Polytech Sorbonne | Sorbonne Université
4 Place Jussieu, 75005, Paris

Laboratoire d'Informatique de Paris 6
4 Place Jussieu, 75005, Paris

Acquisition et traitement temps réel de flux vidéo

(Stage Technique de 4ème année)

Michaël BAUDEUR

Étudiant ingénieur en spécialité

Électronique, Informatique et Systèmes Embarqués (EI-SE)

Du 5 Juin 2023 au 28 Juillet 2023

Tuteur : Adrien CASSAGNE, Maître de conférences au LIP6

Superviseur académique : Thibault HILAIRE

Pour l'année universitaire

2022-2023

Abstract

This document details all the research done during my internship at Paris 6 IT laboratory (LIP6) entitled “Real-time acquisition and processing of video stream”. The ALSOC team from LIP6 laboratory, located at Sorbonne University to 4 Place Jussieu in the 5th district of Paris, they plan to create a real-time meteor detection chain. The team works on two chains. A first one called FMDT for Fast Meteor Detection Toolchain tailored for balloon embedded camera detection by calibrating itself on stars and searching for linear moving object. And a second one called Meteorix made to detect meteor from space with a camera facing the Earth.

These tools are already working but they haven’t been tested in real conditions yet. The goal of this internship is to integrate these tools to a real video stream in order to catch meteors on it. For this purpose, we’ll use two streams, the first one will be a live from the international space station, a camera facing the Earth to test the Meteorix detection chain. The second one will be a camera (FRIPON) installed on Jussieu campus roof, facing the sky to test the FMDT detection chain.

In the first part of this document, we’ll rapidly give details about work and time planning. Then we’ll see how to get the stream from ISS. On a third part we’ll see research on how to use the FRIPON camera. The fourth part will explain how to transfer images from the acquisition chain to the detection chain. In the fifth part, we’ll discuss about research around image quality and how to get scientifically exploitable images with the camera. Then, we’ll see research on how to ensure that real time constraints are respected and what have been implemented to do so. And finally, we’ll conclude by setting a reminder of what has been made, what can be improved and what must be done next and a brief paragraph about possible personal work improvement.

Remerciements

Je tiens tout particulièrement à remercier toutes les personnes qui m'ont permis de réaliser ce stage dans de bonnes conditions et qui m'ont permis d'en apprendre davantage dans mon domaine d'étude.

Je remercie donc M. Adrien Cassagne pour m'avoir offert ce poste, m'avoir accueilli au LIP6 et m'avoir fait confiance pour la réalisation de ce projet, je le remercie également pour toute l'aide qu'il m'a apporté au cours de ce stage et les conseils qu'il m'a donné pour mieux effectuer mon travail et mieux connaitre le domaine de la recherche et les méthodes qui en découlent.

Je remercie tous mes collègues de bureau, M. Maxime Millet, M. Mathuran Kandeepan, Mme. Clara Ciocan, M. Ali Oudrhiri et M. Nathan Maurice pour m'avoir accueilli et conseillé pendant toute la durée du stage.

Je remercie également M. Manuel Bouyer et M. Jean-Paul Chaput pour leur aide technique et informatique lorsque j'ai rencontré des difficultés avec le système du LIP6.

Enfin je remercie M. Lionel Lacassagne pour tous ses conseils, pour m'avoir orienté vers un travail scientifique de qualité, pour m'avoir appris à mieux appliquer la démarche scientifique dans mon travail et à prouver qualitativement et quantitativement mes résultats, pour ses critiques constructives et pour m'avoir fait confiance pour ce travail de recherche malgré mes compétences de débutant dans le domaine du traitement d'images.

Sommaire

Listes des abréviations, cigles et acronymes

ALSOC	**A**rchitecture et **L**ogiciels pour **S**ystèmes **E**mbarqués sur **P**uce
LIP6	**L**aboratoire d'**I**nformatique de **P**aris **6**
Hz	**Hertz** (fréquence/occurrences par secondes)
FMDT	**F**ast **M**eteor **D**etection **T**oolbox
Meteorix	Mission CubeSat universitaire dédiée à la détection et à la caractérisation des météores
FRIPON	**F**ireball **R**ecovery and **I**nter**P**lanetary **O**bservation **N**etwork
API	**A**pplication **P**rogramming **I**nterface (interface de programmation d'application)
AVI	**A**udio **V**ideo **I**nterleave
RHEL	**R**ed**H**at **E**ntreprise **L**inux
GigE Vision	**Gig**a **E**thernet **Vision** (standard de transmission vidéo rapide par ethernet)
FIFO	**F**irst **I**n **F**irst **O**ut (structure de donnée consistant en une file de données)
GIMP	**G**NU **I**mage **M**anipulation **P**rogram (outil de manipulation d'images)
PGM	**P**ortable **G**ray **M**ap (format d'images numériques en niveaux de gris)

Introduction

L'équipe ALSOC du LIP6, situé à Sorbonne Université au 4 Place Jussieu dans le 5ème arrondissement de Paris, a pour projet de réaliser une chaîne de détection[1] de météores en temps réel[2]. L'équipe travail sur deux chaînes :

- FMDT : Cette chaîne de détection est destinée à des enregistrement de caméras embarquées dans des ballons à haute altitude. Cette dernière a été développée pour des caméras qui bougent en permanence en se calibrant sur les étoiles et en relevant les objets linéairement mobiles (les météores).
- Meteorix : Cette chaîne de détection est conçue pour les vidéos prises depuis l'espace vers la Terre et est développée pour repérer les météores sur des vidéos contenant plein de perturbation dues à la Terre, aux nuages et aux lumières issues de l'activité humaine sur la Terre.

Ces deux prototypes sont fonctionnels mais n'ont encore jamais été déployés en conditions réelles.

L'objectif de ce stage est d'intégrer ce travail à un flux vidéo pour récupérer des séquences d'images contenant des météores en temps réel. Les enjeux de ce stage sont de produire une solution capable de s'exécuter sans interruption et de palier aux différents problèmes de gestion de ressources (allocation mémoire / affectation CPU) et de résilience lorsque des problèmes sont rencontrés. Il y a deux principaux flux vidéo à prendre en charge. Le premier est le flux vidéo continu fournit par la station spatiale internationale orienté vers la Terre pour le déploiement de la chaîne Meteorix. Le second flux est celui issue d'une caméra FRIPON[3] installée sur le toit du laboratoire au début du stage et qui sera par la suite le point principal de ce stage. Ce nouveau système à vocation à être embarqué dans un futur nanosatellite (mission Meteorix). ([1, p. 3], modifié)

L'objectif de départ est de développer la chaîne d'acquisition des images de la source vers la chaîne de détection et de gérer les enregistrements afin de ne garder que les séquences vidéo sur lesquelles nous observons des météores, dans l'objectif d'éviter de saturer les espaces de stockages avec des heures d'enregistrements inutiles. Il fallait dans un premier temps, assurer les enregistrements sur une machine du laboratoire. Et dans un second temps, adapter la solution à un système embarqué afin de limiter l'utilisation des ressources tout en assurant un fonctionnement temps réel.

[1] Programme de traitement d'image dont l'architecture est organisée selon une chaîne de blocs de différentes fonctions visant la détection/le repérage d'un élément sur l'image.
[2] Dont le temps d'exécution est contraint par des événements physiques extérieurs (ici l'acquisition des images à 20 Hz minimum)
[3] Voir la page du projet fripon

L'architecture du projet peut être résumée par le schéma suivant :

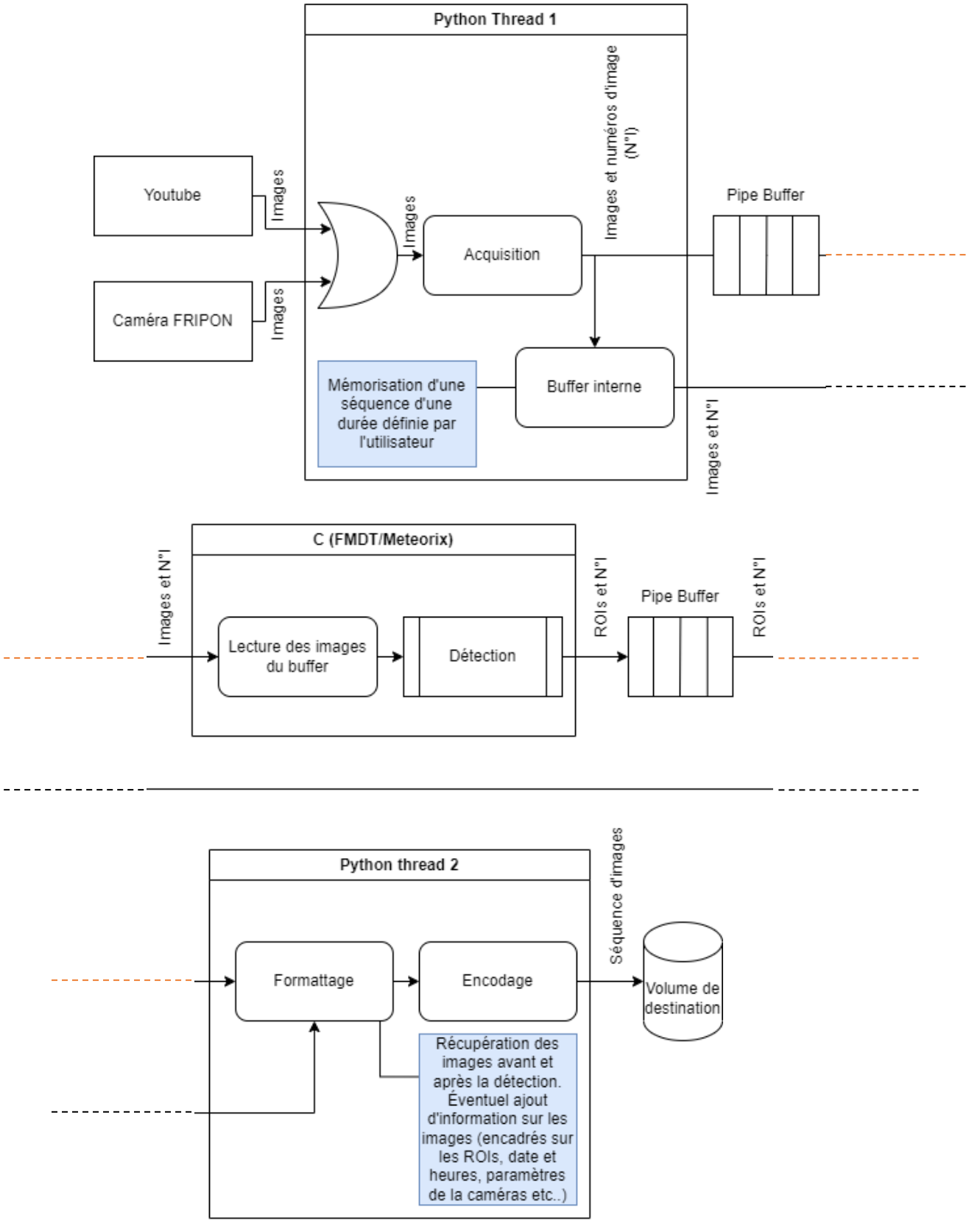


Figure 1 : Schéma de l'architecture de la chaîne d'acquisition (A. Cassagne, modifié)

L'objectif est d'obtenir 2 threads python, le premier fait l'acquisition, il envoie ces images à la chaîne de détection et mémorise un certain nombre d'images (par exemple si l'on souhaite pouvoir enregistrer une seconde avant la détection du météore, on choisira un buffer de 30 images, dans le cas où la caméra fait une acquisition à 30 images par secondes). La chaîne de détection nous renvoie les positions et dimensions des météores sur l'image (en pixels) ainsi que le numéro de l'image traitée. Ainsi on pourra, lors d'une détection, enregistrer quelques images avant la détection (par exemple 1 seconde de vidéo), enregistrer les images ayant fait l'objet d'une détection et y ajouter les informations nécessaire (par exemple ajouter des carrés autour des météores l'horodatage de chaque image et les paramètres de la caméra à l'instant de la capture). On pourrait également ajouter toutes les informations relatives à l'image avec des métadonnées.

Dans ce document figure une première partie expliquant quelques détails concernant la planification du projet et la gestion du temps de travail au cours de ce stage. Dans la seconde partie de ce rapport figurera les recherches pour obtenir le flux vidéo issue de l'ISS. Ensuite vous trouverez le déploiement de la caméra et les informations nécessaire à son utilisation. Dans la quatrième partie de ce document figurera les recherches pour transférer les données de l'acquisition à la chaîne de détection. Dans une cinquième partie vous trouverez l'acquisition du flux vidéo de la caméra FRIPON. Figurera ensuite le détail des recherches et des problèmes rencontrés sur la qualité des images reçues compte tenu des objectifs du projet. Puis vous trouverez les recherches et expériences concernant les défauts d'acquisition temps réel des images. Et enfin vous trouverez une conclusion sur les aboutissements de ce stage, les points d'amélioration de la solution proposée ainsi que les fonctionnalités restant à implémenter.

Instructions pour la lecture de ce rapport

*À LIRE PAR LA PERSONNE CHARGÉE DE LA SUITE DU PROJET D'ACQUISITION DE FLUX VIDÉO POUR LES CHAÎNES DE DÉTECTIONS FMDT ET METEORIX AU LIP6

*Ce rapport a pour objectif de rendre compte du travail réalisé lors de ce stage afin de valider la 4ème année de formation en Électronique, Informatique et Système Embarqués de Polytech sorbonne (EI-SE 4). Cependant ce rapport a également pour objectif de fournir un état de l'art et un guide simple et concis du travail de recherche réalisé au cours de ce stage, des solutions explorées et des solutions retenues, afin de permettre à un futur stagiaire ou chercheur de reprendre le travail là où il a été arrêté. Pour cette raisons les paragraphes importants seront précédés d'un astérisque rouge [*] pour accélérer la lecture de ce rapport. Les autres parties restent tout de même importantes pour une bonne compréhension du sujet.

I. Planification du projet

1. *Informations pratiques

Le projet et toutes ses sources sont sur le Gitlab du projet, consultez la section « Issues » et « boards » pour voir toutes les tâches qui ont été réalisées au cours de ce stage. Je recommande fortement l'utilisation des outils de gestion de projet de Gitlab pour rester concentré sur le projet.

2. Détail de la planification

Gitlab permet simplement l'application de la méthode Scrum avec la création d'«issues» classées dans trois colonnes (« à faire », « en cours » et « terminé ». Vous pouvez également ajouter vos propres colonnes). Le travail a été décomposé en plusieurs tâches visibles sur le projet Gitlab. Je recommande fortement l'utilisation de cet outil, il permet notamment de voir la progression du projet, et pour ma part, encourage le travail personnel malgré l'impression d'inefficacité que j'ai pu souvent rencontrer dans mes recherches. Le projet est décomposé en trois grandes parties (« milestones » sur Gitlab). La première est la réalisation du système d'acquisition des images. La deuxième est la liaison entre l'acquisition et la chaîne de détection. Enfin la troisième est l'enregistrement des images « positives », c'est-à-dire les images sur lesquelles figurent des météores détectés par la chaîne. De mon expérience, ces objectifs donnent une ligne à suivre, vous dévierez sûrement de ces objectifs aux cours de vos recherches, cependant garder ces quelques objectifs facilement consultables permet de mieux cerner l'intérêt de la tâche sur laquelle vous travaillez et permettra de ne pas partir vers des tâches inutiles.

3. Gestion du temps

Si vous disposez comme moi de 7 heures de travail par jour en horaire libre, je conseille de ne pas arrêter votre travail lorsque vous êtes productifs (que vous êtes inspirés), faites des heures supplémentaires le soir et compenser ses heures lorsque vous souhaiter faire quelque chose de particulier en semaine ou que vous vous sentez découragé. Cela m'a permis de continuer mon travail dans mes périodes les plus productives et de venir plus tard ou partir plus tôt dans mes périodes les moins productives ou les recherches étaient plus compliquées. Ce qui encourage à bien travailler et rend moins pénible les périodes où vous bloquez.

II. Flux vidéo de l'ISS

1. Introduction et objectifs

La station spatiale internationale met à disposition un live d'une de ses caméras embarquées sur youtube. Il s'agit donc d'une vidéo vue de l'espace orientée vers la Terre. Le premier objectif est donc de récupérer ce flux vidéo sous forme d'images exploitables, c'est-à-dire une vidéo sous formes d'images dans un format lisible par la chaîne de détection (Des données bruts de pixels RGB). Nous avons alors choisi le langage python pour réaliser cette première fonctionnalité, car ce langage est simple et permet d'implémenter rapidement des solutions de ce type, notamment en utilisant le package streamlink.

2. Recherches

*Installation de l'outil

Pour la réalisation de cette tâche, il est nécessaire de disposer des packages : streamlink, ffmpeg, vlc-python, opencv-python. Nous ne disposons pas des droits de super-utilisateur sur les machines du LIP6. Il est donc nécessaire de contacter le responsable du réseau à chaque fois que l'on souhaite installer un package sur le système (à chaque fois qu'il est nécessaire d'exécuter une commande super utilisateur « sudo », ex : sudo apt-get install <package>). Au cours de ce stage le responsable du réseau était Jean-Paul Chaput. Les machines du LIP6 fonctionnent sous le système d'exploitation Almalinux qui est un dérivé des distributions RHEL et donc compatible avec tous les outils disponibles pour les distributions Fedora.

Exploration de l'outil

Streamlink est une bibliothèque python et une interface en ligne de commande permettant de rediriger des flux (streams) depuis différents services. L'ensemble des informations concernant l'API (**A**pplication **P**rogramming **I**nterface ou interface de programmation d'application) Python de cette bibliothèque sont disponible ici. Dans le fichier « ACQ.py » vous trouverez toutes les fonctions implémentées. Pour réaliser ces fonctions, j'ai utilisé la documentation de l'API python, plusieurs postes sur Stack Overflow [2] ainsi que chatGPT [3]. Bien que ce dernier fasse l'objet de nombreuses controverses, il reste un outil efficace pour trouver rapidement de l'information sur un sujet simple et déjà exploré. Cet outil m'a été très utile pour réaliser les fonctions d'enregistrement et de replay utilisant ffmpeg, vlc et OpenCV. Une vérification de ces informations sur la documentation de chacune de ces API reste cependant important [4] [5] [6] [7]. Cela m'a également été utile pour réaliser des fonctions simples sur python [8].

*Récupération des données brutes

On retiendra que pour obtenir les données brutes des images on utilise les fonctions de script suivantes :

```
from streamlink import streamlink
streams = streamlink.streams(in_URL)
fd = streams["best"].open()
data = fd.read(size) #size in bytes
fd.close()
```

Cela permet de récupérer les données de l'image sous forme de vecteur de valeurs RGB partant du haut gauche de l'image et défilant dans le sens de lecture (gauche à droite, haut en bas).

3. Résultats

Cette partie du projet a été laissé en suspens pour d'autres tâches plus importantes, de nombreux tests restent à effectuer, voir la section commentaires et suggestion plus bas. Il n'y a pas de solution définitive pour l'acquisition du live de l'ISS.

4. Description de la solution retenue

Fonctionnalités

Ce script contient différentes fonctions permettant de visualiser le stream avec ffmpeg ou VLC et de l'enregistrer en vidéo au format AVI. Le script est disponible sur le gitlab du projet. Il y'a également une fonctionnalité permettant de n'enregistrer qu'une portion de l'image, cette fonctionnalité a été développée pour n'enregistrer que les météores sur les images acquises.

*Les fonctions définies dans le script peuvent être résumées par le tableau suivant :

Nom de la fonction	Rôle de la fonction
get_current_time_filename_string	Dater les fichier vidéo enregistrés
record_ROI	Enregistrer une région du flux d'origine
record_stream	Enregistrer avec ffmpeg
vlc_play_record	Lire un enregistrement avec vlc
ffmpger_get_video_duration	Récupérer la durée d'un fichier vidéo
ffmpeg_play_record	Lire un enregistrement avec ffmpeg
get_stream	Récupérer un flux vidéo
ffmpeg_play_stream	Lire un flux vidéo avec ffmpeg
vlc_play_stream	Lire un flux vidéo avec vlc
check_output_dir	Vérifier l'existence d'un répertoire

Figure 2 : Fonctions implémentée pour le l'acquisition du flux de l'ISS

*Les fonctions du buffer vidéo dans ce fichier ne sont pas fonctionnelles ([get_video_buffer] et [record_buffer]), cette fonctionnalité a été mise de côté pour une tâche plus importante, puis reprise plus tard (voir Transfert des images vers la chaîne de détection).

5. Commentaires et suggestions

À ce stade du projet, les objectifs étaient encore peu clairs, mon travail n'était pas bien organisé. Je suis trop parti dans l'exploration de l'outil, pas assez concentré sur les fonctionnalités importantes du projet, à savoir la récupération des images brutes issues du flux vidéo de l'ISS. Ce qui m'a conduit à la réalisation de travaux pouvant être considéré comme inutile.

La fonctionnalité permettant de n'enregistrer qu'une portion du flux sera inutile car nous récupérerons toujours l'image d'origine en entier pour la fournir à l'algorithme d'acquisition. Il est donc inutile voire problématique de tronquer le flux d'origine, surtout que l'on ne peut pas connaitre la région d'intérêt à ce niveau. Tronquer les images peut être intéressant en aval de la détection des météores, il faudrait donc implémenter une fonctionnalité permettant de tronquer les images brutes qui ont déjà fait l'objet d'une détection afin de ne récupérer que les régions de l'image contenant les météores détectés. Ces images pourront ensuite être enregistrées dans un répertoire dédié.

*Pour la suite du projet voici une liste de suggestions pour avancer le script d'acquisition du flux direct :

- Vérifier le contenu binaire des images (vérifier grossièrement la cohérence des valeurs)
- Enregistrer les images à partir des données binaires obtenues et vérifier si l'image est correcte.
- Connecter ce flux d'images à la chaîne de détection Meteorix et vérifier son fonctionnement.
- Créer un buffer d'images pour garder une seconde ou plus d'images avant détection afin de récupérer la séquence complète.
- Implémenter les fonctions permettant de n'enregistrer que les images positives (contenant des météores)

III. Caméra FRIPON

1. Introduction et objectifs

Une caméra a été installée sur le toit du laboratoire dirigée vers le ciel. Cette caméra a pour but d'observer le ciel afin de tester la chaîne de détection FMDT sur un flux vidéo réel pouvant potentiellement repérer un météore. L'objectif ici est donc de récupérer les données des images brutes issues de la caméra afin de les envoyer dans la chaîne de détection. Les contraintes à respecter sont une fréquence d'images suffisante pour l'application. Pour une détection temps réel, il faudrait au minimum une fréquence constante de 25 images par secondes. Il faut également que les images soient exploitables par l'algorithme de détection. Ce dernier se calibre sur les étoiles et détecte les objets dont le mouvement est linéaire. Il faut alors que la lune, les étoiles et les météores soient visibles[4] sur le flux vidéo. L'objectif dans un premier temps est de réussir à contrôler la caméra et d'y effectuer des tests pour trouver comment l'adapter à notre application.

Nous décrirons ci-dessous comment installer le logiciel Aravis, comment visualiser le flux de la caméra et comment exploiter les fonctionnalités de la caméra avec l'API python. Des informations techniques sur la caméra et le script se trouverons également dans cette partie.

2. Recherches

Installation d'Aravis

Aravis est un logiciel permettant d'interagir avec les caméras basées sur GeniCam, il s'agit d'une interface de programmation standardisée pour les caméras professionnelles utilisant notamment le protocole GigE Vision [9]. Ce logiciel propose également une API C et python permettant de manipuler les caméras.

**Procédure d'installation d'Aravis*

Pour installer Aravis, suivez la procédure suivante : ([10], modifié)

1. Téléchargez l'archive d'Aravis ici. [11]
2. Installez les dépendances selon votre système d'exploitation, pour Fedora (Almalinux, RHEL)
   ```
   $ sudo dnf install libxml2-devel glib2-devel cmake libusb1-devel
   gobject-introspection \ gobject-introspection-devel gstreamer1-
   plugins-base-devel gtk3-devel \ gtk-doc libxslt gstreamer1-devel
   gstreamer1-plugins-good python3-gobject \ g++ meson gettext
   ```
   Si vous n'avez pas les droit super-utilisateur, demandez à votre responsable réseau de les installer pour vous.
3. Installez ninja et meson
   ```
   $ pip install ninja
   $ pip install meson
   ```
4. Décompressez l'archive et ouvrez un terminal dans le dossier d'Aravis
5. Compilez le logiciel avec meson
   ```
   $ meson setup build
   ```
   Si vous n'avez pas les droits super-utilisateur sur votre machine, définissez le chemin de la

[4] Les étoiles et les météores doivent être détectables par la chaîne d'acquisition, ce qui ne veut pas forcément dire qu'ils doivent être visible sur les images acquises observées sur un écran. Cependant s'ils sont détectables ils pourront toujours être observés sur un écran après un post-traitement. (Voir V. Qualité des images)

compilation avec meson vers un répertoire dans votre espace utilisateur.

```
$ meson -Dprefix=<chemin dans l'espace utilisateur> build
```

6. Installez le logiciel avec ninja

```
$ ninja
$ ninja install
```

7. Dans le dossier « build » devrait se trouver l'exécutable d'Aravis Viewer. Allez dans build et lancez Aravis Viewer

```
$ aravis-viewer0.8
```

8. Pour ne pas avoir à rouvrir le dossier à chaque fois, ajouter cet exécutable à la variable d'environnement PATH[5]

```
$ export PATH=$PATH:<chemin du dossier aravis-0.8.XX>/build/src
$ export PATH=$PATH:<chemin du dossier aravis-0.8.XX>/build/viewer
```

9. Vous pouvez maintenant lancer Aravis Viewer depuis n'importe quel terminal

```
$ aravis-viewer0.8
```

10. Pour utiliser l'API python d'Aravis, il faut installer gi

```
$ sudo apt-get install python3-gi
$ pip install gi
```

11. Enfin ajoutez le chemin du « girespository » d'Aravis à la variable « GI_TYPELIB_PATH » et le chemin de « libaravis-0.8.so » à la variable « LD_LIBRARY_PATH »

```
$ export GI_TYPELIB_PATH=$GI_TYPELIB_PATH:<chemin vers aravis-
0.8.XX>/build/src
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<chemin vers aravis-
0.8.XX>/build/src
```

12. Vous pouvez également ajouter ces lignes de commandes à votre fichier « .bashrc » pour ne pas avoir remodifier les variables d'environnement à chaque fois que vous redémarrer votre ordinateur.

Visualisation de la vidéo

Lorsque l'on ouvre Aravis Viewer, la caméra devrait apparaitre dans la liste en dessous de « Fake 1 », si ce n'est pas le cas, c'est surement que la caméra n'a pas été correctement connectée avec la machine. Dans le Viewer vous avez en paramètres, le taux de rafraichissement (frame rate), le temps d'exposition (Exposure time), le gain et le niveau de noir (black level).

*Le Taux de rafraichissement est bien évidemment le nombre d'images par secondes qui seront envoyées par la caméra sur le réseau, on peut cependant noter que ce facteur est limité par le temps d'exposition. En effet le temps d'exposition correspond à la durée pendant laquelle le capteur, c'est-à-dire les cellules CCD photosensibles, seront exposées à la lumière avant l'acquisition des valeurs. Sur le logiciel on contrôle le temps d'exposition en microsecondes, donc si le temps d'exposition est supérieur à 33333 µs par exemple, alors la caméra ne pourra plus assurer un débit de 30 images par seconde. L'option de gain contrôle le gain du convertisseur analogique-numérique (ADC), en amont de la conversion, la valeur du gain d'Aravis et de son API n'est pas en dB, la formule de conversion est donnée dans la documentation technique de la caméra. Le niveau de noir change la luminosité globale de la caméra, il s'agit d'un offset ajouté aux pixels permettant de limiter l'impact

[5] La variable d'environnement PATH est une variable du système Linux contenant tous les chemins d'accès des exécutables désirés sous forme d'une chaine de caractère afin de pouvoir y accéder n'importe où depuis le terminal et ainsi éviter de réécrire à chaque fois le chemin des exécutables concernés.

des bruits dus aux courants d'obscurité[6], cependant ce paramètre réduit la précision en supprimant les signaux de faible intensité (pixels très faiblement excités). [12]

Dans un premier temps, on règle la fréquence à 30 images/s, le temps d'exposition à 37 µs, le gain au minimum et le niveau de noir à 0. Si la caméra fonctionne cela donne l'image suivante :

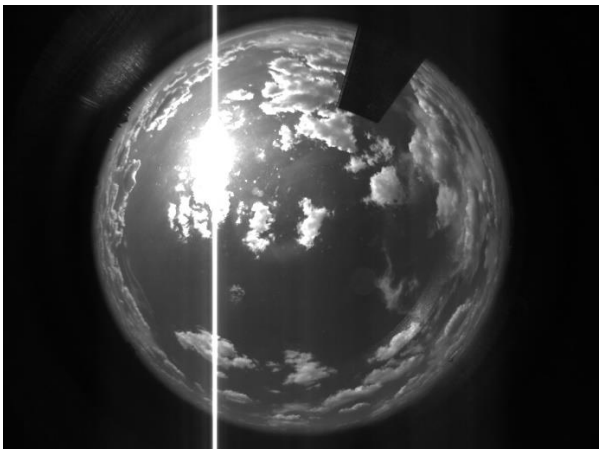


Figure 3 : Image de jour de la caméra FRIPON du LIP6

La caméra fonctionne et on arrive à observer des images. Il faut maintenant pouvoir automatiser le process avec l'API Python. Les scripts « fripon.py » et « fripon_recorder.py » ont été développé pour pouvoir enregistrer une vidéo au format AVI, afficher le flux direct de la caméra, prendre une seule image et modifier les paramètres de la caméra avec une interface en ligne de commande. Pour utiliser l'API Python d'Aravis, référez-vous à la documentation de l'API. [13]

*On notera que pour utilisez l'API Python il faut importer Aravis avec GiRepository en ajoutant les lignes suivante au début de votre script

```
import gi
gi.require_version('Aravis', '0.8')
from gi.repository import Aravis
```

Pour enregistrer des vidéos de nuit et tester la caméra en condition réelle, nous avons utilisé Cron, permettant de planifier l'exécution d'un programme.

[6] Bruits présents dans les cellules photosensibles quand ces dernières ne sont pas excitées par suffisamment de photons.

*Mode opératoire pour l'enregistrement de nuit

Vous trouverez ici comment utilisez Cron pour planifier les enregistrements, pour savoir comment utiliser l'interface en ligne de commande de la caméra, voir Description de la solution retenue.

1. Entrez dans le menu d'édition de Cron

```
$ crontab -e
```

2. Ajouter une ligne avec les informations de minuterie et entrez votre ligne de commande selon le format suivant :

```
# ┌──────────── minutes (0 - 59)
# │ ┌──────────── heures (0 - 23)
# │ │ ┌──────────── jour du mois(1 - 31)
# │ │ │ ┌──────────── mois (1 - 12)
# │ │ │ │ ┌──────────── jour de la semaine(0 - 6) (de dimanche à vendredi;
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
# 0 2 * * * python fripon_recorder.py -v -tm 10 -asc -o C:/Users/Desktop
```

   L'exemple ci-dessus permet d'enregistrer une vidéo de 10 minutes sur le bureau chaque jour à 2 heures du matin avec les paramètres automatiques de la caméra.
3. Enregistrer la modification en utilisant les raccourcis indiqués dans le menu de Cron
4. Quittez Cron et vérifiez que votre planification a bien été enregistrer avec la commande suivante

```
$ crontab -l
```

3. Résultats

À ce stade, l'enregistrement donne de mauvais résultats, la durée d'enregistrement ne correspond pas à la vidéo, on voit le soleil se lever alors que l'enregistrement se fait de nuit et ne dure que quelques heures. Enfaite La configuration automatique de la caméra augmente le temps d'exposition jusqu'à une demi-seconde, le frame rate descend alors à 2 FPS et comme l'enregistrement génère une vidéo au format AVI à 30 FPS constant, on se retrouve avec une séquence vidéo défilant 15 fois plus vite, d'où la première observation. De plus, aucune étoile n'est visible, nous avons cependant observé quelque chose ressemblant à un météore, bien qu'il s'agisse sans doute d'autre chose compte tenu de l'erreur dans cet enregistrement. Cette observation même erronée nous a tout de même permis de faire un premier test avec la chaîne de détection, la chaîne FMDT détecte correctement le « météore » sans faux positifs. La qualité de l'image reste donc correcte (pas trop de bruit) pour détecter un éventuel météore.

4. Description de la solution retenue

*Informations sur la caméra

La caméra FRIPON est un modèle Basler ace, acA1300-30gm, modèle mono (sans couleurs),

La caméra dispose de trois formats différents pour les pixels : 8 bits alignés, 12 bits alignés sur 16 bits avec 4 bits de remplissage (0) et 12 bits empaquetés, c'est-à-dire des paires de 12 bits réparties sur 3 octets pour éviter le remplissage et réduire le volume des images. L'objectif utilisé (de marque Focusave) a une focale de 1.25mm, une ouverture de f/2 et un champ de 185°. [14] Toutes les informations concernant la caméra peuvent être retrouvées sur la documentation technique de la caméra Basler et sur le site de Vigie Ciel. [15]

La caméra Fripon est connectée au réseau du LIP6 selon l'architecture suivante :

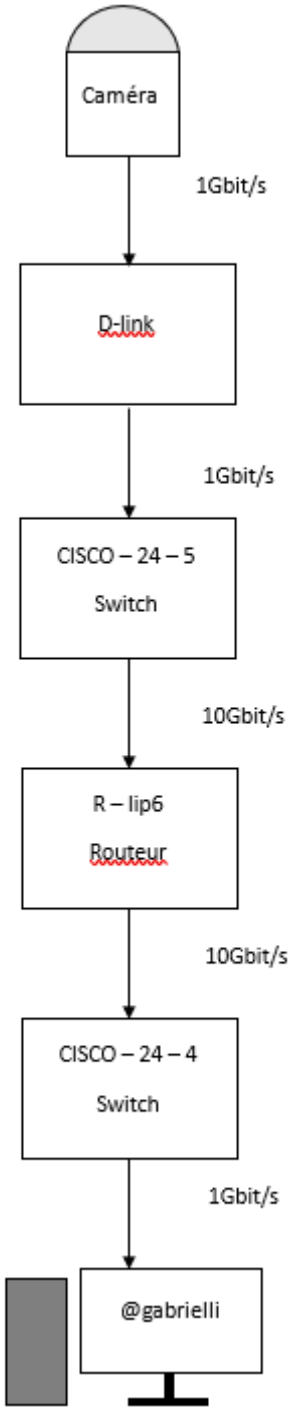


Figure 4 : Schéma de l'architecture du réseau entre la caméra FRIPON et L'ordinateur (Manuel Bouyer, modifié)

Sur ce schéma figure l'ensemble des appareils par lesquels passent les informations fournies par la caméra ainsi que les débits de chacune des lignes concernées par la connexion entre l'ordinateur et la caméra.

La caméra est située exactement sur le toit de la tour 24 sur le campus Jussieu de Sorbonne Université (4 Place Jussieu 75005) Coordonnée 48.847532, 2.355392 (voir carte). [16]

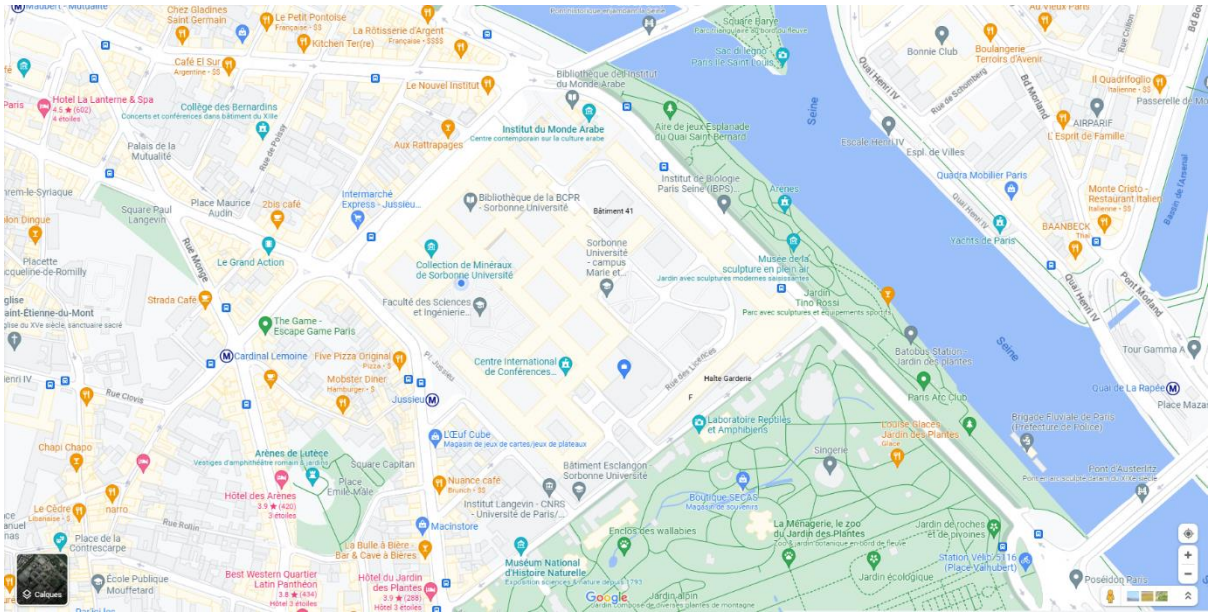


Figure 5 : Emplacement de la caméra Fripon du LIP6 (Google Maps, 2023)

Figure 6 : Photographie de la caméra Fripon sur le toit de la tour 24 devant la Tour Zamansky (A. Cassagne, 2023)

*Manuel d'utilisation de l'enregistreur

L'enregistreur « fripon_recorder.py » fonctionne avec une interface en ligne de commande, pensez à lancer cette ligne de commande depuis le répertoire contenant le script, ou alors ajoutez ce répertoire à la variable d'environnement « PATH ». Ci-dessous le manuel de la commande :

```
usage: fripon_recorder.py [-h] [-v] [-p] [-ts TSECONDS] [-tm
TMINUTES] [-th THOURS] [-fr FRAMERATE] [-o OUTPUT] [-exp EXPOSURE]
[-g GAIN] [-bl BLACKLEVEL] [-rx ROIX] [-ry ROIY] [-rw ROIWIDTH] [-rh
ROIHEIGHT] [-as]
                          [-asc] [-sh] [-lvs] [-i] [-bch] [-ag] [-
ae]

Enregistre une video avec la camera FRIPON

options:
  -h, --help            affiche ce message d'aide et sort
  -v, --video           enregistre une video
  -p, --picture         enregistre une image
  -ts TSECONDS, --tseconds TSECONDS
                        duree en secondes
  -tm TMINUTES, --tminutes TMINUTES
                        duree en minutes
  -th THOURS, --thours THOURS
                        duree en heures
  -fr FRAMERATE, --framerate FRAMERATE
                        taux de rafraichissment en images par
                        secondes
  -o OUTPUT, --output OUTPUT
                        chemin d'acces vers le fichier
  -exp EXPOSURE, --exposure EXPOSURE
                        temps d'exposition en µs
  -g GAIN, --gain GAIN  gain
  -bl BLACKLEVEL, --blacklevel BLACKLEVEL
                        niveau de noir
  -rx ROIX, --roix ROIX
                        origine en x de la region d'interet en
                        pixels
  -ry ROIY, --roiy ROIY
                        origine en y de la region d'interet en
                        pixels
  -rw ROIWIDTH, --roiwidth ROIWIDTH
                        largeur de la region d'interet
  -rh ROIHEIGHT, --roiheight ROIHEIGHT
                        hauteur de la region d'interet
  -as, --autosettings   parametrage automatique de la camera
  -asc, --autosetcontinuous
                        parametrage automatique continu de la camera
  -sh, --show           afficher l'image enregistree
  -lvs, --livestream    affiche le flux en direct
  -i, --info            ajoute les informations sur le flux video
  -bch, --bench         lance un test des paramètres de la camera
  -ag, --autogain       active le gain automatique en continu
  -ae, --autoexposure   active l'exposition automatique en continu
```

5. Commentaire et suggestion

Le script fonctionne, il permet d'enregistrer des vidéos avec la caméra fripon de façon simple. Ce script ne servira pas au projet final, il a pour but de tester la caméra et lancer des enregistrements de nuit pour voir si les conditions permettent la capture de météores. Les premiers résultats ne sont pas très convaincants. On ne voit pas les étoiles, on arrive tout juste à voir la Lune surexposée avec un temps d'exposition supérieur à la durée maximale entre deux images (33 millisecondes). La pollution atmosphérique de Paris ainsi que la tour Zamansky avec ses lumières qui restent souvent allumées la nuit empêche la bonne observation du ciel. (Voir V. Qualité des images)

IV. Transfert des images vers la chaîne de détection

1. Introduction et objectifs

Dans cette partie on cherche à établir le buffer d'images entre l'acquisition et la chaîne de détection (voir Introduction). On cherche à faire passer des images brutes du script python d'acquisition au programme en C de la chaîne de détection.

2. Recherches

Pour faire ce transfert, il existe plusieurs solutions, nous avons retenu 3 solutions à explorer :

- En passant par les flux d'entrée sortie « stdin » et « stdout » des deux programmes,
- En utilisant des mémoires partagées,
- Ou en utilisant des « pipes » nommés ou FIFO[7]

La première méthode consiste à envoyer les images directement sur le flux de sortie du programme et de rediriger ce flux vers le flux d'entrée du programme de la chaîne de détection. (Voir les fichiers « pipe_test.c » et « pipe_test.py »)

La méthode de la mémoire partagée consiste en la création d'un fichier binaire accessible en lecture et écriture. Ainsi les deux programmes pourraient interagir à travers ce fichier. (Voir les fichiers « shared_memory.c » et « shared_memory.py »)

*Enfin les « pipes » nommés consiste en la création d'un fichier de file (FIFO) accessible en écriture par un programme et en lecture par un autre, cette solution est donc parfaitement raccord avec l'architecture logicielle vu en introduction. Noter que pour que cela fonctionne correctement il faut bien s'assurer que l'accès en lecture aux FIFOs soit non bloquant. Pour cela on utilise le code suivant en python

```python
fifo_w_fd = os.open(FIFO, os.O_WRONLY) #en écriture
flags = fcntl.fcntl(fifo_w_fd, fcntl.F_GETFL)
fcntl.fcntl(fifo_w_fd, fcntl.F_SETFL, flags | os.O_NONBLOCK)

fifo_r_fd = os.open(FIFORD, os.O_RDONLY | os.O_NONBLOCK) #en lecture
flags = fcntl.fcntl(fifo_r_fd, fcntl.F_GETFL)
fcntl.fcntl(fifo_r_fd, fcntl.F_SETFL, flags | os.O_NONBLOCK)
```

Et celui-ci en C.

```c
fd1 = open(myfifo,O_RDONLY | O_NONBLOCK); //en lecture
fd2 = open(fifowr, O_WRONLY); //en écriture
```

(Voir les fichiers « pipe.c » et « pipe.py »)

Pour que la chaîne de détection puisse renvoyer les données des régions d'intérêt au script python, il faut créer un deuxième « pipe » dans l'autre sens.

3. Résultats

Après plusieurs essais, la méthode de redirection des flux d'entrée et sortie semble ne pas fonctionner correctement, à la réception les données sont absente ou erronée, sans doute qu'une mauvaise manipulation a été faite.

[7] First In First Out, il s'agit d'une structure de donnée consistant en une file de données les unes à la suite des autres

De même pour la méthode de mémoire partagée après plusieurs essais nous ne sommes pas parvenus à échanger des images entre les deux programmes.

Les « pipes » nommés fonctionnent très bien lorsque l'accès en lecture est non bloquant, on retrouve bien les images. Un test rapide en comparant les valeur envoyées et reçues permet de vérifier que le programme fonctionne. On peut également réencoder l'image reçue en C et vérifier que l'image reste intègre, ou encore renvoyer l'image au script python et réencoder l'image avec les fonctions déjà réalisées pour s'assurer que la méthode fonctionne dans les deux sens.

4. *Description de la solution retenue

La solution la plus efficace et accessoirement la seule que nous avons réussi à faire fonctionner et la méthode des « pipes » nommés. Pour cette solution nous avons donc le script python (« pipe.py » représentant le script d'acquisition) qui crée deux fichier FIFOs. Ce script ouvre l'un en écriture pour y transférer les images et l'autre en lecture pour y récupérer les régions d'intérêt, ce même script exécute le programme C (« pipe.c » représentant ici la chaîne de détection) via un « subprocess[8] » qui va ouvrir le premier fichier en lecture pour récupérer les images et le deuxième en écriture pour y transférer les régions d'intérêt qui seront récupérées par le script d'acquisition afin d'enregistrer les images détectées.

5. Commentaires et suggestions

La méthode des « pipes » nommés fonctionne correctement et a également été testée avec le débit de données des images envoyées à 30 images par secondes (Voir fichier « LIP6_Meteor_Stream_v0.3.py »). Il faudrait maintenant intégrer ce code à la chaîne d'acquisition et le formater proprement pour le rendu du projet final.

[8] En python, on peut exécuter d'autre programme en parallèle du script en utilisant un « subprocess », cela peut être un programme C tout comme un autre script python ou n'importe quel exécutable.

V. Qualité des images

1. Introduction et objectifs

La chaîne d'acquisition doit fournir des images scientifiquement exploitables, cela signifie plusieurs choses. Les images doivent fournir des informations claires (on doit pouvoir distinguer les étoiles et les météores pour pouvoir les détecter). Il faut conserver l'intégrité des informations contenues dans l'image. Notamment, les météores ne doivent pas être surexposés car cela entraine une saturation des valeurs de leurs pixels, fait perdre des informations sur leur rayonnement et empêche certaines mesures permettant de connaitre leur nature. Par exemple cela empêche complètement l'étude du plasma du météore permettant d'obtenir des informations sur sa composition. Nous verrons dans cette partie toutes les pistes explorées pour obtenir de bonnes images.

2. Recherches

La première chose à faire et d'essayer de trouver les valeurs de gain et de temps d'exposition permettant d'observer des étoiles. Pour cela la fonctionnalité de banc d'essai (option « --bench » de l'enregistreur) a été ajoutée. Cette fonction balaye des couples de valeur sur toute la plage possible des paramètres de gain et de temps d'exposition. Comme ces paramètres interagissent directement avec l'électronique de la caméra, il faut laisser un délai court entre chaque changement de valeur, sinon la modification ne s'applique que sur une partie de l'image et ne permet pas de l'évaluer qualitativement. Il faut donc choisir des écarts de valeurs suffisant pour que le banc d'essai soit pertinent sans prendre trop de temps à s'exécuter.

Nous avons également essayé de changer la profondeur des pixels pour gagner en précision (12 bits au lieu de 8 bits). Pour cela il faut modifier l'enregistreur de sorte qu'il encode les pixels sur 16 bits, sinon l'image sera corrompue (pixels non alignés correctement formant une image illisible).

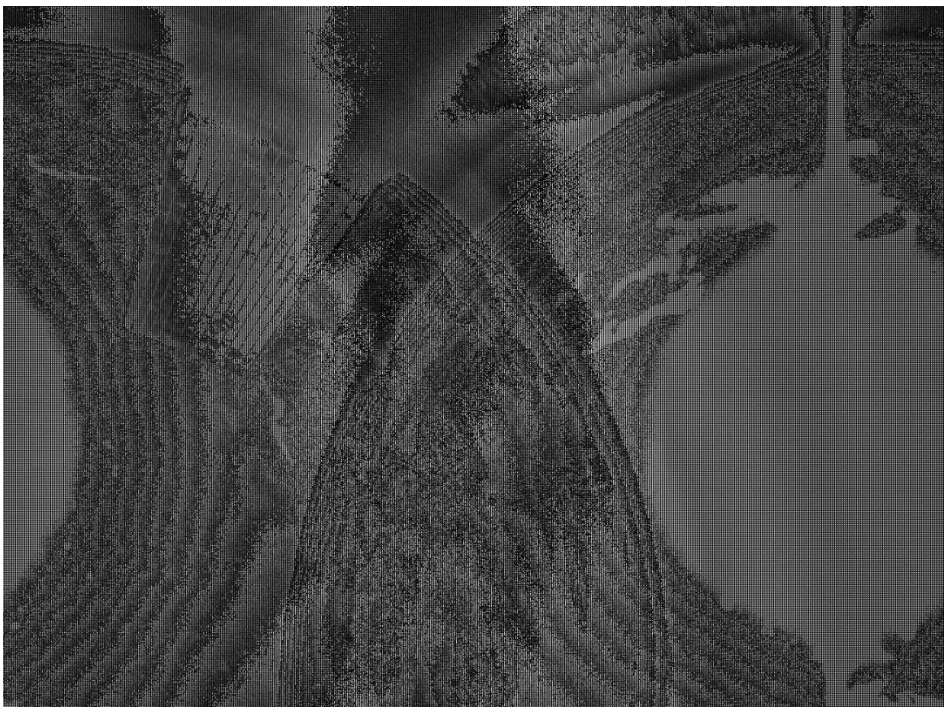


Figure 7 : Image de pixels 12 bits aligné sur 16 bits encodée en 8 bits

Les pixels sont alignés sur 16 bits sur les bits de poids faibles, lorsque l'on visualise ces images elles apparaissent très sombres car l'on voit en réalité 1/15 de la plage de valeur totale affichable par l'écran.

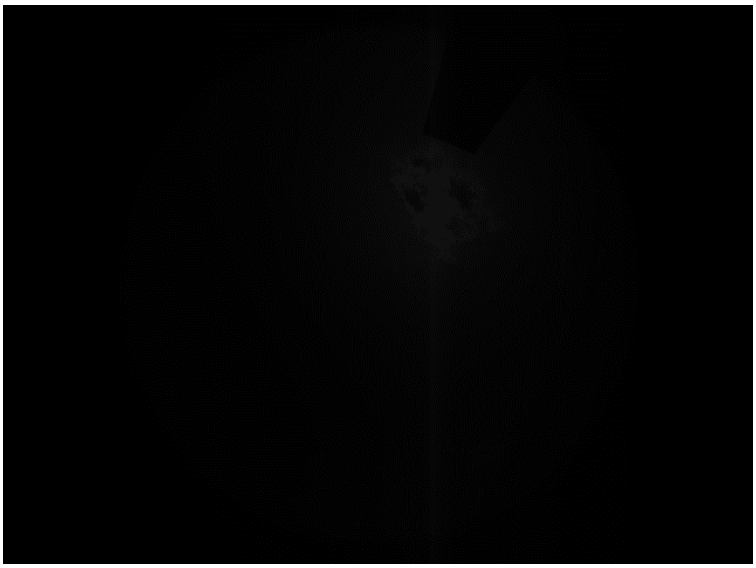
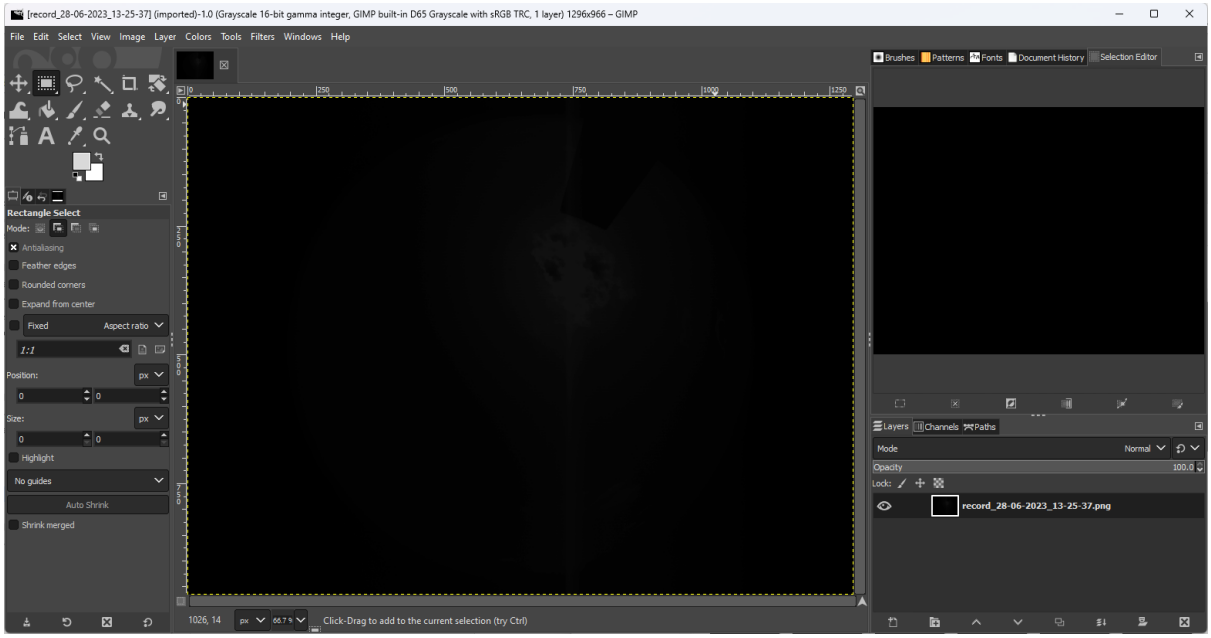


Figure 8 : Image de jour avec des pixel de 12 bits aligné sur 16 bits sur les bits de poids faible

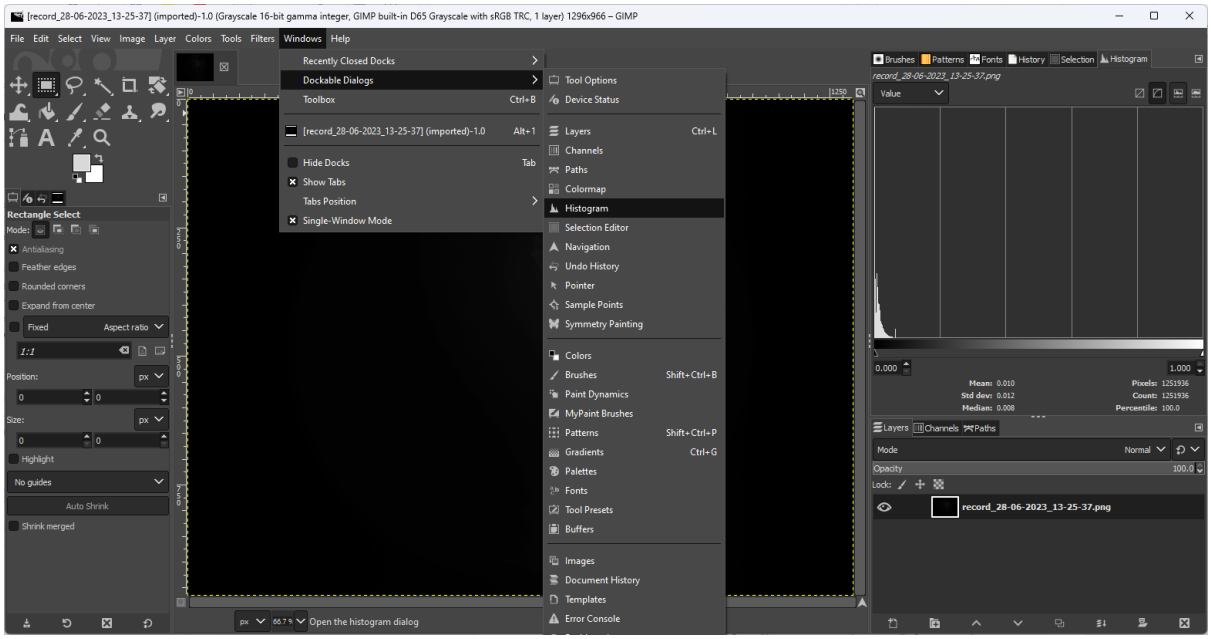
Pour corriger cela, on applique une égalisation d'histogramme à l'image. Cela peut être fait facilement sur le logiciel GIMP2 (GNU Image Manipulation Program).

*Méthode pour égaliser l'histogramme avec GIMP2

1. Ouvrez GIMP2 et glissez déposez votre image dans la zone de travail principale.

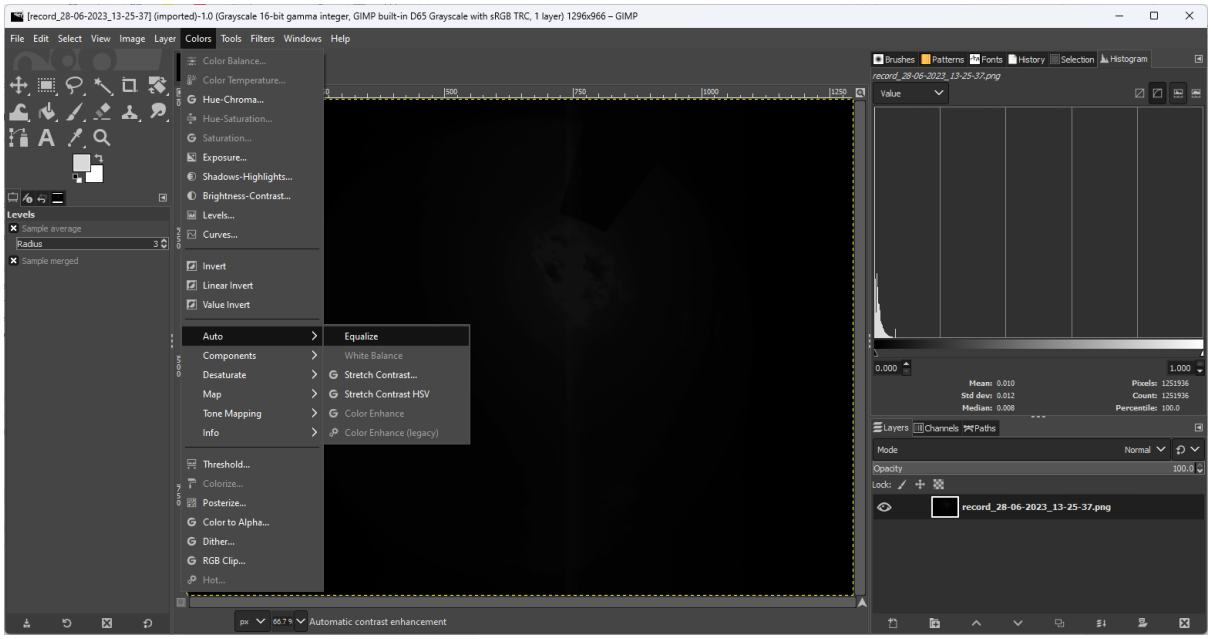


2. Afficher la fenêtre d'histogramme dans « Windows -> Dockable Dialogs -> Histogram ».

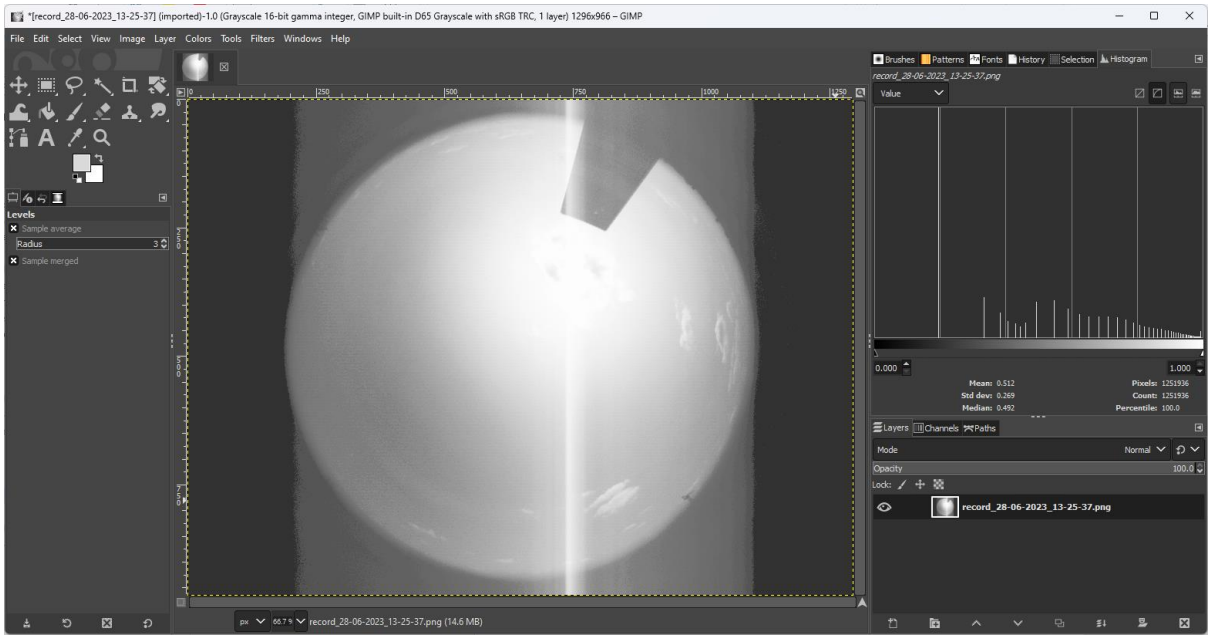


On remarque que l'histogramme occupe 1/15ème de la plage totale.

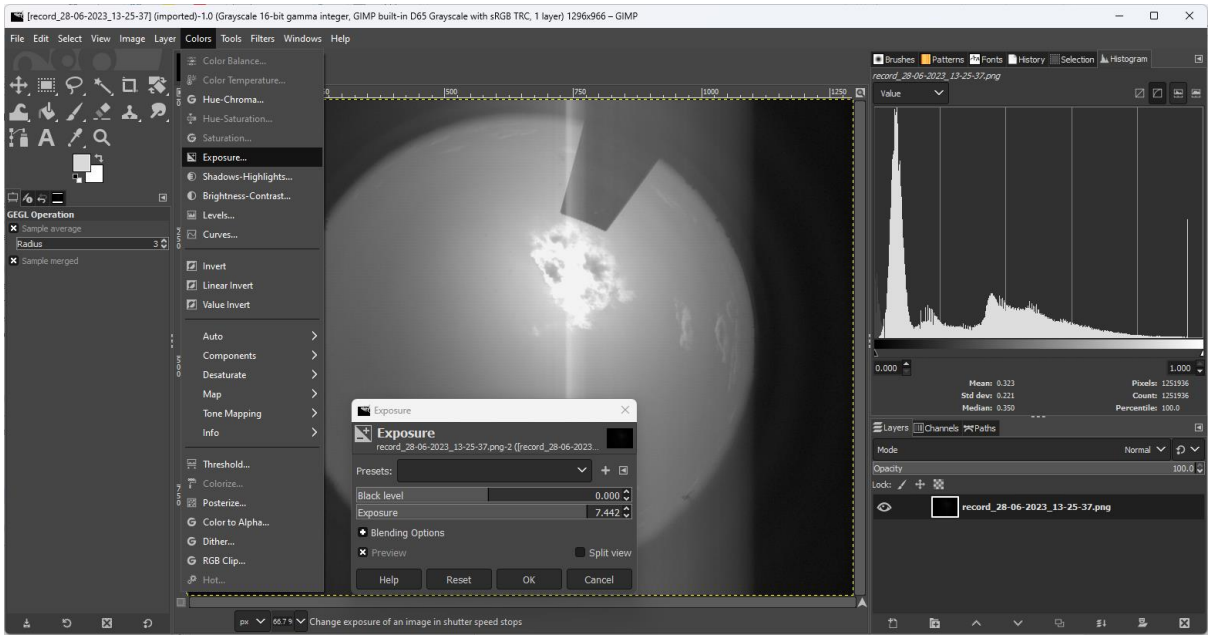
3. Pour égaliser le spectre allez dans « Colors -> Auto -> Equalize ».



4. On voit alors que le spectre a été étalé et l'on distingue beaucoup mieux l'image.



5. On peut également obtenir une image « plus jolie » (bien que cela n'ait que peu d'intérêt) en utilisant « Colors -> Exposure ».



Cette dernière méthode modifie l'image d'origine. À n'utiliser qu'à des fins d'observation visuelle sur un écran.

Pour trouver de bonnes images, lancer le banc d'essai n'est pas suffisant ou peu efficace (délai d'une journée entre chaque essai), il faut un retour direct des images avec les valeurs testées. Comme nous ne pouvons pas accéder au laboratoire de nuit, nous avons réalisé certaines acquisitions à distance en utilisant SSH. Ainsi nous avons pu observer les résultats de nuit en juste après les tests et faire plusieurs tentatives en modifiant des paramètres. Il faut donc se connecter en SSH à la machine du LIP6 (avec la commande « ssh <serveur> »), lancer un enregistrement avec l'enregistreur python puis copier le fichier généré par l'enregistreur avec la commande « rsync ».

3. *Résultats

La caméra ne permet clairement pas d'observer les étoiles, même l'étoile polaire n'apparait pas. En poussant le temps d'exposition et le gain au maximum (ainsi qu'une égalisation d'histogramme), on n'arrive pas à distinguer d'étoiles, la pollution atmosphérique et la pollution lumineuse de Paris empêche l'observation des étoiles. Cependant cela fonctionne peut-être avec des météores, nous n'avons pas encore capturé de météores pour vérifier cette hypothèse.

Le gain bruite et sature l'image lorsqu'il est trop élevé. Le temps d'exposition sature également les images. Nous avons pu observer la Lune avec la caméra, cependant il n'est pas possible de l'observer sans saturer, une bonne partie des pixels au centre sont à la valeur maximale ce qui nous fait perdre de l'information.

Il faut laisser le « black level » à 0 car ce paramètre fait perdre d'avantage d'information, d'autant plus que la chaîne de détection a été conçue pour détecter des météores sur des images bruitées.

L'utilisation des pixels sur 12 bits n'améliore que très peu les résultats, on peu distinguer des éléments plus sombres, mais toujours aucune étoile.

4. Description de la solution retenue

Nous n'avons pas trouvé de solution pour observer des images exploitables pour le moment.

5. *Commentaires et suggestions

Enregistrer des images exploitables scientifiquement avec la caméra installée sur le toit du LIP6 semble très compliqué, voire impossible dû à l'environnement de la caméra. Il faut faire des compromis entre des images avec des éléments visibles, un frame rate suffisant impliquant un temps d'exposition pas trop élevé (< 50 ms pour 20 FPS) et une image qui ne soit pas saturée. Or pour faire ces compromis et obtenir un résultat optimal, il faudrait d'abord capturer plusieurs météores afin de juger la façon dont ils apparaissent avec les paramètres obtenus.

Pour cette raison il semble tout à fait justifié de contacter l'observatoire de Paris qui utilise une caméra FRIPON sur le toit du Museum d'histoire naturelle (à quelques mètres de la caméra du LIP6), pour connaitre les paramètres qu'ils utilisent. (À cet endroit la caméra doit être soumises à des contraintes quasi-identiques) Nous avons d'ailleurs déjà tenté de contacter un membre de l'observatoire de Paris, mais sans réponses. Cela éviterait de faire beaucoup de test en capturant des météores. Et en attendant le prochain pour chaque test.

VI. Acquisition temps réel des images

1. Introduction et objectifs

L'acquisition en temps réel des images est essentiel pour le bon fonctionnement de la chaîne de détection. Pour que la détection s'opère correctement il faut que les images soient acquises à débit constant et qu'aucune image ne soit ratée. (Bien que la chaîne de détection soit capable d'extrapoler les météores afin de ne pas perdre leur trace, il reste néanmoins indispensable de ne pas manquer d'images pour assurer la bonne détection des météores. Tous les scripts et algorithme réalisé jusqu'alors présentaient des défauts dans leur fonctionnement en temps réel, des images se perdent à chaque enregistrement, le nombre d'images et la durée des enregistrements ne correspondent pas.

2. Recherches

Nous avons d'abord observé des incohérences temporelles sur les séquences enregistrées, notamment sur les séquences utilisant les paramètres automatiques. (Les nuits défilaient trop vite). Pour mieux comprendre le problème, nous avons ajouter les informations de la caméra sur les vidéos enregistrées comme cela :

Figure 9 : Enregistrement de nuit avec affichage des informations

L'indicateur « FPS » correspond au réglage du paramètre dans le logiciel, il ne correspond pas au nombre d'images par secondes reçues lors de l'acquisition. Les autres paramètres sont visibles et on peut observer que le contrôle automatique du temps d'exposition l'a poussé jusqu'à 500 ms, soit une demi-seconde, d'où les défauts temporels observés au début. Le temps en bas à gauche se met à jour toutes les secondes, or dans cet enregistrement, ce temps fait des sauts de plusieurs secondes (15 secondes environ à chaque rafraichissement). Avec cette méthode nous avons pu repérer les gros défauts comme celui-ci. (Par la suite nous avons toujours régler les paramètres en mode manuel). Nous avons ensuite ajouté à l'affichage le calcul du nombre d'images par secondes reçues.

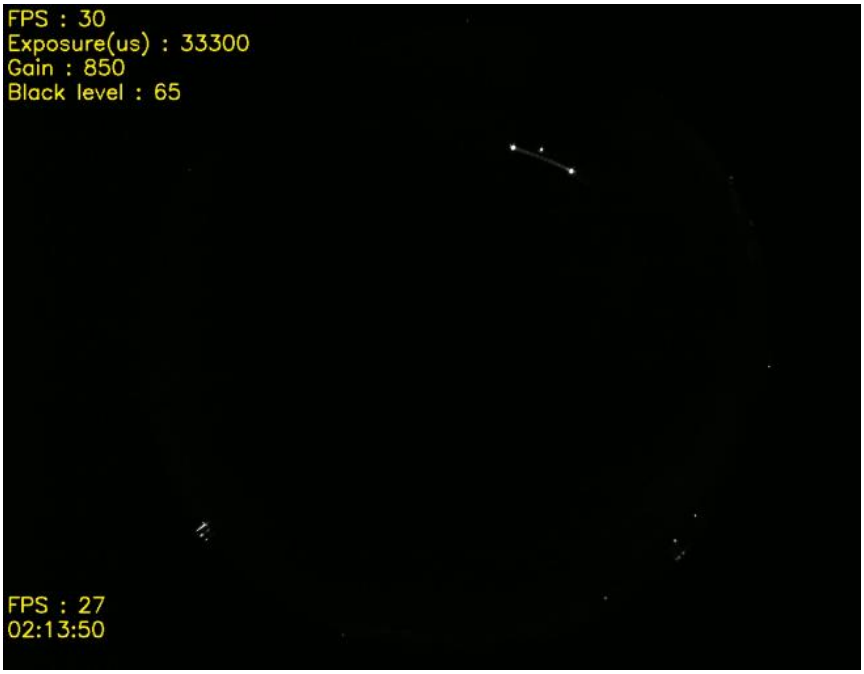


Figure 10 : Enregistrement de nuit avec ajout du calcul du nombre d'images par secondes

On a en moyenne toujours 3 à 5 images manquées par secondes.

Afin de mesurer avec plus de précision ces défauts. Nous avons changé la méthode d'acquisition des images. (Voir fichier « LIP6_Meteor_Stream_envents.py ») Certaines fonctions précédemment développées ont été déplacée dans deux autres fichier pour alléger le code sur les fichiers en cours de développement (voir fichier « common_utils.py » et « Basler_camera_utils.py »).

À partir de là nous avons chercher à optimiser la capture des images, notamment en parallélisant les tâches. Un premier thread[9] place les images dans un buffer circulaire et un autre récupère les images du buffer pour les enregistrer.

Pour assurer la synchronisation des threads, on utilise des évènements (« events »). On crée un évènement pour chaque emplacement du buffer d'image, lorsque le premier thread écrit dans une case mémoire du buffer, il enclenche l'évènement associé, le thread d'enregistrement attend que l'évènement soit enclenché pour procéder à l'enregistrement et désamorce l'évènement une fois l'image enregistrée. Ainsi, si le thread d'enregistrement arrive au début du buffer et que le thread d'acquisition est déjà en train de remplir l'une des dernières cases, le thread d'enregistrement pourra rattraper toutes les images (ce qui permettrait à l'avenir d'enregistrer une seconde avant la détection[10]).

L'enregistrement a également été modifié, désormais les images sont enregistrées au format PGM (Portable Gray Map) en séquences d'images (il ne s'agit plus d'un format vidéo, une manipulation

[9] Un thread (fil en français) est un programme exécuté en parallèle d'un programme principal (qui sera appelé thread principal), dans le cas d'un processeur à plusieurs cœurs (multicœurs), les instructions s'exécutent simultanément sur chacun des cœurs. Dans le cas d'un processeur monocœur, le système d'exploitation alterne entre les deux programmes. Dans les deux cas cette méthode permet d'améliorer les performances et de répondre à des contraintes de temps réel.

[10] On notera que pour arriver à enregistrer une seconde avant la détection il faut qu'au démarrage de l'enregistrement, celui-ci se calibre sur la case mémoire consécutive à celle en cours d'acquisition. Cela est possible avec une variable dont l'accès fonctionne avec un mutex.

particulière[11] est nécessaire pour lire la séquence d'image en vidéo). Ainsi il est beaucoup plus simple de compter le nombre d'images enregistrées. De plus le programme affiche dans la console le nombre d'images qui ont été acquises et mises dans le buffer d'images. On peut donc vérifier séparément le fonctionnement de l'acquisition et de l'enregistrement en comparant le nombre d'images. Dans la majorité des cas le nombre d'images entre acquises et enregistrées est identique (car l'enregistrement est bien plus rapide que le rafraîchissement de la caméra).

Le thread principal agit comme un minuteur pour s'assurer que l'acquisition ne dépasse pas du temps d'enregistrement défini par l'utilisateur. (Dans les premiers programmes un nombre d'images à enregistrer était calculé à partir du temps d'enregistrement demandé, d'où les problèmes qui sont survenus).

Nous avons pu effectuer quelques mesures très précises. Pour un enregistrement de 15h39 (durée de la nuit la plus longue à Paris au solstice d'hiver) à 20 FPS avec un buffer de 20 images, on est supposé obtenir 1126800 images, mais nous en n'avons obtenu que 1122818, soit 3982 images manquées, l'équivalent de 3 minutes et 19 secondes perdues sur l'acquisition. Ce qui nous donne un taux d'erreur de 0.35%. La classe Buffer d'Aravis[12] dispose d'une méthode[13] « get_status » permettant de connaitre l'état du buffer. Cette méthode renvoie « SUCCESS » si l'image a bien été acquises. Dans le premier test, on observe que le buffer bloque dans un état « TIMEOUT » dans lequel il prend beaucoup plus de temps pour acquérir les images et repart normalement au bout d'un certain temps (comme si le buffer était plein et qu'il attendait qu'il se libère pour repartir, d'autant plus que ce phénomène arrive plus tardivement lorsque l'on augmente la taille du buffer Aravis).

Nous avons effectué un autre test à 30 FPS, on est alors supposé obtenir 1690200 images. Nous en avons obtenu 1621574 soit 68626 images manquées, l'équivalent de 38 minutes et 7 secondes. Ce qui nous donne un taux d'erreur de 4.06%, ce qui est bien plus élevé que le premier essai à 20 FPS. On remarque donc que le script a du mal à assurer le respect des contraintes temps réel de l'application.

Nous avons donc mis au point un deuxième script (voir fichier « LIP6_Meteor_Stream_v0.2.py ») dont la fréquence des images de la caméra est adaptative. Lorsque l'on observe un « timeout » sur le buffer, on réduit la fréquence des images et on repart à la fréquence de départ (ex : 20 FPS) dès que le « timeout » s'arrête. Cette méthode réduit le taux d'erreur (de l'ordre de 0.1%) mais ne convient pas à la chaîne de détection qui doit repérer des mouvements linéaires sur un flux à fréquence constante.

Nous avons ensuite cherché à observer l'impact de l'utilisation du réseau sur les performances de l'acquisition (voir fichier « LIP6_Meteor_Stream_v0.3_Bench.py »). Ce script a pour

[11] VLC Media Player propose un outil pour lire des séquences d'images numérotées dans l'ordre dans un répertoire.

[12] Le buffer d'Aravis est le buffer qui reçoit les images envoyées par la caméra sur le réseau, le script copie ces images vers un autre buffer que nous avons implémenté pour notre application.

[13] En programmation orienté objet, une méthode est une fonction d'une classe représentant un objet, ici l'objet est le buffer d'images de la caméra. La méthode est la fonction

but de mesurer le taux d'erreur sur des enregistrement de 10 minutes effectués toutes les heures. Ce test a donné les résultats suivants :

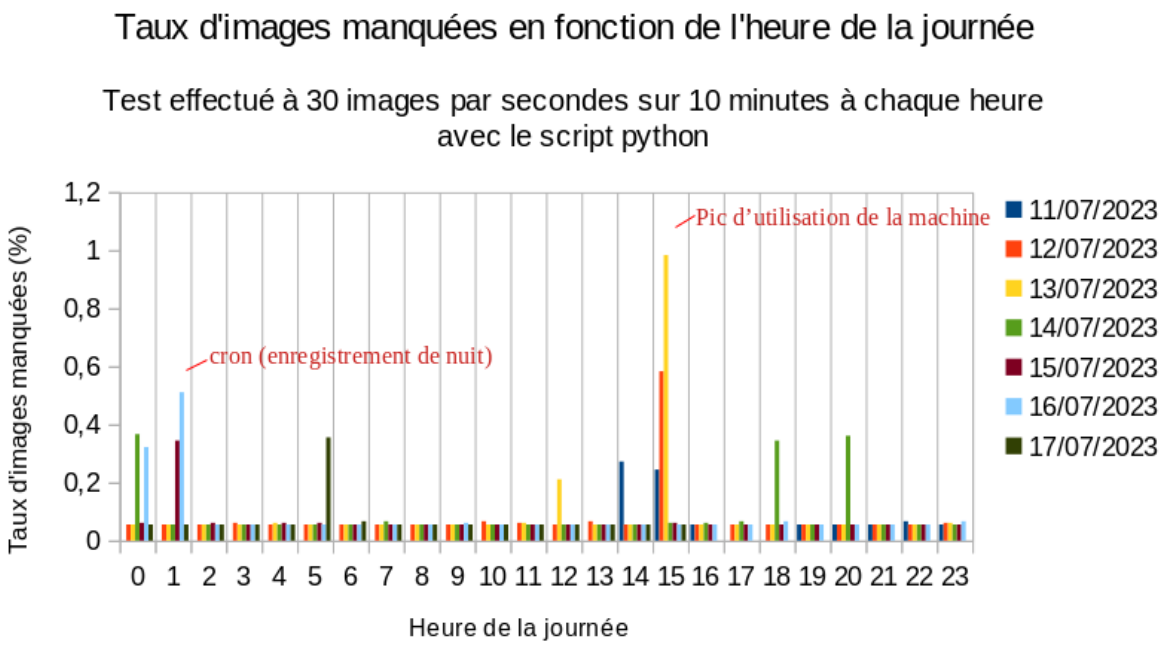


Figure 11 : Diagramme en barre du taux d'erreur en fonction de l'heure de la journée (test effectué sur 7 jours)

On remarque que l'acquisition a le plus grand taux d'erreur non pas dans les périodes d'utilisation du réseau dans quel cas ce dernier serait très faible pendant la nuit mais lors des périodes d'utilisation de la machine. En effet entre 14 heures et 15 heures, il s'agit des heures où j'effectue le plus de tests sur la machine. De même ce taux d'erreur augmente à l'heure de l'enregistrement de nuit (On note que l'enregistrement de nuit échoue à ce moment-là car la caméra est déjà utilisée par le script de test).

Ce test prouve que le script n'est pas assez optimisé pour s'assurer que les contraintes de temps réel soient respectées.

*Pour mieux visualiser les performances du script, nous avons mis en place un banc d'essai mesurant les intervalles de temps entre chaque image (voir fichier « LIP6_Meteor_Stream_v0.3_Bench_timestamp.py »). Le script mesure l'intervalle de temps entre chaque image acquise et enregistre cet intervalle dans un fichier CSV (Comma-Separated Value). Un autre script d'automatisation (voir fichier « histogram_inbetween_generator.py »), écrit une colonne utilisant la fonction d'Excel pour compter le nombre d'occurrences des intervalles de temps dans une

plage donnée afin d'obtenir un histogramme. On supprime ensuite les intervalles de temps n'ayant aucune occurrence. Pour le script v0.3 on obtient l'histogramme suivant :

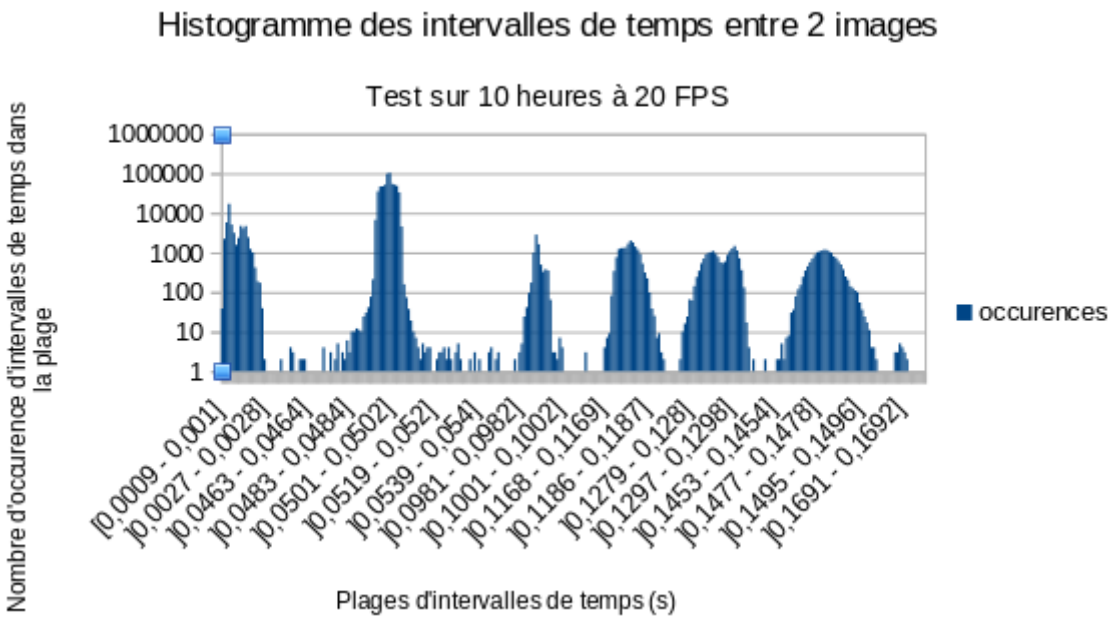


Figure 12 : Histogramme du script v0.3

On remarque qu'il y a une majorité d'images dont l'intervalle de temps est dans la période d'acquisition (0.05 secondes), On remarque un pic conséquent à gauche dans les intervalles de temps très courts et Plusieurs pics à droite. Les pics à 0.1 secondes et à 0.15 secondes sont cohérents car il s'agit de multiples de la période entre deux images, ce qui signifie qu'une ou deux images ont été manquées, les autres pics montrent clairement que le script déconne. Le pic de gauche pourrait correspondre au rattrapage des images manquées lorsqu'un paquet est corrompu et qu'il est renvoyé. (La documentation de la caméra spécifie que les donnée des images sont renvoyées si elles ne sont pas correctement acquittées). Pour les autres pics, le script ralenti. Une hypothèse est que python est trop lent.

Le script v0.4 (fichier « LIP6_Meteor_Stream_v0.4.py ») est un test vérifiant l'état du buffer avant de réaliser l'acquisition, autrement dit ce script attend que le buffer passe à « SUCCES » pour réaliser l'acquisition, mais cette méthode donne des résultats catastrophiques et a été abandonnée.

*La version 0.5 du script corrige un défaut majeur. Jusqu'ici les images étaient acquises depuis le buffer d'Aravis avec une boucle associant un buffer vide au flux « stream.push_buffer », la fonction de callback[14] s'enclenche et récupère une image. Ensuite la fonction d'acquisition reprend la main et un retire le buffer du flux « stream.pop_buffer ». Or le callback a déjà récupéré cette image, cette instruction permet donc seulement de libérer le buffer du flux pour en mettre un nouveau. La version 0.5 (voir fichier « LIP6_Meteor_Stream_v0.5.py ») associe (« push ») le buffer une seule fois

[14] Une fonction callback est une fonction associée à une interruption, une interruption est un évènement qui « met en pause » un programme et redirige le pointeur d'instruction vers la fonction callback, une fois l'exécution de cette fonction terminée, le pointeur d'instruction revient sur le programme principal pour le reprendre.

et c'est le callback qui refait le « push » du buffer après l'acquisition, ainsi le callback est réappelé automatiquement et aucune fonction ne bloque le programme.

En refaisant le test des intervalles on obtient l'histogramme suivant :

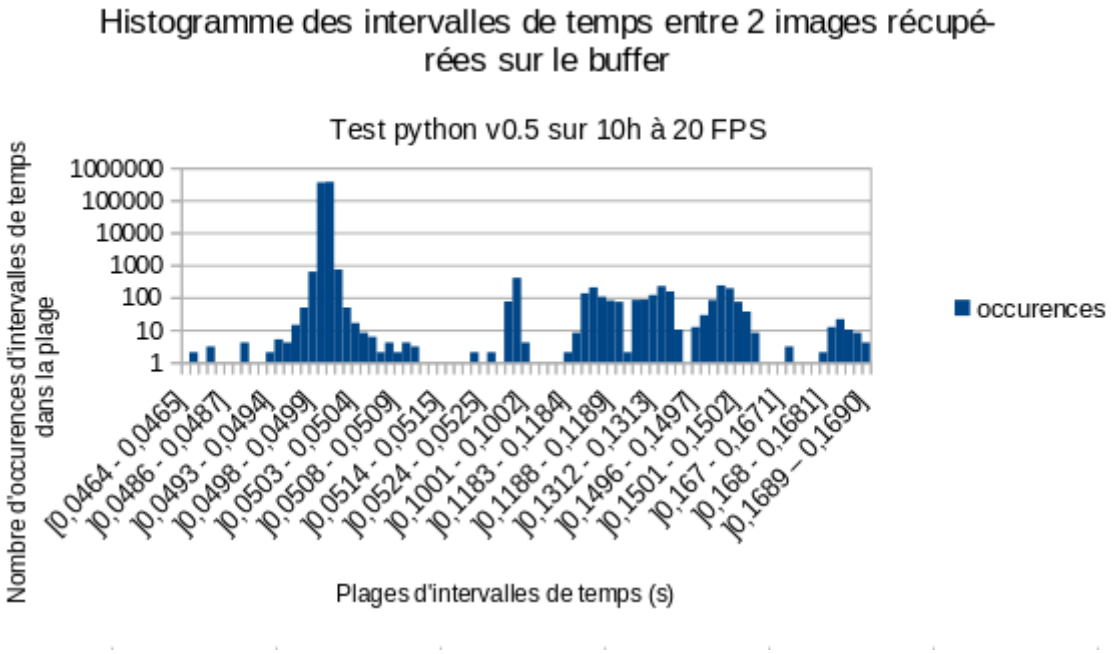


Figure 13 : Histogramme du script v0.5

*On observe des résultats bien meilleurs, il y a beaucoup plus d'images dans l'intervalle correspondant à la fréquence des images (0.5s) et beaucoup moins dans les intervalles supérieurs. Le pic dans les intervalles très court a complètement disparu ce qui rend les résultats beaucoup plus cohérents. Pour les pics dans les intervalles plus grands, cela reste des ralentissements dans l'acquisition. Cette version perd toujours des images, mais avec un taux d'erreur moindre.

La version 0.6 consiste en une méthode brute qui utilise le processeur à 100%. En exécutant en boucle l'acquisition avec une fonction non bloquante « try_pop_buffer ». Les résultats sont

similaires à la version 0.5 mais avec une surcharge du processeur. Cette solution ne sera donc pas retenue.

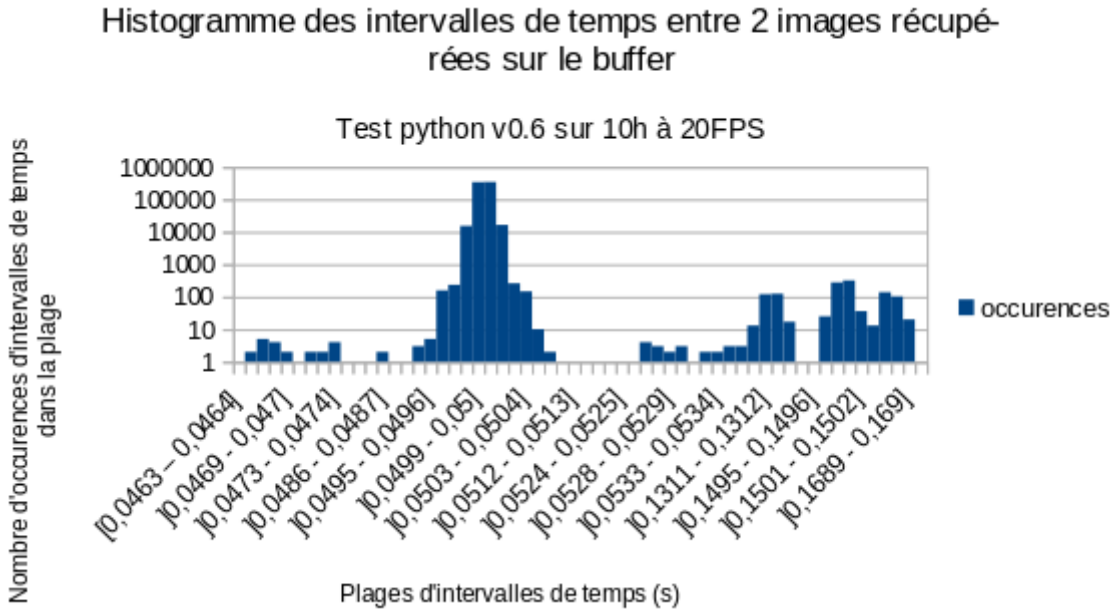


Figure 14 : Histogramme du script v0.6

Enfin afin de vérifier si les performances de Python sont en causes dans ces défauts d'acquisition nous avons essayé de réaliser la même fonction avec un programme en C (voir fichier « ACQ.c »). Ce dernier donne les résultats suivants :

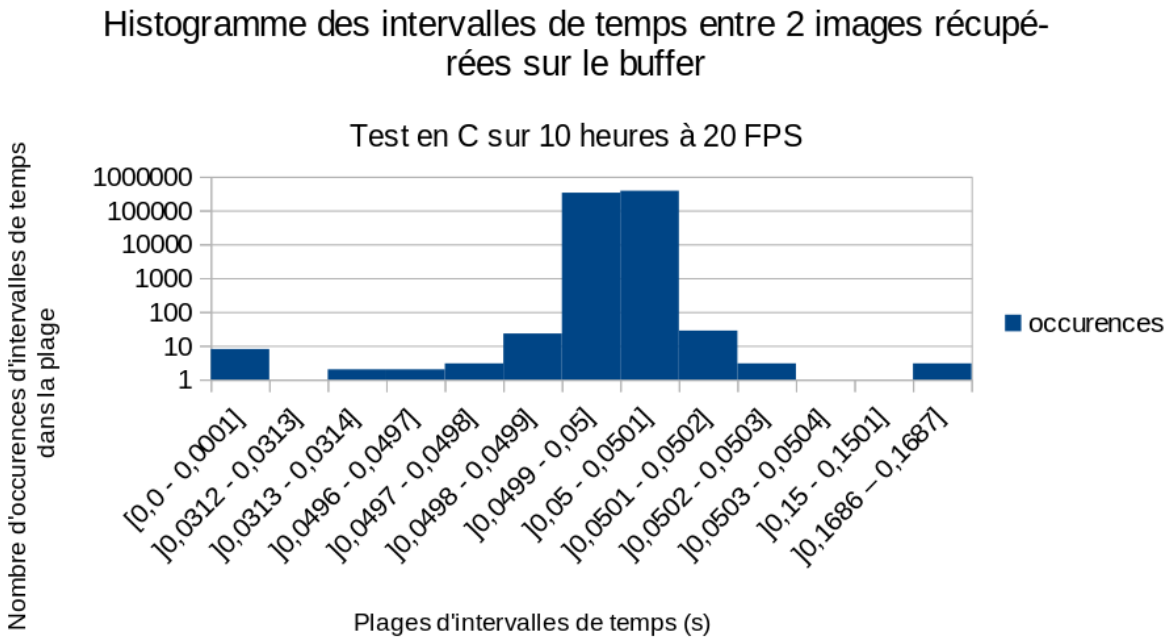


Figure 15 : Histogramme du programme en C

Le test du compteur d'image donne aucune perte (on obtient rigoureusement 720000 images acquises). Sur l'histogramme, on observe que les intervalles de temps dévient très peu de la période de rafraîchissement de 0.5 secondes, de plus les pics d'intervalle supérieurs sont dans des multiples

de la période, ce qui est parfaitement cohérent, et sont intégralement compensé par le pic de gauche ou les images sont rattrapées quasi-instantanément. Enfin comme le programme en C donne des résultats très satisfaisants à 20 FPS nous l'avons testé à 30 FPS. Les résultats sont les suivants :

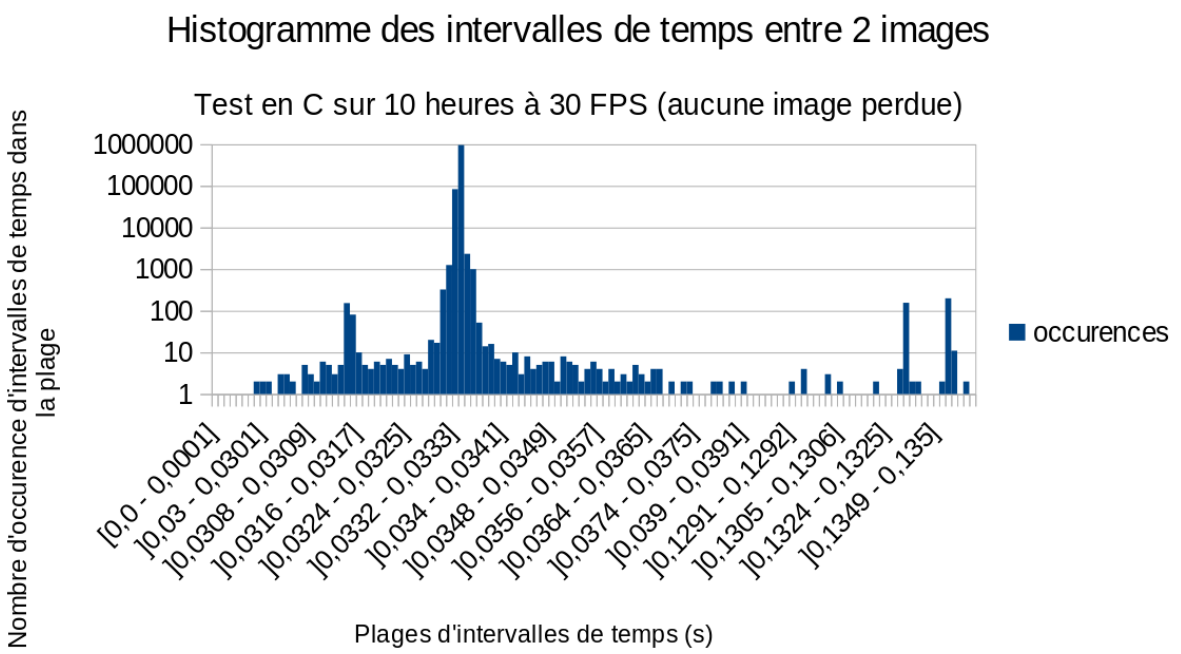


Figure 16 : Histogramme du programme C à 30 FPS

Encore une fois nous n'avons perdu aucune image, ici cependant nous avons plus d'images retardées, car les contraintes sont plus fortes, les erreurs de paquets doivent arriver plus souvent. Mais ces images restent intégralement compensées par des images acquises plus vites. D'ailleurs si l'on fait une moyenne de tous ces intervalles de temps, on obtient une valeur très proche de 0.0333 secondes.

3. *Commentaires et suggestions

Le programme en C fonctionne donc beaucoup mieux que les scripts pythons. En effet celui-ci donne les résultats attendus du projet. Cette expérience confirme une chose : pour une application en temps réel à destination d'un système embarqué il vaut mieux toujours préférer le C au Python. De plus on peut mentionner un fait très important qui est la documentation de l'API C d'Aravis qui est bien plus complète (et avec des exemples) que la documentation de l'API Python. Ainsi, le programme C d'acquisition m'a pris que quelques minutes, alors que le script Python m'aura pris plusieurs jours. (Cependant cela n'est pas tout à fait comparable car les heures passées à faire fonctionner le script Python m'ont permis de bien maitriser le sujet, et ainsi, le code d'exemple en C était facile à comprendre et à modifier). Je conseil fortement de choisir définitivement le C pour la suite de ce projet et d'abandonner tout le travail effectué en Python pour l'acquisition des images de la caméra FRIPON. En effet les expériences et les résultats ont prouvé que la solution en C est bien meilleure.

Si l'on choisi la solution en C il sera sans doute beaucoup plus simple d'intégrer la chaîne d'acquisition à la chaîne de détection. En effet on pourrait faire un programme C utilisant deux threads : un pour l'acquisition et un autre pour la détection (ce qui pourrait éviter des complications avec les « pipes »).

Conclusion

Ce qui a été fait

Nous avons exploré différents outils pour mettre en place la chaîne d'acquisition du projet.

On peut récupérer les images provenant du flux direct de l'ISS avec un script python en utilisant streamlink et les transférer via des « pipes » nommés à la chaîne de détection.

La caméra FRIPON a été testée et il est difficile d'en obtenir des images exploitables scientifiquement compte tenu de l'environnement dans lequel elle a été posée.

Pour réaliser l'acquisition des images en temps réel de la caméra FRIPON, il vaut mieux utiliser un programme en C car celui-ci répond bien mieux aux contraintes du projet. Le programme d'acquisition en C est donc fonctionnel.

Ce qu'il reste à faire et les points à améliorer

Le travail restant est conséquent, le meilleur moyen de détailler le travail restant est d'en dresser une liste que voici :

- Ajouter le transfert des images par « pipes » au script d'acquisition des images directes de l'ISS et le relier à la chaîne de détection Meteorix.
- Contacter l'observatoire de Paris pour connaître les paramètres de la caméra qu'ils utilisent.
- Ajuster les paramètres de la caméra en fonctions des météores observé (cette tâche dépend des météores et peut donc être très longue).
- Ajouter l'enregistrement en séquence d'images au format PGM au programme C d'acquisition.
- Transférer les images acquises du programme d'acquisition à la chaîne de détection (en ajoutant une numérotation aux images).
- Enregistrer les images faisant l'objet d'une détection.
- Récupérer les régions d'intérêt et proposer un affichage sur l'enregistrement. (FMDT peut déjà fournir une vidéo avec des carrés autour des météores. Voir comment l'implémenter dans la solution complète).
- Ajouter les fonctionnalités permettant de paramétrer les enregistrements (durée avant et après la détection).
- Ajouter d'autres fonctionnalités pour rendre le système plus simple d'utilisation.

Commentaires et améliorations possibles

Ce stage m'a appris beaucoup sur le travail en laboratoire. Chaque solution doit être testée et évaluée qualitativement et quantitativement afin d'en assurer la fiabilité. L'équipe du LIP6 m'a particulièrement bien guidé dans la façon de produire un travail de qualité et d'en justifier la pertinence par des expériences, des tests et des mesures précises.

J'ai pu constater pas mal de défauts dans le travail que j'ai réalisé au cours de ce stage. Tout d'abord mon travail n'est pas suffisamment bien planifié, il m'est facile de partir sur des tâches annexes, d'importance moindre. Les objectifs et les tâches ne sont pas clairement posés à l'avance. L'utilisation de la méthode Scrum m'a aidé à rester focus le plus possible sur les tâches essentielles mais l'organisation de mon travail n'est pas encore optimale. De plus mes travaux informatiques sont assez mal organisés, les fichiers sont mal organisés, leurs noms ne sont pas très clairs, le code n'est pas très bien commenté. Et lorsque je rends un code propre, cela me prend beaucoup de temps. Il faudrait que je trouve un bon compromis entre rédaction rapide et clarté du code. J'ai remarqué au cours de ce stage qu'il me manque une compétence essentielle : l'utilisation du versionnage de git aboutissant à

plein de fichiers avec des versions différentes. (Peut-être approfondir ce sujet en cours d'Outils Pour l'Informatique) (OPI).

Pour conclure, ce stage est une très belle expérience scientifique, technique et humaine qui m'a appris beaucoup pour ma carrière d'ingénieur en électronique, informatique et systèmes embarqués et pour mon projet professionnel de travailler dans le spatial. C'est une expérience que je referais volontiers, avec peut-être plus de temps pour avancer sur le projet.

Index/Bibliographie

[1] A. Cassagne, «Convention de stage,» Paris, Sorbonne Université - Polytech Sorbonne | Laboratoire d'Informatique de Paris 6, 2023.

[2] A. multiples, «Stack Overflow,» [En ligne]. Available: https://stackoverflow.com/.

[3] OpenAI, «ChatGPT,» OpenAI, [En ligne]. Available: https://chat.openai.com.

[4] «ffmpeg Documentation,» [En ligne]. Available: https://ffmpeg.org/ffmpeg.html.

[5] O. Aubert, «API Documentation (VLC Python API),» [En ligne]. Available: https://www.olivieraubert.net/vlc/python-ctypes/doc/.

[6] OpenCV, «opencv/opencv-python,» OpenCV, 08 04 2016. [En ligne]. Available: https://github.com/opencv/opencv-python.

[7] OpenCV, «OpenCV: OpenCV modules,» OpenCV, [En ligne]. Available: https://docs.opencv.org/4.x/.

[8] The Python Software Foundation, «Python 3.11.5 Documentation,» The Python Software Foundation, [En ligne]. Available: https://docs.python.org/3/.

[9] Wikipedia, «GenICam,» Wikipedia, 10 07 2023. [En ligne]. Available: https://en.wikipedia.org/w/index.php?title=GenICam&oldid=1164684267. [Accès le 08 09 2023].

[10] aravisproject, «Aravis: Installation and Debug,» [En ligne]. Available: https://aravisproject.github.io/aravis/building.html.

[11] EmmanuelP et anathesys, «Releases · AravisProject/aravis,» [En ligne]. Available: https://github.com/AravisProject/aravis/releases.

[12] «Features | Basler Product Documentation,» [En ligne]. Available: https://docs.baslerweb.com/features.

[13] «Aravis 0.8 (0.8.28) - Aravis 0.8,» [En ligne]. Available: https://lazka.github.io/pgi-docs/Aravis-0.8/index.html.

[14] Vigie Ciel FRIPON, «Observer le ciel avec votre caméra – Vigie-ciel,» 09 11 2017. [En ligne]. Available: https://www.vigie-ciel.org/observer-ciel-camera/.

[15] «BD00054801_Basler_acA1300-30gm_EMVA_Standard_1288.pdf».

[16] «Google Maps,» Google Maps, [En ligne]. Available: https://www.google.fr/maps/@48.8474743,2.3554044,20.52z?entry=ttu.

[17] FRIPON Team - Hosted and maintained by OSU Pytheas institute and Paris Observatory, «Archives,» 19 février 2023. [En ligne]. Available: https://www.fripon.org/.

[18] NASA, «Live High-Definition Views from the International Space Station (Official NASA Stream),» 5 juin 2023. [En ligne]. Available: https://www.youtube.com/watch?v=KG6SL6Mf7ak.

[19] Streamlink, «streamlink,» PyPI, 20 juillet 2023. [En ligne]. Available: https://pypi.org/project/streamlink/. [Accès le 25 07 2023].

[20] Streamlink, «API Guide - Streamlink 6.0.0 documentation,» Streamlink 6.0.0 documentation, [En ligne]. Available: https://streamlink.github.io/api_guide.html. [Accès le 25 07 2023].

[21] Streamlink, «API Reference - Streamlink 6.0.0 documentation,» Streamlink 6.0.0 documentation, [En ligne]. Available: https://streamlink.github.io/api.html. [Accès le 25 07 2023].

[22] «meteor-stream,» GitLab, [En ligne]. Available: https://gitlab.lip6.fr/sog/meteor-stream. [Accès le 25 07 2023].

[23] O. Aubert, «python-vlc: VLC bindings for python.,» PyPI, [En ligne]. Available: http://wiki.videolan.org/PythonBinding.

[24] Olivier Aubert, «python-vlc,» PyPi, [En ligne]. Available: https://pypi.org/project/python-vlc/.

[25] OpenCV, «opencv-python: Wrapper package for OpenCV python bindings.,» [En ligne]. Available: https://github.com/opencv/opencv-python.

[26] K. Kroening, «ffmpeg-python: Python bindings for FFmpeg - with complex filtering support,» [En ligne]. Available: https://github.com/kkroening/ffmpeg-python.

Annexes

POLYTECH SORBONNE

SORBONNE UNIVERSITÉ

Sujet de stage obligatoire : Acquisition et traitement temps réel de flux vidéo

Date du stage : Du 05/06/2023 Au 28/07/2023
Lieu de stage (si différent de l'adresse de l'organisme d'accueil) :
4, Place Jussieu 75005 Paris FRANCE
DUREE DU STAGE : ☐ Heures ☐ Jours ☑ Semaines ☐ Mois (choisir la mention)

Nombre de semaines de présence effective en stage : 8

Durée horaire hebdomadaire : 35h

AMENAGEMENTS PARTICULIERS (par exemple : présence la nuit, jours fériés ou dimanche, etc.) :

MONTANT BRUT DE LA GRATIFICATION : 4,05 € / h
☑ Montant minimum réglementaire défini à l'article L.124-6 du code de l'éducation
☐ Montant supérieur au minimum réglementaire défini à l'article L.124-6 du code de l'éducation
☐ Stage non gratifié

Autres avantages accordés : Remboursement 50% titre de transport

MODALITES DES CONGES ET DES AUTORISATIONS D'ABSENCE DURANT LE STAGE :
Voir la convention collective de l'établissement d'accueil

COMPETENCES A ACQUERIR OU A DEVELOPPER AU COURS DU STAGE :
Compétences techniques, Esprit pratique, Autonomie, Sens de l'analyse, recul, abstraction, Investissement personnel, Ouverture d'esprit, curiosité, Intégration au sein de l'équipe, Expression orale et écrite, Aptitude à diriger.

ACTIVITES CONFIEES (selon des objectifs de formation et les compétences à acquérir) :

Le stage portera sur l'acquisition et le traitement temps réel de flux vidéo. L'équipe ALSoC du LIP6 a développé une chaîne de traitement de l'image capable de détecter des météores depuis une vidéo. Ce prototype est fonctionnel, mais n'a jamais été déployé en conditions réelles. L'objectif de ce stage est d'intégrer ce travail à un flux vidéo pour récupérer des séquences d'images contenant des météores en temps réel. Un des enjeux du stage est de produire une solution capable de s'exécuter sans interruption. Des problèmes de gestion de ressource (allocation mémoire / affectation CPU) et de résilience seront des enjeux importants de ce stage. La station spatiale ISS fournit un flux vidéo continue, ce flux est un des objectifs visé. D'autres flux peuvent être envisagés, comme l'installation d'une caméra FRIPON sur le toit de Jussieu. Ce nouveau système à vocation à être embarqué dans un futur nanosatellite (mission Meteorix).

MODALITES DE VALIDATION DE STAGE (NB d'ECTS et modalités alternatives en cas de rupture de la convention à l'initiative de Sorbonne Université) :
Rapport de stage et Soutenance

Page 3

Figure 17 : Page 3 de la convention de stage mentionnant le titre et les activités du stage

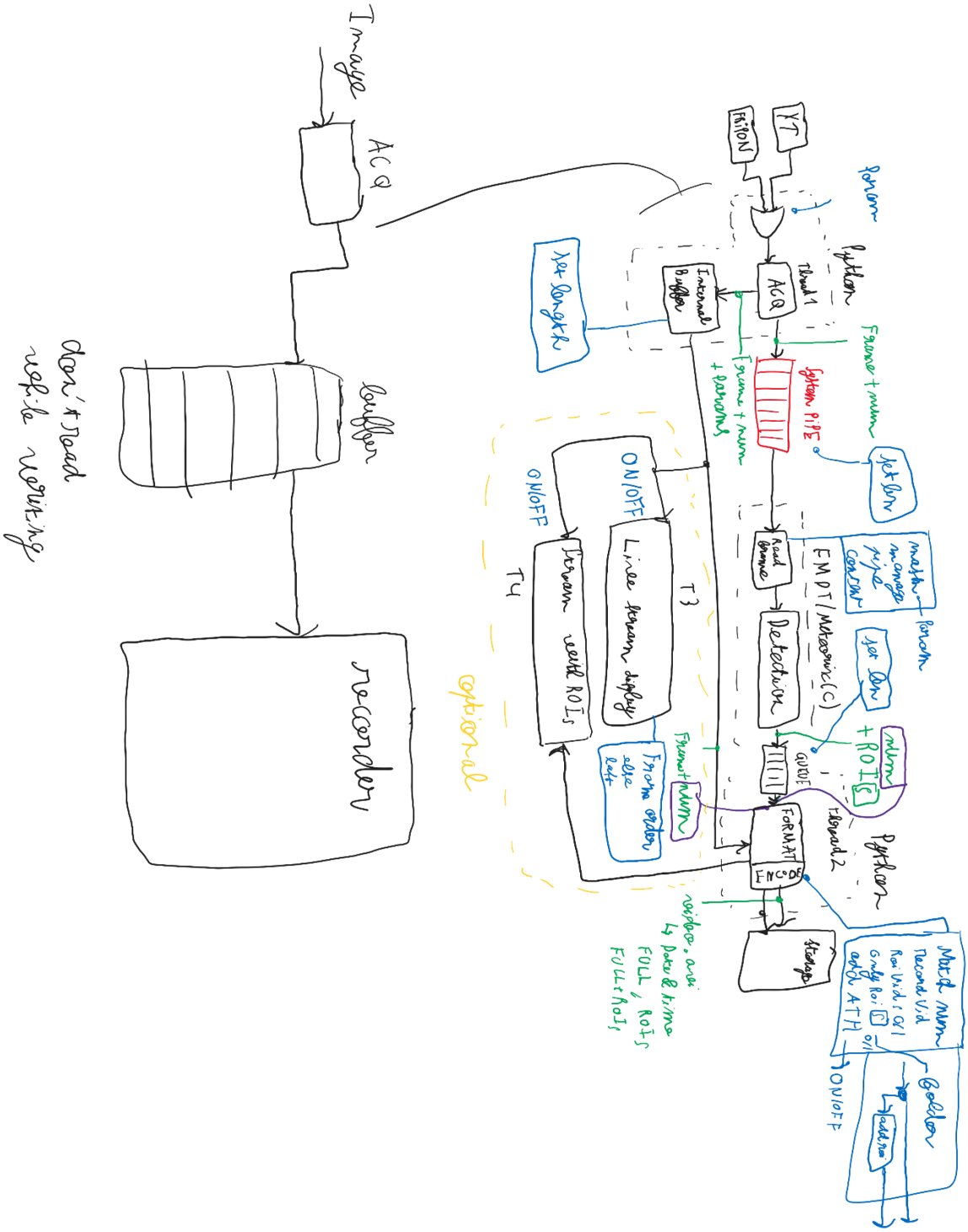


Figure 18 : Croquis de l'architecture du projet (A. Cassagne, modifié)