Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

# Filière Systèmes industriels

## Orientation Power & Control

# Travail de bachelor
# Diplôme 2020

*Mario Brunetti*

## *Capteur de la distribution de l'intensité lumineuse dans le ciel*

*Professeur*
Joseph Moerschell

*Expert*
Marc Nicollerat

*Date de la remise du rapport*
27.08.2020

# Données du travail de diplôme
## *Daten der Diplomarbeit*

| Filière / Studiengang<br>**SYND** | Année académique **/** *Studienjahr*<br>**2019/20** | No TD */ Nr. DA*<br>**pc/2020/62** |
|---|---|---|
| Mandant **/** *Auftraggeber*<br>☒ HES—SO Valais<br>☐ Industrie<br>☐ Etablissement partenaire<br>*Partnerinstitution* | Etudiant **/** *Student*<br>**Mario Brunetti**<br><br>Professeur / *Dozent*<br>**Joseph Moerschell** | Lieu d'exécution */ Ausführungsort*<br>☒ HES—SO Valais<br>☐ Industrie<br>☐ Etablissement partenaire<br>*Partnerinstitution* |
| Travail confidentiel / *vertrauliche Arbeit*<br>☐ oui / ja [1] ☒ non / nein | Expert */ Experte (données complètes)*<br>**Marc Nicollerat**<br>Microdev Sàrl, Chemin du Milieu 21B, 1920 Martigny | |

Titre */ Titel*

**Capteur de la distribution de l'intensité lumineuse dans le ciel**

Description */ Beschreibung*

Comme les puissances installées des panneaux photovoltaïques augmentent rapidement, il devient de plus en plus intéressant de pouvoir prédire à court terme, dans un horizon inférieur au quart d'heure, leur production. Ceci permettrait p.ex. d'optimiser l'emploi de batteries de stockage capables de lisser la production solaire. Un dispositif de mesure sur lequel une telle prédiction pourra s'appuyer, est un capteur mesurant la distribution de l'intensité lumineuse dans le ciel. Sa réalisation se heurte au problème que l'intensité provenant du soleil non voilé est plusieurs ordres de grandeurs supérieure à celle du reste du ciel. Dans ce projet, nous proposons de réaliser la mesure par une matrice CCD à lecture progressive, où chaque élément reçoit le rayonnement d'un secteur du ciel, sans passer par une optique. Le projet consiste à concevoir, réaliser et tester un prototype du capteur, ensemble avec une acquisition des mesures sur PC. Il comporte une partie électronique (module capteur), mécanique (impression 3D des puits) et informatique (acquisition des mesures).

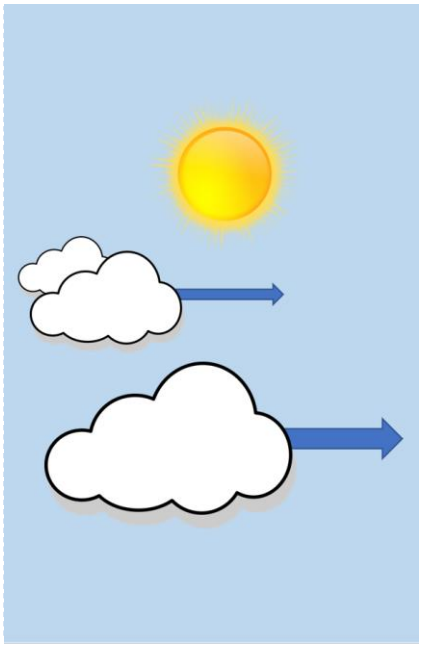
Objectifs / *Ziele*

— Etude de l'architecture du système, avec ses propriétés, spécification des performances de mesure
— Conception et réalisation d'un prototype de capteur avec électronique de traitement et acquisition des données sur PC
— Conception et réalisation de la partie mécanique (puits de lumière) et optique (miroir d'acquisition)
— Développement du logiciel d'acquisition sur PC, stockage et affichage des mesures
— Définition des algorithmes de traitement des données acquises, pour obtenir les quantités de rayonnement direct et diffus, la vitesse et la hauteur des nuages
— Proposition d'un concept de prédiction d'ensoleillement local.

| Signature ou visa / *Unterschrift oder Visum* | Délais / *Termine* |
|---|---|
| Responsable de l'orientation / filière<br>*Leiter der Vertiefungsrichtung / Studiengang:*<br>F. Moghaddam<br><br>[1] Etudiant / *Student* :<br>Mario Brunetti | Attribution du thème / *Ausgabe des Auftrags:*<br>**25.05.2020**<br><br>Présentation intermédiaire / *Zwischenpräsentation*<br>**Semaine / Woche 26** (22.06 – 26.06.2020)<br><br>Remise du rapport / *Abgabe des Schlussberichts:*<br>**27.08.2020, 12:00**<br><br>Exposition / *Ausstellung der Diplomarbeiten:*<br>**28.08.2020** (si autorisé / *falls genehmigt*)<br><br>Défense orale / *Mündliche Verfechtung:*<br>**Semaine / Woche 36** (31.08 – 04.09.2020) |

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
*Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.*

Rapport reçu le / *Schlussbericht erhalten am* …….. Visa du secrétariat / *Visum des Sekretariats* ……..



# Capteur de la distribution de l'intensité lumineuse dans le ciel

Diplômant Mario Brunetti

## Objectif du projet

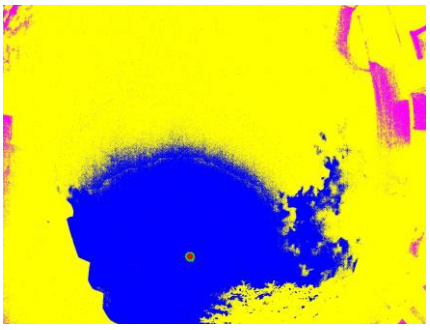
La production d'énergie à l'aide de panneaux photovoltaïques étant en pleine croissance, il devient de plus en plus intéressant de pouvoir prédire à court terme, dans un horizon inférieur au quart d'heure, l'irradiation solaire et par conséquent leur production.
Un dispositif de mesure sur lequel une telle prédiction pourra s'appuyer, est un système d'imagerie mesurant la distribution de l'intensité lumineuse dans le ciel. Sa réalisation se heurte au problème que l'intensité provenant du soleil non voilé est de plusieurs ordres de grandeurs supérieure à celle du reste du ciel.

## Méthodes | Expériences | Résultats

Afin d'atténuer cette intensité lumineuse, un filtre absorbant est placé devant le capteur photographique. Plusieurs captures sont réalisées avec des durées d'exposition différentes puis fusionnées afin d'augmenter la plage de mesure du capteur.

L'acquisition est réalisée à l'aide d'un module camera bon marché. Deux modules caméras ont été testés durant le projet afin de comparer leurs performances.

Le filtre absorbant utilisé ne convient pas pour cette application. Il faut utiliser un filtre plus fort pour empêcher complètement la saturation des mesures. Cependant la méthode convient afin d'exploiter les images fusionnées dans l'objectif de réaliser la détection de nuages dans le ciel.



Définition des différentes zones d'intensité de de l'image.

Image résultante puis convertie dans une plage dynamique visible pour l'afficher

Travail de diplôme
| édition 2020 |

Filière
*Systèmes industriels*

Domaine d'application
*Power and Control*

Professeur responsable
*Joseph Moerschell*
*joseph.moerschell@hevs.ch*

# Table des matières

# Liste des acronymes

**ADC** Analog to Digital Converter. 0, 8, 10, 12

**CCD** Charge-coupled device. 0

**HDR** High Dynamic Range. 0, 20, 28, 33, 35

**ND** Neutral density. 0, 18, 19, 35

**PV** Photo-voltaïque. 0, 2

**SSH** Secure Shell. 0, 8

# 1 Introduction

La production d'énergie à l'aide de panneaux photo-voltaïques étant en pleine croissance, il devient de plus en plus intéressant de pouvoir prédire à court terme, dans un horizon inférieur au quart d'heure, l'irradiation solaire et par conséquent leur production. Ceci permettrait par exemple d'optimiser l'emploi de batteries de stockage capables de lisser la production solaire.[1]
Comme nous pouvons le distinguer sur la figure 1, l'irradions solaire absorbée par une cellule photo-voltaïques peut subir d'importantes fluctuations causées par les passages nuageux.

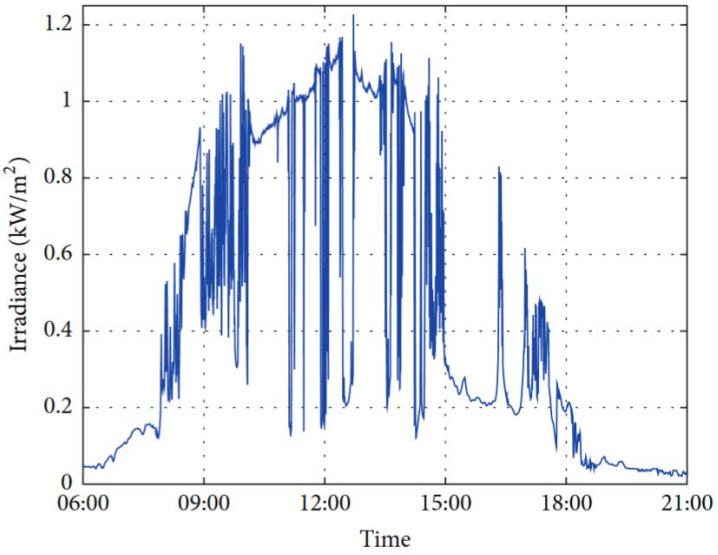



Figure 1 – Fluctuation de l'irradiation solaire absorbée par une cellule PV causé par le passage de nuages[2]

Un dispositif de mesure sur lequel une telle prédiction pourra s'appuyer, est un système d'imagerie mesurant la distribution de l'intensité lumineuse dans le ciel. Sa réalisation se heurte au problème que l'intensité provenant du soleil non voilé est de plusieurs ordres de grandeurs supérieure à celle du reste du ciel.
Différents fournisseurs proposent déjà des solutions compactes mais elles s'avèrent être coûteuse. L'objectif de ce travail est donc de proposer une solution bon marché afin de pouvoir réaliser les captures du ciel. Afin de réaliser cette solution économique, un module caméra connectée à un nano-ordinateur Raspberry pi est utilisée pour les mesures.
Par la suite, une méthode de détection et prévision des mouvements des nuages devra être développée, ainsi qu'un boîtier étanche permettant de loger le système d'imagerie.
Un script codé en Python 3.7 est exécuté sur le Raspberry afin d'effectuer les captures, puis les captures sont transférer sur un ordinateur et analysée à l'aide de Matlab R2019b.

## 1.1 Solutions existantes

InstaCast de Reuniwatt, horizon 30 min
https://reuniwatt.com/en/intrahour-solar-forecasts-instacast/
All sky System Aplhea par Alcor System
http://www.alcor-system.com/new/AllSky/Alphea_camera.html
ASI-16 All Sky par Schreder-CMS
http://www.schreder-cms.com/en/camera.htm

## 1.2 Cahier des charges

Définir une solution afin de réaliser des captures du ciel qui soit exploitable dans l'objectif de prédiction d'ensoleillement.

## 1.3 Lumière du soleil et panneaux photo-voltaïques

La lumière qui atteint le panneaux solaires peut être catégorisée en trois composantes selon son type de rayonnement.

- Le rayonnement direct qui provient directement du soleil sans obstacles sur sa trajectoire (nuages, ...)
- Le rayonnement diffus qui provient de multiples diffractions par les nuages
- Le rayonnement réfléchi appelé aussi albédo qui résultent de la réflexion du rayonnement direct par le sol, qui dépends du pouvoir réfléchissant de sa surface.

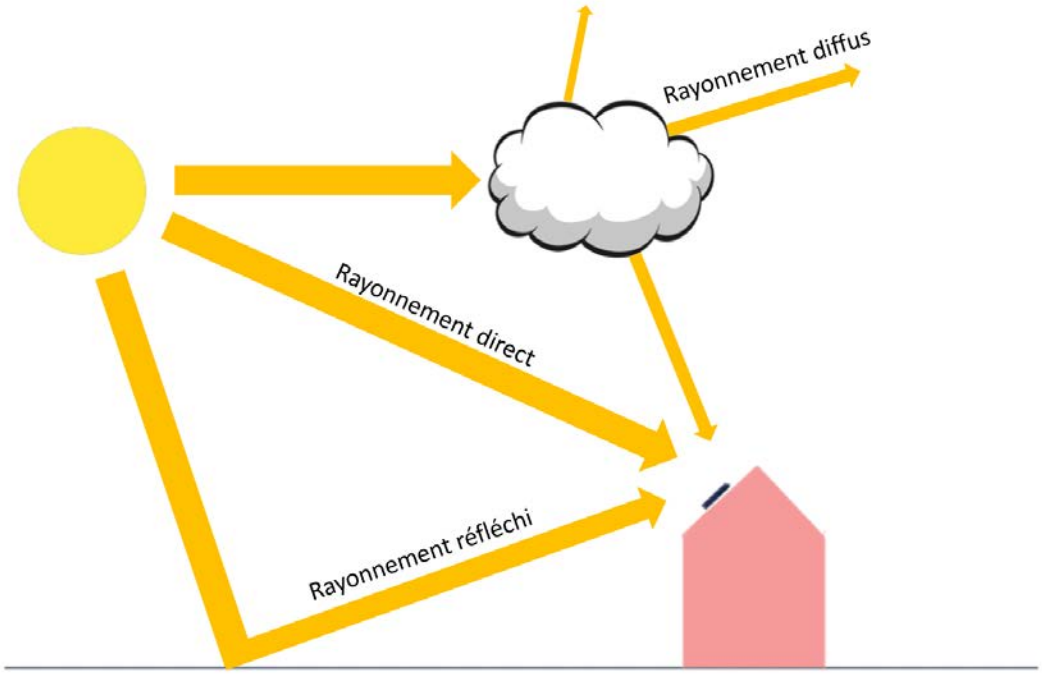



Figure 2 – types de rayonnement

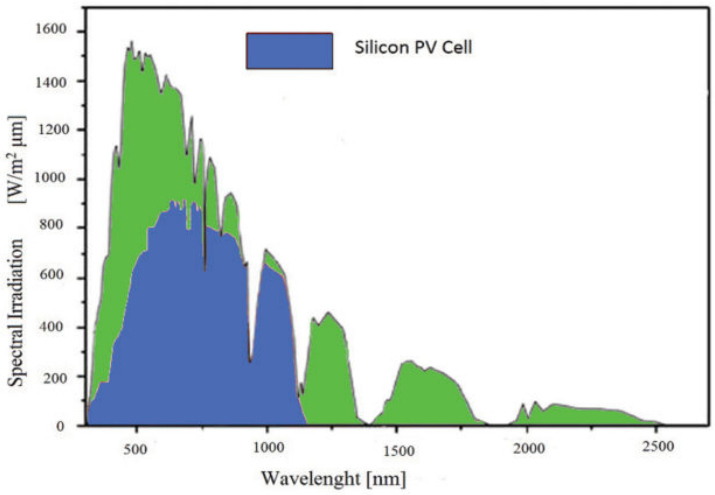



Figure 3 – Réponse spectrale d'une cellule PV face à la lumière du soleil[3]

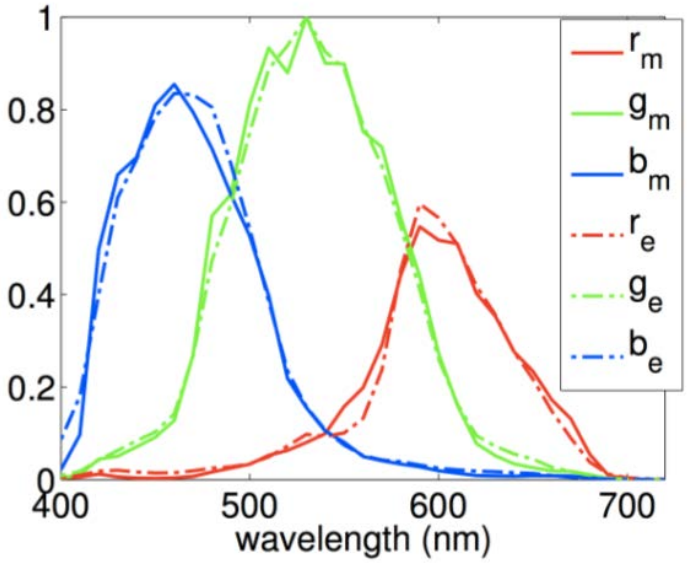



Figure 4 – Réponse spectrale caméra digital [4]

Sur la figure 3, nous pouvons observer la bande de réponse spectrale d'une cellule photo-voltaïque conventionnel face à la lumière du soleil. La zone en bleu représente la partie du spectre qui peut être absorbée par une cellule PV au silicium à jonction unique. Le spectre terrestre lumineux du soleil est représenté en vert. Le reste de l'énergie est converti en chaleur et autres pertes.[3]

Une autre étude réalisée [4] permet de montrer la sensibilité spectrale d'une caméra digitale. (cf. figure 4). Cette figure nous permet d'illustrer la page dans laquelle travaille un capteur photographique. Ainsi un capteur de caméra digitale et une cellule photo-voltaïque sont sensibles dans la même bande spectrale (environ 400nm à 1000nm). C'est par conséquent pertinent d'utiliser un capteur photographique pour notre application.

## 1.4 Plage dynamique nécessaire

Afin d'estimer la résolution nécessaire de plage de mesure de l'intensité lumineuse du capteur, il est indispensable d'estimer dans le pire des cas, la différence d'intensité des rayons qui proviennent directement du soleil par rapport au reste du ciel. Le pire des cas est lorsque le ciel est complètement dégagé et ne contient pas de nuages.
Concrètement l'objectif est d'éviter que la zone occupée par le soleil soit complètement saturée. Cela étant causé par la surexposition du capteur(c.f figure 5).

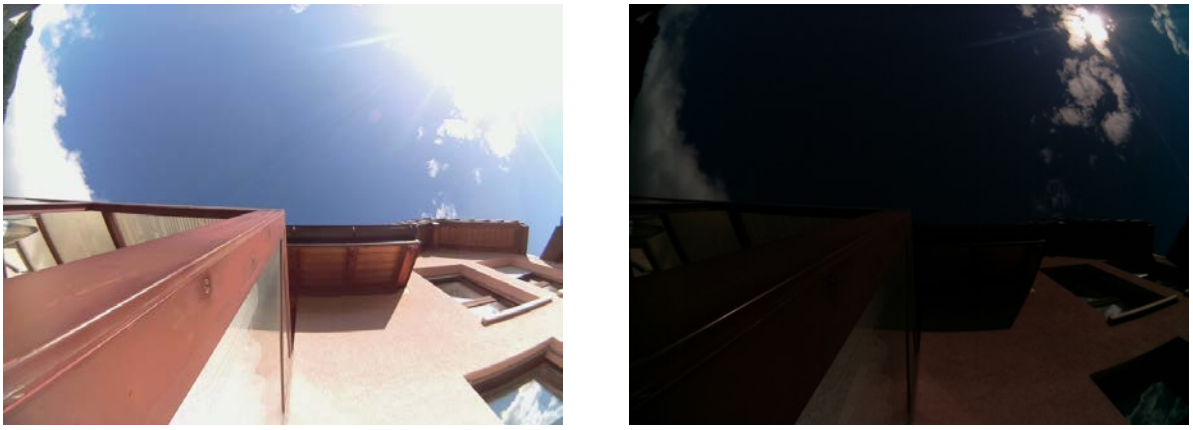


Figure 5 – Comparaison capture à droite réalisée quelques secondes plus tard que l'image de gauche mais avec un temps d'exposition plus faible. La zone occupée par le soleil étant complètement saturée, on ne distingue pas les nuages qui s'y situent

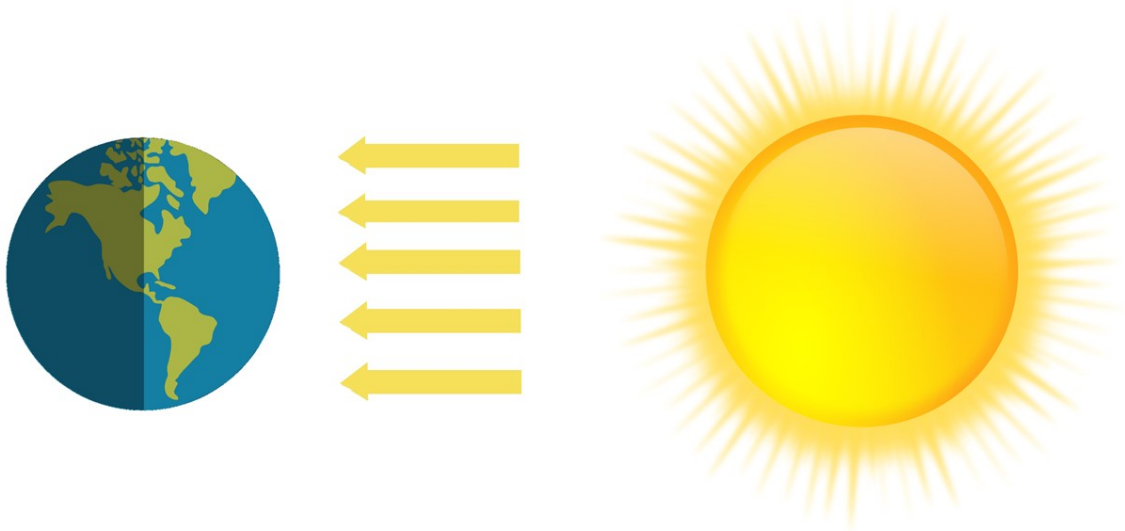


Figure 6 – Rayonnement solaire sur la Terre

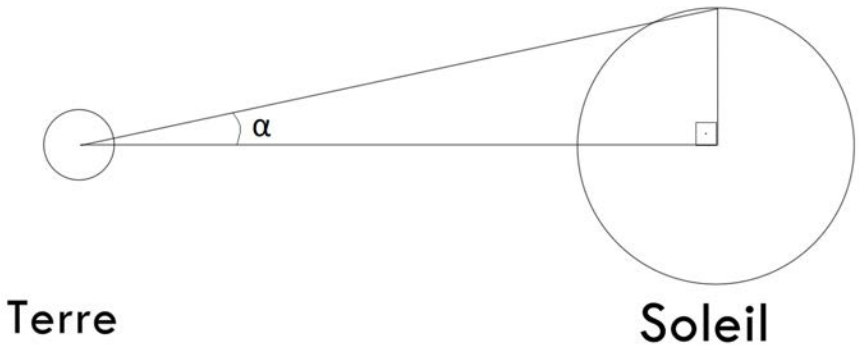



Figure 7 – Calcul de l'angle alpha

| | | |
|---|---|---|
| diamètre.terre | 12'742 | [km] |
| diamètre.soleil | $1.3927 \cdot 10^6$ | [km] |
| distance.terre.soleil | 149'597'870 | [km] |

$$\alpha = \arctan \frac{\frac{\text{diamètre.soleil}}{2}}{\text{distance.terre.soleil}} = 0.266[°] \tag{1}$$

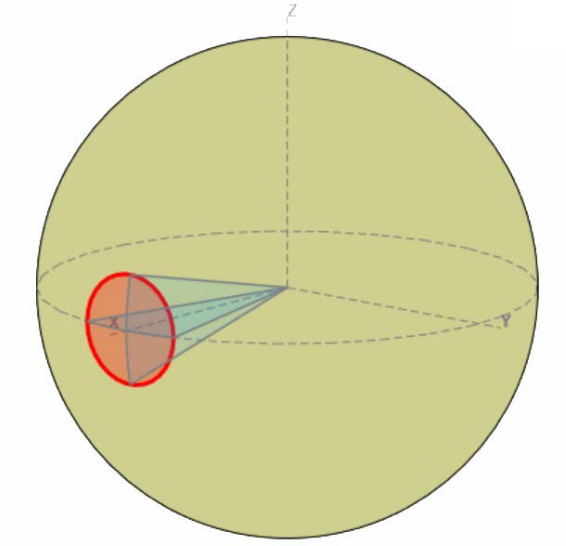



Figure 8 – Cône de révolution interceptant une sphère

La surface totale de l'hémisphère vaut $2\pi$ stéradian. Depuis la Terre, le diamètre apparent du Soleil représente un demi-angle $\alpha$ d'environ 0.266° (cf. équation: 1).
Dans le cas d'un cône de révolution de demi-angle $\alpha$ au sommet, l'angle solide $\Omega$ (cf. équation 2) se calcule par intégration sur la sphère, dans les domaines angulaires des coordonnées sphériques ce qui donne:[5]

$$\Omega = \iint \mathrm{d}^2\Omega = \int_0^{2\pi} \mathrm{d}\phi \int_0^{\alpha} \sin\theta \; \mathrm{d}\theta \; = 2\pi \int_0^{\alpha} \sin\theta \; \mathrm{d}\theta = 2\pi \left[-\cos\theta\right]_0^{\alpha}, \tag{2}$$

$$\Omega = 2\pi \left(1 - \cos\alpha\right) \tag{3}$$

$$\Omega = 2\pi(1 - \cos\alpha) = 2\pi(1 - \cos(\frac{0.266}{2})) = 2\pi \cdot 1.0296 \cdot 10^{-5}[\text{stéradian}] \tag{4}$$

L'estimation du rapport entre la proportion de ciel occupée par le soleil et le reste de l'hémisphère est par conséquent (cf. équation: 5):

$$rapport = \frac{\text{Angle.solide.cône}}{\text{Angle.solide.hémisphère}} = \frac{\Omega}{2\pi} = 10^{-5} = \frac{1}{100'000} \tag{5}$$

Dans le cas extrême, on a donc toute la puissance de rayonnement concentrée dans une portion du ciel qui vaut environ $\frac{1}{100'000}$ de la surface de l'hémisphère.

Si on désire pouvoir mesurer plus finement le reste du ciel, il faut augmenter la résolution de plage de mesure du capteur. Par conséquent, une résolution de la plage de mesure de $\frac{1}{10'000'000}$ serait idéale.

# 2 Matériels et méthodes

Comme précisé dans la section précédente, la résolution de la plage de mesure nécessaire doit être $\frac{1}{10'000'000}$. Une caméra digitale ne possède pas une plage dynamique aussi grande car elle sont proposée pour suffire à la plage dynamique de l'oeil humain.[6]
Pour exemple, une caméra avec un doté d'un convertisseur analogique/digital (ADC) d'une résolution de 12 bit, permet d'avoir une plage pouvant atteindre les valeurs de 0 à 4096. A cela il faut enlever le bruit. Si on prend les caractéristiques du module caméra V1 Raspberry sa plage dynamique est donnée à 68 dB ce qui correspond a une plage de 2512 valeurs. C'est loin d'être suffisant par rapport à l'estimation de $\frac{1}{10'000'000}$.
Pour compenser ce problème, la solution proposée est de filtrer la lumière de manière à l'atténuer puis de combiner plusieurs images capturées avec des temps d'expositions différents afin de reconstituer une image bénéficiant d'une plage dynamique augmentée.

## 2.1 Principe de capture

Le Raspberry est d'abord configuré afin de permettre une connexion SSH, permettant ainsi d'accéder à distance au Raspberry. Le but est de faciliter les captures qui seront réalisées à l'extérieur. Il faut pouvoir exécuter les commandes sans avoir besoin de déplacer un écran, un clavier et une souris. Un réseau Wi-fi est déployé à l'aide d'un smartphone de cette façon, le Raspberry peut être facilement piloter depuis l'ordinateur portable. Pour ce projet, le logiciel VNC viewer est utilisé pour piloter le Raspberry depuis l'ordinateur portable. En ce qui concerne le transfert de fichiers, le client FTP Filezilla est utilisé.

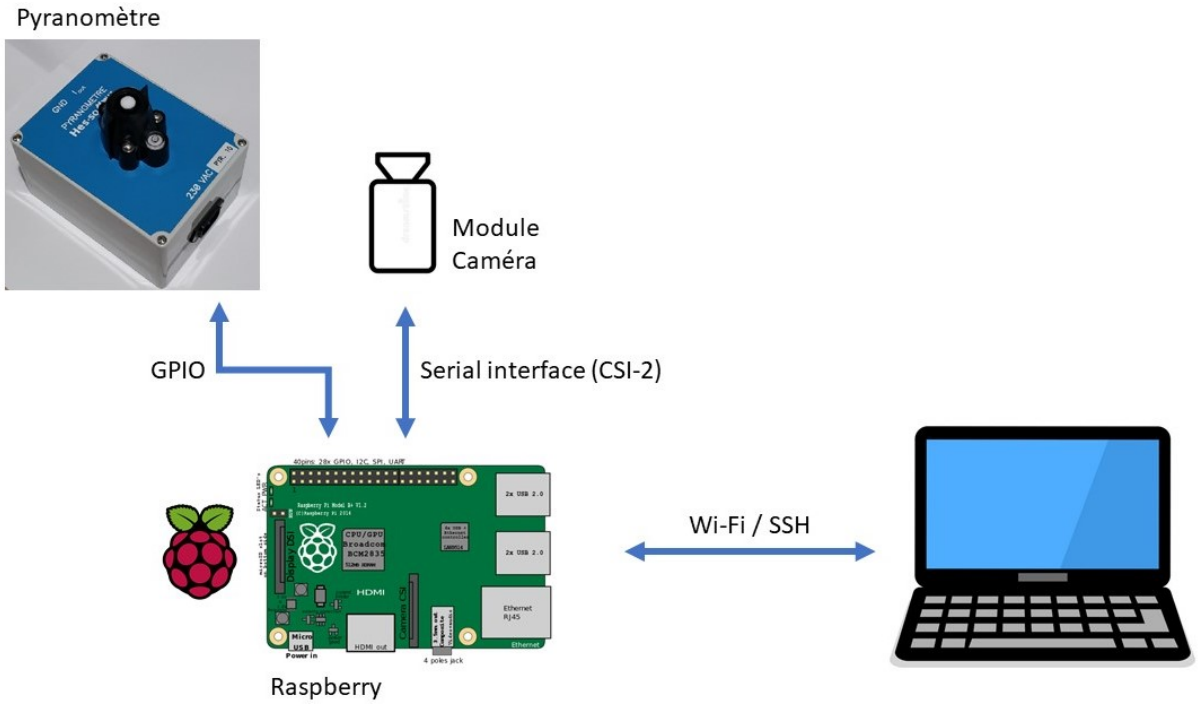



Figure 9 – Schéma

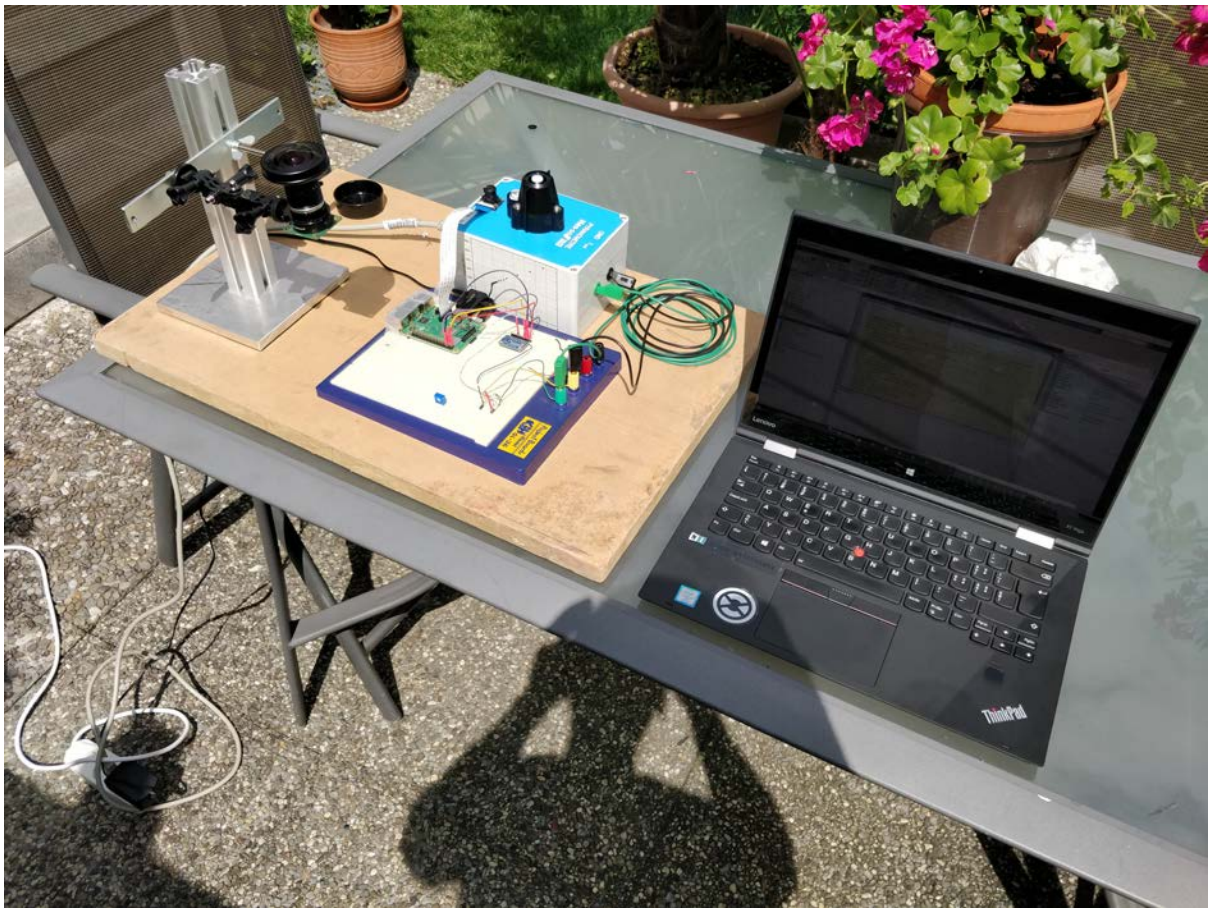


Figure 10 – Montage

## 2.2 Pyranomètre

Le pyranomètre ou capteur de rayonnement solaire mesure la somme des rayonnements directs et diffuses de l'irradiation solaire. Le transducteur du capteur, qui convertit le rayonnement incident en courant électrique, est une photo-diode au silicium à large réponse spectrale.

Figure 11 – Pyranomètre Davis 6450

| | |
|---|---|
| Fabricant | Davis Instruments |
| Modèle | Solar Radiation Sensor (6450) |
| Spectral Response | 400 to 1100 nanometers |
| Prix | 185 usd |

Un montage déjà réalisé par l'Hes-so Valais/Wallis permet de directement fournir une sortie de 4-20mA en fonction du rayonnement solaire. Une résistance de 250Ω est placé en sortie du montage et à l'aide d'un convertisseur analogique numérique ADS1115/16bits, la chute de tension est mesurée sur la résistance. La libraire Python "Adafruit ADS1x15" utilisée pour la lecture du signal est fournie par le fabricant et des exemples d'utilisations sont disponibles sur le site Web du fournisseur. [7]
La relation entre le signal mesuré par l'ADC et puissance du rayonnement solaire total en watts par mètre carré est définie par:

$$\text{Irradiation}[\frac{W}{m^2}] = \frac{400[\frac{W}{m^2}]}{5.3[mA]} * \frac{20[mA]}{5[V]} * \frac{4.096[V]}{32767[bit]} * \text{signal mesuré [bit]} + 80[\frac{W}{m^2}] \quad (6)$$

La mesure réalisée par le pyranomètre pourra être exploitée afin de comparer les mesures d'intensité effectuées par la caméra et l'irradiation totale reçue par le pyranomètre.


Figure 12 – Montage Hes-so pyranomètre

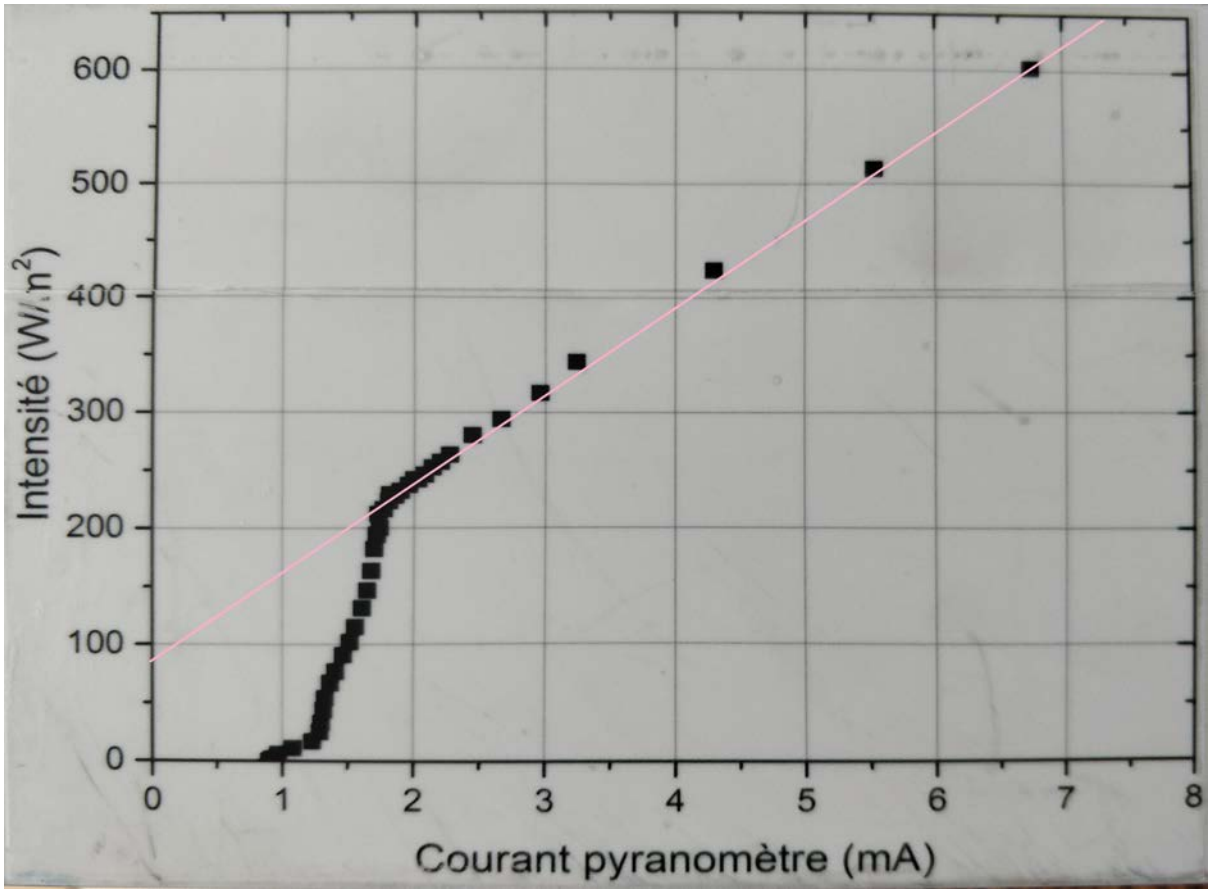



Figure 13 – Graphe montage Hes-so pyranomètre

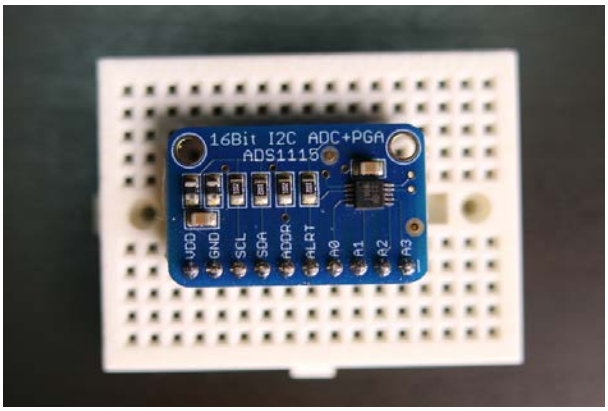



Figure 14 – Convertisseur analogique/digitale ADS1115[7]

| Convertisseur ADC | Adafruit ADS1115 |
|---|---|
| Résolution | 16 bits |

## 2.3 Capture de l'image: librairie Raspistill

Les captures sont réalisées à l'aide d'un script Python 3.7 exécuté sur le Raspberry. Une série de captures sont réalisées avec des temps d'exposition différents puis sauvegardées dans un dossier dans la carte mémoire du Raspberry.
"Raspistill" est la librairie utilisée pour effectuer les captures. Elle s'exécute via le terminal, le script python permet d'ouvrir un terminal est d'exécuter la commande de capture. En sortie un fichier de type jpg est sauvegardé sur la carte mémoire. Ce fichier jpg est déjà compressé (8 bits) par l'algorithme de capture. Afin de stocker les données bruts du capteur, il faut ajouter le flag '–raw' à la commande (cf. Déballage des données bruts: 2.5).
La boucle de capture du script Python réalise les captures pour les temps d'exposition définis suivants: 100us, 1ms, 10ms, 100ms, 200ms, 400ms, 600ms, 800ms, 1s, 2s, 3s,4s et 5s.

| Flag | Value | Description |
|---|---|---|
| –width | 4056 | Set the width of image |
| –height | 3040 | Set the width of image |
| –quality | 100 | Set the quality of jpeg |
| -sh | 0 | Set image sharpness (-100 - 100) |
| -co | 0 | Set image contrast (-100 - 100) |
| -br | 50 | Set image brightness (0 - 100) |
| -sa | 0 | Set image saturation (-100 - 100) |
| –ISO | 100 | Set the ISO to be used for captures (100 - 800) |
| -ev | 0 | Set EV compensation (-10 - 10) |
| –exposure | sports | Set exposure mode |
| –awb | sun | Set Automatic White Balance (AWB) mode |
| -drc | off | Enable/disable dynamic range compression |
| –metering | matrix | Set mettering mode |
| –raw' | | Ajoute les données bruts dans le fichier jpg |

Table 1 – Paramètres de commande raspistill (Flags) utilisés

## 2.4 Caractéristiques des modules caméras Raspberry

### 2.4.1 Éblouissement ("Blooming")

Un pixel est saturé lorsque sa capacité à puits plein est atteinte. Quand un puits de potentiel est rempli, la charge excédentaire peut déborder sur les puits voisins des pixels adjacents. [8] Il existe deux types d'exposition afin d'acquérir les images : L'acquisition peut être faite « ligne par ligne », le capteur exposant chaque ligne avec un décalage dans le temps. Chaque ligne fonctionne de façon indépendante, c'est ce que l'on appelle le "Rolling Shutter".
Le capteur peut également exposer l'intégralité de la matrice à un instant T puis faire la lecture et le transfert ligne par ligne par la suite. C'est le "Global Shutter".
Pour notre application, nous préférons utiliser un capteur de type "Rolling Shutter" afin d'éventuellement limiter ces effets de débordement.

### 2.4.2 Module caméra Raspberry v1

Les essais sont réalisé à l'aide d'un Raspberry pi 3 b+ et son module caméra v1. Un objectif grand angle est monté devant le capteur. Le désavantage avec cette caméra est lorsque l'on veut tester différent filtre ND. Il faut placer le filtre entre le sensor et l'objectif. Mais pour réaliser cela, il faut dévisser l'objectif et de ce fait, il faut à chaque fois réglé la focale de l'objectif.

| | |
|---|---|
| Sensor model | OmniVision ov5647 |
| Definition | 5 Megapixel |
| Active array size | 2592 (H) x 1944 (V) |
| ADC resolution | 10 bits |
| Output format | RAW10 Bayer BGGR |
| Minimal shutter speed | 1/83333 sec |
| Maximal shutter speed | 6 sec |
| S/N ratio | 36dB |
| Dynamic range | 68dB |

Table 2 – Caractéristiques du module v1 Raspberry

### 2.4.3 Module Caméra Raspberry HQ

Des essais sur une deuxième module caméra sont réalisée à l'aide d'un Raspberry pi 3 b+. Cette caméra dispose d'une monture C-mount standardisé pour y visser les objectifs. Ce qui facilite les essais. L'objectif utilisé est de type fisheye 185°.

| | |
|---|---|
| Sensor model | Sony IMX477 |
| Optical format | 1/2.3 inch |
| Pixel size | 1.55um x 1.55um |
| definition | 12.3 Megapixel |
| Active array size | 4056 (H) x 3040 (V) |
| ADC resolution | 12 bits |
| Output format | RAW12 Bayer BGGR |
| Minimal shutter speed | 1/8771.9 sec |
| Maximal shutter speed | 20 sec |

Table 3 – Caractéristiques du module hq Raspberry

Les données brut représentent le capteur en entier.

L'ouverture de cet objectif peut être réglé manuellement de f1.8 f2 f4 f8 f16. Plus le nombre est grand, moins la lumière passe. (cf. figure: 17)

L'ouverture du diaphragme est l'un des trois paramètres (avec la vitesse d'obturation et la sensibilité ISO) qui permet de contrôler l'exposition d'une photo pendant la capture.


Figure 15 – Fujinon FE185C086HA-1 Lens

| | |
|---|---|
| Fabricant | Fujinon |
| Modèle | FE185C086HA-1 |
| Prix | 449 usd |


Figure 16 – Image en fonction de la taille du capteur photographique, le capteur du module HQ est 1/2" [9]


Figure 17 – Ouvertures de diaphragme[10]

## 2.5 Déballage des données brutes 12 bits

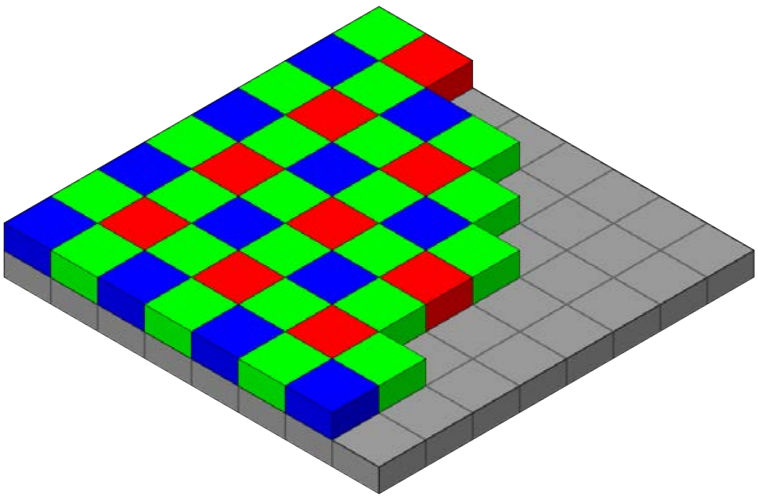


Figure 18 – Bayer

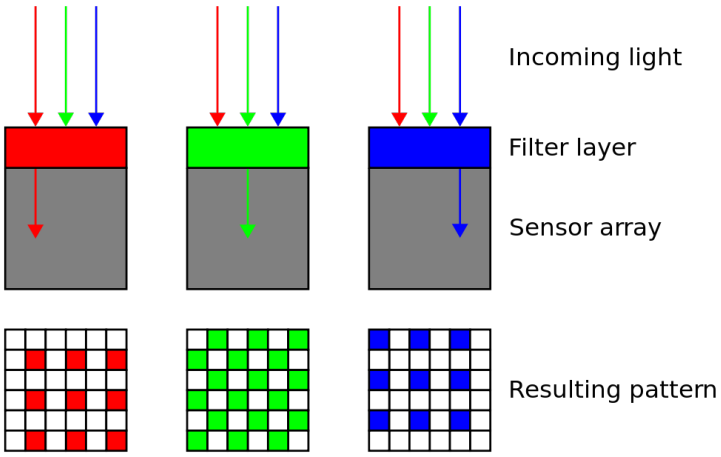



Figure 19 – filtre Bayer

Lorsque -r ou –raw est ajouter à la commande raspistill les données CFA (Color Filter Array) 12 bits sont ajoutées au fichier de type "jpg" 8 bits fournit en output par la capture raspistill. Ces données sont identifiables à la suite de la chaîne de caractère "BCRM" dans le fichier "jpg".

La première partie du bloc est un en-tête de $2^{15}$ octets, suivi des données du tableau de capteurs (3040+16)x(4056+28) écrites ligne par ligne. Il y a donc 3056 lignes au total.

Chaque ligne est constituée de 4056 éléments de 12 bits (4056*12/8 octets), suivis de 12 octets de zéros et de 16 octets de données non imageuses. Les 12 octets de zéros marquent la fin de la zone active jusqu'à la 3040e ligne. Ensuite, il y a 16 rangées supplémentaires de données système.

Du début de chaque ligne aux douze zéros, les valeurs brutes des pixels sont emballées en triplets : trois octets de 8 bits sont écrits pour chaque deux pixels de 12 bits dans le format suivant

AAAAAAAAA BBBBBBBBBB BBBBAAAA

Les deux premiers octets représentent les 8 bits de poids fort individuels tandis que le troisième contient les bits de poids faible respectifs de 4×2 comme indiqué.
Nous avons donc maintenant les données CFA brutes 12 bits de la caméra du Raspberry Pi sous forme d'entier 16 bits

En ce qui concerne la partie décodage de la matrice, toutes les détails sont fourni par l'article disponible içi: https://www.strollswithmydog.com/open-raspberry-pi-high-quality-camera-raw/ Un algorithme pour récupérer une image codée sur 16 bits est même disponible. Il est renommé read_PIHQraw_from_jpg.m pour ce projet (cf. annexe 6). [11]

Par contre ce script est utilisable pour le module HQ. En ce qui concerne le module v1, ce script ne fonctionne pas. J'ai donc travaillé avec les images 8 bits pour cette caméra.

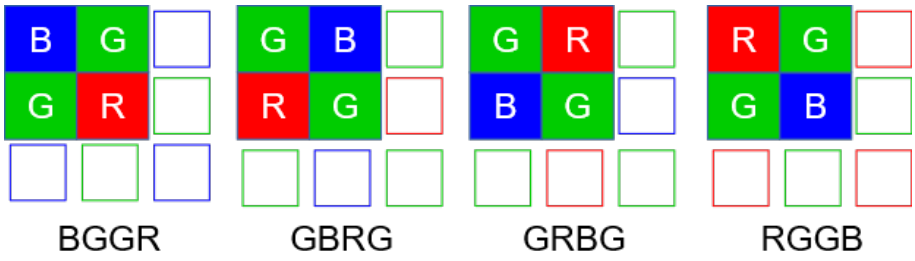



Figure 20 – Type de pattern Bayer, pour notre cas le type est BGGR

## 2.6 Filtre à densité neutre


Figure 21 – Feuille filtre ND 3.00

Un filtre à densité neutre, appelé filtre ND, est placé entre le capteur et l'objectif. L'idée est d'atténuer le surplus de lumière pour avoir une marge de manoeuvre plus grande avec le temps d'exposition.
Plusieurs filtres différents ont été mis à disposition par le laboratoire optique de l'Hes-so Valais/Wallis. Les essais pour déterminer le filtre à utiliser ont été réalisé à l'aide de la caméra HQ car elle possède un temps minimum de durée d'exposition (100us) plus élevé que la caméra V1 (10us). Par conséquent si une image peut être capturée sans saturation avec ce filtre sur la caméra HQ cela doit convenir également pour la caméra V1.
Après divers essais, on peut rapidement constater qu'il faut utiliser un filtre qui atténue fortement la lumière. Le filtre le plus absorbant disponible est donc utilisé soit le filtre 3.0 (cf. figure :21). Le filtre utilisé pour ce projet ne laisse passer qu'1/1024 de la lumière et permet d'augmenter le temps d'exposition d'un facteur 1024 en théorie.[12]

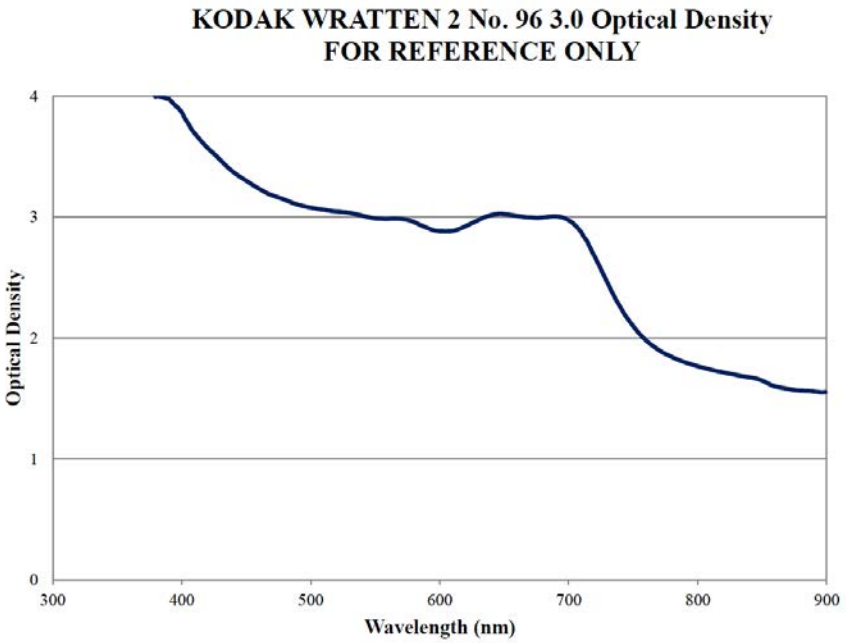
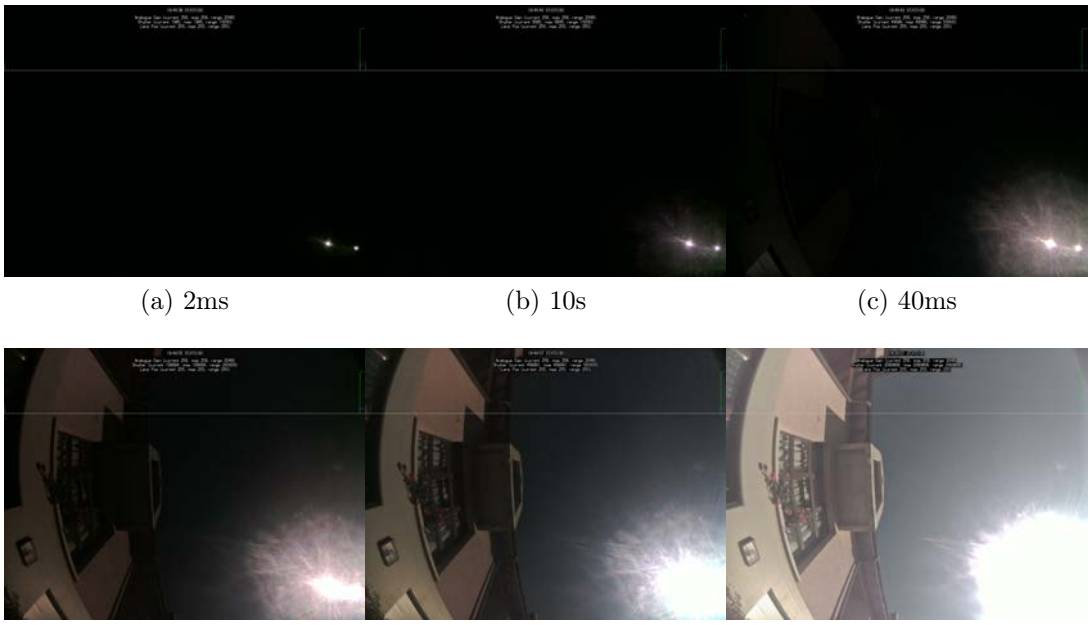



Figure 22 – Graphe influence spectrale filtre ND 3.00 utilisé

Des essais ont également été réalisé en superposant deux feuilles de filtre ND, malheureusement les images ne sont pas exploitables. La zone du soleil est altérée (cf. figures 23).



(a) 2ms (b) 10s (c) 40ms

(d) 200ms (e) 500ms (f) 3s

Figure 23 – Série de captures avec filtres superposés: module v1

## 2.7 Reconstitution d'une image non saturée

A partir d'une série de 5 captures réalisées avec différents temps d'exposition, nous pouvons reconstituer une image qui ne contient aucun pixels saturés.
(Remarque: Pour la suite du projet, il faudra envisagé plusieurs caméras pour réalisé les captures au même moment. Car une différence de temps d'exposition de quelques secondes permet au nuages de se déplacer entre les captures).
L'image brut 16 bit est d'abord convertie en image en niveaux de gris à l'aide de la fonction RGBtoGrey de Matlab.
L'algorithme pour la transformation en niveaux de gris utilise la pondération suivante sur les canaux rouge, vert, bleu [13]:

$$0.2989 * R + 0.5870 * G + 0.1140 * B \tag{7}$$

Une matrice représentant l'image HDR est ensuite reconstituée à partir des différents temps d'exposition, Les pixels compris entre une valeur supérieur à un seuil de saturation considéré à 65'530 et un seuil de 8'000 en dessous duquel les valeurs d'intensité sont considérées comme bruitées. Les valeurs des pixels à l'intérieur de cette plage sont récupérées et sauvegardées dans une image finale ne contenant ni pixels saturé, ni pixels bruités.
Remarque: Pour le module v1, les images étant en 8 bit, les seuils sont 10 et 254.
Une image représentant en différentes couleurs quels zones appartiennent à quels temps d'exposition est créée. Elle représente les différentes zones d'intensités lumineuses aussi.
Le script réalisé 'fusion_hq_ponderation.m' pour le module hq et 'fusion_v1_ponderation.m' pour le module v1 permettent de créer cette image (cf. script matlab : annexe 6, annexe 6).

Une matrice 32 bit est ensuite créée. Dans cette matrice, une pondération est réalisée sur les valeurs de pixels possédant un faible temps d'exposition afin de les ajuster par rapport au pixels qui proviennent de captures possédant un plus grand temps d'exposition (cf. tables : 4, 5).

Pour cette pondération, j'ai fait l'hypothèse que la relation entre le temps d'exposition et l'intensité lumineuse calculée a partir de l'image en niveaux de gris était proportionnelle.

| Temps d'exposition | Facteurs appliqués au pixels |
|---|---|
| 100us | 10000 |
| 10ms | 100 |
| 100ms | 10 |
| 600ms | 1.667 |
| 1s | 1 |

Table 4 – Temps d'exposition choisis et pondération: module hq

| Temps d'exposition | Facteurs appliqués au pixels |
|---|---|
| 10us | 40000 |
| 100us | 4000 |
| 10ms | 40 |
| 100ms | 4 |
| 400ms | 1 |

Table 5 – Temps d'exposition choisis et pondération: module v1

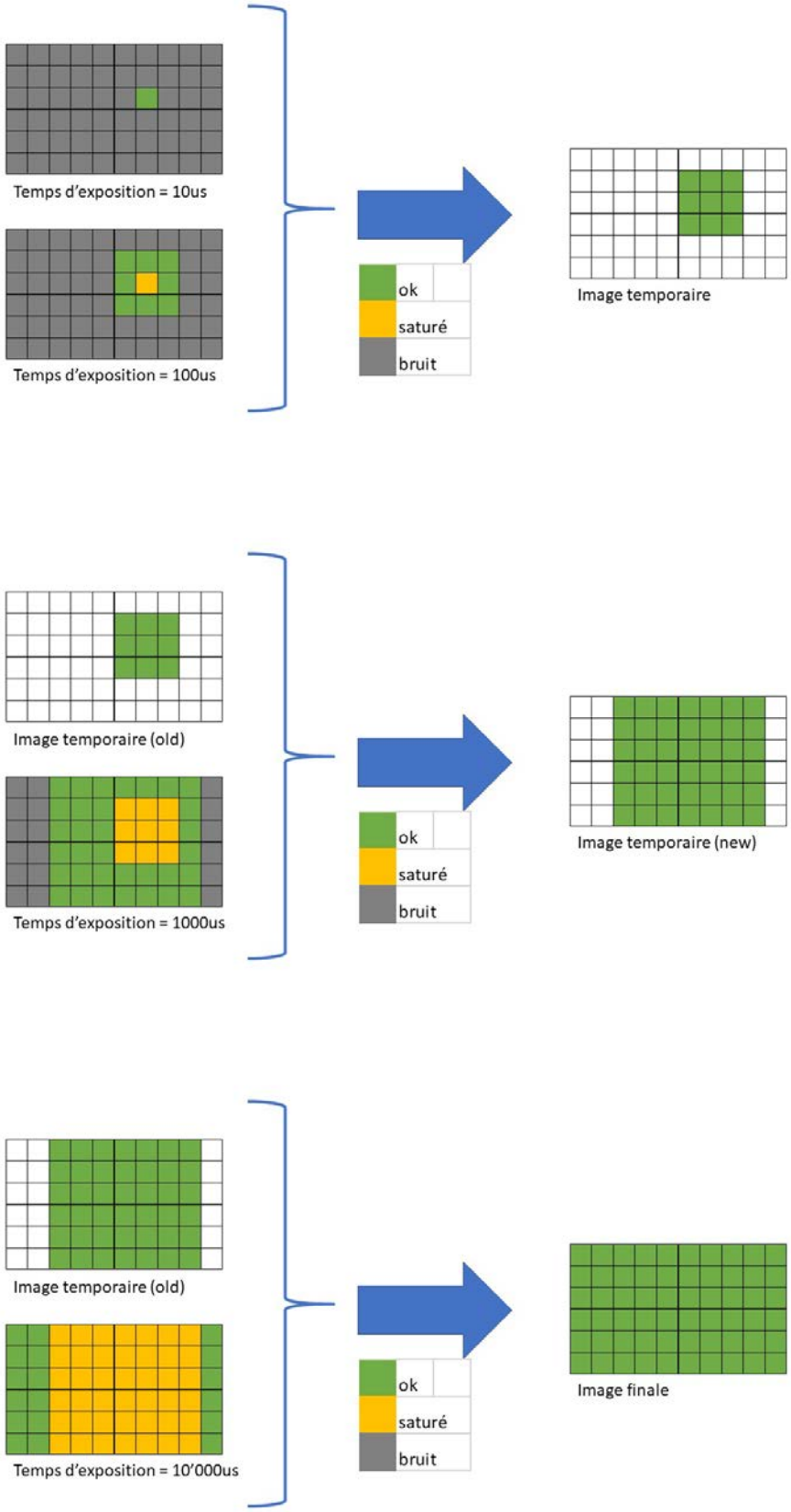
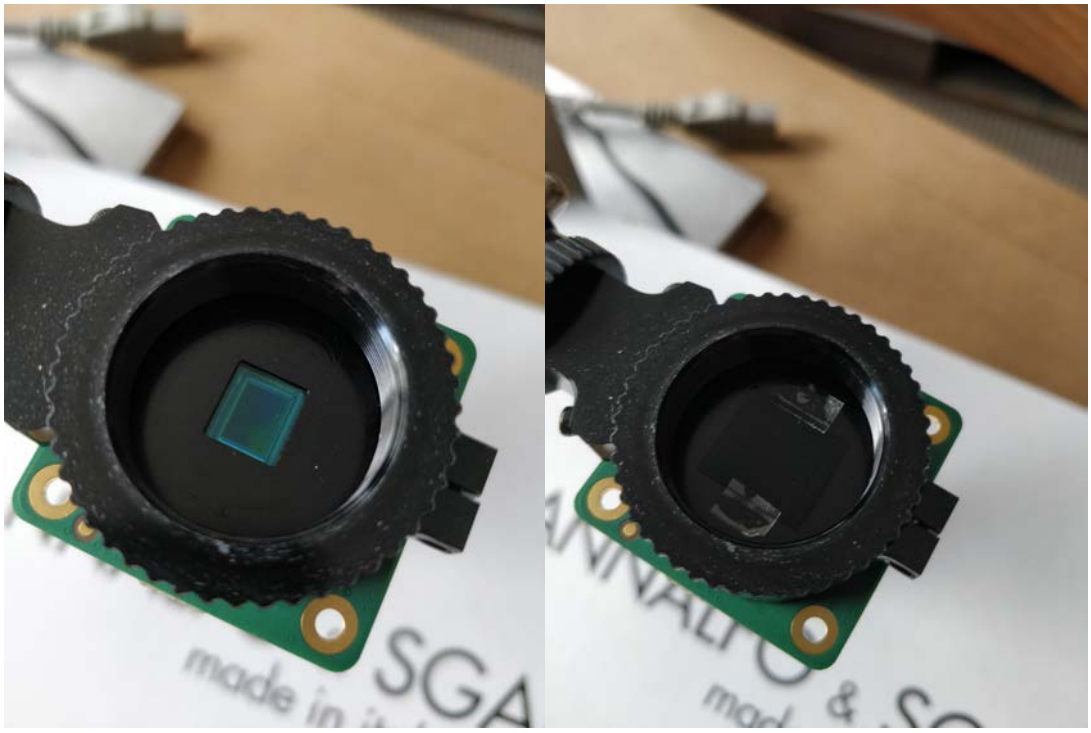



Figure 24 – Reconstitution d'une image

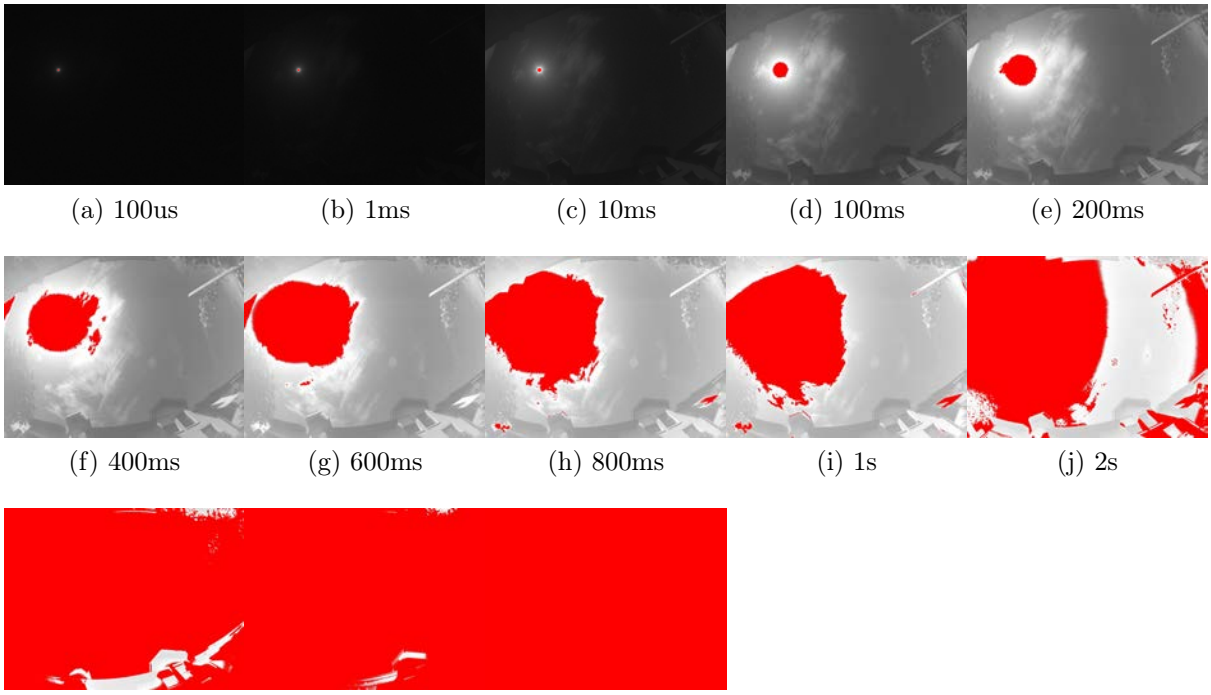
# 3 Test du module HQ



(a) sans filtre (b) filtre 3.00

Figure 25 – Filtre 3.00 placé devant le capteur du module HQ

## 3.1 Essais avec différentes ouvertures sur le fisheye



(a) 100us (b) 1ms (c) 10ms (d) 100ms (e) 200ms

(f) 400ms (g) 600ms (h) 800ms (i) 1s (j) 2s

(k) 3s (l) 4s (m) 5s

Figure 26 – Série de captures avec temps d'exposition croissants, les pixels rouges représentent les pixels saturés: module hq ouverture f1.8

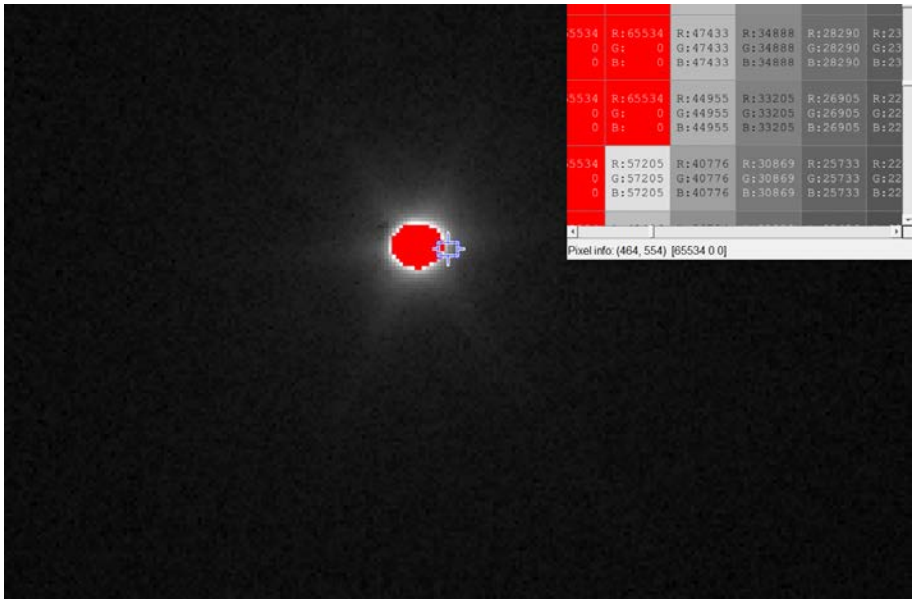
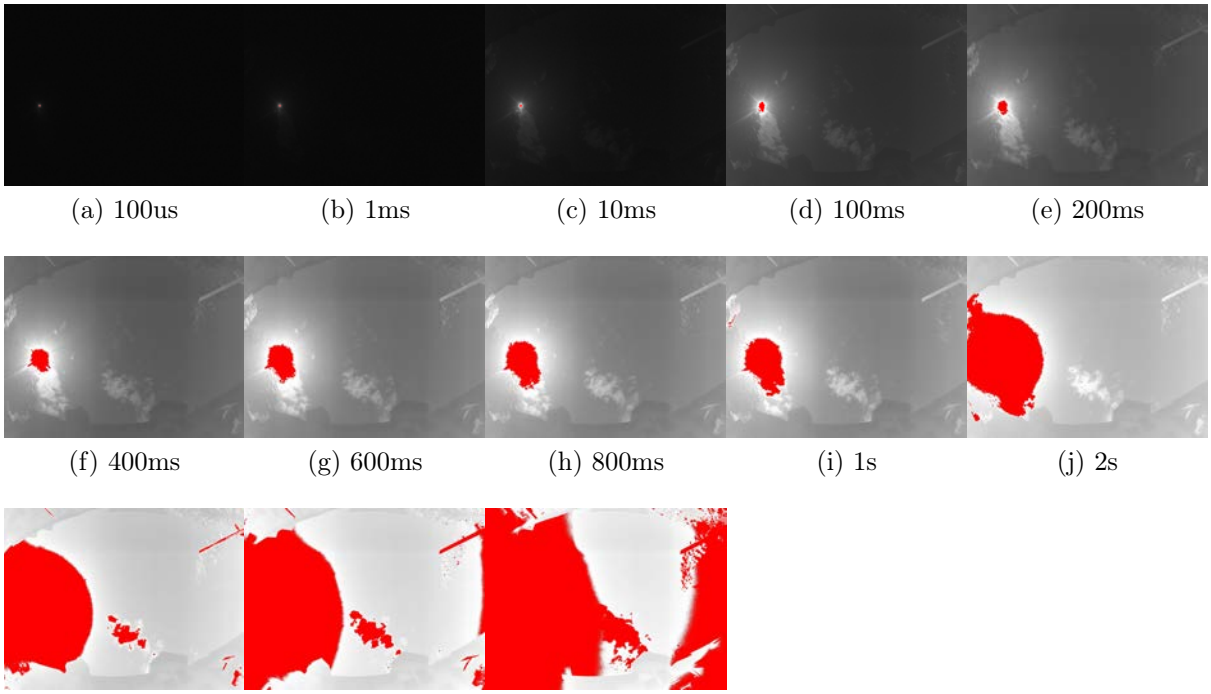



Figure 27 – Zoom sur soleil :durée d'exposition 100us, module hq ouverture f1.8



(a) 100us (b) 1ms (c) 10ms (d) 100ms (e) 200ms

(f) 400ms (g) 600ms (h) 800ms (i) 1s (j) 2s

(k) 3s (l) 4s (m) 5s

Figure 28 – Série de captures avec temps d'exposition croissants, les pixels rouges représentent les pixels saturés: module hq ouverture f4

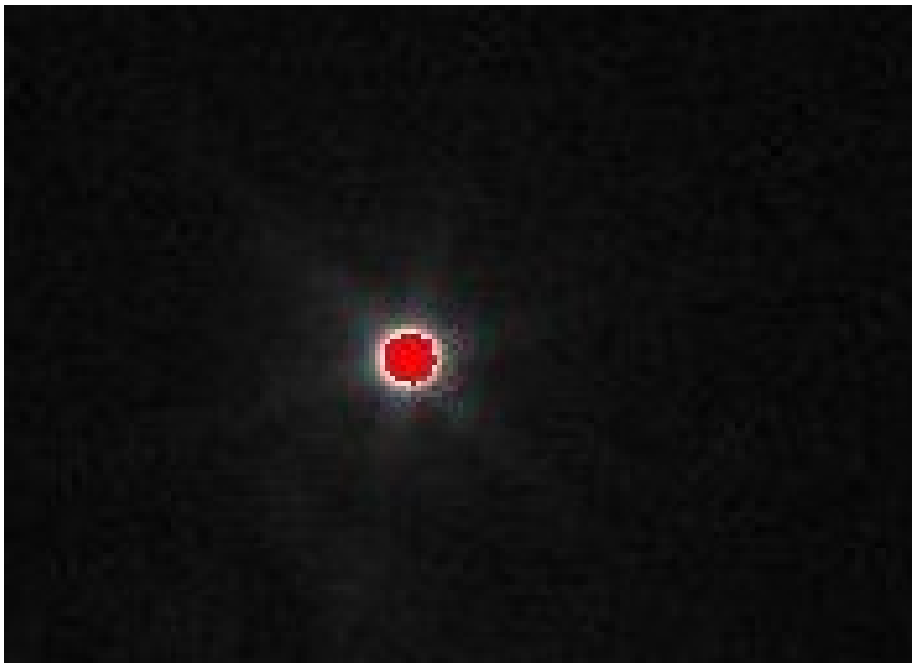
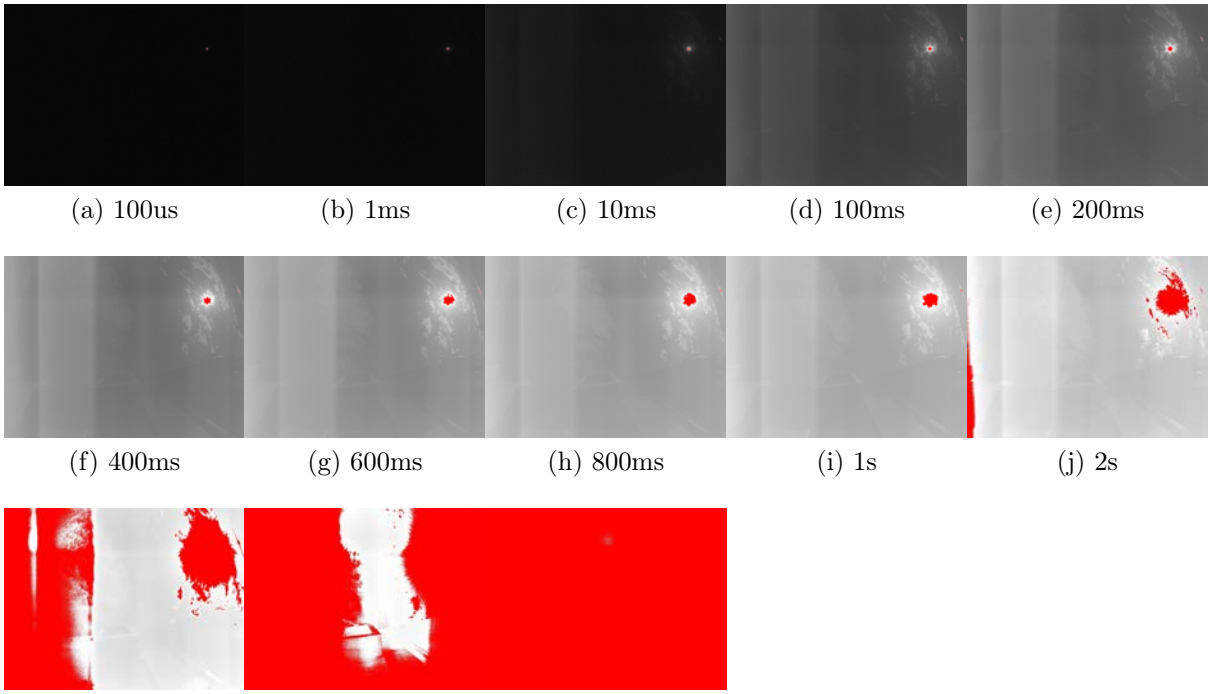


Figure 29 – Zoom sur soleil :durée d'exposition 100us, module hq ouverture f4 , heure 15:55



(a) 100us (b) 1ms (c) 10ms (d) 100ms (e) 200ms

(f) 400ms (g) 600ms (h) 800ms (i) 1s (j) 2s

(k) 3s (l) 4s (m) 5s

Figure 30 – Série de captures avec temps d'exposition croissants, les pixels rouges représentent les pixels saturés: module hq ouverture f4

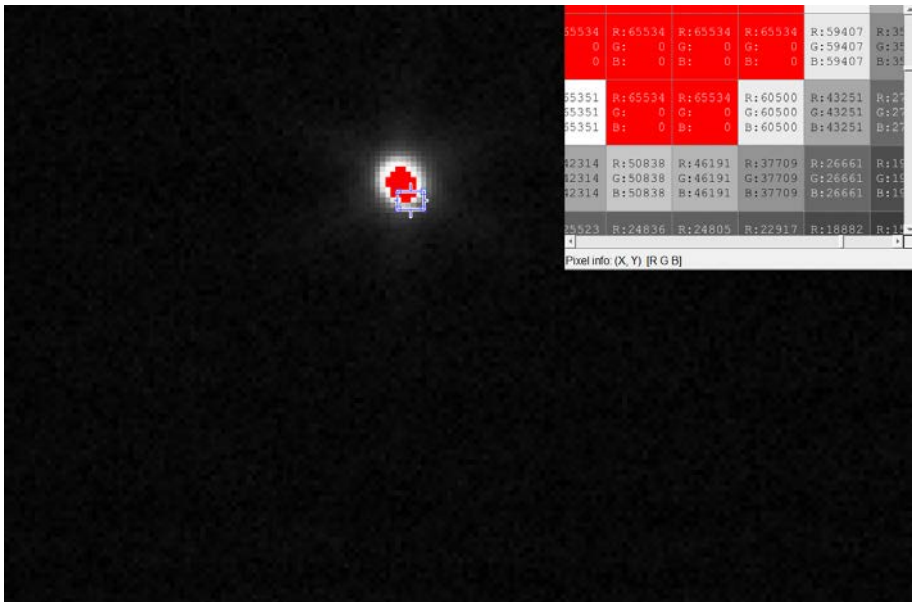
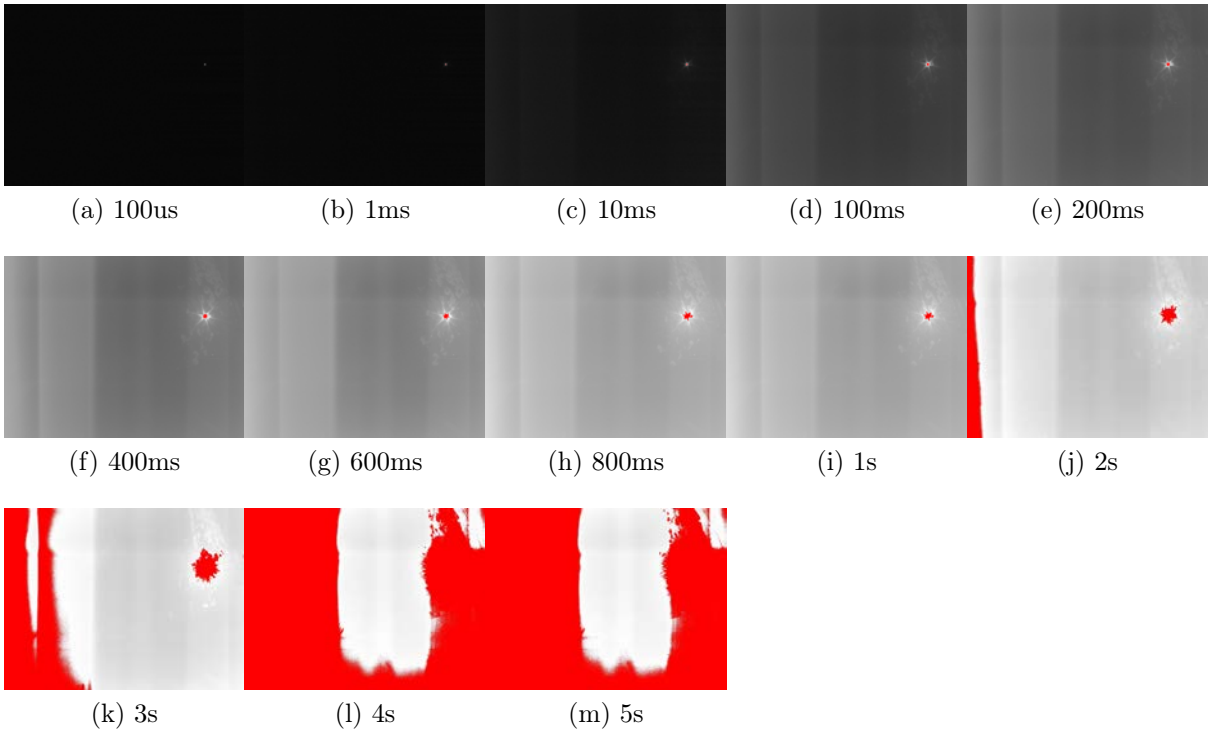



Figure 31 – Zoom sur soleil :durée d'exposition 100us, module hq ouverture f8



(a) 100us (b) 1ms (c) 10ms (d) 100ms (e) 200ms

(f) 400ms (g) 600ms (h) 800ms (i) 1s (j) 2s

(k) 3s (l) 4s (m) 5s

Figure 32 – Série de captures avec temps d'exposition croissants, les pixels rouges représentent les pixels saturés: module hq ouverture f16

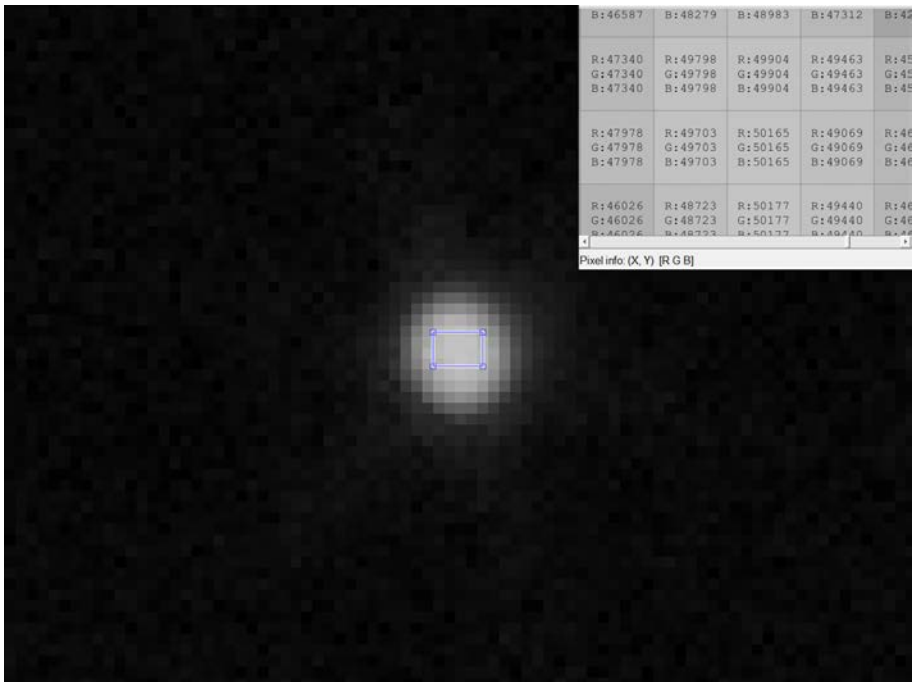



Figure 33 – Zoom sur soleil: durée d'exposition 100us, module hq ouverture f16

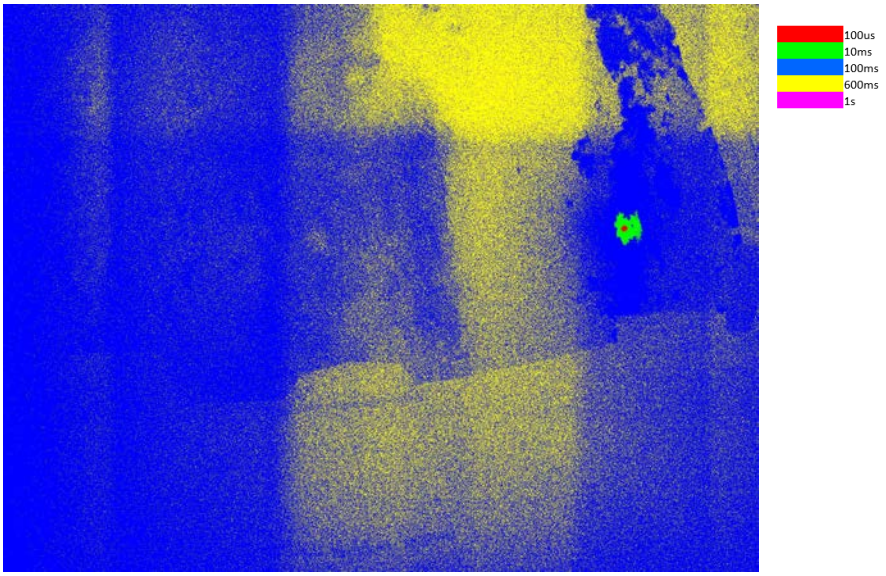
Le script Matlab 'color_saturated_pixels_hq.m' (cf. annexe 6) réalisé me permet de colorer en rouge les pixels saturé sur une image.
D'abord, une série de captures avec des temps d'exposition différents est réalisée à l'aide du module HQ avec l'ouverture du fisheye réglée en f1.8 (cf. figure 26). On constate que même avec le temps d'exposition le plus faible la zone du disque solaire contient des pixels saturés (cf. figure: 27). Puis l'ouverture est diminuée jusqu'à obtenir une image avec un soleil non voilé qui ne sature pas le capteur.
La capture réalisée avec un temps d 'exposition de 100us et l'ouverture réglée au minimum en f16 (cf. figure 33), ne contient plus de pixels saturés.
Comme on peut le constater sur les séries de captures en ouverture f16 (cf. figure 32), avec le filtre 3.00 utilisé les images capturée en dessus de 2s ne possèdent plus d'informations exploitables.

## 3.2 Fusion

(a) 100us (b) 10ms (c) 100ms

(d) 600ms (e) 1s

Figure 34 – Série 01 de captures avec temps d'exposition différents, ouverture f16, filtre 3.00 : module hq




(a) (b)

Figure 35 – Zones d'exposition différentes: série 01

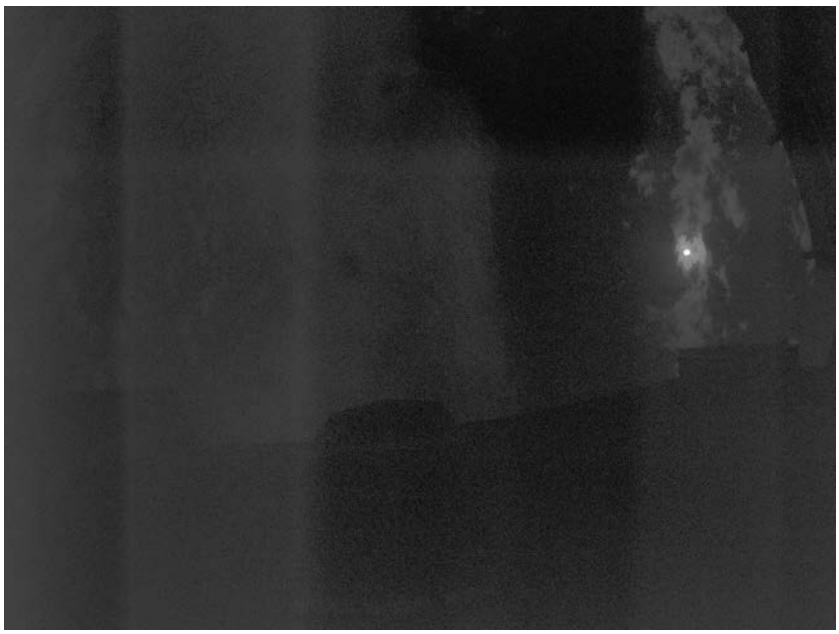
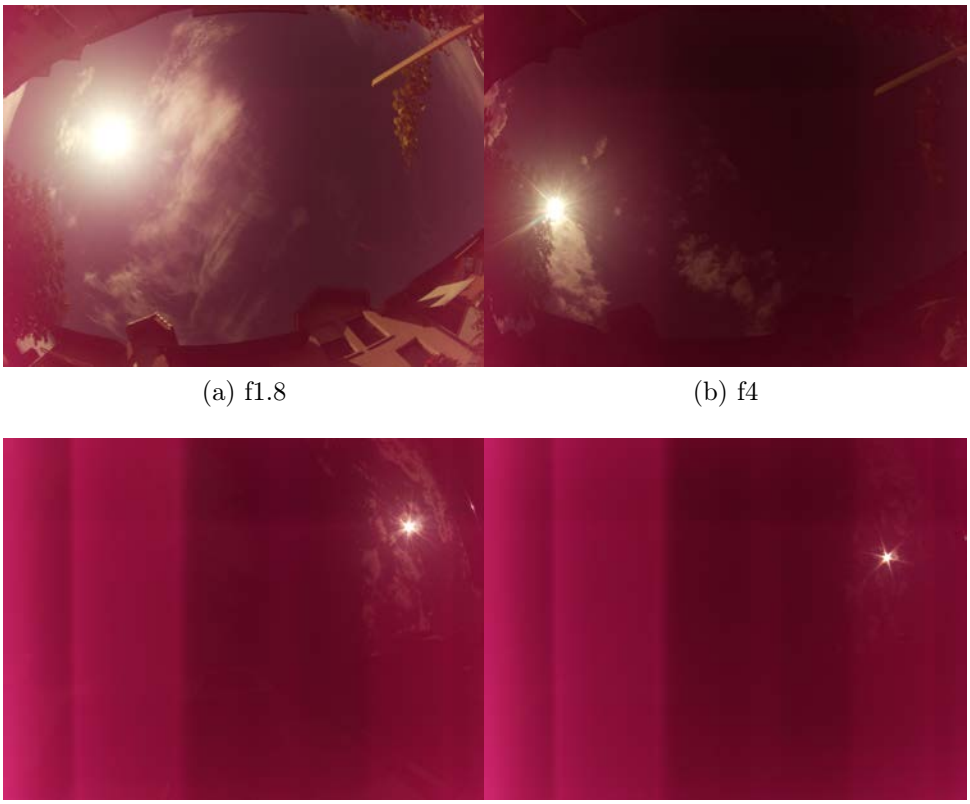


Figure 36 – Images HDR fusionnées, convertie dans une plage dynamique visible [14] , module hq

## 3.3 Discussion des résultats module caméra hq

Après essais, il s'avère que la lumière reçue par le capteur est altérée dans la zone spectrale rouge. Par conséquent, l'image résultante apparaît rose. Le filtre ND est d'abord soupçonné dû a son influence spectrale (cf. figure 22)
Cependant il ne devrait pas être la cause étant donné que les captures avec le même filtre sur le module v1 ne fournit pas ces images roses. Je suppose donc que cela provienne du filtre IR ou du revêtement de protection du capteur.
Un autre problème lié cette fois au filtre ND survient lorsque l'ouverture est faible. Des bandes apparaissent: on les distingue clairement sur la figure 37 Je suppose qu'elles sont causées par le filtre ND 3.00. Je ne sais pas si c'est possible d'améliorer cela.
C'est un problème à résoudre car on peut confondre les nuages et ces bandes lors de l'analyse de la matrice HDR (cf. figure: 36)
Remarque: L'image 36 est convertie dans une plage dynamique de manière à la visualiser mais ce n'est pas la véritable matrice. L'algorithme utilisé est indiqué ici: [14].



(a) f1.8 (b) f4

(c) f8 (d) f16

Figure 37 – Série de captures avec temps d'exposition de 100ms, différentes ouvertures : module hq

Plusieurs sessions de captures ont été réalisée, c'est à dire une série de 5 images consécutives (100us,10ms,100ms,600ms et 1s) comme précédemment. Pour chaque sessions, une matrice 32 bits est créée par l'algorithme de fusion (cf. section 2.7). J'ai réalisé la somme des valeurs contenues dans cette matrice afin de la comparer à l'irradiation totale mesurée par le pyranomètre. La valeur mesurée par le pyranomètre est moyennée sur les 5 mesures.

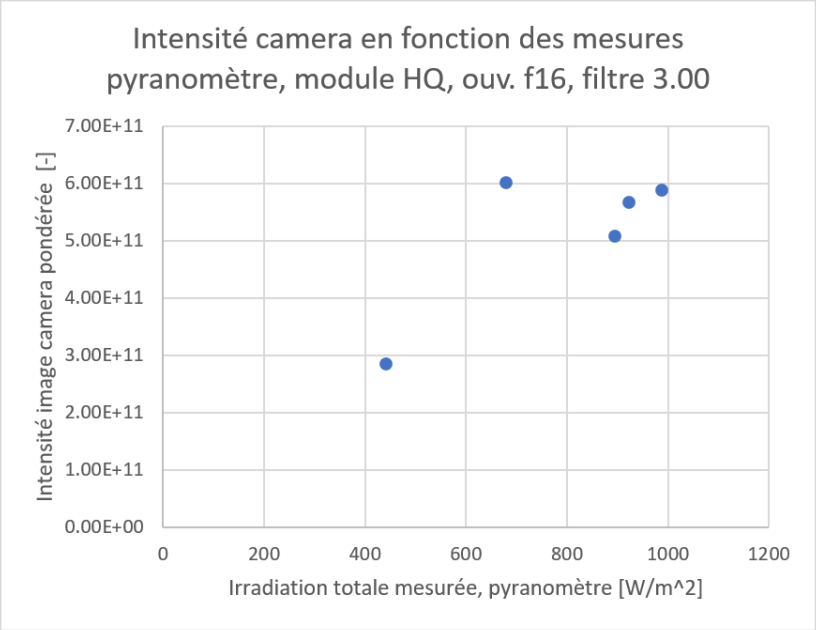
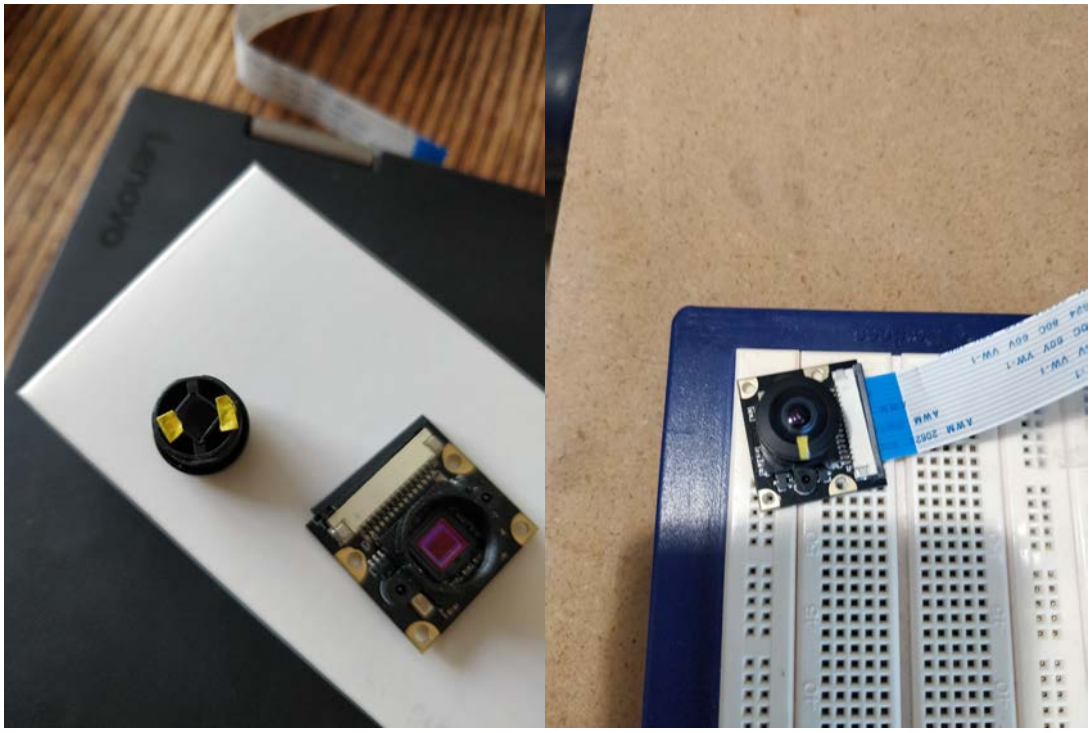



Figure 38 – module hq, ouverture f16, filtre 3.00

Ils faudraient plus de mesures afin de tirer des conclusions. Mais les problèmes de bandes précédemment cités influences forcément les mesures. Étant donné que ces bandes n'apparaissent pas lorsque l'ouverture est plus grande, Il serait intéressant de tester un filtre qui atténue plus fortement la lumière afin de pouvoir augmenter l'ouverture de l'objectif.

# 4 Test du module v1



(a) démontage objectif (b) module v1

Figure 39 – Filtre 3.00 placé devant le capteur du module V1

Pour les captures réaliser avec le module v1, je n'avais pas d'algorithme à ma disposition afin d'extraire les données bruts du capteur. Il faudrait s'attarder sur l'article [11] afin de reproduire l'algorithme pour le module V1.
J'ai donc travaillé avec des images compressées en 8 bit fournit par la commande 'Raspistill'. J'ai aussi utilisé le flag '-drc = high' ajouter à la commande 'raspistill' afin d'utiliser la compression de la plage dynamique que réalise l'algorithme de capture (cf. section: 2.3). L'outil de compression de plage dynamique est normalement utilisé dans l'objectif de réduire les hautes lumières et les ombres. Comme les images sont compressées, j'ai fait l'hypothèse que cette compression serait bénéfique.

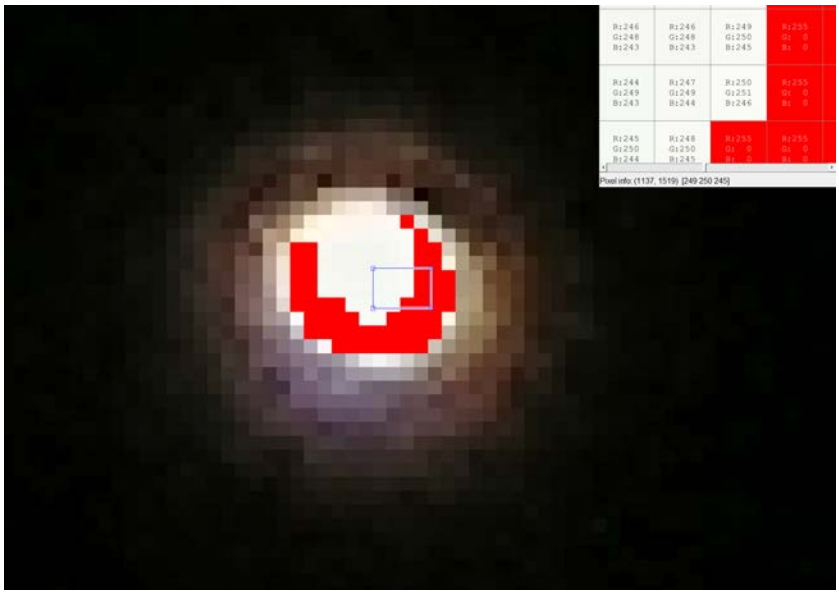
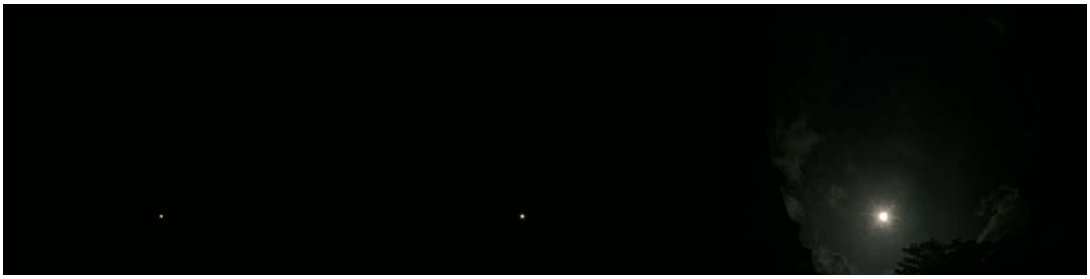


Figure 40 – Zoom sur soleil, durée d'exposition 10us, module v1

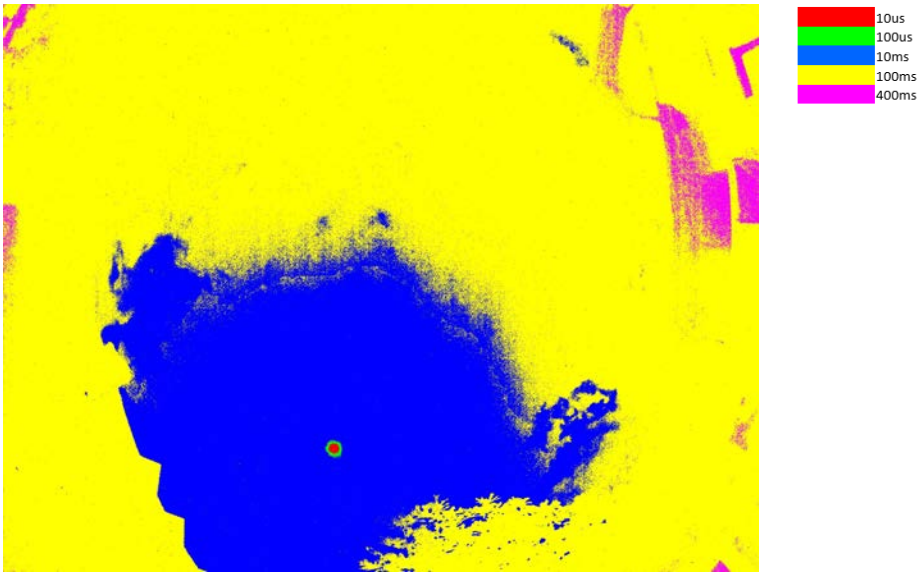
On peut constater sur la figure 40, malgré un temps d'exposition de 10us, des pixels saturés demeure au niveau du disque solaire. Ils sont disposés sur les bords du disque solaire. Je suppose que cela est causé par la compression de la plage dynamique car sur les images capturées avec le module HQ, les pixels saturés se trouvaient au centre du soleil.
Une fusion des images est quand même réalisée malgré le fait qu'il reste de la saturation sur l'image avec le plus faible des temps d'exposition. Le script de fusion des images utilisé pour le module est 'fusion_v1_ponderation.m' (cf. annexe: 6)



(a) 10us (b) 100us (c) 10ms

(d) 100ms (e) 400ms

Figure 41 – Série 01 de captures avec temps d'exposition différents, filtre 3.00 : module v1




(a) (b)

Figure 42 – Zones d'exposition différentes: série 01, module v1

Figure 43 – Image HDR fusionnée, convertie dans une plage dynamique visible[14], module v1

Comme pour le module hq, plusieurs sessions de captures ont été réalisée, c'est à dire une série de 5 images consécutives (10us,100us,10ms,100ms et 400ms). Pour chaque sessions, une matrice est créée par l'algorithme de fusion (cf. section 2.7). J'ai réalisé la somme des valeurs contenues dans cette matrice afin de la comparer à l'irradiation totale mesurée par le pyranomètre. La valeur mesurée par le pyranomètre est moyennée sur les 5 mesures.

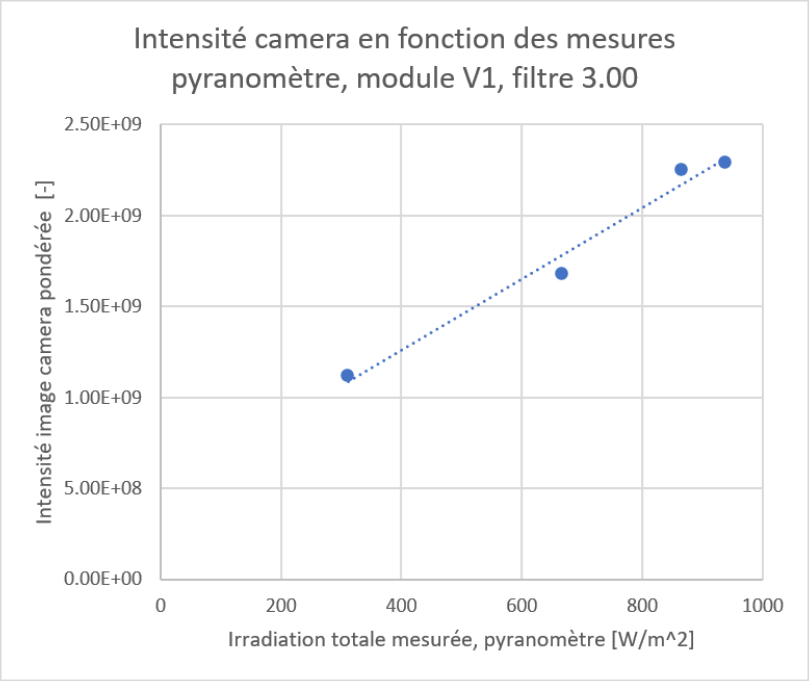



Figure 44 – module v1, filtre 3.00

# 5 Discussion de résultats

Il sera effectivement problématique de réaliser les captures avec une seule caméra. Les nuages se déplacent rapidement et quelques secondes suffisent à varier fortement l'irradiation solaire.
Sur la figure 45, les captures ont été réalisées de manière successive. Les nuages ont le temps de cacher le soleil entre les captures. Alors qu'elles devrait représenté la même image afin de recréer une matrice HDR.

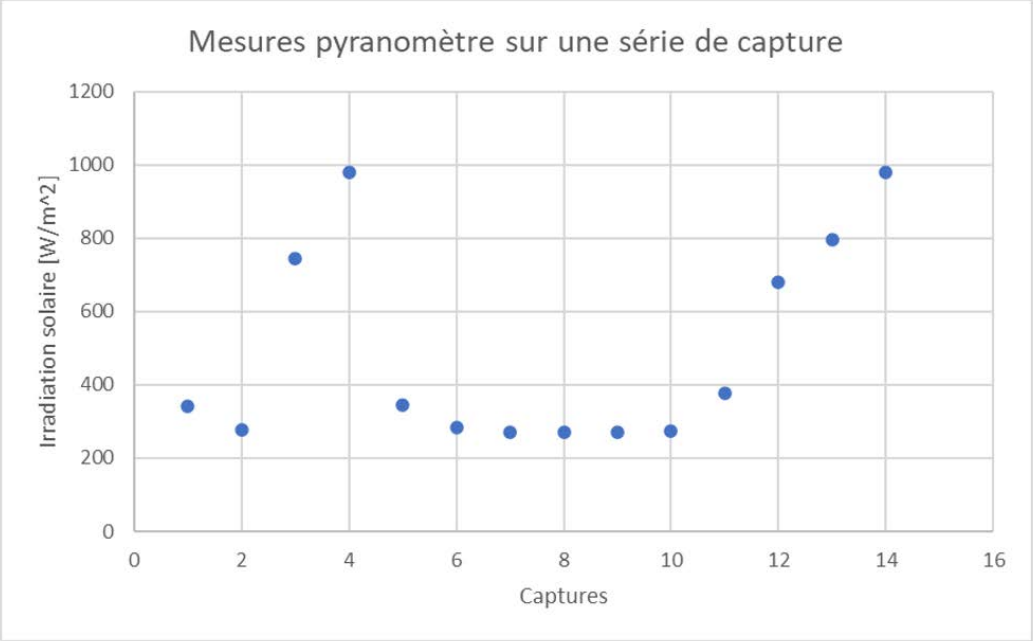



Figure 45 – Mesures du pyranomètre pour une série de captures

Des essais supplémentaires doivent être réalisée pour déterminer une solution efficace. Idéalement il faudrait placer la caméra sur un toit de manière à ne voir que le ciel et également afin d'avoir le même ciel pour toutes les mesures.
Le module caméra HQ avec le filtre ND 3.00 réalise des captures 'rose'. Cependant le même filtre placé devant le module V1 réalise des captures correct au niveaux des couleurs. Je ne pense pas que les images fournit par la caméra HQ soit exploitable à cause des bandes qui vont faire apparaître le ciel plus lumineux qui ne le devrait.
Le graphe 44 semble correct pour les séries de captures utilisées. Mais à nouveau, il faudrait plus de situations du ciel afin de confirmer cela.
La pondération utilisée pour la matrice HDR est possiblement incorrect. J'ai utilisé ces facteurs afin de mettre plus de poids aux pixels provenant de captures à faible temps d'exposition.
Idéalement il faut construire une base de donnée d'images de manière à avoir le plus de situation possible (ciel bleu, nuageux, ...). Ensuite alors une meilleure calibration pourra être effectuée.
La matrice HDR devrait être en mesure de définir des zones du ciel simplement avec l'intensité de ces pixels.

| | |
|---|---|
| Ciel bleu | 0...10% |
| Nuages faiblement opaque | 10...40% |
| Nuages fortement opaque | 40..90% |
| Soleil | 90...100% |

Table 6 – Proposition de détection de zones dans le ciel

La séparation de l'irradiation solaire peut être réalisée à partir des différentes zones définie dans le tableau 6. L'image peut contenir une zone soleil, une ou plusieurs zones nuages et une ou plusieurs zones ciel bleu. En fonction des différents zones, la zone occupée par le soleil représente le rayonnement direct provenant du soleil. Celle occupée par les nuages représente le rayonnement diffusé.
L'algorithme de fusion des images pour créer une image bénéficiant d'une plage dynamique augmentée utilisé dans ce projet n'est peut-être pas la meilleure solution. J'ai découvert des méthodes plus avancées en lisant des publications sur le sujet. Il serait judicieux de comparer les méthodes est ensuite déterminer si elles s'avèrent plus efficace pour la détection des nuages. [15], [16],[17] L'article 'High Dynamic Range Imaging for Cloud Segmentation' fournit même le code utilisé dans le projet sur github. https://github.com/Soumyabrata/HDR-cloud-segmentation

# 6 Conclusion et proposition pour la suite du projet

Les différents essais réalisé ne sont pas suffisant pour déterminer une solution pour les mesures d'intensité du ciel.
La méthode est correcte par contre le matériel utilisé ne suffit pas à répondre au besoin du projet. Le filtre 3.00 utilisé n'absorbe pas assez la lumière pour éliminer totalement la saturation de la caméra au niveau du disque solaire. Une image ne contenant pas de saturation a pu être réalisé en diminuant mécaniquement l'ouverture de l'objectif sur la caméra HQ. Cependant le fait de diminuer l'ouverture amène un problème de 'bandes lumineuses' sur l'image et par conséquent la rend inexploitable.
Les captures réalisées à l'aide de la caméra v1, possèdent des pixels saturés. L'image HDR reconstituée par les séries de captures réalisées avec ce module ne sont pas si mauvaises. Il est peut-être possible de quand même exploiter ces résultats dans le but de détecter les nuages.
La prédiction est toutefois possible malgré le fait que le disque solaire sature un peu l'image. Ce projet [6] réalisé à l'aide d'un module caméra v2 pour raspberry en témoigne.
Concernant la prédiction de production photo-voltaïques, il faut être en mesure de détecter si le soleil est présent ou non sur l'image et être capable de trouver sa position s'il y est. Idem avec les nuages et en plus il faut pouvoir mesurer la vitesse et la direction de ces derniers. Une publication propose d'utiliser des méthodes de détection de flux optique (Optical flow) pour le mouvement des nuages. [18]
Il faudra penser également corriger la distorsion causé par l'objectif fish-eye. Une méthode pour réaliser cela est disponible en open-source içi: [19].
En ce qui concerne le prix du matériel utilisé, il faut considérer que l'objectif fisheye utilisé doit être remplacé par un objectif meilleur marché. La définition de l'objectif à utiliser définira certainement la caméra qui va avec.

Sion, August 27, 2020

Mario Brunetti

# Annexes

# Annexe 1

```
function s = read_PIHQraw_from_jpg(file)
    %s = loadRaspistill; Loads via GUI and extracts uint16 raw data from
    %Raspberry Pi HQ camera jpg file captured by raspistill -md 3 -r, full
    %resolution 12-bit switches. Output struct s contains a 'BGGR' (cfa)
    %image, a rough/half (rgb) sRGB image and basic exif information.
    %Matrix (M) takes black subtracted, white balanced raw data to linear sRGB.
    %s = loadRaspistill(1) also displays the sRGB image.
    %Apply brightness/contrast to s.rgb image to taste. For details see:
    %www.strollswithmydog.com/raw-file-conversion-steps/
    %By Jack Hogan -  Version 0.17  -  May 12, 2020

    [filepath,name,ext] = fileparts(file);
    s.fileName = strcat(name,ext);
    s.filePath = strcat(filepath,'\');
    getExif;                                        %get auxiliary data

    f = fopen(fullfile(s.filePath,s.fileName),'r'); %load the jpeg file
    bin= fread(f,'*uint8');
    fclose(f);

    roi = strfind(bin','BRCM');                     %find start of raw data
    if isempty(roi)
        disp('No raw data was found in this Jpeg file')
        return
    end

    c = 4056;                                       %active image width
    r = 3040;                                       %active image height

    bin = bin(roi+2^15:end);                        %ignore header
    bin = reshape(bin,[],r+16);                     %transposed sensor array
    bin = uint16(bin(1:c*12/8,1:r));                %select packed area

    s.cfa = zeros(c*r,1,'uint16');
    s.cfa(1:2:end) = bitshift(bin(1:3:end),4) + bitand(bin(3:3:end),15);
    s.cfa(2:2:end) = bitshift(bin(2:3:end),4) + bitshift(bin(3:3:end),-4);
    s.cfa = reshape(s.cfa,[],r)';                   %12 bit raw in uint16

    %% ------------------------------------------------
    %sRGB half render (similar to dcraw -h), or use built-in demosaic func
    B = (single(s.cfa(1:2:end,1:2:end)) - s.black) / s.AsShotNeutral(3);
    G = single((s.cfa(2:2:end,1:2:end)+s.cfa(1:2:end,2:2:end)))/2 - s.black;
    R = (single(s.cfa(2:2:end,2:2:end)) - s.black) / s.AsShotNeutral(1);
    s.rgb = cat(3,R,G,B) / (2^12-1-s.black);        %normalize to 1

    s.rgb(s.rgb>1) =1;                              %clip
```

```
    s.rgb = reshape(reshape(s.rgb,[],3)*s.M',size(s.rgb)); %project to sRGB
    s.rgb = uint16(real(s.rgb.^0.45) * (2^16-1));   %approximate gamma

    if nargin
        warning('off','images:initSize:adjustingMag')
        %figure; imshow(s.rgb)
    end

    function getExif
        i = imfinfo(fullfile(s.filePath,s.fileName));
        s.FileModDate = i.FileModDate;
        s.cam = i.Model;
        s.bayer = 'BGGR';
        s.iso = single(i.DigitalCamera.ISOSpeedRatings);
        s.ss = single(i.DigitalCamera.ExposureTime);    %ss from exif
        s.black = single(256.3);                        %my module
        s.white = 2^12-1-s.black;

        m = char(i.DigitalCamera.MakerNote);
        disp(['File ' s.fileName ', Camera ' s.cam ', Maker Notes: ' m])
        m = split(m,["="," "]);
        s.exp = str2double(m(find(strcmp(m,'exp'))+1))/10^6;  %reported ss
        s.ag = str2double(m(find(strcmp(m,'ag'))+1))/256;     %analog gain
        rg = single(str2double(m(find(strcmp(m,'gain_r'))+1)));
        bg = single(str2double(m(find(strcmp(m,'gain_b'))+1)));
        s.AsShotNeutral = 1 ./ [rg 1 bg];
        s.M = str2double(split(m(find(strcmp(m,'ccm'))+1),","));
        s.M = reshape(s.M(1:9),3,3)'/10000;             %matrix wbraw->sRGB
        s.M = single(s.M / sum(s.M(2,:)));              %Norm sum y row 1
        s.ActiveArea = uint16([0,0,i.Height,i.Width]);
        % ...
    end
end
```

# Annexe 2

```
function s = color_saturated_pixels(plotFlag)
    %Loads via GUI and color saturated pixels in red
    [s.fileName,s.filePath] = uigetfile({'*.jpg'},...
            'Select Raspberry Pi Camera jpg file');
    saturation_threshold = 250;
    img = imread([s.filePath s.fileName]);
    [rows, columns, numberOfColorChannels] = size(img);
    %Creation d'un masque a zero
    mask = zeros(rows,columns);
    %Conversion vers niv de gris
    img_gray = rgb2gray(img);
    %mise a 1 des pixels ok dans le masque
    mask = img_gray <= saturation_threshold;
    %mise a zero des pixel saturé
    new_img = img.*uint8(mask);
    %creéation d'une image rouge de la bonne taille
    img_red = zeros(rows,columns,3);
    img_red(:,:,1)=255;
    %mise a zeros des pixels qui ne sont pas saturé
    inv_mask = ~mask;
    inv_mask = 1-inv_mask;
    inv_mask = (inv_mask == 0);
    img_red = uint8(img_red).*uint8(inv_mask);
    %combinaison de l'image rouge et de l'image de base
    new_img = uint8(new_img) + uint8(img_red);
    imwrite(new_img,'img_red.jpg');
    imtool(new_img);
end
```

# Annexe 3

```
 %Loads via GUI and color saturated pixels in red
[s.fileName,s.filePath] = uigetfile({'*.jpg'},...
            'Select Raspberry Pi Camera jpg file');
saturation_threshold = 65530;
s = read_PIHQraw_from_jpg([s.filePath s.fileName]);
[rows, columns, numberOfColorChannels] = size(s.rgb);
% Initilisation d'un masque a zero
mask = zeros(rows,columns);
%Conversion de l'image vers img en niveau de gris
img_gray = rgb2gray(s.rgb);
%mise a 1 des pixels ok dans le masque
mask = img_gray <= saturation_threshold;
%mise a zero des pixel saturé
new_img = img_gray .*uint16(mask);
%création d'une image rouge de la bonne taille
img_red = zeros(rows,columns,3);
img_red(:,:,1)=65534;
% permutation des 1 et 0
inv_mask = ~mask;
inv_mask = 1-inv_mask;
inv_mask = (inv_mask == 0);
%mise a zeros des pixels qui ne sont pas saturé dans l'image rouge
img_red = img_red.*inv_mask;
%combinaison de l'image rouge et de l'image de base
color = uint16(new_img) + uint16(img_red);
imwrite(uint8(color/256),'saturation\img_red_01.jpg');
%imshow(new_img)
%imtool(color)
%imshow(img_red)
```

# Annexe 4

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fusion de plusieurs images captures realisées avec la camera v1 avec des
% temps d'exposition differents.
% 0 - 255
% rows 1944,columns 2592
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filename    : fusion_hq_ponderation.m
% Context     : HES-SO-VS bachelor work
% Authors     : Mario Brunetti
% Version     : 1.0.0
% Last modif. : 23.08.2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input variables
%   myFolder : name of the folder containing the 5 images
% Output variables
%
% Input files
%   the file the name of which is passed as input parameter
% Output files
%   none
% Loaded variables
%   none
% Saved variables
%   none
% Global variables
%   none
% Used figures
%   none
% Tested Matlab versions
%   9.6.0.1174912 (R2019b) Update 1
% Needed toolboxes and tested versions
%   Image Processing Toolbox for imfinfo
% Called specific M-files (i.e. not in any toolbox)
%   none
% Remarks
%   Fusion de 5 images captures realisées avec la camera v1 avec des
%   temps d'exposition diferents. Les 5 captures à analysée doivent se
%   trouver dans le dossier myFolder dont le lien est à spécifier.
%   Elle sont egalement nommées de manières a avoir le temps d'exp du plus
%   faible au plus elevé.
%   Les valeurs de seuil pour la saturation et pour le niveau de bruit sont
%   à specifier dans les variables saturation_threshold et noise_threshold.
%   la resolution de l'image est de rows 1944, columns 2592
%   et codée sur 8 bits (0 - 255).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear ,clear,clc
```

```
%% Parcourir le dossier
saturation_threshold = 254;  %Set the threshold value for saturated pixels
noise_threshold = 10;        %Set the threshold value for noised pixels
%%
myFolder = 'path_to_folder';
if ~isfolder(myFolder)
  errorMessage = sprintf('Error: The following folder does not exist:\n%s',
  myFolder);
  uiwait(warndlg(errorMessage));
  return;
end
filePattern = fullfile(myFolder, '*.jpg');
jpegFiles = dir(filePattern);

%% check la taille des images
file = [myFolder '\' jpegFiles(1).name];
%Extraction des données bruts 12bits du fichier jpg vers images rgb 16bits
s = imread(file);
rows = size(s,1);
columns = size(s,2);


%% Initialisation du masque
mask = zeros(rows,columns);
img = zeros(rows,columns);
mask_rgb = zeros(rows,columns,3);
map= zeros(rows,columns,3);
map_all= zeros(rows,columns,3);
ponderation_matrix = zeros(rows,columns);
ponderation = [40000 4000 40 4 1];

%% Définition des couleurs pour le plot des zones
img_red = zeros(rows,columns,3);
img_green = zeros(rows,columns,3);
img_blue = zeros(rows,columns,3);
img_yellow = zeros(rows,columns,3);
img_pink = zeros(rows,columns,3);
img_red(:,:,1)=255;
img_green(:,:,2)=255;
img_blue(:,:,3)=255;
img_yellow(:,:,1)=255;
img_yellow(:,:,2)=255;
img_pink(:,:,1)=255;
img_pink(:,:,3)=255;
img_colored(:,:,:,1)= img_red;
img_colored(:,:,:,2)= img_green;
img_colored(:,:,:,3)= img_blue;
img_colored(:,:,:,4)= img_yellow;
```

```
img_colored(:,:,:,5)= img_pink;
%% Boucle
t_exp = [];
intensit_cam = [];
for k = 1:length(jpegFiles)
  file = [myFolder '\' jpegFiles(k).name];
  i = imfinfo(fullfile(myFolder,jpegFiles(k).name));
  m = char(i.DigitalCamera.MakerNote);

  exp = str2double(m(find(strcmp(m,'exp'))+1))/10^6;  %reported ss
  %Extraction des données bruts 12bits du fichier jpg vers images rgb 16bits
  s = imread(file);

  %Conversion vers image en niveaux de gris
  img_gray = rgb2gray(s);

  % Sauvegarde le temps d'exposition dans un vecteur et la somme des rgb
  %t_exp = [t_exp,exp*1000000];%us
  intensit_cam = [intensit_cam,sum(s(:))];

  %Permutation du masque precedant 0 à la place de 1 et inversemment
  old_mask = mask;
  inv_old_mask = ~old_mask;
  inv_old_mask = 1-inv_old_mask;
  inv_old_mask = (inv_old_mask == 0);

  %Création d'un nouveau masque binaire à partir de l'image
  %avec 0 = saturé ou bruit, 1 = pixel ok
  mask = img_gray <= saturation_threshold & img_gray >= noise_threshold;
  %Mise a zeros des pixels deja recupéré de l'image precedante
  %Les pixels déjà recupérer dans la precedante image sont a 1 dans mask
  % donc sont a 0 dans inv_old_mask
  mask_tmp = inv_old_mask.*mask;
  old_ponderation_matrix = ponderation_matrix;
  ponderation_matrix = mask_tmp .* ponderation(k);
  ponderation_matrix = uint8(ponderation_matrix) +uint8(old_ponderation_matrix);
  %%
  old_map = map_all;
  map = mask .* img_colored(:,:,:,k);
  % mise a zero des pixel deja ok
  map_tmp_sans_old = map.*inv_old_mask;
  map_all = old_map + map_tmp_sans_old;
  %% création d'une nouvelle image avec les valeurs des pixels ok
  %mask_tmp 1 = nouveaux element a ajouter
  img_old = img;

  %img_tmp = la nouvelle image avec les nouvelles valeurs
  img_tmp = uint8(img_gray).*uint8(mask_tmp);
```

```
    img = uint8(img_old) + uint8(img_tmp);
    mask = mask_tmp + old_mask;
  end
    %figure,imshow(img)
    imwrite(ponderation_matrix,'results/ponderation_v1.jpg');
    imwrite(img,'results/fusion_v1.jpg');
    %figure,imshow(map_all)
    imwrite(map_all,'results/map_v1.jpg','BitDepth',8);
    %imtool(map_all);
    img_pond = uint16(ponderation_matrix) .* uint16(img);
    % Création d'une image hdr visible
    hdr_image= double(img_pond);
    rgb = tonemap(hdr_image);
    %imtool(rgb);
    imwrite(rgb,'results\rgb.jpg','BitDepth',8);
    % Sommes des valeurs d'intenstié de toute l'image fusionnée
    intensite_sans_sat_not_pond = sum(img(:));
    % Sommes des valeurs d'intenstié de toute l'image fusionnée et pondérée
    intensite_sans_saturation_pond = sum(img_pond(:));
```

# Annexe 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filename    : fusion_hq_ponderation.m
% Context     : HES-SO-VS bachelor work
% Authors     : Mario Brunetti
% Version     : 1.0.0
% Last modif. : 23.08.2020
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input variables
%   myFoler : name of the folder containing the 5 images
% Output variables
%
% Input files
%   the file the name of which is passed as input parameter
% Output files
%   none
% Loaded variables
%   none
% Saved variables
%   none
% Global variables
%   none
% Used figures
%   none
% Tested Matlab versions
%   9.6.0.1174912 (R2019b) Update 1
% Needed toolboxes and tested versions
%   Image Processing Toolbox for imfinfo
% Called specific M-files (i.e. not in any toolbox)
%   read_PIHQraw_from_jpg.m
% Remarks
%   Fusion de 5 images captures realisées avec la camera HQ avec des
%   temps d'exposition diferents. Les 5 captures à analysée doivent se
%   trouver dans le dossier myFolder dont le lien est à spécifier.
%   Elle sont egalement nommées de manières à avoir le temps d'exp du plus
%   faible au plus elevé. capture 01, capture 02 au lieu de capture 1,
%   capture 2
%
%   Les valeurs de seuil pour la saturation et pour le niveau de bruit sont
%   à specifier dans les variables saturation_threshold et noise_threshold.
%   le fichiers brut de l'image est recuperer a l'aide du script
%   read_PIHQraw_from_jpg.m la resolution de l'image est de
%   rows 1520, columns 2028 et codée sur 16 bits (0 - 65536).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all,clear,clc
%% Settings
saturation_threshold = 65530; %Set the threshold value for saturated pixels
```

```
noise_threshold = 8000;       %Set the threshold value for noised pixels
%% Parcourir le dossier
myFolder = 'path_to_folder';
filePattern = fullfile(myFolder, '*.jpg');
jpegFiles = dir(filePattern);

%% Recupere info taille des images
file = [myFolder '\' jpegFiles(1).name];
%Extraction des données bruts 16bits du fichier jpg vers images rgb 16bits
s = read_PIHQraw_from_jpg(file);
rows = size(s.rgb,1);
columns = size(s.rgb,2);

%% Initialisation des masques
mask = zeros(rows,columns);
img = zeros(rows,columns);
mask_rgb = zeros(rows,columns,3);
map= zeros(rows,columns,3);
map_all= zeros(rows,columns,3);
ponderation_matrix = zeros(rows,columns);
% Ponderation en fonction des temps d'exposition
ponderation = [10000 100 10 1.6667 1];
%% Définition des couleurs pour le plot des zones
img_red = zeros(rows,columns,3);
img_green = zeros(rows,columns,3);
img_blue = zeros(rows,columns,3);
img_yellow = zeros(rows,columns,3);
img_pink = zeros(rows,columns,3);


img_red(:,:,1)=255;
img_green(:,:,2)=255;
img_blue(:,:,3)=255;
img_yellow(:,:,1)=255;
img_yellow(:,:,2)=255;
img_pink(:,:,1)=255;
img_pink(:,:,3)=255;


img_colored(:,:,:,1)= img_red;
img_colored(:,:,:,2)= img_green;
img_colored(:,:,:,3)= img_blue;
img_colored(:,:,:,4)= img_yellow;
img_colored(:,:,:,5)= img_pink;
%% Init vecteur des temps d'exposition et de la somme des valeurs de
%   tous les pixels (gray img)
t_exp = [];
intensit_cam = [];
```

```
for k = 1:length(jpegFiles)
  file = [myFolder '\' jpegFiles(k).name];

  %Extraction des données bruts 16bits du fichier jpg vers images rgb 16bits
  s = read_PIHQraw_from_jpg(file);
  imwrite(uint8(s.rgb/256),['results\' sprintf('%d.jpg',k)])

  %Conversion vers image en niveaux de gris
  img_gray = rgb2gray(s.rgb);


  % Sauvegarde le temps d'exposition dans un vecteur et la somme des rgb
  t_exp = [t_exp,s.exp*1000000];%us
  intensit_cam = [intensit_cam,sum(s.rgb(:))];

  %Inversion du masque precedant 0 à la place de 1 et inversemment
  old_mask = mask;
  inv_old_mask = ~old_mask;
  inv_old_mask = 1-inv_old_mask;
  inv_old_mask = (inv_old_mask == 0);

  %Création d'un nouveau masque binaire à partir de l'image
  % avec 0 = saturé ou bruit, 1 = pixel ok
  mask = img_gray <= saturation_threshold & img_gray >= noise_threshold;
  %Mise a zeros des pixels deja recupéré de l'image precedante
  %Les pixels déjà recupérer dans la precedante image sont a 1 dans mask
  % donc sont a 0 dans inv_old_mask
  mask_tmp = inv_old_mask.*mask;
  old_ponderation_matrix = ponderation_matrix;
  ponderation_matrix = mask_tmp .* ponderation(k);
  ponderation_matrix = uint16(ponderation_matrix) + uint16(old_ponderation_matrix);
  %%
  old_map = map_all;
  map = mask .* img_colored(:,:,:,k);
  % mise a zero des pixel deja ok
  map_tmp_sans_old = map.*inv_old_mask;
  map_all = old_map + map_tmp_sans_old;
  %% création d'une nouvelle image avec les valeurs des pixels ok
  %mask_tmp 1 = nouveaux element a ajouter
  img_old = img;

  %img_tmp = la nouvelle image avec les nouvelles valeurs
  img_tmp = uint16(img_gray).*uint16(mask_tmp);
  img = uint16(img_old) + uint16(img_tmp);
  mask = mask_tmp + old_mask;
end
  figure,imshow(img)
  % Ecriture de l'image hdr sans ponderation des intensités
```

```
imwrite(uint8(img/256),'results\fusion_hq.jpg', 'BitDepth',8);
figure,imshow(map_all)
% Ecriture de l'image contenant les zones de couleurs
imwrite(map_all,'results\map.jpg','BitDepth',8);
% Ecriture de la matrice de ponderation
% imwrite(ponderation_matrix,'ponderation_hq.jpg','BitDepth',16);
% Ecriture de l'image hdr ponderée
img_pond = uint32(ponderation_matrix) .* uint32(img);

imwrite(uint8(img_pond/65536),'results\img_pond.jpg','BitDepth',8);
% Sommes des valeurs d'intenstié de toute l'image fusionnée
intensite_sans_sat_not_pond = sum(img(:));
% Sommes des valeurs d'intenstié de toute l'image fusionnée et pondérée
intensite_sans_saturation_pond = sum(img_pond(:));
% Création d'une image hdr visible
hdr_image= double(img_pond);
rgb = tonemap(hdr_image);
imtool(rgb);
imwrite(rgb,'results\rgb.jpg','BitDepth',8);
```

# Annexe 6

```
# Boucle de capture utilisant raspistill et commande shell
# Script pour la camera v1 avec le filtre ND 3.0
import subprocess
import time
# Import the ADS1x15 module.
import Adafruit_ADS1x15
# Create an ADS1115 ADC (16-bit) instance.
adc = Adafruit_ADS1x15.ADS1115()
# See table 3 in the ADS1015/ADS1115 datasheet for more info on gain.
GAIN = 1
# Flags for raspistill capture commande
resolution = ' --width 2592 --height 1944'; # Set the resolution
quality = ' --quality 100';                 # Set the quality of jpeg
sharpness = ' -sh 0';                       # Set image sharpness (-100 - 100)
constrast = ' -co 0';                       # Set image contrast (-100 - 100)
brightness = ' -br 50';                     # Set image brightness (0 - 100)
saturation = ' -sa 0';                      # Set image saturation (-100 - 100)
iso = ' --ISO 100';                         # Sets the ISO to be used for captures
ev = ' -ev 0';                              # Set EV compensation (-10 - 10)
exposureMode = ' --exposure sports';        # Set exposure mode
awb = ' --awb sun';                         # Set Automatic White Balance (AWB) mode
drc = ' -drc high';                          # Enable/disable dynamic range compression
metteringMode = ' --metering matrix';       # Set mettering mode
k=1;                                        # Innit the counter of images
# Set the list of shutter open time to the specified value (in microseconds)
shutterList = [10,100,1000,10000,100000,200000,400000,600000,800000,1000000
    ,2000000,3000000,4000000,5000000]
# Initialisation des mesures ADC du pyranometre
pyranoValueList = []
for shutter in shutterList:
    shutterSpeedCmd = ' -ss {0}'.format(str(shutter));
    print(shutterSpeedCmd)
    fileName = "capture_{0}.jpg".format(k)
    print(fileName)
    outputFileCmd = ' -o /home/pi/captures/' + fileName
    print(outputFileCmd)
    cmd = "raspistill -t 1000 --raw --nopreview" + resolution + quality + \
          sharpness + constrast + brightness + saturation + iso + ev + \
          exposureMode + awb + shutterSpeedCmd + drc + quality + \
          metteringMode + annotations + outputFileCmd + resolution
    subprocess.call(cmd, shell=True)
    # Read the specified ADC channel using the previously set gain value.
    pyranoValue = adc.read_adc(0, gain=GAIN)
    pyranoValueList.insert(0, pyranoValue)        ## insert elem at index 0
    print(pyranoValue)
    k = k+1
```

```
    time.sleep(2)
#Save the list of values of the adc read into a file
f=open('/home/pi/captures/f1.txt','w')
for ele in pyranoValueList:
    f.write(str(ele)+'\n')
f.close()
```

# Annexe 7

```
# Boucle de capture utilisant raspistill et commande shell
# Script pour la camera hq avec le filtre ND 3.0
import subprocess
import time
# Import the ADS1x15 module.
import Adafruit_ADS1x15
# Create an ADS1115 ADC (16-bit) instance.
adc = Adafruit_ADS1x15.ADS1115()
# See table 3 in the ADS1015/ADS1115 datasheet for more info on gain.
GAIN = 1
# Flags for raspistill capture commande
resolution = ' --width 4056 --height 3040'; # Set the resolution
quality = ' --quality 100';                 # Set the quality of jpeg
sharpness = ' -sh 0';                       # Set image sharpness (-100 - 100)
constrast = ' -co 0';                       # Set image contrast (-100 - 100)
brightness = ' -br 50';                     # Set image brightness (0 - 100)
saturation = ' -sa 0';                      # Set image saturation (-100 - 100)
iso = ' --ISO 100';                         # Sets the ISO to be used for captures
ev = ' -ev 0';                              # Set EV compensation (-10 - 10)
exposureMode = ' --exposure sports';        # Set exposure mode
awb = ' --awb sun';                         # Set Automatic White Balance (AWB) mode
drc = ' -drc off';                          # Enable/disable dynamic range compression
metteringMode = ' --metering matrix';       # Set mettering mode
k=1;                                        # Innit the counter of images
# Set the list of shutter open time to the specified value (in microseconds)
shutterList = [10,100,1000,10000,100000,200000,400000,600000,800000,1000000
    ,2000000,3000000,4000000,5000000]
# Initialisation des mesures ADC du pyranometre
pyranoValueList = []
for shutter in shutterList:
    shutterSpeedCmd = ' -ss {0}'.format(str(shutter));
    print(shutterSpeedCmd)
    fileName = "capture_{0}.jpg".format(k)
    print(fileName)
    outputFileCmd = ' -o /home/pi/captures/' + fileName
    print(outputFileCmd)
    cmd = "raspistill -t 1000 --raw --nopreview" + resolution + quality + \
          sharpness + constrast + brightness + saturation + iso + ev + \
          exposureMode + awb + shutterSpeedCmd + drc + quality + \
          metteringMode + annotations + outputFileCmd + resolution
    subprocess.call(cmd, shell=True)
    # Read the specified ADC channel using the previously set gain value.
    pyranoValue = adc.read_adc(0, gain=GAIN)
    pyranoValueList.insert(0, pyranoValue)        ## insert elem at index 0
    print(pyranoValue)
    k = k+1
```

```
    time.sleep(2)
#Save the list of values of the adc read into a file
f=open('/home/pi/captures/f1.txt','w')
for ele in pyranoValueList:
    f.write(str(ele)+'\n')
f.close()
```

# Annexe 8

module v1, filtre 3.00

| | | | Valeurs mesurées par l'ADC [bit] | | | | | Valeurs convertie en [W/m^2] | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| capture n° | durée d'exposition [us] | facteur de ponderation | session01 | session02 | session03 | session04 | session05 | session01 | session02 | session03 | session04 | session05 |
| 1 | 10 | 40000 | 22659 | 20744 | 17541 | 6166 | 3508 | 935 | 863 | 742 | 313 | 212 |
| 2 | 100 | 4000 | 22698 | 20780 | 16529 | 6270 | 3456 | 937 | 864 | 704 | 317 | 210 |
| 3 | 10000 | 40 | 22499 | 20759 | 14395 | 6060 | 3442 | 929 | 863 | 623 | 309 | 210 |
| 4 | 100000 | 4 | 22897 | 20747 | 14427 | 6069 | 3449 | 944 | 863 | 624 | 309 | 210 |
| 5 | 400000 | 1 | 22923 | 20971 | 14813 | 6144 | 3456 | 945 | 871 | 639 | 312 | 210 |
| | | moyenne | 22735 | 20800 | 15541 | 6142 | 3462 | 938 | 865 | 666 | 312 | 211 |
| | | sommes des valeurs des pixels pondérés | 2.29E+09 | 2.25E+09 | 1.68E+09 | 1.12E+09 | 671714402 | | | | | |

module HQ, ouverture f16, filtre 3.00

| | | | Valeurs mesurées par l'ADC [bit] | | | | | Valeurs convertie en [W/m^2] | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| capture n° | durée d'exposition [us] | facteur de ponderation | session01 | session02 | session03 | session04 | session05 | session01 | session02 | session03 | session04 | session05 |
| 1 | 100 | 10000 | 6912 | 21949 | 22051 | 4692 | 24057 | 341 | 908 | 912 | 257 | 988 |
| 2 | 10000 | 100 | 23820 | 22100 | 22086 | 13112 | 24175 | 979 | 914 | 913 | 575 | 992 |
| 3 | 100000 | 10 | 7013 | 22108 | 22069 | 19877 | 24175 | 345 | 914 | 913 | 830 | 992 |
| 4 | 600000 | 1.666666667 | 5015 | 22473 | 22021 | 19127 | 24080 | 269 | 928 | 911 | 802 | 989 |
| 5 | 1000000 | 1 | 5180 | 22932 | 19769 | 22677 | 23761 | 275 | 945 | 826 | 936 | 977 |
| | | moyenne | 9588 | 22312 | 21599 | 15897 | 24050 | 442 | 922 | 895 | 680 | 988 |
| | | sommes des valeurs des pixels pondérés | 2.84E+11 | 5.67E+11 | 5.08E+11 | 6.02E+11 | 5.87646E+11 | | | | | |

# References


[1] Ken Birman, Lakshmi Ganesh, Robbert Renessee, Michael Ferris, Andreas Hofmann, Brian Williams, Janos Sztipanovits, Graham Hemingway, Anjan Bose, Anurag Stivastava, Santiago Grijalva, Sarah Ryan, James Mccalley, David Woodruff, Jinjun Xiong, Emrah Acar, Bhavna Agrawal, Andrew Conn, Gary Ditlow, and Basil Smith. Computational needs for the next generation electric grid proceedings. 10 2011.

[2] Seppo Valkealahti Julius Schnabel. Energy storage requirements for pv powerramp rate control in northern europe. 06 2016.

[3] PV Spectral response. https://www.ee.co.za/article/thermal-photovoltaics-energy-generation-storage.html/. Accessed: 2020-08-13.

[4] J. Jiang, D. Liu, J. Gu, and S. Süsstrunk. What is the space of spectral sensitivity functions for digital color cameras? In *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, pages 168–179, 2013.

[5] Angle solide d'un cone de révolution. https://fr.wikipedia.org/wiki/Angle_solide. Accessed: 2020-07-13.

[6] Walter Richardson, Hariharan Krishnaswami, Rolando Vega, and Michael Cervantes. A low cost, edge computing, all-sky imager for cloud tracking and intra-hour irradiance forecasting. *Sustainability*, 9:482, 03 2017.

[7] How to read analog signals from Python with an analog to digital converter and Raspberry Pi. https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads1115. Accessed: 2020-07-10.

[8] Eblouissement (« Blooming »). http://www.optique-ingenieur.org/fr/cours/OPI_fr_M05_C06/co/Contenu_12.html. Accessed: 2020-07-10.

[9] Datasheet fe185c086ha-1.

[10] Ouverture de diaphragme. https://www.fabienbeilhe.com/blog/ouverture-diaphragme-photo. Accessed: 2020-07-25.

[11] Opening Raspberry Pi High Quality Camera Raw Files. https://www.strollswithmydog.com/open-raspberry-pi-high-quality-camera-raw/. Accessed: 2020-07-27.

[12] Filtre à densité neutre. https://fr.wikipedia.org/wiki/Filtre_%C3%A0_densit%C3%A9_neutre. Accessed: 2020-07-23.

[13] Convert RGB Image to Grayscale Image. https://www.mathworks.com/help/matlab/ref/rgb2gray.html. Accessed: 2020-08-25.

[14] Display High Dynamic Range Image. https://ch.mathworks.com/help/images/display-high-dynamic-range-image.html. Accessed: 2020-08-25.

[15] Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion. pages 382 – 390, 10 2007.

[16] Yann Traonmilin and Cecilia Aguerrebere. Simultaneous high dynamic range and super-resolution imaging without regularization. *SIAM Journal on Imaging Sciences*, 7:1624–1644, 08 2014.

[17] Soumyabrata Dev, Florian Savoy, Yee Hui Lee, and Stefan Winkler. High-dynamic-range imaging for cloud segmentation. *Atmospheric Measurement Techniques Discussions*, pages 1–14, 09 2017.

[18] Sylvain Cros, Nicolas Sébastien, Olivier Liandrat, Samuel Jolivet, and Nicolas Schmutz. Solar power forecasting for a massive and secure injection of photovoltaics on the grid. 11 2014.

[19] Calibrate fisheye lens using OpenCV . https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b0. Accessed: 2020-07-25.