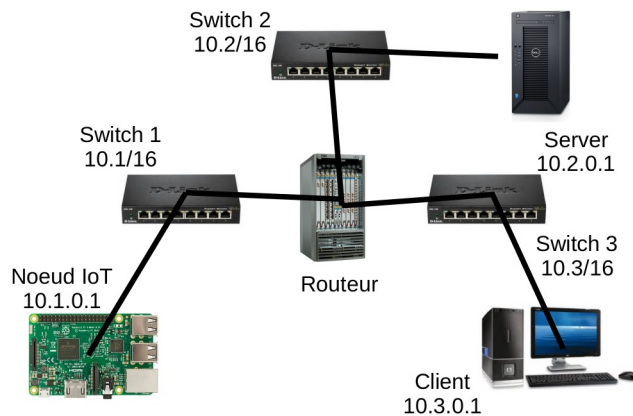


TD Internet of Things (IoT) avec Raspberry Pi

1 Création d'un réseau virtuel (NE PAS FAIRE)

Avec le logiciel NEmu, créer le réseau virtuel suivant :



1. Récupérez un noyau GNU/Linux pour Raspberry Pi émulé par QEMU :

<https://github.com/dhruvvyas90/qemu-rpi-kernel>

2. Sur le noeud IoT, installez le système Raspbian. Dans NEmu, utilisez les commandes suivantes en vous assurant du bon *path* pour le noyau :

```
VHostConf("rpi", "qemu-system-arm", M="versatilepb", no_reboot="", cpu="arm1176",
kernel="./qemu-rpi-kernel/kernel-qemu", append="root=/dev/sda2@panic=1@rootfstype=ext4@rw")

VHost('rpi1', conf='rpi', hds=[VFs('rpi1.img', 'cow', tag='rpi1.img')],
nics=[VNic(model='smc91c111')], m='256')
```

3. Sur le serveur et sur le client, installez Debian.

2 Préparation du Raspberry Pi

1. Installez la **Caméra V2** et le **Sense HAT** sur le Raspberry Pi si cela n'est pas déjà fait. Passez la nappe de la caméra au travers du Sense HAT avant de fixer celle-ci sur le port CSI situé entre le port HDMI et la prise mini-jack audio. Fixez ensuite le Sense HAT sur le **header GPIO à 40-pin** à l'aide des supports fournis (8 vis et 4 écrous).
2. Rendez vous sur le site officiel pour télécharger un OS pour le Raspberry Pi. Choisissez celui nommé **Raspbian Stretch Lite** (vous pouvez aussi le récupérer dans le dossier NFS **/net/stockage/dmagoni/**) :

<https://www.raspberrypi.org/downloads/raspbian>

3. Dézippez le pour obtenir un fichier **.img**, puis copiez le sur la carte SD comme suit :

```
# dd bs=4M if=/home/magoni/rpi/2018-10-09-raspbian-stretch-lite.img of=/dev/mmcblk0
```

4. Insérez la carte SD dans le slot du Raspberry Pi situé sous la carte à l'opposé des ports USB.

5. Branchez le Raspberry Pi 3 sur secteur avec une alimentation délivrant au moins 2,5A. Ne pas l'alimenter via le port USB d'un PC car l'ampérage sera insuffisant. Branchez un clavier sur un port USB et un écran sur le port HDMI.

6. Le Raspberry Pi boote son Linux et vous récupérez un **login** sur le **tty1**. Connectez vous avec le login **pi** et le mot de passe **raspberrypi**. Attention le clavier est QWERTY.

7. Lancez le menu de configuration et allez dans le sous menu **localisation** (choix 4), puis changez les **locales** (choix 1) avec **fr_FR.UTF-8** par défaut, changez la **time zone** (choix 2) avec Europe/Paris, configurez le clavier (choix 3) en AZERTY, et fixez le Wifi à **FR** :

```
$ sudo raspi-config
```

8. Toujours dans le menu, configurez le réseau (choix 2), puis activez la caméra et le serveur SSH dans le sous menu **interfacing options** (choix 5).

9. Optionnel : dans **Advanced options** (choix 7), sélectionnez **Expand Filesystem**.

10. A la fin des opérations, tapez sur **Finish** puis rebootez le Raspberry Pi.

11. Afin de se connecter au réseau Wifi Eduroam, créez le fichier `/etc/certs/ca.pem` et modifiez les fichiers `/etc/wpa_supplicant/wpa_supplicant.conf` et `/etc/network/interfaces` avec les instructions disponibles ici :

<http://dept-info.labri.fr/~magoni/risc/config-wpa2-raspberry-pi.txt>

12. Vérifiez avec **ifconfig** que l'interface Ethernet et/ou Wifi est bien configurée.

13. Testez la caméra en prenant une photo avec **raspistill** en JPEG (-e pour d'autres formats)

```
$ raspistill -o testcapture.jpg
```

14. Testez une capture vidéo (taille en pixels, durée en ms) avec **raspivid** as H.264

```
$ raspivid -t 60000 -w 1280 -h 720 -o hdvideo.h264
```

15. Mettez à jour le Pi :

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

16. Installez le paquet pour Sense HAT :

```
$ sudo apt-get install sense-hat
```

17. Créez un script Python et lancez le avec Python 3, vérifiez que le message s'affiche :

```
from sense_hat import SenseHat

sense = SenseHat()

sense.show_message("Hello world")
```

18. Pour effacer :

```
sense.clear()
```

19. Pour mesurer la pression :

```
pressure = sense.get_pressure()
print(pressure)
```

20. Pour mesurer la température :

```
temp = sense.get_temperature()
print(temp)
```

21. Pour mesurer le taux d'humidité :

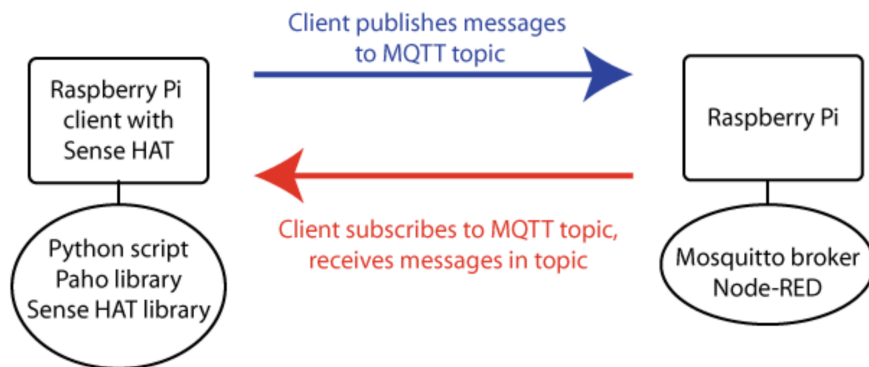
```
humidity = sense.get_humidity()
print(humidity)
```

22. Installez PIP, le gestionnaire de paquets pour Python :

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

3 Installation du client MQTT Paho et du serveur MQTT Mosquitto

L'architecture est la suivante avec la différence que le broker tourne sur un serveur virtuel dans un premier temps :



1. Définissez les **topics** MQTT auquel les messages appartiendront (e.g., sense/temp, sense/humid, etc).

2. Installez Paho, un client MQTT en python, sur le Raspberry Pi :

```
$ sudo pip3 install paho-mqtt
```

3. Ecrivez un script python clientEditeur.py qui publie les données des capteurs aux **topics** MQTT définies ci dessus puis lancez le sur le Raspberry Pi :

```
import paho.mqtt.client as mqtt
import time
from sense_hat import SenseHat
sense = SenseHat()
# set up mqtt client
client = mqtt.Client("python_pub")
# set mqtt username/pw
client.username_pw_set(username="pi", password="<mdp>")
# set server to publish to
client.connect("<broker-ip>", 1883)
client.loop_start()
```

```

try:
    while True:
        # publish temp to topic
        client.publish("sense/temp", sense.get_temperature())
        # publish humidity
        client.publish("sense/humid", sense.get_humidity())
        # pause for 10 seconds
        time.sleep(10)
        # deal nicely with ^C
        except KeyboardInterrupt:
            print("interrupted!")
client.loop_stop()

```

4. Installez la version LTS de NodeJS sur la machine serveur (source : <https://tecadmin.net/install-latest-nodejs-npm-on-debian/>) :

```

# apt-get install curl software-properties-common
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo bash -
# apt-get install nodejs

```

5. Vérifiez l'installation de NodeJS :

```

magoni@phoenix:~/rpi$ node -v
v8.12.0
magoni@phoenix:~/rpi$ npm -v
6.4.1

```

6. Installez Node-RED :

```
# npm install -g --unsafe-perm node-red
```

7. Configurez les dépôts pour Mosquitto (valable pour **stretch**, **jessie**, **wheezy**) :

```

$ wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
$ sudo apt-key add mosquitto-repo.gpg.key
$ rm mosquitto-repo.gpg.key
$ cd /etc/apt/sources.list.d/
$ sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
$ sudo apt-get update

```

8. Installez le serveur MQTT Mosquitto (**broker**) sur la machine serveur et installez les clients en ligne de commande :

```
# apt install mosquitto mosquitto-clients
```

9. Le broker démarre dès son installation. Vérifiez qu'il s'exécute correctement :

```
# systemctl status mosquitto
```

10. Arrêtez le et configurez un accès sécurisé en créant et modifiant un fichier mosquitto.conf :

```

# systemctl stop mosquitto
# cd /etc/mosquitto/conf.d/
# nano mosquitto.conf

```

11. Bloquez les clients anonymes et contrôlez les accès client au broker en définissant des noms et mots de passe (mdp) valides (cf. <http://mosquitto.org/man/mosquitto-conf-5.html>).

Ajoutez les lignes :

```

allow_anonymous false
password_file /etc/mosquitto/conf.d/passwd
require_certificate false

```

12. Sauvez et quittez nano.

13. Dans le dossier courant /conf.d, créez le fichier vide de mdps :

```
# touch passwd
```

14. Utilisez mosquitto_passwd pour créer le hash du mdp de l'utilisateur **pi** :

```
# mosquitto_passwd -c /etc/mosquitto/conf.d/passwd pi
```

15. Il faut entrer le mdp deux fois.

16. Relancez le broker et testez le. Pour cela, ouvrez un terminal et lancez un client souscripteur :

```
$ mosquitto_sub -v -t 'topic/test' -u pi -P <mdp>
```

17. Dans un autre terminal, lancez un client publiant :

```
$ mosquitto_pub -t 'topic/test' -m 'helloWorld' -u pi -P <mdp>
```

18. Si le broker fonctionne bien, le 1er terminal du souscripteur doit faire apparaître :

```
topic/test helloWorld
```

19. Lancez Node-RED sur le serveur et allez sur <http://localhost:1880>. Sur le canevas **Flow 1**, placez un élément **input mqtt** par drag'n drop depuis le panneau de gauche pour le topic sense/temp et configurez le. Faites de même pour le topic sense/humid. Placez un élément **output debug** par drag'n drop depuis le panneau de gauche. Reliez la sortie des deux éléments **mqtt** à l'entrée de l'élément **debug**.

20. Démarrez un traçage des messages échangés entre le Raspberry Pi et le broker MQTT avec Wireshark. Appuyez sur le bouton **Deploy** en haut à droite, qu'observez vous dans Node-RED et dans Wireshark ?

21. Sur le Raspberry Pi, écrivez un script python clientAbonne.py qui souscrit à un nouveau **topic** MQTT nommé **officeTemp** puis affiche les données reçues sur le Sense HAT. Lancez le sur le Raspberry Pi :

```
import paho.mqtt.client as mqtt
import time
from sense_hat import SenseHat
sense = SenseHat()
# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("officeTemp")
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print("got message on topic %s : %s" % (msg.topic, msg.payload))
    sense.show_message("MQTT Temp = %.2f" % (float(msg.payload)))
client = mqtt.Client()
client.username_pw_set(username="pi", password="<mdp>")
client.on_connect = on_connect
client.on_message = on_message
client.connect("<broker-ip>", 1883, 60)
try:
    client.loop_forever()
    # deal nicely with ^C
except KeyboardInterrupt:
    print("interrupted!")
```

22. Sur le canevas **Flow 1**, rajoutez un élément **output mqtt** par drag'n drop depuis le panneau de gauche pour le topic **officeTemp** et configurez le. Reliez la sortie de l'élément **input mqtt sense/temp** à l'entrée de l'élément **output mqtt officeTemp**.
23. Démarrez un traçage des messages échangés entre le Raspberry Pi et le broker MQTT avec Wireshark. Appuyez sur le bouton **Deploy** en haut à droite, qu'observez vous dans Node-RED et dans Wireshark ?
24. Installez maintenant un broker Mosquitto et un Node-RED sur votre Raspberry Pi (attention c'est un peu différent que sur un serveur Debian classique) afin que les autres groupes de vos collègues puissent récupérer les informations de votre Sense HAT. De même, modifiez votre clientAbonne.py afin qu'il collecte les informations des Sense HAT de tous les autres groupes.
25. Changez les paramètres de QoS dans Node-RED. Modifiez le code python en conséquence en lisant la doc : <http://www.eclipse.org/paho/clients/python/docs/>. Observez les messages échangés avec Wireshark.

Source : <https://hos.se/files/custom/ProductTemplate/sense-hat-application-note-0-en.pdf>

4 CoAP

1. Installez Git et la bibliothèque **Twisted** sur le Raspberry Pi :

```
$ sudo apt install git
```

```
$ pip install Twisted
```

2. Installez la bibliothèque **txThings** sur le Raspberry Pi et la machine client :

```
$ git clone https://github.com/mwasilak/txThings
```

```
$ cd txThings/
```

```
$ python setup.py build
```

```
$ sudo python setup.py install
```

3. Dans le sous répertoire **examples/** éditez le fichier server.py pour changer l'adresse IP du serveur puis lancez le serveur :

```
python server.py
```

4. Modifiez ce programme afin d'envoyer les valeurs de température, de pression et d'humidité fournies par le Sense HAT. Définissez des URIs auxquels ces valeurs seront rattachées.
5. Tracez les messages échangés entre le Raspberry Pi et la machine cliente avec Wireshark.
6. Sur la machine cliente, modifiez et lancez le programme client afin qu'il se connecte au serveur CoAP et récupère les valeurs produites par le Sense HAT :

```
python clientGET.py
```

7. Optionnel : Installez l'add-on **Copper** pour Firefox et accéder aux valeurs du serveur CoAP via l'URL : `coap://<raspberrypi-ip>:5683` .

5 Détection de mouvement avec OpenCV

1. Supprimez Wolfram pour gagner de l'espace (si vous avez la version **full**) :

```
sudo apt-get purge wolfram-engine
```

2. Installez les dépendances d'OpenCV :

```
sudo apt-get install build-essential cmake pkg-config
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libgtk2.0-dev
sudo apt-get install libatlas-base-dev gfortran
sudo apt-get install python2.7-dev python3-dev
```

3. Nous utiliserons dans la suite un environnement virtuel pour Python. Un *environnement virtuel* est un outil utilisé pour garder les dépendances requises par différents projets dans des emplacements séparés en créant des environnements Python *isolés et indépendants* pour chacun d'eux. Cet outil résout le problème "Projet X dépend de la version 1.x, mais Projet Y dépend de la version 4.x". Cela permet aussi de conserver le dossier global de Python nommé site-packages propre et net. (cf. <https://realpython.com/blog/python/python-virtual-environments-a-primer/>)

```
sudo pip install virtualenv virtualenvwrapper
sudo rm -rf ~/.cache/pip
```

4. Editez votre ~/.profile et ajoutez les lignes suivantes à la fin du fichier :

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

5. Rechargez votre profil :

```
source ~/.profile
```

6. Créez votre environnement virtuel utilisant Python 2.7 :

```
mkvirtualenv cv -p python2
```

7. Placez vous dans votre environnement virtuel. En cas de reboot, de relogging, ou de lancement d'un nouveau terminal, se replacer dans l'environnement virtuel en tapant :

```
$ source ~/.profile
$ workon cv
```

8. Vous devez avoir (cv) en tête du prompt.

```
(cv) pi@raspberrypi:~ $
```

9. Installez Numpy :

```
$ pip install numpy
```

10. (optionnel) Afin d'éviter la compilation d'OpenCV, téléchargez les libs compilées ici :

```
http://dept-info.labri.fr/~magoni/risc/opencv2-compiled.tar.gz
```

11. (optionnel) Puis replacez les dans /usr/local et dans /usr/local/lib/python2.7{python3.5}/site-packages. Rendez-vous directement à l'étape 15.

12. Récupérez le code d'OpenCV et de ses contributions. Assurez vous d'avoir la même version pour les deux archives (ici 3.4.3) :

```
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.3.zip
$ unzip opencv.zip
```

```
$ wget -O opencv_contrib.zip \  
https://github.com/opencv/opencv_contrib/archive/3.4.3.zip  
$ unzip opencv_contrib.zip
```

13. Configurez la compilation d'OpenCV :

```
$ cd opencv-3.4.3/  
$ mkdir build  
$ cd build/  
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D  
INSTALL_PYTHON_EXAMPLES=ON -D OPENCV_EXTRA_MODULES_PATH=~/.opencv_contrib-  
3.4.3/modules -D BUILD_EXAMPLES=ON ..
```

14. Compilez OpenCV (prend ~1h30 sur Raspberry Pi 3), puis installez :

```
$ make -j4  
$ sudo make install  
$ sudo ldconfig
```

15. Vérifiez que la bibliothèque est bien installée :

```
pi@raspberrypi:~ $ ls -l /usr/local/lib/python2.7/site-packages/  
total 4628  
-rw-r--r-- 1 root staff 4736996 nov. 8 20:53 cv2.so
```

16. Créez un lien symbolique dans votre environnement virtuel :

```
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/  
$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

17. Testez la bibliothèque :

```
pi@raspberrypi:~ $ workon cv  
(cv) pi@raspberrypi:~ $ python  
Python 2.7.13 (default, Sep 26 2018, 18:42:22)  
[GCC 6.3.0 20170516] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import cv2  
>>> cv2.__version__  
'3.4.3'
```

18. Effacez les dossiers source et objet pour récupérer de la place :

```
$ rm -rf opencv-3.4.3 opencv_contrib-3.4.3
```

19. Installez picamera et XFCE4 pour avoir un environnement graphique :

```
$ pip install "picamera[array]"  
$ sudo apt-get install xfce4  
$ startxfce4
```

20. Créez un fichier test_image.py avec le code suivant :

```
# import the necessary packages  
from picamera.array import PiRGBArray  
from picamera import PiCamera  
import time  
import cv2  
  
# initialize the camera and grab a reference to the raw camera capture  
camera = PiCamera()  
rawCapture = PiRGBArray(camera)  
  
# allow the camera to warmup  
time.sleep(0.1)  
  
# grab an image from the camera  
camera.capture(rawCapture, format="bgr")  
image = rawCapture.array  
  
# display the image on screen and wait for a keypress
```



```
cv2.imshow("Image", image)
cv2.waitKey(0)
```

21. Testez la capture d'une photo :

```
$ python test_image.py
```

22. Créez un fichier test_video.py avec le code suivant :

```
# import the necessary packages
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))

# allow the camera to warmup
time.sleep(0.1)

# capture frames from the camera
for frame in camera.capture_continuous(rawCapture,
    format="bgr", use_video_port=True):
# grab the raw NumPy array representing the image, then initialize the timestamp
# and occupied/unoccupied text
    image = frame.array
    # show the frame
    cv2.imshow("Frame", image)
    key = cv2.waitKey(1) & 0xFF
    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)
    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break
```

23. Testez la capture d'une video :

```
$ python test_video.py
```

24. Installez libffi et Paramiko (une bibliothèque Python qui implémente SSHv2) :

```
$ sudo apt install libffi-dev
```

```
$ sudo apt install libssl-dev
```

```
$ pip install paramiko
```

25. Créez un fichier pi_surveillance.py qui contient le code suivant :

```
# import the necessary packages
from pyimagesearch.tempimage import TempImage
from picamera.array import PiRGBArray
from picamera import PiCamera
import argparse
import warnings
import datetime
import dropbox
import imutils
import json
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
```

```

ap.add_argument("-c", "--conf", required=True, help="path to the JSON
configuration file")
args = vars(ap.parse_args())

# filter warnings, load the configuration and initialize the Dropbox
# client
warnings.filterwarnings("ignore")
conf = json.load(open(args["conf"]))
client = None
# check to see if the Dropbox should be used
if conf["use_dropbox"]:
    # connect to dropbox and start the session authorization process
    client = dropbox.Dropbox(conf["dropbox_access_token"])
    print("[SUCCESS] dropbox account linked")

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = tuple(conf["resolution"])
camera.framerate = conf["fps"]
rawCapture = PiRGBArray(camera, size=tuple(conf["resolution"]))

# allow the camera to warmup, then initialize the average frame, last
# uploaded timestamp, and frame motion counter
print("[INFO] warming up...")
time.sleep(conf["camera_warmup_time"])
avg = None
lastUploaded = datetime.datetime.now()
motionCounter = 0

# capture frames from the camera
for f in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    # grab the raw NumPy array representing the image and initialize
    # the timestamp and occupied/unoccupied text
    frame = f.array
    timestamp = datetime.datetime.now()
    text = "Unoccupied"
    # resize the frame, convert it to grayscale, and blur it
    frame = imutils.resize(frame, width=500)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21, 21), 0)
    # if the average frame is None, initialize it
    if avg is None:
        print("[INFO] starting background model...")
        avg = gray.copy().astype("float")
        rawCapture.truncate(0)
        continue
    # accumulate the weighted average between the current frame and
    # previous frames, then compute the difference between the current
    # frame and running average
    cv2.accumulateWeighted(gray, avg, 0.5)
    frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(avg))

    # threshold the delta image, dilate the thresholded image to fill
    # in holes, then find contours on thresholded image
    thresh = cv2.threshold(frameDelta, conf["delta_thresh"], 255,
cv2.THRESH_BINARY)[1]
    thresh = cv2.dilate(thresh, None, iterations=2)
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if imutils.is_cv2() else cnts[1]

    # loop over the contours
    for c in cnts:

```

```

# if the contour is too small, ignore it
if cv2.contourArea(c) < conf["min_area"]:
    continue
# compute the bounding box for the contour, draw it on the frame,
# and update the text
(x, y, w, h) = cv2.boundingRect(c)
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
text = "Occupied"

# draw the text and timestamp on the frame
ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
cv2.putText(frame, "Room Status: {}".format(text), (10, 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
cv2.putText(frame, ts, (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.35, (0, 0, 255), 1)

# check to see if the room is occupied
if text == "Occupied":
    # check to see if enough time has passed between uploads
    if (timestamp - lastUploaded).seconds >= conf["min_upload_seconds"]:
        # increment the motion counter
        motionCounter += 1

        # check to see if the number of frames with consistent motion is
        # high enough
        if motionCounter >= conf["min_motion_frames"]:
            # check to see if dropbox should be used
            if conf["use_dropbox"]:
                # write the image to temporary file
                t = TempImage()
                cv2.imwrite(t.path, frame)
                # upload the image to Dropbox and cleanup the tempory image
                print("[UPLOAD] {}".format(ts))
                path = "{base_path}/
{timestamp}.jpg".format(base_path=conf["dropbox_base_path"], timestamp=ts)
                client.files_upload(open(t.path, "rb").read(), path)
                t.cleanup()
            # update the last uploaded timestamp and reset the motion
            # counter
            lastUploaded = timestamp
            motionCounter = 0
# otherwise, the room is not occupied
else:
    motionCounter = 0

# check to see if the frames should be displayed to screen
if conf["show_video"]:
    # display the security feed
    cv2.imshow("Security Feed", frame)
    key = cv2.waitKey(1) & 0xFF
    # if the `q` key is pressed, break from the lop
    if key == ord("q"):
        break
# clear the stream in preparation for the next frame
rawCapture.truncate(0)

```

26. Créez un fichier conf.json utilisé par pi_surveillance.py pour sa configuration :

```

{
    "show_video": true,
    "use_dropbox": false,
    "dropbox_access_token": "YOUR_DROPBOX_KEY",
    "dropbox_base_path": "YOUR_DROPBOX_PATH",
    "min_upload_seconds": 3.0,
    "min_motion_frames": 8,

```

```

    "camera_warmup_time": 2.5,
    "delta_thresh": 5,
    "resolution": [640, 480],
    "fps": 16,
    "min_area": 5000
}

```

27. Modifiez les deux fichiers ci-dessus afin de gérer l'upload de l'image sur un serveur SFTP en vous inspirant du code ci-dessous :

```

import paramiko

host = "HOST_FQDN"          #hard-coded
port = 22
transport = paramiko.Transport((host, port))

password = "PASSWORD"     #hard-coded
username = "USERNAME"     #hard-coded
transport.connect(username = username, password = password)

sftp = paramiko.SFTPClient.from_transport(transport)

path = './TARGETDIRECTORY/FILENAME' #hard-coded
localpath = './pyimagesearch/' + FILENAME
sftp.put(localpath, path)

sftp.close()
transport.close()

```

28. Créez un dossier ~/pyimagesearch/ et placez un fichier vide nommé __init__.py dedans. Dans ce même dossier, créez un fichier tempimage.py avec le code suivant :

```

# import the necessary packages
import uuid
import os

class TempImage:
    def __init__(self, basePath=".", ext=".jpg"):
        # construct the file path
        self.path = "{base_path}/{rand}{ext}".format(base_path=basePath,
            rand=str(uuid.uuid4()), ext=ext)

        def cleanup(self):
            # remove the file
            os.remove(self.path)

```

29. Lancez le programme de surveillance :

```
$ python pi_surveillance.py --conf conf.json
```

30. Vérifiez que les images sont bien uploadées sur le serveur en SFTP.

Source : <https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/>