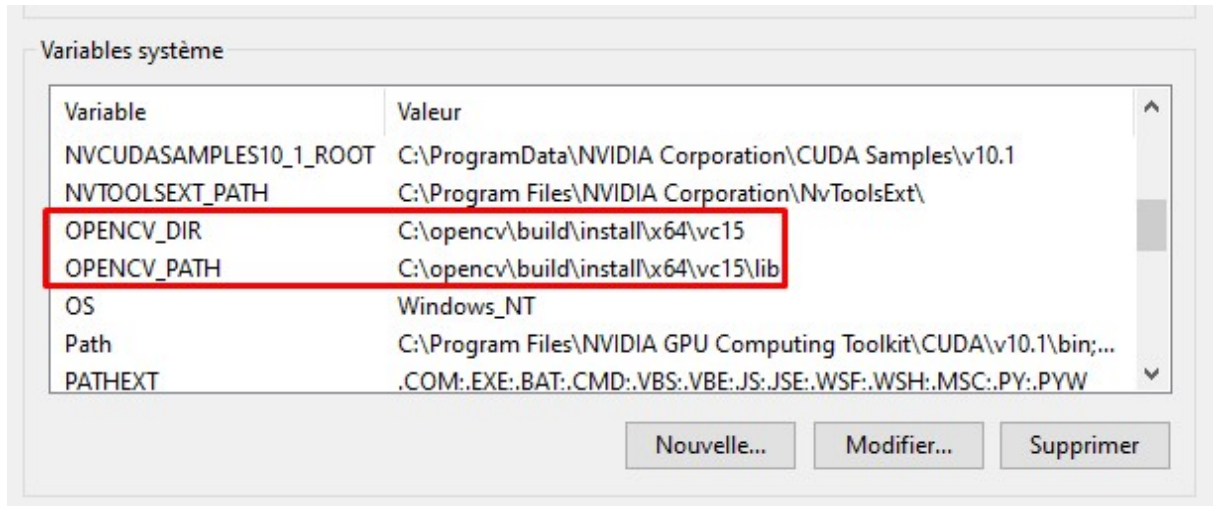
```
"C:\Program Files\CMake\bin\cmake.exe" -B"%openCvBuild%" -H"%openCvSource%" -G"%generator%" -DCMAKE_BUILD_TYPE=%buildType% -DBUILD_opencv_world=ON -DBUILD_opencv_gapi=OFF -DWITH_NVCUVID=OFF -DWITH_CUDA=ON -DCUDA_TOOLKIT_ROOT_DIR="C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v10.1" -DCUDA_FAST_MATH=ON -DWITH_CUBLAS=ON -DINSTALL_TESTS=ON -DINSTALL_C_EXAMPLES=ON -DBUILD_EXAMPLES=ON -DWITH_OPENGL=ON -DOPENCV_EXTRA_MODULES_PATH="%openCVExtraModules%" -DOPENCV_ENABLE_NONFREE=ON -DCUDA_ARCH_PTX=7.5 -DBUILD_opencv_python3=ON -DBUILD_opencv_hdf=OFF -DWITH_NVCUVID=ON -DWITH_MFX=ON
```

Le build peut être démarré via cette commande :

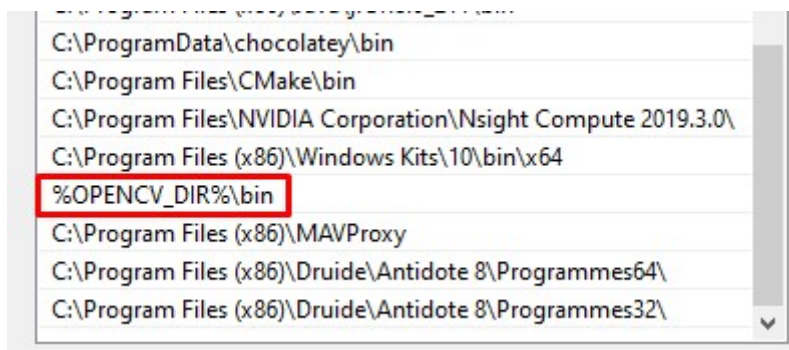
```
"C:\Program Files\CMake\bin\cmake.exe" --build %openCvBuild% --target install
```

Le compilé va durer pendant un certain temps.

Rajoutez ensuite des variables environnement comme suivi :



Puis dans la variable Path :



Pour vérifier qu'opencv soit installé correctement, vous pouvez lancer la commande suivante :

```
"%openCvBuild%\install\x64\vc15\bin\opencv_perf_cudaarithm.exe" --gtest_filter=Sz_Type_Flags_GEMM.GEMM/29
```

Vous devriez avoir les lignes suivantes s'afficher :

```

OpenCL Platforms:
  NVIDIA CUDA
    dGPU: GeForce GTX 980M (OpenCL 1.2 CUDA)
Current OpenCL device:
  Type = dGPU
  Name = GeForce GTX 980M
  Version = OpenCL 1.2 CUDA
  Driver version = 431.36
  Address bits = 64
  Compute units = 12
  Max work group size = 1024
  Local memory size = 48 KB
  Max memory allocation size = 1 GB
  Double support = Yes
  Host unified memory = No
  Device extensions:
    cl_khr_global_int32_base_atomics
    cl_khr_global_int32_extended_atomics
    cl_khr_local_int32_base_atomics
    cl_khr_local_int32_extended_atomics
    cl_khr_fp64
    cl_khr_byte_addressable_store
    cl_khr_icd
    cl_khr_gl_sharing
    cl_nv_compiler_options
    cl_nv_device_attribute_query
    cl_nv_pragma_unroll
    cl_nv_d3d10_sharing
    cl_khr_d3d10_sharing
    cl_nv_d3d11_sharing
    cl_nv_copy_opts
    cl_nv_create_buffer
  Has AMD Blas = No
  Has AMD Fft = No
  Preferred vector width char = 1
  Preferred vector width short = 1
  Preferred vector width int = 1
  Preferred vector width long = 1
  Preferred vector width float = 1
  Preferred vector width double = 1
Note: Google Test filter = Sz_Type_Flags_GEMM.GEMM/29
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from Sz_Type_Flags_GEMM
[ RUN     ] Sz_Type_Flags_GEMM.GEMM/29, where GetParam() = (1024x1024, 32FC2, 0|cv::GEMM_1_T)
[ PERFSTAT ] (samples=13 mean=4.56 median=4.53 min=4.43 stddev=0.11 (2.5%))
[ OK      ] Sz_Type_Flags_GEMM.GEMM/29 (627 ms)
[-----] 1 test from Sz_Type_Flags_GEMM (630 ms total)

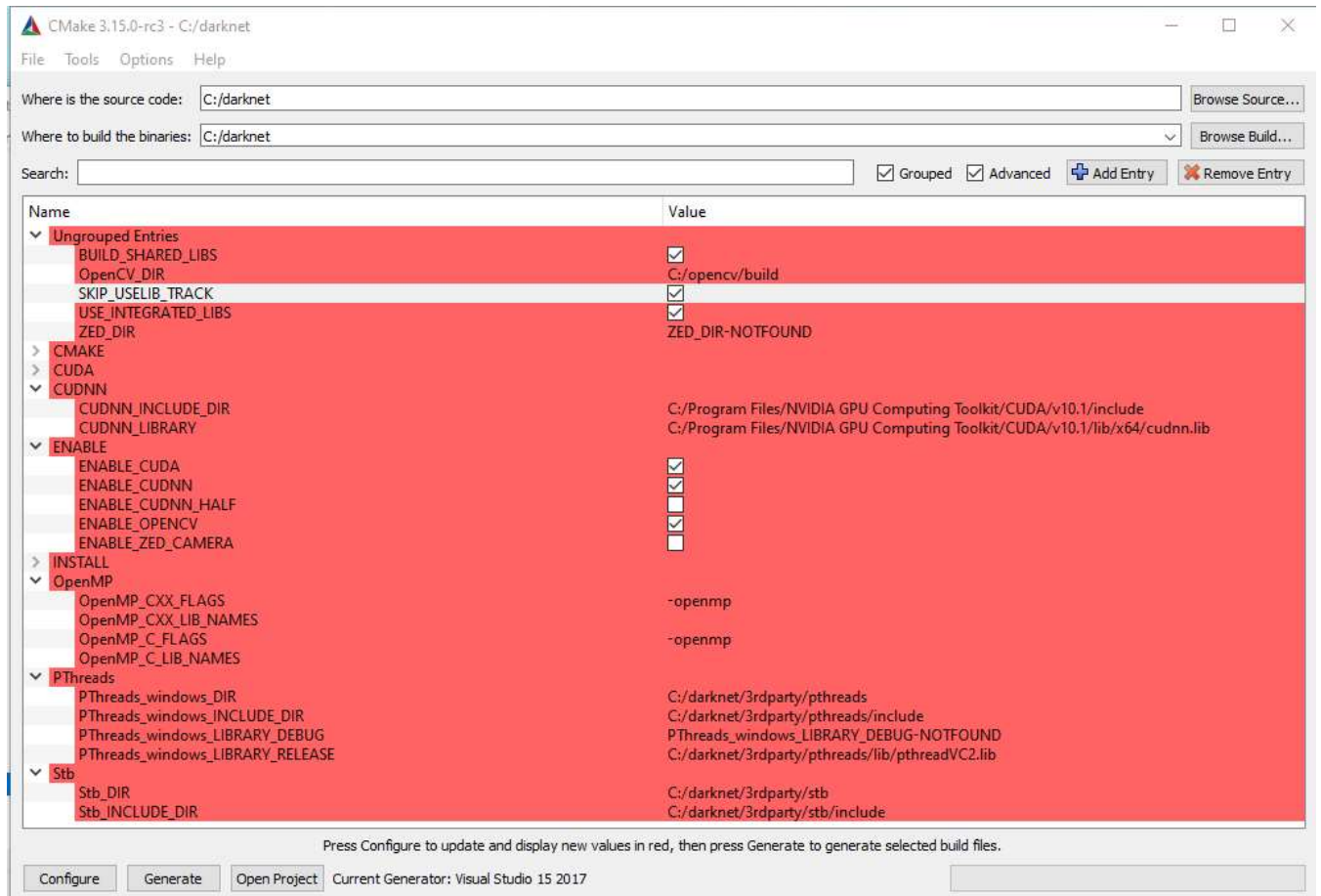
[-----] Global test environment tear-down
[=====] 1 test from 1 test case ran. (636 ms total)
[ PASSED ] 1 test.

```

Opencv est alors installé correctement.

Installation de Darknet :

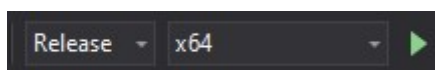
Ouvrez Cmake GUI, puis reglez correctement de la manière suivante :



Appuyez sur configure et sélectionner x64 pour « Optional platform for generator », puis « Finish » et ensuite « Generate ».

Ne fermez pas la fenêtre, et cliquez sur « open project ». Une fenêtre Visual studio devrait s'ouvrir.

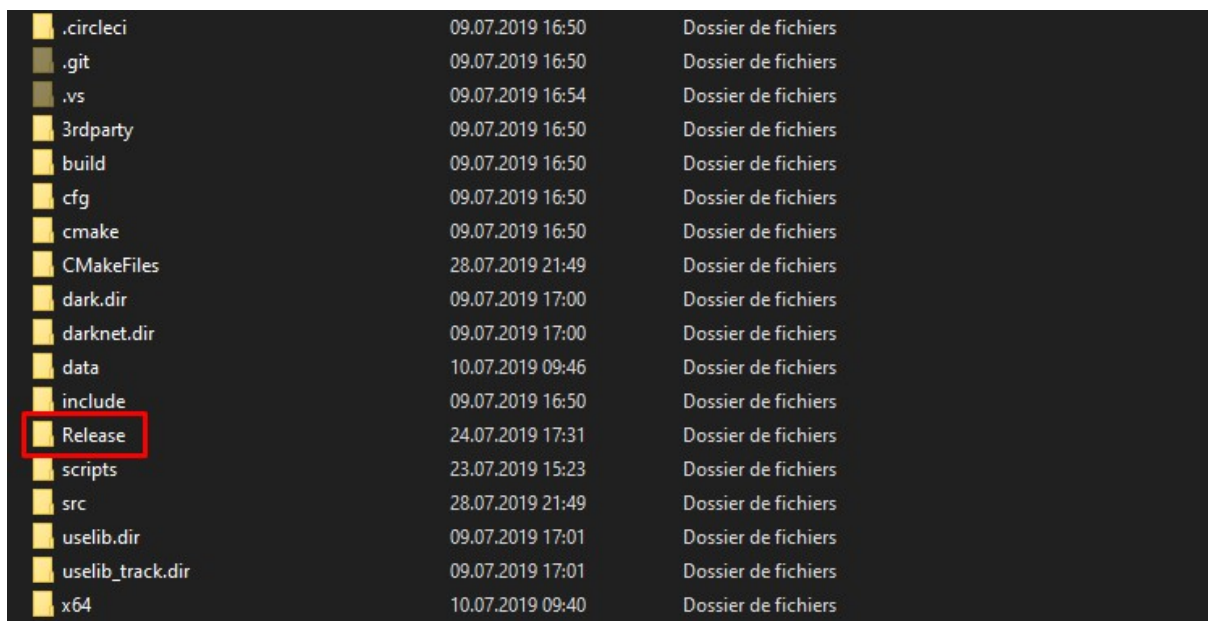
Vérifiez d'avoir bien « x64 » et « Release » dans les options de build.



Lancez la compilation via Build/Build solution.

Celle-ci va durer un certain temps

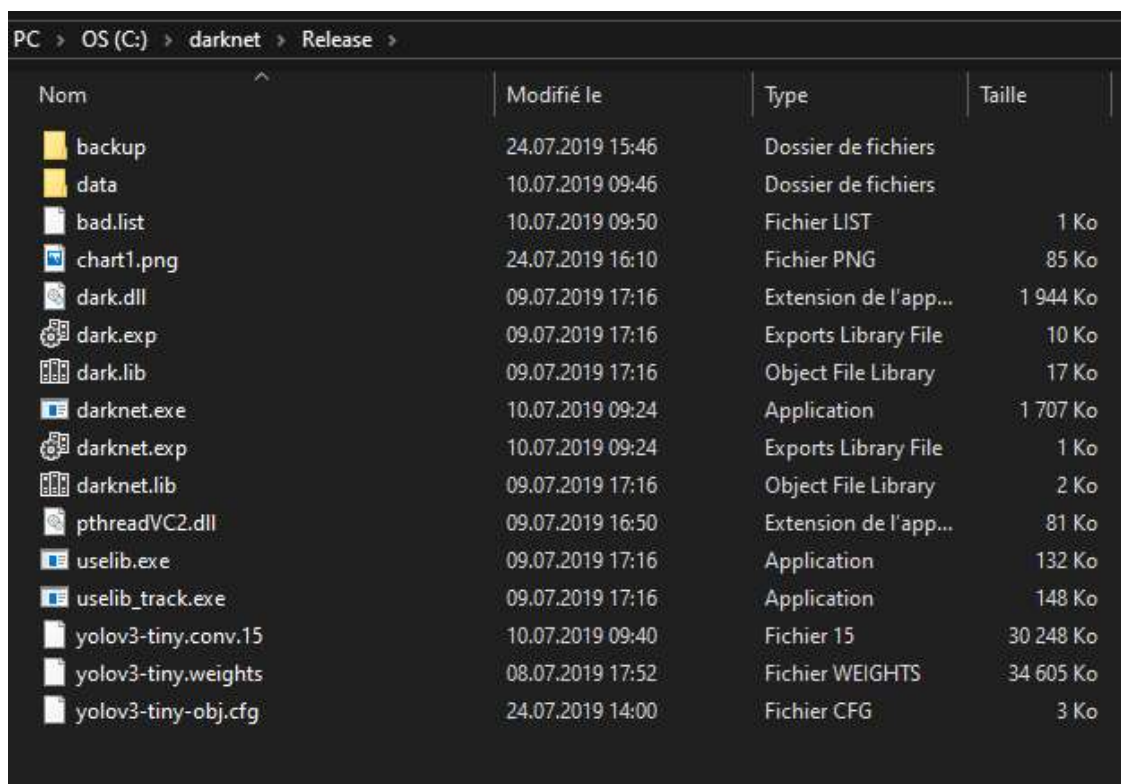
Vous devriez avoir un nouveau dossier nommé Release dans votre emplacement darknet :



Nom	Modifié le	Type
.circleci	09.07.2019 16:50	Dossier de fichiers
.git	09.07.2019 16:50	Dossier de fichiers
.vs	09.07.2019 16:54	Dossier de fichiers
3rdparty	09.07.2019 16:50	Dossier de fichiers
build	09.07.2019 16:50	Dossier de fichiers
cfg	09.07.2019 16:50	Dossier de fichiers
cmake	09.07.2019 16:50	Dossier de fichiers
CMakeFiles	28.07.2019 21:49	Dossier de fichiers
dark.dir	09.07.2019 17:00	Dossier de fichiers
darknet.dir	09.07.2019 17:00	Dossier de fichiers
data	10.07.2019 09:46	Dossier de fichiers
include	09.07.2019 16:50	Dossier de fichiers
Release	24.07.2019 17:31	Dossier de fichiers
scripts	23.07.2019 15:23	Dossier de fichiers
src	28.07.2019 21:49	Dossier de fichiers
uselib.dir	09.07.2019 17:01	Dossier de fichiers
uselib_track.dir	09.07.2019 17:01	Dossier de fichiers
x64	10.07.2019 09:40	Dossier de fichiers

Votre darknet est alors correctement installé !

Nous avons eu un problème quant à une librairie que notre programme n'arrivait pas à trouver. Nous avons alors copier-coller pthreadVC2.dll depuis le dossier 3rdparty/bin à l'intérieur de Release, afin que celui-ci trouve correctement la .dll. Notre dossier Release :



Nom	Modifié le	Type	Taille
backup	24.07.2019 15:46	Dossier de fichiers	
data	10.07.2019 09:46	Dossier de fichiers	
bad.list	10.07.2019 09:50	Fichier LIST	1 Ko
chart1.png	24.07.2019 16:10	Fichier PNG	85 Ko
dark.dll	09.07.2019 17:16	Extension de l'app...	1 944 Ko
dark.exp	09.07.2019 17:16	Exports Library File	10 Ko
dark.lib	09.07.2019 17:16	Object File Library	17 Ko
darknet.exe	10.07.2019 09:24	Application	1 707 Ko
darknet.exp	10.07.2019 09:24	Exports Library File	1 Ko
darknet.lib	09.07.2019 17:16	Object File Library	2 Ko
pthreadVC2.dll	09.07.2019 16:50	Extension de l'app...	81 Ko
uselib.exe	09.07.2019 17:16	Application	132 Ko
uselib_track.exe	09.07.2019 17:16	Application	148 Ko
yolov3-tiny.conv.15	10.07.2019 09:40	Fichier 15	30 248 Ko
yolov3-tiny.weights	08.07.2019 17:52	Fichier WEIGHTS	34 605 Ko
yolov3-tiny-obj.cfg	24.07.2019 14:00	Fichier CFG	3 Ko

Pour lancer un entraînement, mettez tous les fichiers à l'intérieur de Release, afin de faciliter la commande. Sinon vérifier les chemins des fichiers lors de la commande

```
darknet.exe detector train data/obj.data yolov3-tiny-obj.cfg yolov3-tiny.conv.15
```

Lors du lancement de l'entraînement voici ce que vous devriez voir :

```
C:\darknet\Release>darknet.exe detector train data/obj.data yolov3-tiny-obj.cfg yolov3-tiny.conv.15
yolov3-tiny-obj
  layer  filters  size/strd(dil)  input  output
  0 conv  16      3 x 3/ 1      416 x 416 x 3 -> 416 x 416 x 16 0.150 BF
  1 max           2 x 2/ 2      416 x 416 x 16 -> 208 x 208 x 16 0.003 BF
  2 conv  32      3 x 3/ 1      208 x 208 x 16 -> 208 x 208 x 32 0.399 BF
  3 max           2 x 2/ 2      208 x 208 x 32 -> 104 x 104 x 32 0.001 BF
  4 conv  64      3 x 3/ 1      104 x 104 x 32 -> 104 x 104 x 64 0.399 BF
  5 max           2 x 2/ 2      104 x 104 x 64 -> 52 x 52 x 64 0.001 BF
  6 conv  128     3 x 3/ 1      52 x 52 x 64 -> 52 x 52 x 128 0.399 BF
  7 max           2 x 2/ 2      52 x 52 x 128 -> 26 x 26 x 128 0.000 BF
  8 conv  256     3 x 3/ 1      26 x 26 x 128 -> 26 x 26 x 256 0.399 BF
  9 max           2 x 2/ 2      26 x 26 x 256 -> 13 x 13 x 256 0.000 BF
 10 conv  512     3 x 3/ 1      13 x 13 x 256 -> 13 x 13 x 512 0.399 BF
 11 max           2 x 2/ 1      13 x 13 x 512 -> 13 x 13 x 512 0.000 BF
 12 conv 1024     3 x 3/ 1      13 x 13 x 512 -> 13 x 13 x1024 1.595 BF
 13 conv  256     1 x 1/ 1      13 x 13 x1024 -> 13 x 13 x 256 0.089 BF
 14 conv  512     3 x 3/ 1      13 x 13 x 256 -> 13 x 13 x 512 0.399 BF
 15 conv  18      1 x 1/ 1      13 x 13 x 512 -> 13 x 13 x 18 0.003 BF
 16 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
 17 route  13
 18 conv  128     1 x 1/ 1      13 x 13 x 256 -> 13 x 13 x 128 0.011 BF
 19 upsample      2x      13 x 13 x 128 -> 26 x 26 x 128
 20 route  19 8
 21 conv  256     3 x 3/ 1      26 x 26 x 384 -> 26 x 26 x 256 1.196 BF
 22 conv  18      1 x 1/ 1      26 x 26 x 256 -> 26 x 26 x 18 0.006 BF
 23 yolo
```

```
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
 17 route  13
 18 conv  128     1 x 1/ 1      13 x 13 x 256 -> 13 x 13 x 128 0.011 BF
 19 upsample      2x      13 x 13 x 128 -> 26 x 26 x 128
 20 route  19 8
 21 conv  256     3 x 3/ 1      26 x 26 x 384 -> 26 x 26 x 256 1.196 BF
 22 conv  18      1 x 1/ 1      26 x 26 x 256 -> 26 x 26 x 18 0.006 BF
 23 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
Total BFLOPS 5.448
Allocate additional workspace_size = 52.43 MB
Loading weights from yolov3-tiny.conv.15...
seen 64
Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
If error occurs - run training with flag: -dont_show
Resizing
608 x 608
try to allocate additional workspace_size = 52.43 MB
CUDA allocate done!
Loaded: 0.000000 seconds
v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 16 Avg (IOU: 0.424
```

ANNEXE II : Script python record.py

```
import os
import datetime
import sys

currentDT = datetime.datetime.now().strftime("%Y%m%d%H%M%S")

os.system(sudo /flyt/flytos/flytcore/share/core_api/scripts/stop_flytOS.sh)

os.system('ffmpeg -t '+sys.argv[1]+' -f video4linux2 -r 10 -s 1024x768 -i /dev/video0 output'+str(currentDT)+'.avi')
```

Annexe III : Bash Script capture.sh :

```
#!/bin/bash
# Timelapse controller for USB webcam
DIR=frames
#counter for filename
n=1
#counter for files saved
x=1
#number of screenshots to take
num=30
#interval at which screenshots get taken in seconds
interval=0.2
]while [ $x -le $num ]; do
filename=$(printf "%05d.jpg" "$n")
let n=n+1
#Capture image
fswebcam -d /dev/video0 -S 3 -r 1280x720 --jpeg 70 --no-banner --save $DIR/$filename
x=$(( $x + 1 ))
sleep $interval;
done;
```


Annexe IV : Script python capture.py

```

import os
import datetime
import time
from threading import Thread
def picFunc():
    os.system("fswebcam -d /dev/video0 -r 1280x720 --no-banner frames/%s.jpeg" %datetime.datetime.utcnow().strftime("%Y-%m-%d-%H:%M:%S:%f")[:-3])
t=0.5 # initialise the pause between pictures in seconds
count=10 # initialise the number of pictures to be taken
i=1 # initialise (reset) the counting sequence
totalTime=(t*count) # Calculate the time in seconds
# Take a series of pictures one every t seconds
while (i<=count):
    # initialise variables
    leftTimeH=0
    leftTimeM=0
    leftTimeS=0
    # taking a picture by calling a command line prompt
    x=Thread(target=picFunc)
    x.start()
    totalTime=(t*(count-i)) # Calculate the time in seconds
    print (i) # print the current count value to show progress
    while (totalTime>=3600):
        leftTimeH=leftTimeH+1
        totalTime=totalTime-3600
    while (totalTime>=60):
        leftTimeM=leftTimeM+1
        totalTime=totalTime-60
    leftTimeS=totalTime
    percentDone=((i/count)*100)
    percentDone=round(percentDone,2)
    message1="Time left to finish " +repr(leftTimeH) + " Hours " + repr(leftTimeM) + " Minutes and " + repr(leftTimeS) + " Seconds"
    message2=""+repr(percentDone) + "% Completed!"
    print (message1)
    print (message2)
    i=i+1
    if (i>count): # leave the loop when count fulfilled (not really necessary)
        break
    time.sleep(t) # wait the defined time t(s) between pictures
print ("Finished!") # print to show when finished

```

ANNEXE V : Tableau de bord





HES-SO Valais	Report Summary														
Title	Machine learning pour un drone agricole avec un "companion computer"														
Student	Olivier Arbellay														
Period	30 / 04 / 2019 - 31 / 07 / 2019														
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Total
Analysis & Planification	1,50	3,50	0,50	1,00	1,00	1,00	0,00	1,00	2,00	0,00	1,00	0,00	0,00	0,00	12,50
Installation	0,00	4,00	2,50	0,00	0,00	0,00	0,00	0,00	1,00	8,00	8,00	3,00	0,00	0,00	26,50
Configuration	0,00	3,50	3,00	10,50	2,00	4,00	6,00	0,00	5,00	17,00	16,00	7,00	0,00	0,00	74,00
Programmation	0,00	1,00	1,00	0,00	1,00	10,00	1,00	0,00	10,00	0,00	5,00	28,00	5,00	0,00	62,00
Research / Lecture	12,50	9,00	4,00	4,50	7,00	5,00	4,00	3,00	8,00	12,00	2,00	2,00	0,00	0,00	73,00
Redaction	0,00	3,00	4,50	7,00	6,00	3,00	1,00	0,00	6,00	8,00	8,50	4,00	36,00	0,00	87,00
Course / working reports / works on data Test	0,00	0,00	1,00	0,00	5,00	0,00	0,00	0,00	0,00	1,00	4,00	3,00	7,00	19,00	40,00
Total	14,00	24,00	16,50	23,00	22,00	23,00	12,00	4,00	32,00	46,00	44,50	47,00	48,00	19,00	375,00

Annexe VI Product Backlog

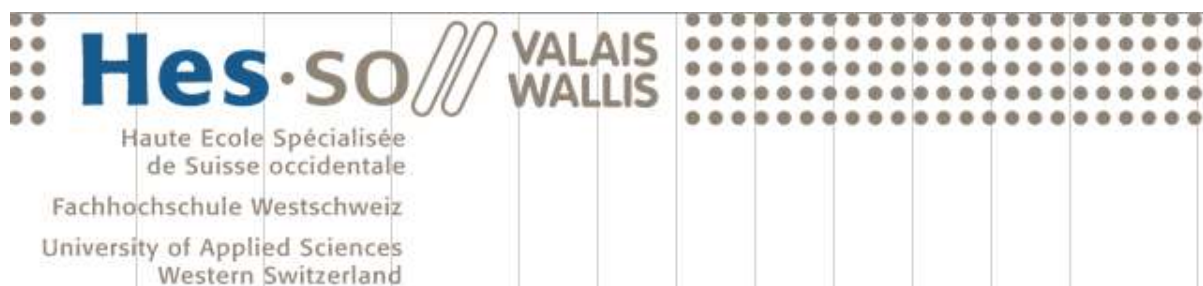
US Nr.	Theme	En tant que ...	je veux	afin que	Acceptance Criteria	Priority	Status	Story Points	Sprint	US accepted (done)	MoSCoW	Deadline
1	Preparatio	Developer	Préparer l'environnement de travail	travailler sans probleme		1000	●	-	0		Must	06.05.2019
2	Etude	Etudiant	pouvoir comprendre le fonctionnement des système d'exploitation des "Companion computer"	Je puisse établir la difficulté des tâches à venir	L'étudiant est capable d'expliquer le fonctionnement d'un drone	1100	●		0		Must	27.05.2019
3	Etude	Etudiant	pouvoir comprendre les communication entre un flight control unit et un companion computer	je puisse intervenir dans ces communications	L'étudiant est capable d'expliquer le fonctionnement des communication	1000	●		0		Must	27.05.2019
4	Pilote	Pilote	que mon drone s'arrête quand un obstacle est détecté	mon drone ne s'écrase pas	Les tests réels du drone sont accepté car le drone s'arrete	800	■		2		Should	
5	Pilotage	Pilote	que mon drone puisse éviter un obstacle détecté	mon drone ne s'écrase pas et qu'il continue son chemin	Les tests réels du drone sont accepté car le drone évite un obstacle	700	■		2		Should	
6	R&D	Ingénieur	pouvoir connaître la faisabilité de l'ajout d'un système de machine learnings sur un drone via un companion computer	je puisse développer les fonctionnalité souhaité de mon drone	un verdict est rendu sur les possibilités de réaliser du machine learning en live	900	■		1		Must	24.07.2019
7	R&D	Ingénieur	extraire les données d'une caméra	mettre en place un système de machine learning en live sur la caméra	un fichier vidéo est disponible pour analyse	890	■		1		Should	10.06.2019
8	Etude	Etudiant	pouvoir connaître les différents OS disponible sur le marché	je puisse les juger et établir un état de l'art	L'état de l'art est approuvé	950	■		1		Must	03.06.2019
9	Machine Learning	Etudiant	Etablir un dataset d'image vidéo pour le machine learning	je puisse entrainer mon modèle	Suffisamment d'image sont disponible pour le datasets	940	■		2		Must	
10	Machine Learning	Etudiant	Etablir un etat des connaissance actuelle en Reconnaissance d'image	je puisse m'orienter vers le bon système	L'étudiant est capable d'expliquer les différents systèmes	920	■		2		Must	
11	Machine Learning	Etudiant	Entrainer mon réseau de neurone	Tenter de la reconnaissance d'image	Le réseau est entraîné et capable de reconnaître des arbres	930	■		2		Must	
12	Machine Learning	Etudiant	Implémenter de la reconnaissance d'image sur un raspberry	de pouvoir évaluer les performances de détection en temps réel	Détection en temps réel possible	920	■		3			

ANNEXE VII : RAPPORT HEBDOMADAIRE


									
HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		30 / 04 / 2019 - 03 / 05 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification			1,50						1,50
Installation									0,00
Configuration									0,00
Programmation									0,00
Research / Lecture			3,50	7,00		2,00			12,50
Redaction									0,00
Course									0,00
Total		0,00	5,00	7,00	0,00	2,00	0,00	0,00	14,00
Working Report Details									
Date	Note								
30.04.2019	Réunion avec les partenaires du projet, discussion du TB, documentation à lire								
01.05.2019	Lecture de la documentation, essai du prototype avec tout le materiel								
03.05.2019	Lecture de documentation dronecode								
Comment									
Priorisation des étape de travail, démonstration sur les différents composants et récupération du matériel de travail(Raspberry, petit simulateur bricolé etc)									


 <p>Hes·SO VALAIS WALLIS Haute Ecole Spécialisée de Suisse occidentale Fachhochschule Westschweiz University of Applied Sciences Western Switzerland</p>									
HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		06 / 05 / 2019 - 10 / 05 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification		1,50	2,00						3,50
Installation			1,00	2,00			1,00		4,00
Configuration		2,00	1,50						3,50
Programmation							1,00		1,00
Research / Lecture		4,50	1,50	2,00			1,00		9,00
Redaction			1,00	2,00					3,00
Course									0,00
Total		8,00	7,00	6,00	0,00	0,00	3,00	0,00	24,00
Working Report Details									
Date	Note								
06.05.2019	Réunion avec Mr Genoud, lecture de documentation, installation de maverick sur une autre carte sd								
07.05.2019	Début de l'écriture du PB, des specifications. Test sur maverick (probleme avec le réseau)								
08.05.2019	Rédaction de l'état de l'art, test de récupération de vidéo live sur maverick, resolution probleme réseau								
11.05.2019	Test script python pour écrire des logs - Ros, pas concluant								
Comment									
Point sur l'avancement de la documentation, proposition d'étude du système Maverick									


HES-SO Valais		Weekly Report						
Title		Machine learning pour un drone agricole avec un "companion computer"						
Name		Olivier Arbellay						
Week		13 / 05 / 2019 - 17 / 05 / 2019						
	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification	0,50							0,50
Installation	1,00		1,00			0,50		2,50
Configuration	1,00	1,00	1,00					3,00
Programmation		1,00						1,00
Research / Lecture	1,00		2,00		1,00			4,00
Redaction	1,00	2,00	1,00			0,50		4,50
Course / Test		1,00						1,00
Total	4,50	5,00	5,00	0,00	1,00	1,00	0,00	16,50
Working Report Details								
Date	Note							
13.05.2019	Réunion avec Mr Genoud avancement ecriture, product backlog							
14.05.2019	Avancement de l'etat de l'art, test live du companion computer à chamoson, pas de crash ! Rencontre avec l'équipe d'aero 41.							
15.05.2019	Redaction de l'état de l'art partie batterie et communication , recherche des logs et export, installation du simulateur (marche pas).							
17.05.2019	Lecture sur le comportement d'un drone							
18.05.2019	Verification du product backlog, essai du simulateur (modification à faire sur machine virtuelle)							
Comment								
Point sur l'avancement de l'Etat de l'art, proposition d'expliquer les composants d'un drone car manque de connaissance et compréhension de ceux-ci. Proposition de User story pour le product backlog								





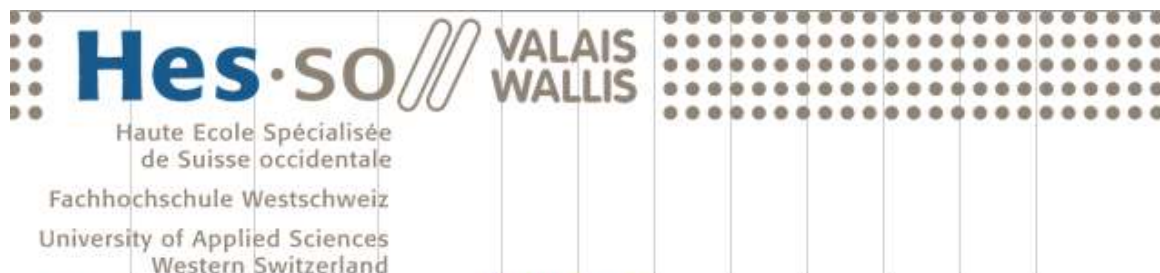
HES-SO Valais		Weekly Report						
Title	Machine learning pour un drone agricole avec un "companion computer"							
Name	Olivier Arbellay							
Week	20 / 05 / 2019 - 24 / 05 / 2019							
	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification	1,00							1,00
Installation								0,00
Configuration	3,00	3,00	4,00		0,50			10,50
Programmation								0,00
Research / Lecture	2,00	1,00	1,00		0,50			4,50
Redaction	2,00	3,00	2,00					7,00
Course								0,00
Total	8,00	7,00	7,00	0,00	1,00	0,00	0,00	23,00
Working Report Details								
Date	Note							
20.05.2019	Etat de l'art partie OS - Maverick, lecture log et information sur flytos, configuration réseau de maverick, Réunion avec Mr Genoud							
21.05.2019	Configuration machine virtuelle pour simulateur (ubuntu 18), rédaction etat de l'art communication							
22.05.2019	Configuration ubuntu, test simulateur docker, etat de l'art							
24.06.2019	Spécification, test d'un script python sur simulateur Flytos							
Comment								
Prochain objectif de la semaine: faire marché la caméra, synchroniser heure des test des logs, Réalisation d'un tableau comparatif des système pour l'état de l'art. Validation du Product Backlog								

																																																																																		
Haute Ecole Spécialisée de Suisse occidentale Fachhochschule Westschweiz University of Applied Sciences Western Switzerland																																																																																		
Weekly Report																																																																																		
Title	Machine learning pour un drone agricole avec un "companion computer"																																																																																	
Name	Olivier Arbella																																																																																	
Week	27 / 05 / 2019 - 31 / 05 / 2019																																																																																	
	<table border="1"> <thead> <tr> <th></th> <th>Mo</th> <th>Tu</th> <th>We</th> <th>Th</th> <th>Fr</th> <th>Sa</th> <th>So</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td>Analysis & Planification</td> <td>1,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1,00</td> </tr> <tr> <td>Installation</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>0,00</td> </tr> <tr> <td>Configuration</td> <td>1,00</td> <td>1,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>2,00</td> </tr> <tr> <td>Programmation/ test</td> <td></td> <td>1,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1,00</td> </tr> <tr> <td>Research / Lecture</td> <td>3,00</td> <td>3,00</td> <td>1,00</td> <td></td> <td></td> <td></td> <td></td> <td>7,00</td> </tr> <tr> <td>Redaction</td> <td>4,00</td> <td>2,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>6,00</td> </tr> <tr> <td>Course / test</td> <td></td> <td></td> <td>5,00</td> <td></td> <td></td> <td></td> <td></td> <td>5,00</td> </tr> <tr> <td>Total</td> <td>9,00</td> <td>7,00</td> <td>6,00</td> <td>0,00</td> <td>0,00</td> <td>0,00</td> <td>0,00</td> <td>22,00</td> </tr> </tbody> </table>		Mo	Tu	We	Th	Fr	Sa	So	Total	Analysis & Planification	1,00							1,00	Installation								0,00	Configuration	1,00	1,00						2,00	Programmation/ test		1,00						1,00	Research / Lecture	3,00	3,00	1,00					7,00	Redaction	4,00	2,00						6,00	Course / test			5,00					5,00	Total	9,00	7,00	6,00	0,00	0,00	0,00	0,00	22,00
	Mo	Tu	We	Th	Fr	Sa	So	Total																																																																										
Analysis & Planification	1,00							1,00																																																																										
Installation								0,00																																																																										
Configuration	1,00	1,00						2,00																																																																										
Programmation/ test		1,00						1,00																																																																										
Research / Lecture	3,00	3,00	1,00					7,00																																																																										
Redaction	4,00	2,00						6,00																																																																										
Course / test			5,00					5,00																																																																										
Total	9,00	7,00	6,00	0,00	0,00	0,00	0,00	22,00																																																																										
Working Report Details																																																																																		
Date	Note																																																																																	
27.05.2019	Ecriture etat de l'art, recherche sur enregistrement de video sur raspberry																																																																																	
28.05.2019	Ecriture specification et tableau comparatif, script python pour record																																																																																	
29.05.2019	Amelioration script python, test d'enregistrement de terrain à Chamoson																																																																																	
Comment																																																																																		
Point sur l'état de l'art partie OS, proposition de commencer à écrire introduction. Réalisation prochaine d'un script pour enregistrer des vidéos en vol. Remplacement du Product backlog par journal de bord, moins difficile à juger																																																																																		

 <p>Haute Ecole Spécialisée de Suisse occidentale Fachhochschule Westschweiz University of Applied Sciences Western Switzerland</p>									
HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		03 / 06 / 2019 - 07 / 06 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification		1,00							1,00
Installation									0,00
Configuration		1,00	3,00						4,00
Programmation			4,00	6,00					10,00
Research / Lecture		3,00	1,00	1,00					5,00
Redaction		2,00		1,00					3,00
Course									0,00
Total		7,00	8,00	8,00	0,00	0,00	0,00	0,00	23,00
Working Report Details									
Date	Note								
04.05.2019	Redaction etat de l'art + lecture documentation dronecode, point sur l'avancement								
05.05.2019	Probleme connexion ssh raspberry, script								
06.05.2019	Réalisation du script pour prendre des photo chaque 0.5 seconde sur raspberry, redaction procédure data								
Comment									
Point sur les données récoltées, décisions de prendre des images plutôt que des vidéos. Point sur l'état de l'art et choix du dernier système à analyser plus en profondeur									

																																																																																		
Haute Ecole Spécialisée de Suisse occidentale Fachhochschule Westschweiz University of Applied Sciences Western Switzerland																																																																																		
Weekly Report																																																																																		
Title	Machine learning pour un drone agricole avec un "companion computer"																																																																																	
Name	Olivier Arbella																																																																																	
Week	10 / 06 / 2019 - 14 / 06 / 2019																																																																																	
	<table border="1"> <thead> <tr> <th></th> <th>Mo</th> <th>Tu</th> <th>We</th> <th>Th</th> <th>Fr</th> <th>Sa</th> <th>So</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td>Analysis & Planification</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>0,00</td> </tr> <tr> <td>Installation</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>0,00</td> </tr> <tr> <td>Configuration</td> <td></td> <td>3,00</td> <td>3,00</td> <td></td> <td></td> <td></td> <td></td> <td>6,00</td> </tr> <tr> <td>Programmation</td> <td>1,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1,00</td> </tr> <tr> <td>Research / Lecture</td> <td>2,00</td> <td>2,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>4,00</td> </tr> <tr> <td>Redaction</td> <td></td> <td>1,00</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1,00</td> </tr> <tr> <td>Course</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>0,00</td> </tr> <tr> <td>Total</td> <td>3,00</td> <td>6,00</td> <td>3,00</td> <td>0,00</td> <td>0,00</td> <td>0,00</td> <td>0,00</td> <td>12,00</td> </tr> </tbody> </table>		Mo	Tu	We	Th	Fr	Sa	So	Total	Analysis & Planification								0,00	Installation								0,00	Configuration		3,00	3,00					6,00	Programmation	1,00							1,00	Research / Lecture	2,00	2,00						4,00	Redaction		1,00						1,00	Course								0,00	Total	3,00	6,00	3,00	0,00	0,00	0,00	0,00	12,00
	Mo	Tu	We	Th	Fr	Sa	So	Total																																																																										
Analysis & Planification								0,00																																																																										
Installation								0,00																																																																										
Configuration		3,00	3,00					6,00																																																																										
Programmation	1,00							1,00																																																																										
Research / Lecture	2,00	2,00						4,00																																																																										
Redaction		1,00						1,00																																																																										
Course								0,00																																																																										
Total	3,00	6,00	3,00	0,00	0,00	0,00	0,00	12,00																																																																										
Working Report Details																																																																																		
Date	Note																																																																																	
10.05.2019	Test d'un script sur simulateur avec gazebo machine virtuelle, crash de celui-ci																																																																																	
11.05.2019	Reprise à 0 de l'installation de la machine virtuelle, crash de celle-ci																																																																																	
12.05.2019	fin de l'installation de la machine, ça rame vraiment vilain et impossible de faire quoi que ce soit lorsque on lance le simulateur(traitement d'image 3d) On va essayer de faire une pci passthrough sur vmware. Ca marche pas sur vmware, on essaie sur vbox																																																																																	
Comment																																																																																		
Point sur la situation personnel. Mise en place des simulateur à effectué, Proposition d'une première version de l'état de l'art partie OS, et Choix technologique: Dronecode/ROS																																																																																		

									
Haute Ecole Spécialisée de Suisse occidentale Fachhochschule Westschweiz University of Applied Sciences Western Switzerland									
HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		17 / 06 / 2019 - 21 / 06 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification		1,00							1,00
Installation									0,00
Configuration									0,00
Programmation									0,00
Research / Lecture		1,00	2,00						3,00
Redaction									0,00
Course									0,00
Total		2,00	2,00	0,00	0,00	0,00	0,00	0,00	4,00
Working Report Details									
Date	Note								
17.05.2019	Révision de l'état de l'art								
18.05.2019	Vérification de pourquoi la machine virtuelle ram avec Jon, pci passthrough pas supporté sur vbox								
Comment									
Revision de l'état de l'art avec les partenaires du projet. Cette semaine sera dédié à l'étude des examens et donc je ne devrais pas travailler sur le bachelor cette semaine, et reconstruction personnelle									



HES-SO Valais	Weekly Report
Title	Machine learning pour un drone agricole avec un "companion computer"
Name	Olivier Arbellay
Week	24 / 06 / 2019 - 28 / 06 / 2019

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification		2,00						2,00
Installation			1,00					1,00
Configuration			4,00	1,00				5,00
Programmation		2,00		6,00	2,00			10,00
Research / Lecture		2,00	3,00	2,00	1,00			8,00
Redaction		2,00			4,00			6,00
Course								0,00
Total	0,00	8,00	8,00	9,00	7,00	0,00	0,00	32,00

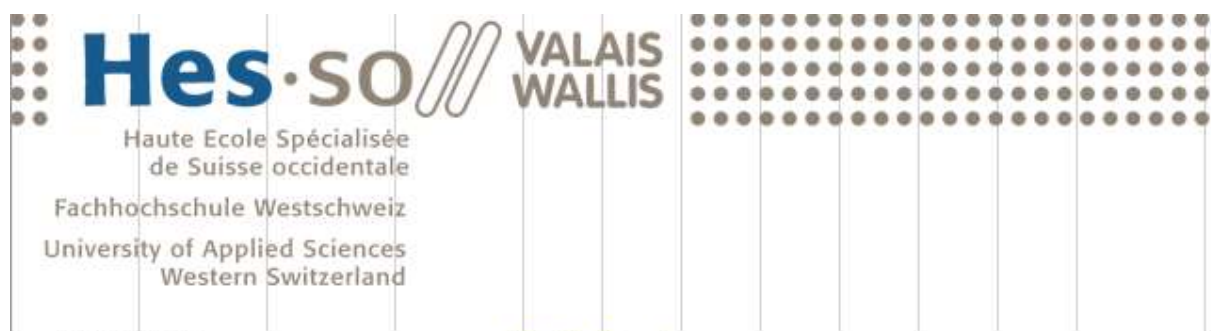
Working Report Details	
Date	Note
24.05.2019	Entretien personnel + jour de conger pour changer d'air
25.05.2019	Correction de l'etat l'art en accord avec ce qu'a dit Doms. Point sur l'etat actuel (moral et travail), quelle seront les prochaines étapes etc. Parti chercher mon Intel Nuc pour mettre le simulateur+ installation. CA MARCHE ET SA RAM PAS !, Rédaction de l'etat de l'art et Recherche pour la reconnaissance d'image (Recherche de tutorial + ce que Calixte m'a donné : "Salut, Voici quelques points de départs pour faire cette détection d'arbres - Un algo de détection d'objets : https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/ - Simple object tracking : https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/ C'est des points de départ. Je te conseille de chercher d'autres méthodes, il existe sans doute plein d'autres variantes, mais ca devrais toujours tourner autour de « détection » et « tracking »"
26.05.2019	variantes, mais ca devrais toujours tourner autour de « détection » et « tracking »"
27.05.2019	Tutorial d'utilisation d'opencv + reconnaissance d'image
28.05.2019	redaction etat de l'art machine learning + test de reconnaissance d'image avec réseaux neuronal yolo

Comment
 Etant donné que l'etat de l'art partie OS est finie, nous établirons un point sur l'etat actuelle de reconnaissance d'image embarqué et générale. Cette semaine, apprentissage de l'utilisation d'openCV et de Début de test sur la reconnaissance d'image.


HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		01 / 07 / 2019 - 05 / 07 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification									0,00
Installation				2,00	5,00	1,00			8,00
Configuration			1,00	7,00	5,00	4,00			17,00
Programmation									0,00
Research / Lecture		5,00	6,00		1,00				12,00
Redaction		3,00	1,00			4,00			8,00
Working Report Details ou annotation, travail sur donnée			1,00						1,00
Total		8,00	9,00	9,00	11,00	9,00	0,00	0,00	46,00
Working Report Details									
Date	Note								
01.07.2019	Lecture et compréhension réseaux de neurone, rédaction état de l'art								
02.07.2019	Lecture des tutoriels pour entrainer un réseau de neurone YOLO il faut utiliser un framework en c++ qui s'appelle darknet, 1h de rédaction et doc photo								
03.07.2019	Configuration de darknet et build, ne marche pas car il faut opencv à installer, le git de darknet pris étant celui de la version linux, il m'a fallu reprendre de 0 la configuration, du coup je vais aussi essayer de faire la config avec Cuda pour utiliser la carte graphique								
04.07.2019	Installation de la version 10.1 de Cuda, Opencv installation et build avec cmake, n'ayant jamais appris à réaliser des build en c++ il m'a fallu un moment avant de réaliser comment faire car plein de probleme lors de build, du coup lecture de comment ça marche etc. Pendant que les build s'effectue, j'ai fait le tri des photo qu'il me fallait d'arbre pour entrainer le réseau de neurone + les annotations (métadata, position de l'arbre etc) à faire sur des fichier txt								
05.07.2019	Le build Opencv ne marche pas car j'avais pas les bonne version correspondante des contrib... fin de matinée, le build sur cmake fonctionne par contre sur visual studio, erreur ! Rédaction/fin de l'état de l'art, rédaction du commencement de la partie machine learning								
Comment									
Etant donné l'efficacité du réseaux de neurone YOLOv3, on va essayer d'utiliser le framework darknet pour l'entrainer. Commencement de l'annotations des images en fonction du réseaux. Idéalement il faudrait terminer la partie machine learning à la fin de la semaine prochaine pour réaliser l'implémentation sur Raspberry rapidement									

HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		08 / 07 / 2019 - 12 / 07 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification			1,00						1,00
Installation		4,00	3,00			1,00			8,00
Configuration		6,00	5,00	3,00		2,00			16,00
Programmation			1,00	4,00					5,00
Research / Lecture						2,00			2,00
Redaction					8,00	0,50			8,50
Working Report Details ou annotation, travail sur donnée				3,00	1,00				4,00
Total		10,00	10,00	10,00	9,00	5,50	0,00	0,00	44,50
Working Report Details									
Date	Note								
08.07.2019	Erreur lors de la compilation de Darknet, rédaction de la doc et test de l'installation d'opencv, darknet ne peut pas être utilisé sans que Opencv soit fini de compilé. Je me suis gouré et j'ai mis en 32bit au lieu de 64... pour opencv et darknet... pas de jugement svp je pouvais pas savoir								
09.07.2019	Compilation toute la nuit d'opencv et des librairies associé, build effectué mais pas dans le bon répertoire, du coup je fais la compilation avec ninja et c'est bon! Point avec dominique sur la marche suivre des prochaines semaine. On va faire un test prototype le 22 juillet. Compilation de darknet avec en 64 cette fois, compilation réussie avec opencv et darknet, cmake et visual studio, mais reste des erreur lors de l'exécution de darknet.exe. Il manque des dll, et je sais pas pourquoi. configuration des couches du réseau de neurone, séparation des images de test et d'entraînement+ ajout d'image pris d'un drone de jérôme, il m'a fallu faire des scripts python pour prendre des images chaque 5 sec, les resize...								
10.07.2019	Remplacement des var environnement pour opencv, et changement de répertoire d'une dll, et la compilation et le programme fonctionne ! Test de l'entraînement de réseau de neurone avec des images test et ça marche + ultra rapide grâce à CUDA !! Nouvelle annotations des images afin d'avoir plus d'arbres pour entraîner le réseau de neurone (j'ai supprimer les ensemble déjà fait car en ajoutant des images, j'ai pas vu qu'elle était en png ...) recommencement annotation ! Entraînement de tiny-yolo (plus léger pour raspberry) avec ma classe d'objet custom ! Test du réseaux nouvellement entraîner ... et CA MARCHE, JE RECONNAIS DES ARBRES, et meme sur video !!!!								
11.07.2019	Rédaction toute la journée partie data mining + point sur l'état des chose avec dominique et choix des étapes de à suivre et comportement du drone lors de reconnaissance d'arbre. On part sur des bruits lors de reconnaissance et arret lors que la frame prend la moitié de l'image (pour un echange avec le drone.) pour la suite il faudrait rajouter des capteur pour la distance, ou calculer l'accélération								
12.07.2019	Rdv médiation, Recherche sur l'implémentation de reconnaissance d'image sur le raspberry et de yolo sur raspberry, flash de la carte sd, installation des composant sur le raspberry, darknet, darknet allégé								
Comment									
Choix de la date de test du prototype: 22 juillet, Présentation de l'entraînement de neurone et de la reconnaissance d'arbre. Choix du comportement du drone: essayer d'envoyer des sons à la ground control quand le drone reconnait un arbre puis stop et hold en cas de frame supérieur									

HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbellay							
Week		15 / 07 / 2019 - 19 / 07 / 2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification									0,00
Installation			2,00			1,00			3,00
Configuration		3,00		2,00	2,00				7,00
Programmation		2,00	4,00	3,00	4,00	5,00	4,00	6,00	28,00
Research / Lecture		1,00	1,00						2,00
Redaction		1,00		2,00		1,00			4,00
Working Report Details			1,00	1,00	1,00				3,00
Total		7,00	8,00	8,00	7,00	7,00	4,00	6,00	47,00
Working Report Details									
Date	Note								
17.07.2019	Implémentation de la reconnaissance d'image sur raspberry, avec un réseau préentraîné, puis changement du réseau de neurone. Erreur du code utilisé, problème avec le réseau de neurone. Réécriture et erreur d'orthographe.								
18.07.2019	Changement de code à implémenter, car il n'accepte pas le réseau de neurone, impossible de savoir pourquoi. Le nouveau code demande d'utiliser opencv 3.4, que je n'ai pas, installation 3x parce que j'arrive pas en un coup comme d'hab. Build pendant la nuit de celui-ci.								
19.07.2019	Code fonctionne ! Et utilise le réseau de neurone entraîné ! Et reconnaît des arbres ! Redaction de la partie yolo								
20.07.2019	Création du nœud ROS en se basant sur le code de démo, impossible de le faire compiler, et simulateur qui marche plus, essai sur sitl, marche pas non plus								
21.07.2019	Test d'utilisation de mavros, marche pas non plus ... je sais vraiment pas coder en c++ , changement pour dronekit et mavproxy. Installation de dronekit. Impossible de se connecter+ simulateur qui marche pas avec dronekit								
22.07.2019	Refonte du code de la reconnaissance d'image pour l'adapter avec dronekit								
23.07.2019	Développement avec l'aide de Mr Genoud, utilisation de dronekit pour se connecter au px4								
Comment									
Point sur l'avancement de l'implémentation sur raspberry OK, plus que la logique pour stopper le drone à code. Après le test, écriture du reste de la documentation									



HES-SO Valais		Weekly Report						
Title	Machine learning pour un drone agricole avec un "companion computer"							
Name	Olivier Arbella							
Week	22 / 07 / 2019 - 26 / 07 / 2019							
	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0,00
Installation								0,00
Configuration								0,00
Programmation	5,00							5,00
Research / Lecture								0,00
Redaction		8,00	9,00	5,00	9,00	2,00	3,00	36,00
Working Report Details/ test	2,00			5,00				7,00
Total	7,00	8,00	9,00	10,00	9,00	2,00	3,00	48,00
Working Report Details								
Date	Note							
22.07.2019	Implémentation du code pour changer de mode sur écoute du switch, puis test en vol, changement de mode validé, reconnaissance d'arbre puis mode loiter marche !							
23.07.2019	Rédaction de la thèse, partie machine learning implémentation							
24.07.2019	Rédaction de la thèse, correction de fautes et modification							
25.07.2019	Rédaction de la thèse Introduction et problématique, remerciement							
26.07.2019	Rédaction de la thèse, Conclusion							
	Poster							
Comment								
Rédaction								

 <p>Haute Ecole Spécialisée de Suisse occidentale Fachhochschule Westschweiz University of Applied Sciences Western Switzerland</p>									
HES-SO Valais		Weekly Report							
Title		Machine learning pour un drone agricole avec un "companion computer"							
Name		Olivier Arbella							
Week		29/07/2019 - 31/07/2019							
		Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification									0,00
Installation									0,00
Configuration									0,00
Programmation									0,00
Research / Lecture									0,00
Redaction									0,00
Working Report Details		10,00	9,00						19,00
Total		10,00	9,00	0,00	0,00	0,00	0,00	0,00	19,00
Working Report Details									
Date	Note								
29.07.2019	Correction des modifs et mise en page poster								
30.07.2019	Mise en page + référence, changement de dernier minutes et annexe								
31.07.2019	Rendu								
Comment									
Correction de TOUT + Rendu									

DÉCLARATION DE L'AUTEUR

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Dominique Genoud

- Professeur HES

- HES-SO // Valais