

RAPPORT TECHNIQUE

Démonstrateur d'un système de défense anti-drone pour
les véhicules de l'armée de terre

Février 2024 - Juillet 2024



Hector de Turckheim
hector.deturckheim@gmail.com

SOMMAIRE

Sujet	3
Description du projet.....	3
Présentation Hardware.....	3
Présentation de l'architecture du code.....	6
Description des fichiers principaux.....;	7
detectnet.py.....	7
angle_calculations.py.....	7
angle_display.py.....	9
Informations pratiques.....	11
Enregister de nouvelles images.....	11
Tester les fonctions de calcul angulaire.....	11
Installation embarqué.....	12
Annoter un dataset.....	14
Entraîner une nouvelle IA.....	14
Le drone.....	15
Limitations et pistes d'amélioration.....	16
Annexes.....	18

SUJET

La finalité de ce projet est la conception d'un démonstrateur d'un système de défense anti-drone étant destiné à être équipé sur les véhicules de l'armée de terre française. Concrètement, le démonstrateur doit être capable, à l'aide d'une caméra haute vitesse et d'un objectif œil de poisson circulaire (angle de vue supérieur ou égal à 180°), de détecter tous les drones entrant dans l'espace aérien du véhicule à défendre et de calculer les coordonnées de leurs points gisement et source. Ces coordonnées seront transmises à une arme automatique qui s'occupera de neutraliser la menace. La gestion de l'arme automatique ne fait en principe pas partie du projet, même si une caméra PTZ (Pan Tilt Zoom) devant mimer le comportement d'une arme automatique a été connectée à la fin du stage afin de rajouter un aspect plus visuel au démonstrateur.

DESCRIPTION DU PROJET (fin juillet 2024)

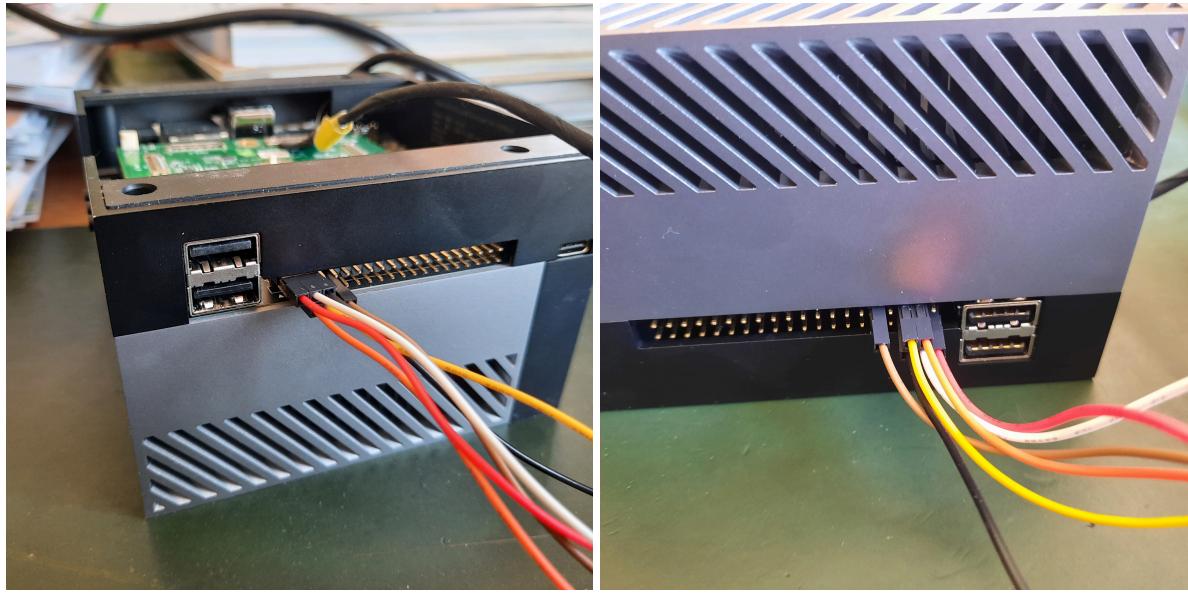
Présentation du matériel et de son installation

Le système actuel (fin juillet 2024) est composé d'un kit de développement de la carte **Jetson AGX Orin de 64GB** (de la marque Nvidia), et d'une caméra **e-CAM56 CUOAGX** (de la marque e-con) de dimension 2432 x 2048 pixels. La caméra est connectée à la carte Jetson avec un câble IPEX et via un adaptateur fourni par le fabricant de la caméra qui se fixe sur le kit.

En plus de cela, une **caméra PTZ IMX477 de Arducam** est branchée sur le kit de développement via le header de 40 branches de ce dernier. Voici comment s'effectuent les branchements :

- câble rouge ⇒ 5V (Servo Vcc) ⇒ PIN 2 et 4
- câble noir ⇒ GND (Servo) ⇒ PIN 6, 9, 14, 20, 25, 30, 34, 39
- câble blanc ⇒ 5V (Caméra) ⇒ PIN 2 et 4
- câble marron ⇒ GND (Caméra) ⇒ PIN 6, 9, 14, 20, 25, 30, 34, 39
- câble jaune ⇒ liaison I2C Clock ⇒ PIN 5 (Bus 7) et 28 (Bus 1)
- câble orange ⇒ liaison I2C Data ⇒ PIN 3 (Bus 7) et 27 (Bus 1)

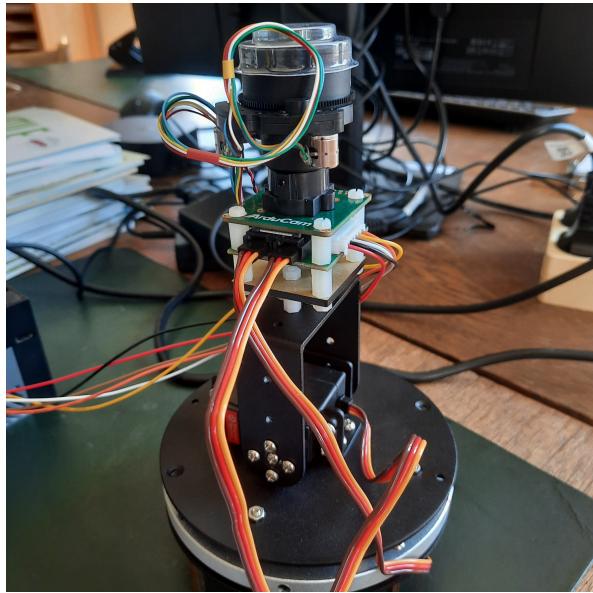
Vous pouvez également vous référer aux images ci-dessous pour effectuer vos branchements.



N.B : Le PIN 1 de la carte Jetson peut être repéré visuellement grâce à une flèche blanche qui indique sa position.



Pour ce qui est des servomoteurs du support PT, branchez le servomoteur panoramique (celui du dessous) sur la prise de gauche de la carte de la caméra, et le servomoteur d'élévation (celui au-dessus) sur la prise de droite. Attention, dans les deux cas, la couleur des câbles de connexion doit suivre l'enchaînement suivant (de gauche à droite) : marron, rouge, jaune (voir l'image ci-dessous)



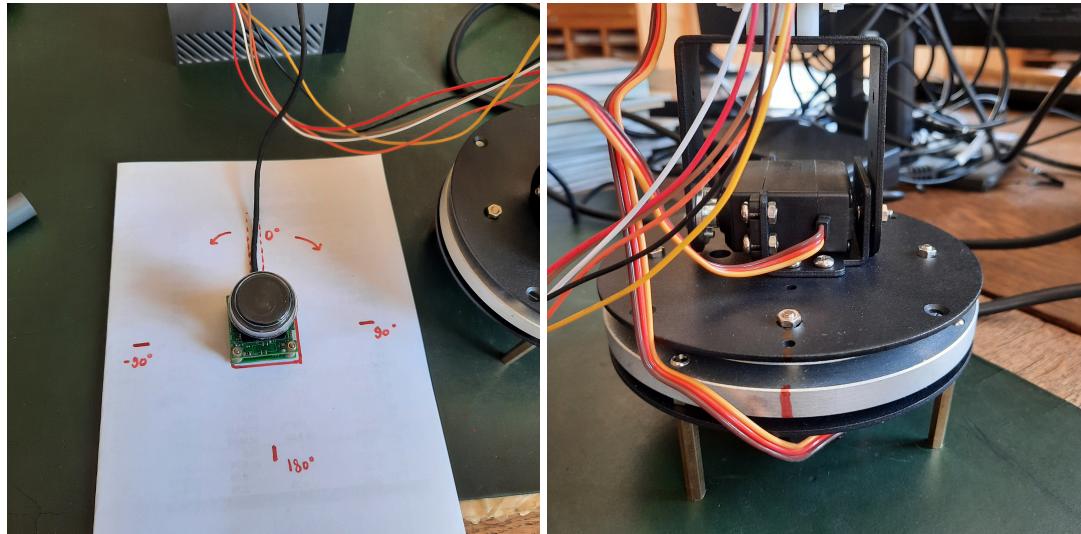
Pour plus d'informations concernant la disposition des pins de la carte Jetson, consultez le lien ci-dessous :

<https://jetsonhacks.com/nvidia-jetson-agx-orin-gpio-header-pinout/>

Pour plus d'information quant à l'installation de la caméra PTZ, consultez le lien ci-dessous :

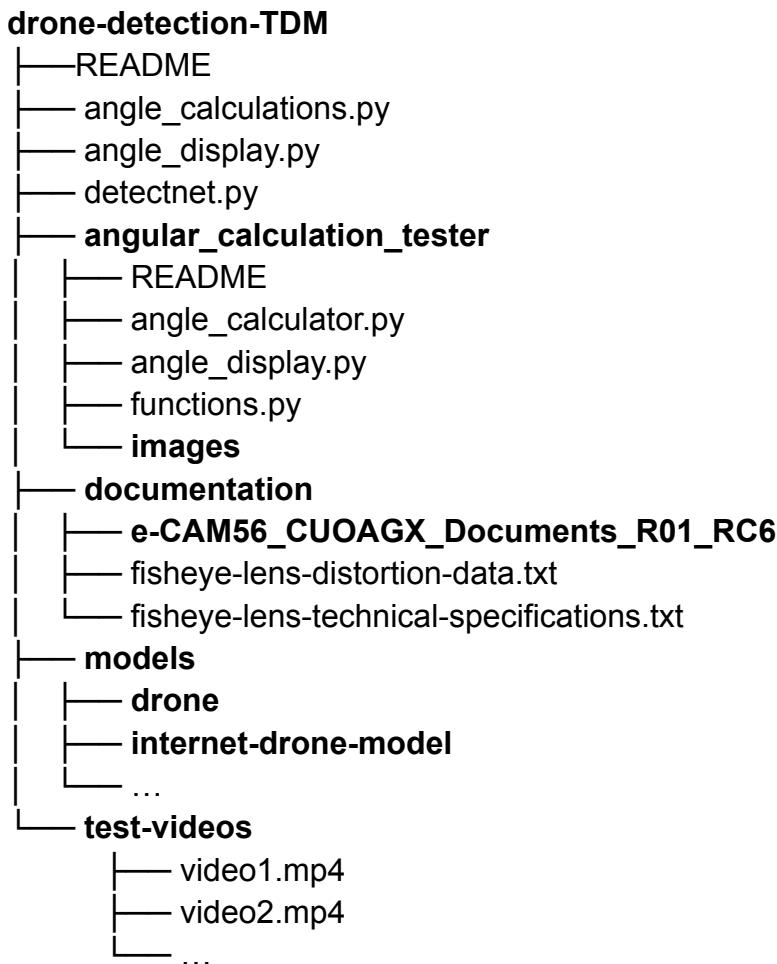
<https://docs.arducam.com/Nvidia-Jetson-Camera/Pan-Tilt-Zoom-Camera/quick-start/>

Enfin, notez qu'il est important d'orienter dans la même direction la caméra œil de poisson et la caméra PTZ afin de maintenir une cohérence entre les angles des objets détectés (par rapport à la caméra œil de poisson) et les mouvements du support PTZ. Pour la caméra œil de poisson, l'avant est indiqué par l'arrivée du câble IPEX (voir l'image de gauche ci-dessous). Pour la caméra PTZ, l'avant est symbolisé par un trait rouge dessiné au feutre (voir l'image de droite ci-dessous). Prenez garde pour la caméra PTZ à ce que la rotation panoramique soit bien nulle.



Présentation de l'architecture du code

Voici l'architecture logicielle du projet :



- Les fichiers `detectnet.py`, `angle_calculations.py` et `angle_display.py` sont les fichiers principaux du projet. La boucle d'acquisition d'images et les détections sont gérés dans le fichier `detectnet.net`; les coordonnées angulaires des objets détectés sont calculées et affichées respectivement dans les fichiers `angle_calculations.py` et `angle_display.py`.
- **angular_calculation_tester** contient les fichiers permettant de tester les fonctions de calcul de coordonnées angulaires (voir page 12 : tester les fonctions de calcul angulaire).
- **documentation** : documentation concernant la caméra et la lentille oeil de poisson.
- **models** contient les modèles d'IA nécessaires pour la détection des drones.
- **test-videos** : contient des vidéos de tests prises avec l'objectif oeil de poisson.
- **README** contient des informations pratiques concernant le projet (lancement, *requirements*, etc.).

Description détaillée des fichiers principaux

Detectnet.py

C'est ici que se trouve la colonne vertébrale du projet, à savoir la boucle d'acquisition d'images. Celle-ci se décompose en 6 sous parties :

- 1) L'acquisition : le script charge une image de la caméra.
- 2) La correction de l'input : La résolution de la caméra étant rectangulaire (2432x2048), de grandes bandes noires apparaissent à droite et à gauche de l'image lorsque l'on fixe un objectif oeil de poisson circulaire sur la caméra. Ces bandes étant inutiles, il convient de s'en débarrasser en rognant les bords pour redimensionner l'image à un format carré (2048x2048) englobant parfaitement la prise de vue circulaire de l'objectif.
- 3) La détection : les images sont confiées au réseau de neurones pour effectuer d'éventuelles détections.
- 4) Les calculs : une fois les détections effectuées, les coordonnées angulaires de chaque objet sont calculées puis affichées en parallèle dans une autre fenêtre.
- 5) La redimension de l'output : une fois les détections effectuées, le résultat (l'image d'entrée + les boîtes de détections) est redimensionné pour pouvoir tenir dans une fenêtre plus petite.
- 6) L'affichage : à la fin, les images sont simplement affichées dans une fenêtre afin de pouvoir suivre l'exécution du programme.

En parallèle de l'acquisition d'images se déroule le calcul des coordonnées angulaires de chaque drone détecté juste après la partie 3 décrite ci-dessus (nous verrons le détail de ces calculs dans la partie suivante – angle_calculation.py).

Pour afficher le résultat de ces calculs, nous utilisons une classe Python afin de manipuler la fenêtre d'affichage de manière simplifiée (voir la partie angle_display.py).

Enfin, pour ce qui est de la connexion et de la communication avec le support PTZ, une fonction a été implémentée : *angle2PTZ*. Cette dernière se charge de convertir les coordonnées angulaires des différents objets détectés pour les faire correspondre au référentiel de la caméra PTZ, avant de les lui transmettre.

N.B : Le README du dossier contient davantage d'informations concernant l'utilisation de ce programme. Je vous y renvoie pour toute information complémentaire.

angle_calculations.py

C'est ici que sont calculés les angles d'élévation et d'azimut des drones détectés. Les deux fonctions présentes dans ce script ne sont que l'application des deux calculs mathématiques présentés ci-dessous. Gardez à l'esprit (notamment pour le calcul de l'angle d'élévation) que les calculs présentés ci-dessous ne sont que des estimations basées sur des modèles théoriques et des mesures étales approximatives. De plus amples recherches seraient nécessaires afin de s'approcher d'un modèle plus précis, donc plus fidèle à la réalité.

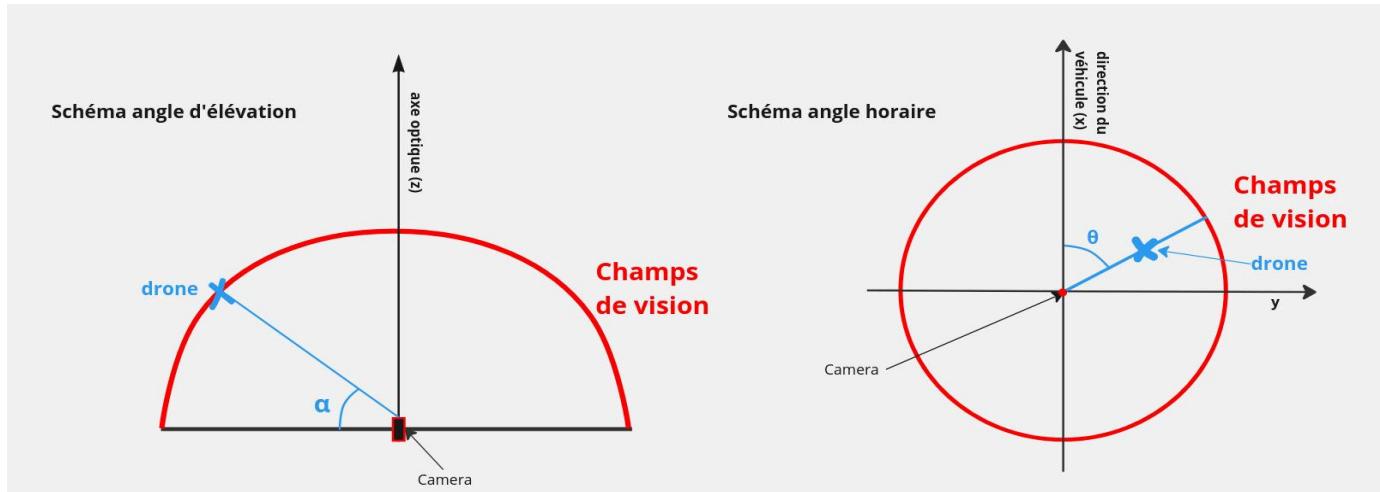


Schéma représentant les angles horaire et d'élévation

Pour calculer l'azimut d'un objet dont les coordonnées sont (x, y) sur une image circulaire de taille (*width x height*), la méthode est assez simple.

On commence par calculer les coordonnées du centre de l'image :

$$C_x = \frac{\text{width}}{2}, C_y = \frac{\text{height}}{2}$$

On calcule ensuite la différence de coordonnées entre le centre et l'objet en question :

$$\Delta x = x - C_x, \Delta y = y - C_y$$

On utilise ensuite la fonction *atan2* pour calculer l'angle en radians entre la partie positive de l'axe des ordonnées et le point de coordonnées (x, y) :

$$\alpha_{\text{radians}} = \text{atan2}(\Delta x, \Delta y)$$

Il suffit ensuite de convertir cet angle en degrés et le tour est joué.

Pour calculer l'élévation, la méthode est légèrement plus compliquée.

Comme précédemment, on commence par calculer les coordonnées du centre de l'image ainsi que la différence de coordonnée ($\Delta y, \Delta x$) entre le centre de l'image et l'objet. On utilise ensuite le résultat pour calculer la distance euclidienne entre le centre de l'image et l'objet détecté :

$$r = \sqrt{\Delta x^2 + \Delta y^2}$$

Intervient ensuite la *mapping function* de l'objectif œil de poisson. La *mapping function* d'un objectif permet de calculer la position r d'un objet par rapport au centre de l'image déformée, avec la distance focale f et l'angle par rapport à la partie positive de l'axe optique θ (en radians). Dans notre cas précis, la *mapping function* dite 'équidistante' semble la plus appropriée :

$$r = f\theta$$

En remenant pour isoler θ , on obtient :

$$\theta = \frac{r}{f}$$

En convertissant cet angle en degrés et en calculant son complémentaire à 90° , on obtient l'élévation par rapport au sol de l'objet détecté.

Dans la pratique, on applique aussi un coefficient de correction α afin de se rapprocher le plus possible de mesures étalons effectués à la main afin de modéliser efficacement la déformation de l'objectif œil de poisson.

angle_display.py

angle_display.py contient la classe RealTimeAnglePlotting qui, comme son nom l'indique, est responsable d'afficher en temps réel les angles d'azimut et d'élévation dans une fenêtre séparée.

Dans le fichier detectnet.py, la classe "RealTimeAnglePlotting" est instanciée dans la variable *rtpA*. Les fonctions d'initialisations *__init__* et *initialize_plot* sont donc appelées pour configurer la liste des angles d'azimut et d'élévation (*self.angles* – une liste de tuple, un tuple par objet détecté), la fenêtre d'affichage (*self.fig* et *self.ax*) ainsi que l'animation *FuncAnimation* pour continuellement mettre à jour les valeurs de la figure en appelant la méthode *update* toutes les 50ms (*self.ani*).

La méthode *update* appelle la fonction *update_plot* qui pour chaque tuple dans la liste *self.angles*, calculera la direction du vecteur défini par les angles d'azimut et d'élévation via la fonction *calculate_direction_vector*, et affichera ledit vecteur sur la

fenêtre d'affichage via la fonction `plot_line`. En plus de cela, la fonction `update_plot` s'occupe également d'afficher les axes X,Y,Z afin de pouvoir mieux interpréter les résultats. L'axe X est dessiné en rouge afin de le différencier avec l'axe Y, et c'est lui qui symbolise la direction du système.

Dans le script `detectnet.py`, la fonction `update_3d_visualization(angles)` est défini pour appeler la méthode `set_angle` de la classe “*RealTimeAnglePlotting*” via l'instance `rtpA` pour modifier la valeur des angles à être affiché. Elle est appelée à chaque tour de boucle, juste après que le calcul des coordonnées angulaires ait été effectué pour tous les drones détectés.

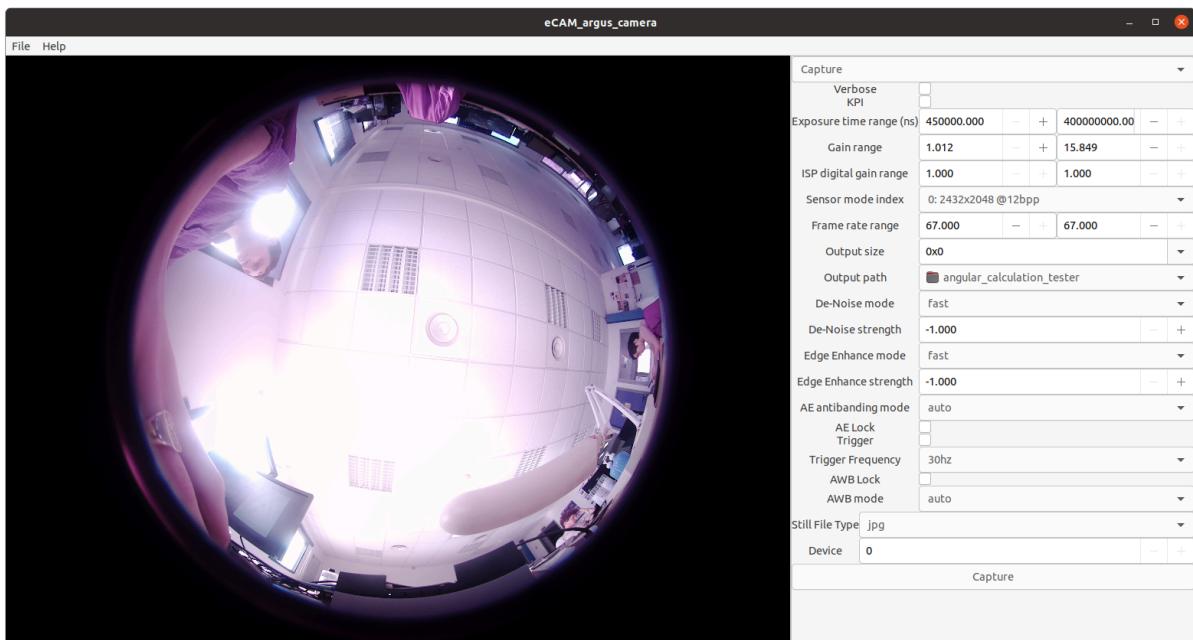
Enfin, un thread est créé au début du script `detectnet.py` afin d'appeler la boucle principale définie dans la fonction `main_loop`, afin d'afficher les résultats de toutes les opérations définies ci-dessus sans bloquer le processus principal¹.

¹ La raison qui nous pousse à reléguer au thread l'exécution de la boucle principale au lieu de l'affichage des angles (ce qui aurait été plus logique structurellement parlant) vient du fait qu'il ne soit pas possible d'exécuter `plt.show()` dans un thread.

INFORMATIONS PRATIQUES

Enregistrer de nouvelles images

Pour enregistrer de nouvelles images, deux solutions s'offrent à vous : utiliser les fonctions de gstreamer dans votre terminal (cf. la documentation de gstreamer à ce sujet²) ou alors utiliser le logiciel “eCAM_argus_camera” (en le lançant depuis le terminal) afin de disposer d'une interface graphique.



Tester les fonctions de calcul angulaire

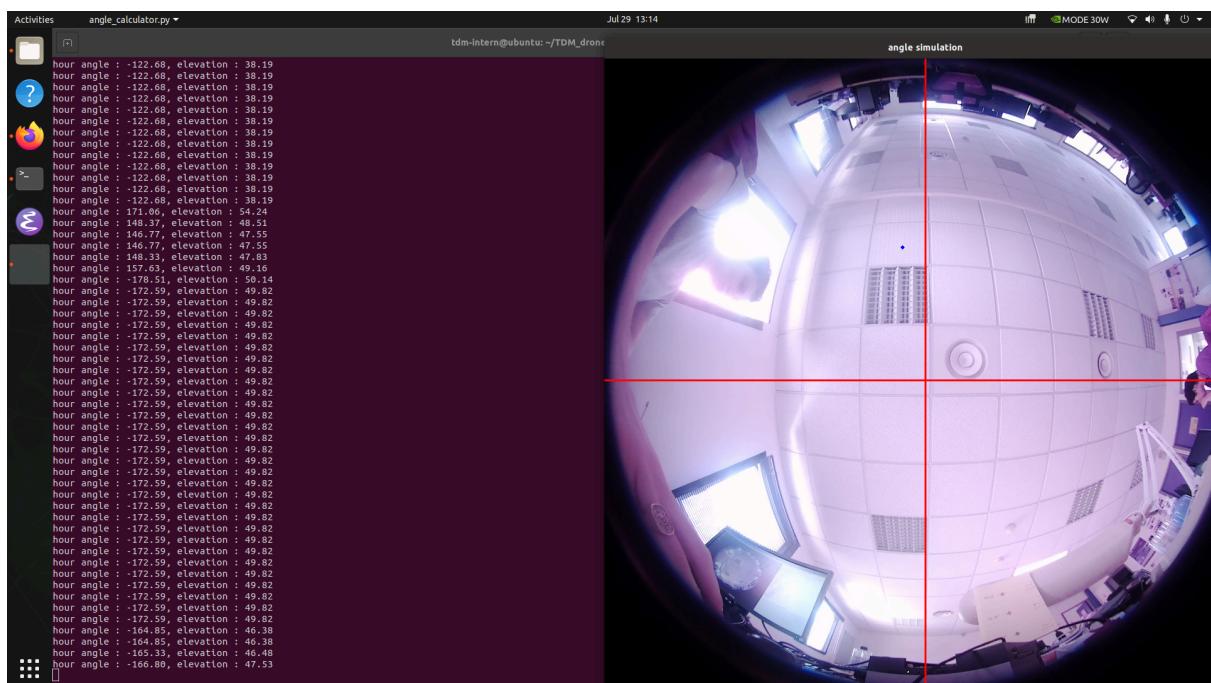
Afin de tester les algorithmes de calcul des coordonnées angulaires des objets détectés, j'ai conçu un petit programme en Python permettant de facilement vérifier les valeurs de sorties de mes fonctions. Ce dernier est stocké dans le dossier *angular_calculation_tester* et est composé de trois scripts : *angle_calculator.py* (le script principal *functions.py* (le script contenant les fonctions de calculs de coordonnées angulaires), et *Focuser.py* (le script contenant la classe permettant de manipuler le support PT de la caméra Arducam).

Lorsqu'on lance le script *angle_calculator.py*, une image spécifiée dans le code est chargée du sous-dossier ‘images’, et on affiche les axes coupant l'image de moitié

² voir le fichier se trouvant à l'emplacement suivant :
documentation/e-CAM56_CUOAGX_Documents_R01_RC6/Software/e-CAM56_CUOAGX_GStreamer_Usage_Guide_Rev_1_4.pdf

dans le sens de la longueur et de la hauteur afin de mieux pouvoir suivre le résultat des tests. On lance ensuite une boucle infini dans laquelle on transmet les coordonnées du curseur de la souris aux fonctions se trouvant dans le dossier *functions.py*. Les valeurs exactes des angles sont affichées en temps réel dans la console. Il est également possible de connecter le support de la caméra PTZ en ajoutant le paramètre [--ptz] lors de l'exécution de la ligne de commande.

Je vous renvoie au README du dossier pour plus d'informations concernant l'utilisation de ce programme.



N.B : Le capteur de la caméra étant rectangulaire, il est nécessaire, après avoir pris une photo, de redimensionner l'image à un format carré et de la centrer afin de ne garder que la prise de vue circulaire de l'objectif oeil de poisson.

Installation embarqué

Pour prendre de nouvelles images de drone pour enrichir le dataset de l'IA, il est souvent nécessaire de devoir se déplacer dans des endroits en extérieur où il est possible de faire voler son drone. Cela implique généralement de devoir se débrouiller sans prise de terre, et donc de devoir trouver une autre source de courant pour alimenter la carte Jetson et pour pouvoir utiliser la caméra. La solution à cette contrainte que nous avons trouvé est tout simplement une batterie portable Nastima de 12V et 8Ah, c'est-à-dire de 96Wh. La batterie peut être rechargée avec un simple générateur et des pinces crocodiles. Son autonomie, dans

le cadre de la simple prise d'images, peut aller jusqu'à 6h environ (la carte Jetson consomme en moyenne 16Wh lors d'une capture vidéo, donc $96/16 = 6$).

Une autre contrainte de l'utilisation embarquée de notre kit de développement (avec sa caméra) est l'absence d'écran, de clavier et de souris. Pour cela, deux solutions sont possibles :

- 1) Utiliser un serveur vnc et se connecter avec son ordinateur portable avec un partage de connexion³.
- 2) Utiliser un logiciel tiers pour pouvoir se connecter sur la carte Jetson avec son ordinateur portable et un partage de connexion.

Pour les avoir testé, je peux affirmer que ces deux solutions fonctionnent toutes les deux très bien. Cependant, afin de gagner en simplicité, j'ai personnellement opté pour la solution n°2 avec le logiciel *NoMachine*. Je vous conseille cependant de tester les deux solutions afin de vous faire votre propre idée concernant la meilleure méthode à employer.

Un dernier détail doit être pris en compte avant d'avoir un système pleinement opérationnel : le *display-manager*. En effet, lorsque l'on se connecte sur la carte Jetson d'une des deux manières cité ci-dessus sans brancher d'écran sur le kit de développement, le *display-manager* de la carte ne se lance tout simplement pas, ce qui a pour fâcheuse conséquence de nous laisser sur un écran vide, sans interface graphique pour pouvoir travailler. La solution à cela est d'utiliser un émulateur d'affichage (*dummy plug* en anglais) que l'on connecte sur le port DisplayPort du kit de développement afin de tromper le *display-manager* en lui faisant croire qu'un écran est branché pour le forcer à se lancer, ce qui résout notre problème.



³ Pour se connecter avec VNC : <https://developer.nvidia.com/embedded/learn/tutorials/vnc-setup>

Annoter un dataset

Pour construire un dataset avec de nouvelles images, la seule chose nécessaire est un logiciel permettant d'annoter vos images. Pour cela, à vous de choisir le logiciel qui vous convient le mieux. Cependant, prenez garde à ce que votre logiciel permette d'exporter votre dataset vers un des formats supporté par les algorithmes de training de Nvidia, à savoir KITTI, COCO et Pascal VOC (voir ci-dessous). Pour ma part, mon choix s'est d'abord porté sur Roboflow, puis sur CVAT (ce deuxième étant recommandé par Nvidia).

Entraîner une nouvelle IA

Pour entraîner une nouvelle IA, il est nécessaire d'utiliser le dossier Github de Nvidia “Hello AI World⁴”. Une fois installé, rendez-vous dans le sous-dossier /jetson-inference-master/python/training/detection/ssd. Déposez votre dataset dans le sous-dossier *data* et lancez ensuite la commande suivante :

```
python3 train_ssd.py --data=data/<dataset-directory>
--model-dir=models/<model-directory> --batch-size=4 --epochs=30
--dataset-type=voc
```

- dataset-directory : le nom du dossier de votre dataset
- model-directory : le nom du dossier où sera stocké les résultats de l'entraînement
- batch-size : nombre d'images qui sont traité en même temps (à augmenter pour aller plus vite si vous avez suffisamment de RAM)
- epoch : nombre de fois où votre dataset sera traité dans sa globalité
- dataset-type : indique le type du dataset utilisé (voc = Pascal VOC, coco = COCO). Par défaut, utilise un dataset de type KITTI.

Une fois l'entraînement terminé, les résultats seront stockés dans le dossier *model-directory* spécifié dans la ligne de commande. Lancez ensuite la commande suivante pour exporter les résultat au format ONNX (exploitable par nos algorithme) :

```
python3 onnx_export.py --model-dir=models/<model-directory>
```

⁴ <https://github.com/dusty-nv/jetson-inference?tab=readme-ov-file>

Copiez (ou déplacez) ensuite le fichier ONNX et le fichier labels.txt dans le dossier de votre code principale (ici, drone-detection-TDM/model pour pouvoir utiliser votre IA.

Le Drone

Afin d'enrichir le dataset et de tester le projet dans son ensemble, nous avons à notre disposition un petit drone : un DJI mini 4 pro. Son utilisation étant relativement intuitive, je ne m'y attarderai pas longtemps. Veuillez noter cependant les points suivants qui pourraient vous être utiles :

- Pour démarrer le drone appuyez deux fois sur le bouton de démarrage : brièvement la première fois puis longtemps la deuxième. Idem pour la télécommande.
- Vous avez à votre disposition des hélices de recharge et un tournevis dont vous pouvez vous servir en cas d'accidents. Pensez à les emmener avec vous.
- Vous avez trois batteries à votre disposition pour votre drone. Une batterie vous permet de faire voler votre drone pendant 30 minutes environ.
- Les joysticks de la télécommande peuvent se dévisser pour faire rentrer ladite télécommande dans la sacoche du drone. Un emplacement est prévu derrière la télécommande pour les ranger.

LIMITATIONS ET PISTES D'AMÉLIORATIONS

1) Trop grande sensibilité à la lumière

La caméra oeil de poisson est trop sensible à la lumière. Lors de la prise d'images, le drone disparaît dans un halo de lumière lorsqu'il vole au-delà de 3 mètres au-dessus de la caméra. En mettant des lunettes de soleil au-dessus de l'objectif, on gagne quelques mètres de vision verticale. Trouver une solution moins artisanale et plus efficace serait intéressant pour améliorer l'efficacité du système.

2) Un seul modèle de drone utilisé

Le modèle d'intelligence artificielle actuel a été réalisé avec un drone dji mini 4 pro, c'est-à-dire un drone avec quatre hélices repliables. Cela limite l'IA dans son éventail de détection et d'autres types de drone devront être intégrés au dataset de l'IA pour la rendre efficace dans toutes les situations (exemple : mono-coptère, tricoptère, hexacoptère, octocoptère, ainsi que toutes les variations de modèles dans toutes ces catégories).

Cependant, la finalité de ce projet n'étant que la réalisation d'un démonstrateur plutôt que d'un système fini, il suffit en réalité de ne s'occuper que des drones qui seront utilisés lors des démonstrations. Cette limitation est donc à relativiser avec les attentes du cahier des charges.

3) Un seul fond utilisé

Afin de simplifier l'intelligence artificielle et d'augmenter son efficacité, son domaine d'utilisation a été restreint. Seuls des drones sur fond de ciel ont ainsi été inclus dans le dataset. Afin de garantir une efficacité de l'IA dans tout type de situation, il serait intéressant d'inclure dans le dataset des images de drones dans des fonds divers et variés (forêt, bâtiments, plafond, etc.).

À l'instar du point précédent, cette limitation est à relativiser avec les paramètres des démonstrations tels qu'ils sont définis dans le cahier des charges.

4) Code c++

Le github de Nvidia [Hello AI World](#) dont nous nous sommes servis jusque-là pour la partie IA de notre projet met à disposition des codes en C++ en plus des scripts Python (que nous utilisons). Le C++ étant un langage plus rapide que le Python, son utilisation pourrait se révéler avantageuse. Cependant, les performances actuelles de la carte étant largement supérieures au framerate

de la caméra œil de poisson (100 à 400 fps affichés contre 67 fps maximum pour la caméra), cette amélioration n'est pas utile dans l'immédiat à moins que la caméra ne soit remplacée.

5) Vitesse de la caméra

Selon les spécifications techniques de la caméra œil de poisson, le framerate maximale en pleine résolution peut atteindre les 67 fps avec une sortie 12 bits, ou 79 fps avec une sortie 10 bits. Cela est relativement faible comparé aux standards des caméras hautes vitesses (framerate généralement supérieure à 150 fps). De plus, les performances de l'unité de calcul étant largement supérieure à celle de la caméra actuelle, se procurer une nouvelle caméra pourrait nous conférer une amélioration significative des performances globale du système.

6) Utilisation du flux vidéo de la caméra PTZ

Bien que nous puissions contrôler le support PT via une liaison I2C, nous n'avons pas encore réussi à connecter le flux vidéo de la caméra IMX477 à notre système, notamment à cause d'un problème de driver (à la base, le driver fournis par le fabricant était destiné à être utilisé sur des cartes Jetson nano et NX, et non pas sur des cartes AGX Orin comme celle que nous utilisons). Résoudre ce problème et approfondir cet aspect-là du projet permettrait d'ajouter un élément visuel convaincant pour le démonstrateur.

7) Problème de positionnement et d'instabilité

Lorsque la caméra PTZ effectue un mouvement trop rapide, le socle de cette dernière bouge, ce qui fausse la cohérence entre les coordonnées angulaires des objets détectés par la caméra œil de poisson et les angles pris par la caméra PTZ. De plus, l'orientation des deux caméras dans le même sens pour assurer cette même cohérence étant assez approximative, fixer les deux caméras sur un socle commun permettrait de régler d'un coup ces deux problèmes.

ANNEXES

- **Explications du choix de la mapping function équidistante :**

Dans une projection équidistante (également appelée projection linéaire), la distance radiale r depuis le centre de l'image jusqu'à un point sur le plan de l'image est directement proportionnelle à l'angle θ entre l'axe optique et le rayon lumineux entrant. Cette relation est donnée par :

$$r = f \cdot \theta$$

où f est la distance focale.

Pour un objectif fisheye avec une projection équidistante, l'angle maximum θ (qui est la moitié du champ de vision) doit correspondre à un rayon maximum r sur le capteur d'image. Le champ de vision fournit (185° en diagonale) correspond à un demi-angle de :

$$\theta_{max} = \frac{185^\circ}{2} = 92,5^\circ \approx 1,615 \text{ rad}$$

En utilisant la distance focale donnée $f = 1,7 \text{ mm}$ dans la formule équidistante :

$$r_{max} = f \cdot \theta_{max} = 1,7 \text{ mm} \cdot 1,615 \text{ rad} \approx 2,7455 \text{ mm}$$

Ce $r_{max} \approx 2,7455 \text{ mm}$ calculé devrait correspondre ou légèrement dépasser le rayon du cercle d'image (2,41 mm), indiquant que l'angle maximum correspond bien au bord du capteur, ce qui est un comportement caractéristique des objectifs à projection équidistante.

La projection équidistante garantit que les angles sont cartographiés uniformément aux distances sur le capteur, en accord avec les spécifications de l'objectif qui montrent des angles de champ uniformes (185°) sur différents formats de capteurs.

Un autre argument en faveur de la projection équidistante est la mention des angles de champ presque constants dans les spécifications⁵. Par exemple, pour différents formats de capteurs (1/1.8", 1/2", 1/2.5"), l'angle de champ diagonal reste à 185° , et l'angle horizontal est également constant à 185° . Seul l'angle vertical varie légèrement avec le format du capteur. Cette uniformité des angles de champ suggère une fonction de cartographie qui s'adapte linéairement avec l'angle, caractéristique de la projection

⁵ Voir documentation/fisheye-lens-technical-specifications.txt

équidistante, où chaque incrément d'angle correspond à une augmentation proportionnelle de la distance radiale sur le capteur.