



Université  
Sorbonne  
Paris Nord

## Rapport de stage



## Observatoire d'une colonie de fourmilions

*Rubinthan Jegatheeswaran*

**INSTRUMENTATION 3**

**M.Vincent Lorent**

**Encadrant**

# Table des matières

<b>Table des matières .....</b>	<b>1</b>
<b>1 Introduction générale.....</b>	<b>2</b>
<b>2 Observation des pièges des fourmillons .....</b>	<b>4</b>
2.1 Présentation du matériel et pièges des fourmilions .....	4
2.2 MotionEye .....	6
2.3 Prise de photo à l'aide de python.....	9
2.4 Résultats et interprétations .....	10
<b>3 Traitement des images .....</b>	<b>15</b>
3.1 Stockage des images à l'aide d'un NAS.....	15
3.2 Création d'une vidéo avec ImageJ .....	16
3.3 Reconnaissance de forme avec OpenCV .....	16
a) Reconnaissance de mouvement .....	17
b) Reconnaissance d'objet(fourmis) .....	19
<b>Conclusion .....</b>	<b>20</b>
<b>Bibliographie.....</b>	<b>21</b>

# 1 Introduction générale

Le stage a consisté à observer une colonie de fourmilions situés dans le campus de Villetaneuse. Le but était de quantifier l'activité autour des pièges des fourmilions : le nombre de proies capturées par fourmilion dans l'intervalle de temps de l'observation, le type de proies capturées. Ces deux mesures sont faites après observation par des algorithmes de reconnaissance de forme et de mouvement.

Le fourmilion est un insecte Névroptère appartenant à la famille des Myrméléontidés. Au stade larvaire, qui dure entre deux à trois ans, cet insecte est un prédateur à l'affût. Les fourmis sont ses proies les plus fréquentes, mais on a observé des larves de fourmilion se nourrir aussi de petits coléoptères, de chenilles, de mites, de guêpes et d'araignées.

La larve du fourmilion creuse un trou en forme d'entonnoir plus ou moins creux selon sa taille. Ce piège en entonnoir est créé à partir d'un sillon circulaire que l'animal a creusé à reculons en jetant le sable vers l'extérieur du cercle initial. Le fourmilion se trouvera à la fin de la construction au fond de son piège où il attend ses proies, ne laissant entrevoir que le bout de sa tête et ses mandibules. Une fois qu'une fourmi tombe dans le piège, la larve essaie de l'agripper avec ses mandibules. Si la fourmi essaie de sortir, la larve lance du sable vers elle afin que cette dernière glisse jusqu'à capture. A la fin de la capture, la larve injecte un venin au travers de ses mandibules, puis des enzymes qui digèrent l'intérieur de la fourmi. A la fin de ce processus de digestion, d'autres canaux s'ouvrent dans l'appareil mandibulaire : la larve se nourrit alors en aspirant le liquide interne.

La simplicité architecturale du piège fait par le fourmilion en fait un modèle idéal pour l'analyse et la compréhension des propriétés physiques d'un piège garantissant son efficacité. Les connaissances sur la biophysique du piège du fourmilion sont encore partielles.

En particulier, Jérôme Crassous, professeur à l'université de Rennes et ses collègues ont analysé les caractéristiques physiques de ce piège en laboratoire en 2017. Ils ont

modélisé ce piège à l'aide d'un plan d'inclinaison variable formé de billes de verre, sur lequel ils ont posé des objets de différentes formes. Ils ont observé dans des conditions comparables à celles du trou de fourmilion que la pression exercée par un objet très léger n'est pas assez forte pour perturber la structure et provoquer une avalanche. Au contraire, un objet trop lourd déplace assez de billes pour creuser son propre trou où il est alors stoppé dans sa chute. Cela expliquerait pourquoi les insectes trop légers ou trop lourds peuvent s'échapper du piège du fourmilion. Quant aux fourmis, qui ont un poids moyen, celles-ci provoquent une avalanche qui les entraîne vers le fond. (J. Crassous, 2017)

Nigel. R. Franks et al. ont publié en 2019 les résultats de plusieurs expériences faites avec seize fourmilions placés dans du sable composé de trois types de grains de diamètre différents. Ils ont remarqué que les parois des pièges des fourmilions sont essentiellement composées de petits grains. (Nigel R. Franks, Digging the optimum pit: antlions, spirals and spontaneous stratification, 2019)

Sebastian Busse et Thies H. Buscher en 2021 ont déterminé des paramètres mécaniques en jeu lors de la modification du piège lorsque une proie y est présente. (Sebastian Büsse, 2021)

L'objectif du stage est cependant différent de ces études mécaniques, sans en être si éloigné. L'activité d'un fourmilion consiste, hors moment de chasse, à fabriquer et maintenir son piège en état : c'est une dépense en énergie alors que la ressource en énergie est liée à son alimentation. Avant d'arriver à un bilan métabolique il apparaît fondamental d'enregistrer et l'activité du fourmilion et son taux de prise de capture.

## 2 Observation des pièges des fourmillons

### 2.1 Présentation du matériel et pièges des fourmilions

Les pièges de fourmilions se situe dans le campus de Villetaneuse en face du restaurant administratif.

Voici quelques photos des pièges pris avec un téléphone :



*Figure1 : Photo des pièges*

En lieu et place d'un téléphone, une caméra pilotée par un microordinateur Raspberry a été placée en surplomb de quelques pièges de cette colonie de fourmilions.

#### **Hardware :**

La liste du matériel utilisé est ci-dessous :

- **Raspberry PI 0**
- **Raspberry PI 4**
- **Camera Raspberry V2 Noir**
- **Camera HQ Raspberry**

#### **Caméra infrarouge 8 Mégapixels**

Résolution photo : 3280 x 2464 pixels

Résolution vidéo : 1080p30, 720p60, 640x480p90  
640x480p90

#### **Caméra infrarouge 12.3 Mégapixels**

Résolution photo : 4056 x 3040 pixels

Résolution vidéo : 1080p30, 720p60,

Compatible avec toutes les cartes Raspberry (A/A+/B/B+)  
(A/A+/B/B+)

Compatible avec toutes les cartes Raspberry

- **Capteur température/humidité (AM2302 )**

3 à 5V pour l'alimentation et l'I/O

2.5mA max durant la conversion (pendant l'acquisition des données)

Lecture d'humidité de 0-100% (avec une précision de 2-5%)

Lecture de température de -40 à 80°C (avec une précision de  $\pm 0.5^{\circ}\text{C}$ )

Echantillonnage à 0.5 Hz (une fois toutes les deux secondes)

- **Batterie externe :**

- 3 à 5V pour l'alimentation et l'I/O

- 2.5mA max durant la conversion (pendant l'acquisition des données)

- Lecture d'humidité de 0-100% (avec une précision de 2-5%)

- Lecture de température de -40 à 80°C (avec une précision de  $\pm 0.5^{\circ}\text{C}$ )

- Echantillonnage à 0.5 Hz (une fois toutes les deux secondes)

L'observation est pilotée par une carte Raspberry Pi 0 afin de minimiser la consommation d'énergie, les traitements d'images après acquisition ont été réalisés avec une carte Raspberry pi 4.

Pour faire cette expérience on a utilisé la Raspberry PI 0 puisqu'elle consomme peu par rapport à la Raspberry 3 ou 4 comme nous montre le tableau ci-dessous :

	Zero	Zero W	A+	A	B+	B	Pi2B	Pi3B	Pi3B+	Pi3A+	Pi4B
	/mA	/mA	/mA	/mA	/mA	/mA	/mA	/mA	/mA	/mA	/mA
Idling	100	120	100	140	200	360	230	230	400	240	575
Loading LXDE	140	160	130	190	230	400	310	310	690	470	885
Watch 1080p Video	140	170	140	200	240	420	290	290	510	280	600
Shoot 1080p Video	240	230	230	320	330	480	350	350	520	350	640

Figure2: Raspberry Pi Power Usage table 1

- On voit dans ce tableau que la Raspberry PI zéro est d'un facteur 4 à 5 plus économique en termes de consommation d'énergie par rapport à la pi 4.

- La caméra HQ a été choisie à la place de la caméra V2 principalement en raison de la possibilité d'y accoler une optique réglable

	Camera Module v1	Camera Module v2	HQ Camera
Net price	\$25	\$25	\$50
Size	Around 25 x 24 x 9 mm		38 x 38 x 18.4mm (excluding lens)
Weight	3g	3g	
Still resolution	5 Megapixels	8 Megapixels	12.3 Megapixels
Video modes	1080p30, 720p60 and 640 x 480p60/90	1080p30, 720p60 and 640 x 480p60/90	1080p30, 720p60 and 640 x 480p60/90
Linux integration	V4L2 driver available	V4L2 driver available	V4L2 driver available
C programming API	OpenMAX IL and others available	OpenMAX IL and others available	
Sensor	OmniVision OV5647	Sony IMX219	<a href="#">Sony IMX477</a>
Sensor resolution	2592 x 1944 pixels	3280 x 2464 pixels	4056 x 3040 pixels
Sensor image area	3.76 x 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)	6.287mm x 4.712 mm (7.9mm diagonal)
Pixel size	1.4 µm x 1.4 µm	1.12 µm x 1.12 µm	1.55 µm x 1.55 µm

*Figure3 : Camera V2 vs HQ Camera*

Dans la capture d'image nous avons expérimenté deux méthodes :

- La première approche était l'utilisation du logiciel MotionEye utilisé pour de la surveillance vidéo
- La deuxième approche était de prendre des photos à intervalle de temps régulier grâce à un script python et traiter ensuite l'information à l'aide d'un logiciel adapté.

## 2.2 MotionEye

MotionEye est un logiciel gratuit développé pour Linux et est utilisé en général pour faire de la surveillance caméra. Il est piloté par lignes de commande et peut fonctionner comme un daemon avec un encombrement plutôt faible et une faible utilisation du processeur. Ce logiciel permet de faire un traitement d'image en temps réel puisqu'il prend une photo et des vidéos uniquement lorsque la caméra détecte un mouvement. L'interface graphique de ce logiciel se présente comme-ci après avoir l'installer sur la Raspberry PI :



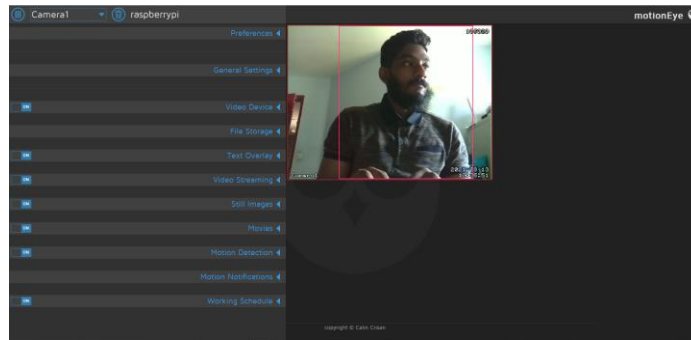


Figure 4 : Example

- Les avantages de ce logiciel sont de pouvoir le paramétrer en enregistrement de mouvements à des heures prédéterminées et d'émettre une notification mail de la détection. Ce logiciel est donc très adapté pour faire de la surveillance vidéo.

Voici les paramètres les plus importants à régler pour la détection de mouvement :

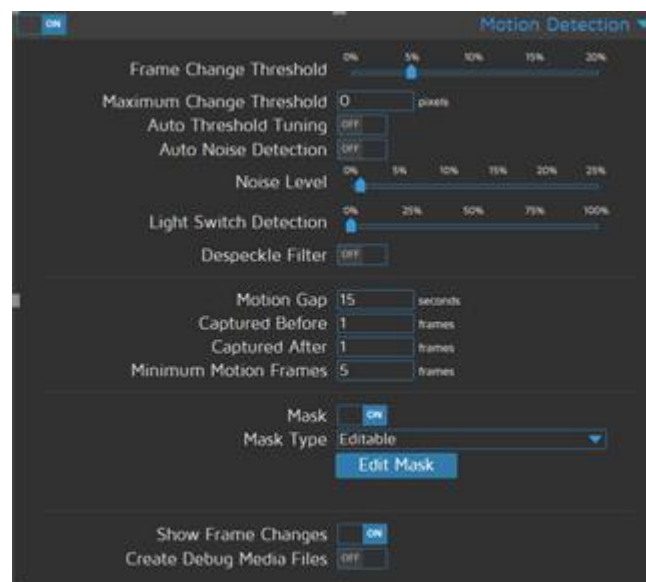


Figure 5 : paramètres de mouvement

- **Frame Change Threshold(seuil de changement d'image)** : permet de régler la sensibilité de la détection(plus il est faible plus il est sensible au mouvement)
- **Mask** : permet d'avoir plus de précision sur la détection.
- **Show Frame Changes** : permet d'avoir un carré rouge autour de l'objet détecté.

Ci-dessous les paramètres à régler pour recevoir une notification lorsque la caméra détecte du mouvement :



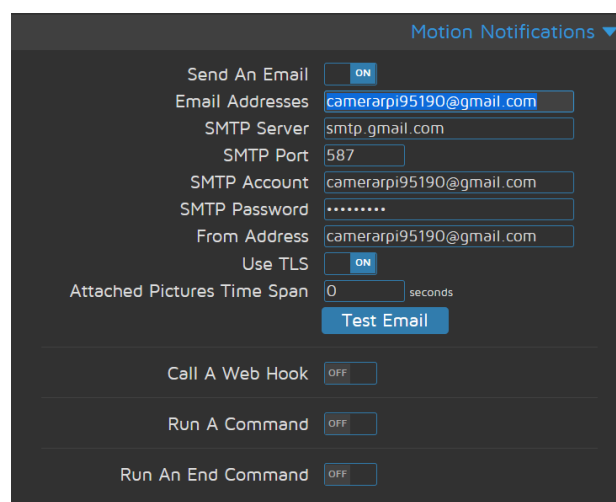


Figure 6 : détection de mouvement

On met le mail du destinataire afin qu'il reçoive une notification lorsque la caméra détecte un mouvement.

Voici un exemple de notification émis par la Raspberry PI pour me prévenir de la détection d'un mouvement.



Figure 7 : Notification Raspberry

De plus, on peut régler à quel moment la caméra doit être en mode surveillance.

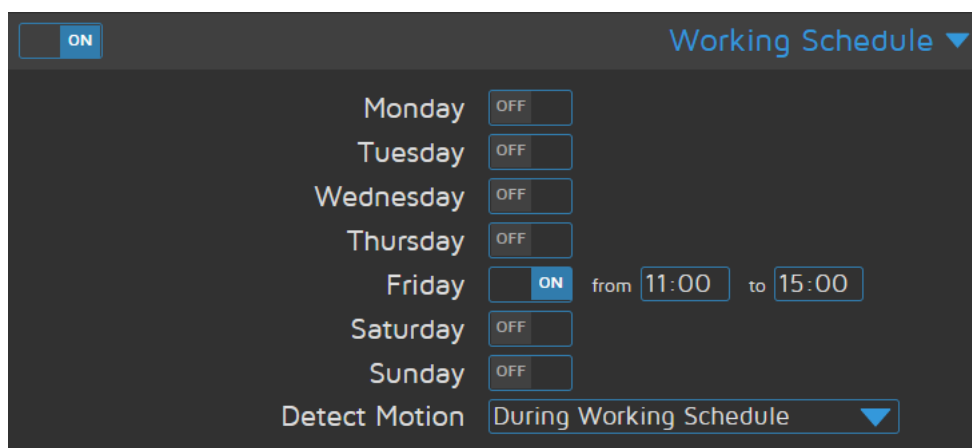


Figure 8 : Mode surveillance

Ce logiciel n'a pas été utilisé à l'endroit de l'observation car il n'y avait de connexion internet sur le site. Ainsi on a surveillé les pièges à l'aide d'une autre méthode à l'aide de Python et **Crontab**.

## 2.3 Prise de photo à l'aide de python

Dans cette partie, l'observation a été faite par des photographies prises à intervalles de temps régulier. A raison d'une photo toutes les minutes, pendant une semaine entière, notre évaluation de consommation d'énergie a été de 30000 mA.h .

A titre de comparaison ... etc

B with keyboard	= 1.89 W -> daily 45 Wh [6]
B+ with keyboard	= 1.21 W -> daily 29 Wh
B+ with LAN/USB chip off (no i/o except GPIO)	= 0.76 W -> daily 18.2 Wh
B+ shut down	= 0.26 W -> daily 6.2 Wh
A idle	= 0.7 W -> daily 17 Wh
A+ idle	= 0.52 W -> daily 12.5 Wh
Pi2 B at idle	= 1.15 W -> daily 28 Wh
Pi Zero at idle	= 0.51 W -> daily 12.2 Wh
Pi3 B at idle	= 1.15 W -> daily 28 Wh
Pi3 B at 100% * 4 CPUs	= 3.6 W -> daily 86 Wh

Figure 8 : Consommation quotidienne d'énergie par une carte raspberry, en fonction de son type.

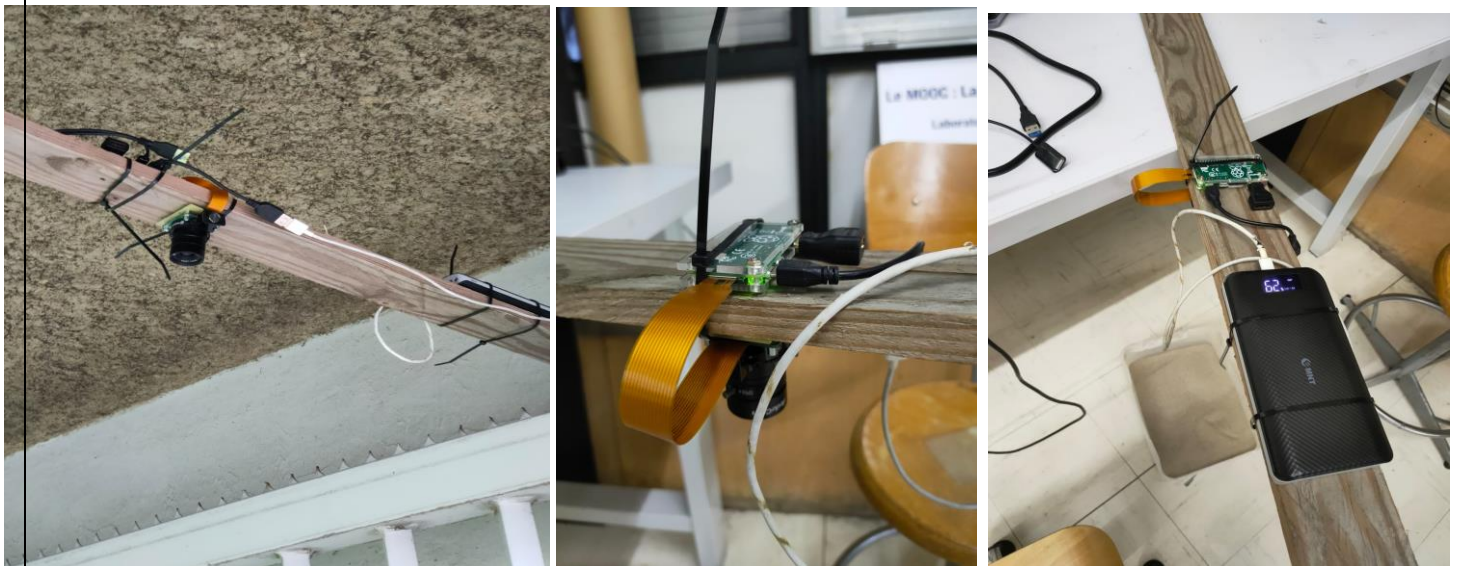


Figure 9 : Montage électrique

## 2.4 Résultats et interprétations

Les photos sont rangées par date : le nom sur chaque photo est la date à laquelle la photo a été prise dans le format (jour/mois/année/heure/minute/seconde)

Nom	Date	Type	Taille	Mots clés
2021-06-28	12/07/2021 18:55	Dossier de fichiers		
2021-06-29	12/07/2021 18:54	Dossier de fichiers		
2021-06-30	12/07/2021 18:51	Dossier de fichiers		
2021-07-01	12/07/2021 18:46	Dossier de fichiers		






				
28_06_21_12_19_0	28_06_21_12_20_0	28_06_21_12_21_0	28_06_21_12_22_0	28_06_21_12_23_0
1	1	1	1	1

Figure 10 : résultats

- ✓ On a obtenu au total 4495 photos pour 4 jours d'expérience du 28/06/21 à 12h06 jusqu'au 01/07/21 à 15h00 avant que la batterie ne se décharge complètement.
- Voici quelques exemples de photos obtenues à des heures et journées différentes :



28/06/21 à 14h08



28/06/21 à 22h06





29/06/21 à 5h46



29/06/21 à 20h47



30/06/21 à 16h38





30/06/21 à 21h31



01/07/21 à 5h35



01/07/21 à 15h00

- Voici le script python utilisé pour obtenir ces photos :

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import time
from datetime import datetime
import os

def PrisePhoto(NomPhoto): #prendre une photo avec Raspistill avec délai de 5 secondes avant photo
    command = "sudo raspistill -t 20000 -w 1200 -h 675 -o "+ NomPhoto +" -q 100"
    os.system(command)

if (os.path.isdir("/home/pi/Desktop/Experience") == False): # si le dossier pour stocker les photos n'existe pas
    os.mkdir("/home/pi/Desktop/Experience") # alors on crée le dossier (sur le bureau)
    os.chmod("/home/pi/Desktop/Experience",0o777) # et on change les droits pour pouvoir effacer des photos

#on génère le nom de la photo avec heure_min_sec
date_today = datetime.now()
nom_image = date_today.strftime('%d_%m_%y_%H_%M_%S')

#on déclenche la prise de photo
chemin_photo = '/home/pi/Desktop/Experience/'+nom_image+'.jpeg'
PrisePhoto(chemin_photo) #Déclenchement de la prise de photo
```

Ce programme est codé pour ne prendre qu'une seule photo mais avec l'aide de Crontab on peut le faire exécuter autant de fois qu'on le veut et quand on le veut.

Crontab est un outil permettant de configurer des tâches planifiées sur les systèmes Unix. Il est utilisé pour planifier l'exécution de commandes ou de scripts de façon périodique et à intervalles fixes.

Pour l'utiliser on tape `Crontab -e` sur le terminal et on configure le fichier comme suit :

```
# * * * * * command to execute  
# T T T T T  
#  
# | | | | |  
# | | | | | day of week (0 - 7) (0 to 6 are Sunday to Saturday,  
# | | | | | month (1 - 12)  
# | | | | | day of month (1 - 31)  
# | | | | | hour (0 - 23)  
# | | | | | min (0 - 59)
```

Dans notre exemple on a mis la ligne suivante dans le fichier nano /etc/crontab :

```
*/1 * * * * pi python3 /home/pi/Desktop/photo.py
```

- Cette ligne permet d'exécuter le programme toutes les minutes.

Crontab permet également de ne pas prendre de vue la nuit :

```
* /1 5-22 * * * pi python3 /home/pi/Desktop/photo.py
```

- Cette ligne sur Crontab permet d'exécuter le script python photo.py toutes les minutes de 5h à 22h.



De plus, on peut ajouter une ligne sur Crontab permettant d'exécuter le programme directement au démarrage de la Raspberry PI avec cette commande :

```
@reboot pi python3 /home/pi/Desktop/photo.py &
```

- Dans ce cas le **&** permet à la Raspberry PI de continuer de démarrer en même temps que le programme photo.py s'exécute.

L'utilisation de **Crontab** est très adaptée à notre expérience puisqu'il permet de limiter la consommation énergétique. En effet, si on utilise un programme sans l'utilisation de Crontab le programme tournera à l'infini ce qui fait consommer la Raspberry. Or avec celui-ci le programme ne s'exécute que quand on lui demande. Ainsi, l'utilisation de Crontab est un enjeu très important pour limiter la consommation énergétique de notre expérience. Dans la suite je vais vous présenter les résultats d'une expérience réalisée sans Crontab pour pouvoir comparer.

On réalise la même expérience mais avec un code qui permet de s'abstenir de l'utilisation Crontab.



- On a obtenu au total 919 photos pour 1 jour et demi d'expérience du 17/07/21 à 13h49 jusqu'au 18/07/21 à 21h03 avant que la batterie se décharge complètement.



- Voici le code utilisé pour faire cette expérience :

```
import time
import picamera
from datetime import datetime, timedelta

def wait():
    #Calculate the delay to the start of the next minute
    next_minute = (datetime.now() + timedelta(minutes=1)).replace(
        second=0, microsecond =0)
    delay = (next_minute - datetime.now()).seconds    #on convertit l'intervalle de temps en seconde
    time.sleep(delay)

with picamera.PiCamera() as camera:
    camera.start_preview()
    wait()
    for filename in camera.capture_continuous('/home/pi/Desktop/img{timestamp:%Y-%m-%d-%H-%M}.jpg'):
        print('Captured %s' % filename)
        wait()
```

Ce programme permet de s'affranchir de l'utilisation de Crontab. En effet il calcul lui-même le délai à attendre c'est-à-dire une minute dans ce cas-ci et exécute le script continuellement.

On voit bien d'après les résultats que cette expérience dure moins longtemps par rapport à la première expérience. Ce constat peut s'expliquer par le fait que dans cette expérience le programme s'exécute continuellement et ainsi a besoin continuellement de l'énergie pour fonctionner tandis que dans la première expérience le programme s'exécute qu'à des temps déjà défini sur Crontab.

### 3 Traitement des images

#### 3.1 Stockage des images à l'aide d'un NAS

Pour pouvoir ainsi récupérer ces photos j'ai créé un NAS (Network Attached Storage) entre mon pc et la Raspberry pi zéro. En effet, à la fin de l'expérience il suffit que je connecte la Raspberry pi zéro et mon pc sur la même connexion internet afin de récupérer les photos.

Pour créer ce NAS (dossier partagé) j'ai utilisé le logiciel samba. Tous d'abord il faut l'installer sur la Raspberry PI et ensuite ajouter à la fin du fichier smb.conf les lignes suivantes :

```
[Partage]
comment = Partage Samba sur Raspberry Pi
path = /home/pi/partage
writable = yes
guest ok = yes
guest only = yes
create mode = 0777
directory mode = 0777
share modes = yes
```

<b>Path</b>	→	chemin du dossier partagé
<b>Writable</b>	→	les utilisateurs pourront écrire dans ce fichier
<b>Guest ok</b>	→	les clients peuvent accéder à ce dossier sans mot de passe
<b>0777</b>	→	permet à l'utilisateur et aux clients d'écrire et de lire les fichiers. (rwxrwxrwx)

Ainsi dans l'explorateur du pc on verra ce dossier partagé avec le nom partage :

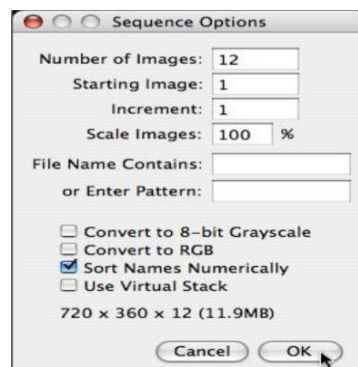
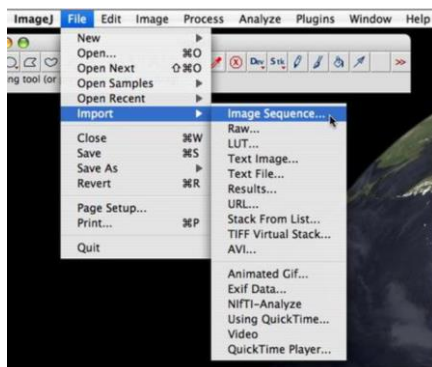


- Ainsi si je mets les photos qui sont stockés dans la Raspberry PI zéro dans ce dossier partagé, je pourrai les récupérer dans mon pc pour les traiter.

### 3.2 Création d'une vidéo avec ImageJ

Pour traiter les photos obtenues on utilise un logiciel de traitement d'image qui s'appelle ImageJ. C'est un logiciel multiplateforme, libre et open source de traitement et d'analyse d'images développé par les National Institutes of Health en 1987. Avec ce logiciel on a fait un empilement d'images (stacking) pour créer une vidéo afin de faire de la reconnaissance de mouvement avec openCV.

Pour créer ce stack on doit importer dans le logiciel les images que l'on veut traiter : Ensuite on définit les paramètres comme suit pour obtenir un stack d'images :



➔ Après avoir obtenu la vidéo on a fait du traitement d'images avec Open CV.

### 3.3 Reconnaissance de forme avec OpenCV

Open CV (Open Computer Vision) est une bibliothèque graphique développée initialement par Intel qui permet de traiter des images (extraction de couleurs, détection de visages, de formes, de mouvement, etc.).

Dans l'objectif de quantifier l'activité autour des pièges des fourmilions on a effectué deux types de reconnaissance différentes :

- la reconnaissance de mouvement (détection des insectes qui sont en mouvement autour des pièges)

- la reconnaissance d'objet (plus particulièrement la reconnaissance de fourmis)

a) *Reconnaissance de mouvement*

Le script python qui a permis de faire de la détection de mouvement se trouve en pièce complémentaire (« movement\_detectfinal1.py »).

Ce script compare l'image d'avant avec l'image d'après dans la vidéo et mesure la différence d'intensité par pixel entre les deux images pour détecter les objets en mouvement avec la fonction `cv2.absdiff()` d'open CV. Ensuite on trouve les contours autour des objets en mouvement `cv2.findContours()` puis on les trace sur l'image avec `cv2.drawContours()`.

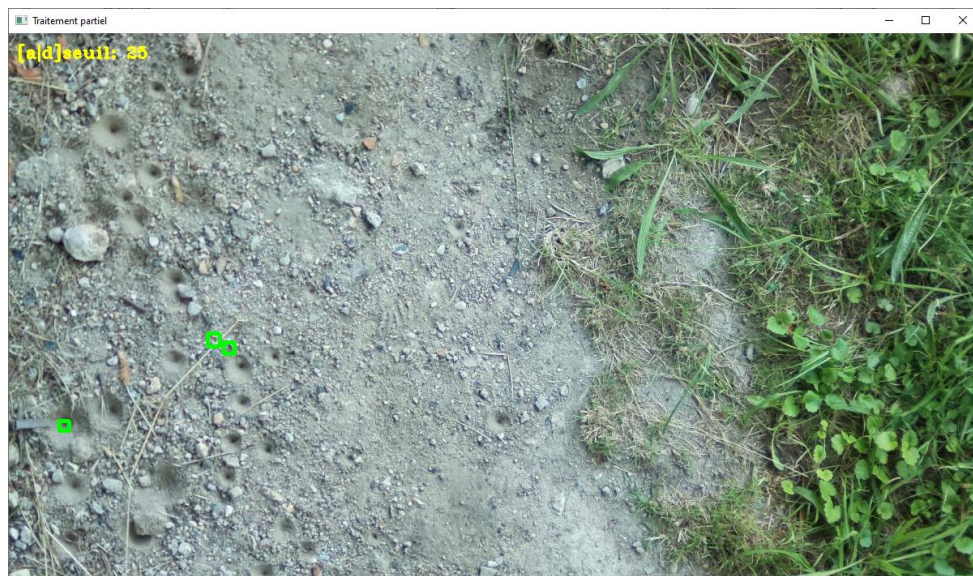
L'exécution de ce script génère quatre fenêtres de visionnage de la vidéo :

- La première fenêtre affiche la vidéo sans traitement d'image.
- La deuxième fenêtre affiche la vidéo avec un traitement sur toutes l'image.
- La troisième fenêtre affiche la vidéo avec un traitement partiel de l'image.
- La dernière affiche la vidéo avec un traitement sur la zone à analyser seulement.

Voici un exemple de capture d'image des quatre fenêtres prises au même moment :







Le principal défaut de ce programme est qu'il détecte tout type de mouvement, même lorsqu'il y a de l'herbe qui bouge à cause du vent il va le détecter comme on peut voir sur la capture du traitement de l'image total. Ainsi pour pallier ce problème on a décidé de faire un traitement d'image uniquement proche des pièges de fourmilion comme sur la capture de la zone à analyser. Ainsi on peut observer que sur la capture du traitement d'image partiel on ne détecte pas le mouvement de l'herbe.

Cette analyse de vidéo a permis de voir l'activité des insectes autour des pièges de fourmilion. Cependant on n'a voulu aller plus loin dans l'analyse en faisant de la reconnaissance de fourmis.

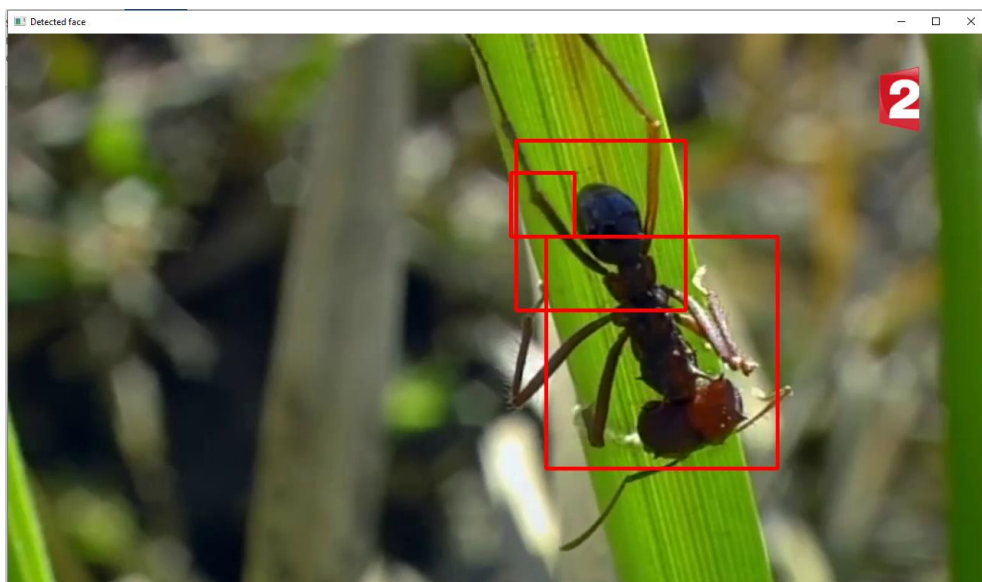
### *b) Reconnaissance d'objet(fourmis)*

Pour aller plus loin dans notre stage, on a décidé de faire de la détection de fourmis car ce sont les proies de prédilection des fourmilions. Cependant on n'a pas pu faire de la détection sur les vidéos qu'on a obtenu avec ImagesJ car les insectes que l'on observe sur les vidéos sont trop petits et on ne peut pas faire de la détection d'objet sur des objets si petit. Ainsi on n'a décidé de travailler sur des vidéos de fourmis trouvées sur internet (« vidéofourmis2.mp4 »).

Pour faire tous type de reconnaissance il faut tous d'abord entraîner un modèle et générer un classifieur. Pour générer ce classifieur, j'ai décidé d'utiliser un logiciel Cascade Trainer GUI. En effet, ce logiciel à partir d'une base de données qu'on lui transmet permet d'entraîner un modèle. Cette base de données doit être composé d'images positives, des images contenant l'objet que l'on veut détecter (dans notre cas ce seront des images de fourmis) et des images négatives (des images représentant des objets autres que l'on veut détecter).

Dans notre cas on n'a pas trouvé de base de données de fourmis sur internet directement. On n'a nous-même créer une base de données avec 150 images de fourmis et 200 images négatives. Ensuite on n'a généré le classifieur à partir de Cascade Trainer GUI. Pour entraîner le modèle à partir de notre base de données le logiciel a mis environ 1h30 pour générer un classifieur sous format XML (« antdetectorfinal.xml »)

Ce classifieur nous a permis d'écrire un script python pour faire de la détection de fourmis (« fourmis\_recognizervideo.py »). Ce script permet de détecter les fourmis dans une vidéo et tracer un contour rouge autour des fourmis qu'il détecte.



## Conclusion

Ce stage avait pour objectif de quantifier l'activité autour des pièges de fourmilion, notamment compter le nombre et le type de proies capturées par les fourmilions. On a réussi à avoir tous d'abord des milliers de photos des pièges dans leur état naturel.

Ensuite grâce à ces photos, on a fait de la reconnaissance de mouvement pour détecter les proies s'approchant des pièges de fourmilion. Pour aller plus loin, on a fait de la reconnaissance de fourmis car ce sont les proies les plus fréquentes des fourmilions.

Ainsi, ce stage nous a permis de comprendre à partir de l'étude des pièges de fourmilion le fonctionnement du machine learning avec la reconnaissance de mouvement et d'objet.

## Bibliographie

J. Crassous, A. H. (2017). Pressure-Dependent Friction on Granular Slopes Close to Avalanche. *Phys. Rev. Lett.*, 058003.

Nigel R. Franks, A. W.-F. (2019). *Digging the optimum pit: antlions, spirals and spontaneous stratification.*

Sebastian Büsse, T. H. (2021). *Sand-throwing behaviour in pit-building antlion larvae: insights from finite-element modelling.*



