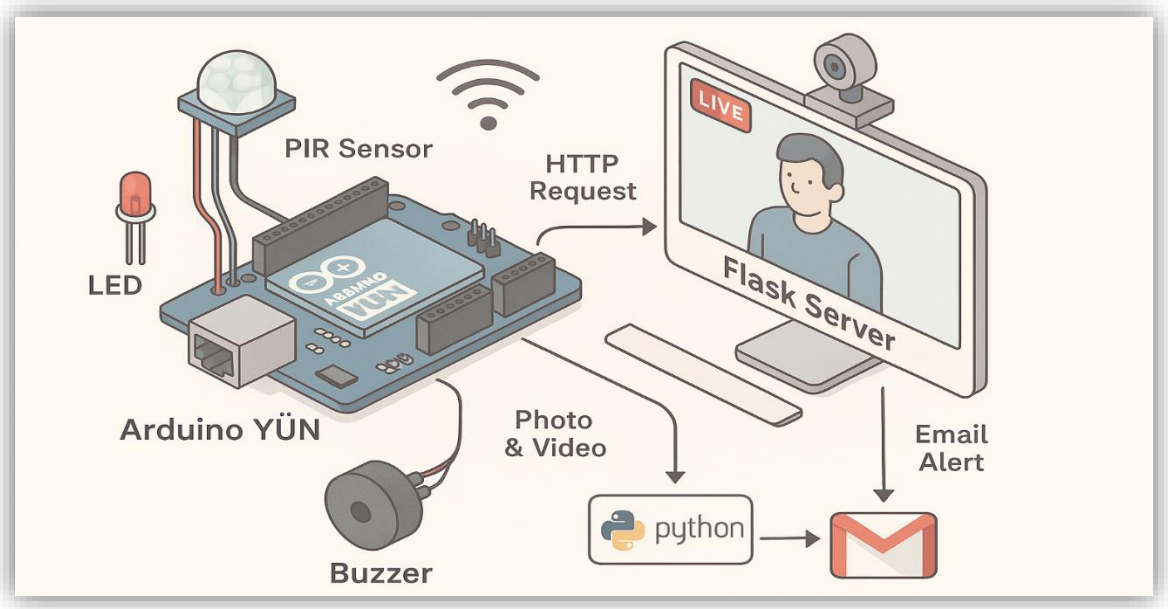
Université Hassan Premier

Faculté des Sciences et Techniques de SETTAT

MST- Réseaux et Systèmes Informatiques

RAPPORT DE PROJET DE FIN D'ANNÉE

SURVEILLANCE INTELLIGENTE : INTEGRATION D'UN CAPTEUR PIR AVEC ALERTE PAR E-MAIL ET STREAMING VIDEO



Réalisé par :

Abdelmounim MOUADILI

Sous la direction de :

M. Abderrahim MARZOUK, professeur à la FST de Settat

Soutenu le : **10 juin 2025**

Jury :

M. Bouchaib NASSEREDDINE, professeur à la FST de Settat

Année Universitaire : 2024-2025

Faculté des Sciences et Techniques
Km 3, B.P. : 577 Route de Casablanca Tél : 0523.40.07.36 Fax : 0523.40.09.69

Remerciements

À l'issue de cette année passée au sein de la Faculté des sciences et techniques Settat, j'ai acquis de nombreuses connaissances intellectuelles, ainsi je tiens à exprimer mes profonds remerciements, dans un premier temps, à toute l'équipe pédagogique et administrative de l'établissement pour la qualité de formation qu'ils ont prodiguée.

Mes remerciements vont tout spécialement à mes parents, qui ont su me soutenir et m'encourager tout au long de ma vie, ainsi que pour leur aide inestimable, leur patience et leur soutien indéfectible.

Je tiens à exprimer ma profonde gratitude à **M. Abderrahim MARZOUK**, mon encadrant, pour ses conseils précieux, son soutien constant et sa disponibilité tout au long de ce projet. Son expertise et son accompagnement ont grandement contribué à la réussite de ce travail.

Je tiens aussi à remercier tous les membres du jury qui ont porté un intérêt particulier à mon projet, et qui ont accepté de juger ce travail.

Je remercie également tous mes amis pour leur aide, leur encouragement et leur motivation qui m'ont permis de surmonter les difficultés rencontrées.

Enfin, je remercie toute personne qui a contribué, de près ou de loin, à l'exécution de ce projet.

Merci à Tous. . .

Sommaires

Résumé

Ce projet présente la conception et la mise en œuvre d'un système de surveillance intelligent basé sur un capteur infrarouge passif (PIR), une carte Arduino Yun et un serveur Python. Lorsqu'un mouvement est détecté, le système active une alerte visuelle et sonore (LED + buzzer), puis l'Arduino envoie une requête HTTP à un serveur local.

Le serveur, développé avec Flask et OpenCV, capture une photo, enregistre une vidéo de 5 secondes, puis envoie un e-mail automatique contenant ces fichiers en pièce jointe, ainsi qu'un lien vers un flux vidéo en direct. Cette architecture répartit les tâches : l'Arduino agit comme déclencheur rapide, tandis que l'ordinateur gère les fonctions avancées (multimédia, réseau).

Le système peut être enrichi par des fonctions comme la journalisation des événements, la mise en veille énergétique ou une interface web de configuration. Ce projet illustre une approche complète et évolutive de la surveillance connectée, mêlant électronique, programmation embarquée et développement web.

Abstract

This project presents the design and implementation of an intelligent surveillance system based on a Passive Infrared (PIR) sensor, an Arduino Yun board, and a Python server. When motion is detected, the system triggers a visual and audio alert (LED + buzzer), then the Arduino sends an HTTP request to a local server.

The server, developed using Flask and OpenCV, captures a photo, records a 5-second video, and automatically sends an email containing these files as attachments, along with a link to a live video stream. This architecture distributes tasks: the Arduino acts as a fast trigger, while the computer handles advanced functions (multimedia, networking).

The system can be enhanced with features such as event logging, energy-saving sleep modes, or a web interface for remote configuration. This project demonstrates a complete and scalable approach to connected surveillance, combining electronics, embedded programming, and web development.

Liste des tableaux

Liste des figures

1. Introduction

1.1 Contexte et Objectifs

1.1.1 Pourquoi la détection de mouvement ?

La détection de mouvement est une technologie largement utilisée dans divers domaines, notamment :

- **Sécurité** : Systèmes d'alarme, surveillance de locaux.
- **Automatisation** : Éclairage intelligent, déclenchement d'actions (caméras, enregistrement).
- **Économie d'énergie** : Activation de dispositifs uniquement en présence de mouvement (ex : éclairage public).

Dans ce projet, nous utilisons un **capteur PIR** (Passive Infrared Sensor) pour détecter les mouvements grâce aux variations de chaleur infrarouge émises par les corps.

1.1.2 Objectifs du Projet

Ce système a trois objectifs principaux :

- ✓ *Détection via capteur PIR :*
  - Utiliser un capteur **HC-SR501** pour repérer les mouvements dans son champ de vision.
  - Configurer la sensibilité et le temps de déclenchement.
- ✓ *Alerte visuelle (LED) :*
  - Faire clignoter une **LED** en cas de détection pour une notification immédiate.
  - Choix d'une broche numérique (ex : D13) pour un feedback simple.
- ✓ *Alerte à distance (e-mail via Python) :*
  - Envoyer une notification par **e-mail** dès qu'un mouvement est détecté.
  - Utiliser un **script Python** sur l'Arduino YUN pour communiquer avec un serveur SMTP (ex : Gmail).

1.2 Public Visé et Applications

1.2.1 Public Cible

Ce projet s'adresse à :

- **Étudiants en électronique/informatique** : Pour apprendre l'interfaçage capteur-microcontrôleur.
- **Makers et hobbyistes** : Un projet accessible avec des composants courants.
- **Débutants en Arduino/Python** : Bonne introduction à la programmation embarquée et aux scripts Linux.

1.2.2 Applications Possibles

Ce système de détection peut être déployé dans plusieurs domaines. Dans un contexte de **surveillance domestique**, il permet de détecter une intrusion et d'envoyer une alerte en temps réel, notamment s'il est couplé à une caméra. Il peut aussi être utilisé pour surveiller des pièces spécifiques comme un bureau ou un garage. En matière d'**automatisation IoT**, il peut déclencher automatiquement l'éclairage lorsqu'un mouvement est détecté, ou encore activer une alarme ou un enregistrement vidéo. Par ailleurs, ce projet constitue un excellent support pour des **activités pédagogiques**, en permettant aux apprenants de se familiariser avec le fonctionnement des capteurs infrarouges et de comprendre la communication entre une carte Arduino et un programme Python.

2. Matériels et Logiciels

2.1 Liste des Composants Électroniques

2.1.1 Arduino YUN

2.1.1.1 Caractéristiques techniques :

La carte **Arduino Yun** se distingue par sa **double architecture** unique qui combine deux processeurs : un microcontrôleur **ATmega32u4**, compatible avec l'Arduino Leonardo, et un **processeur Atheros AR9331** capable de faire tourner un système Linux embarqué (OpenWRT). Cette configuration hybride permet d'exécuter à la fois du code Arduino et des scripts Python ou Bash directement sur la carte. En termes de **connectivité**, l'Arduino Yun est très bien équipée, avec un module **Wi-Fi intégré** compatible 802.11b/g/n, un **port Ethernet 10/100 Mbps** pour les connexions filaires, ainsi qu'un **port USB Host** permettant de raccorder des périphériques externes (caméras, clés USB, etc.). La carte propose également **20 broches d'entrées/sorties numériques**, dont **7 sont compatibles PWM**, ainsi que **12 entrées analogiques**, offrant une grande flexibilité pour les projets électroniques. Enfin, elle dispose d'une **mémoire flash de 32 Mo**, dédiée au système Linux, ce qui permet d'y stocker des scripts et de faire tourner des services réseau embarqués.

2.1.1.2 Pourquoi choisir l'Arduino YUN ?

Sa capacité à exécuter simultanément des programmes Arduino classiques et des scripts Python via son processeur Linux en fait un choix idéal pour ce projet combinant détection matérielle et notification réseau.

2.1.2 Capteur PIR HC-SR501

2.1.2.1 Principe de fonctionnement :

Le capteur **PIR (Passive InfraRed)** permet la détection de mouvements en captant les variations de chaleur corporelle dans le spectre infrarouge, généralement entre **8 et 14 µm**. Il offre un **angle de détection d'environ 110°**, avec une **portée réglable** située entre **3 et 7 mètres**, ce qui le rend particulièrement adapté aux projets de surveillance de pièces de taille moyenne. Il fonctionne avec une **tension d'alimentation comprise entre 5V et 20V DC**, bien que l'usage à **5V soit recommandé** pour une compatibilité optimale avec l'Arduino.

2.1.2.2 Réglages importants :

Deux réglages mécaniques sont essentiels au bon fonctionnement du capteur. Le premier concerne le **temps d'activation (Tx)** : un petit potentiomètre permet de définir la durée pendant laquelle le signal de détection reste actif, allant de **0,3 seconde à 5 minutes**. Pour ce projet, un réglage entre **2 et 5 secondes** est conseillé afin d'éviter les latences trop longues. Le second réglage touche à la **sensibilité**, permettant d'ajuster la **distance de détection**. Il convient d'adapter ce paramètre en fonction de l'environnement physique afin de **minimiser les faux positifs** (causés, par exemple, par des sources de chaleur non humaines).

En fonctionnement typique, lorsque le capteur détecte un mouvement, sa **sortie numérique passe à HIGH (3,3V)** pendant toute la durée définie par Tx. Passé ce délai, en l'absence de nouveau mouvement, la sortie revient automatiquement à **LOW**, ce qui permet à l'Arduino de détecter clairement les événements discrets de mouvement.

2.1.3 LED et Résistance

2.1.3.1 Calcul de la résistance :

- Formule :

$$R = \frac{Vcc - V_f}{I_f}$$

  - Vcc = 5V (tension Arduino)

- V_f = 1.8-2.2V (chute de tension LED standard)
- I_f = 10-20mA (courant nominal)

- Exemple pour If=15mA :

$$R = \frac{5V - 2V}{0.015A} = 200\Omega$$

- Choix pratique : **résistance 220Ω** (valeur standard la plus proche)

2.1.3.2 Spécifications techniques :

Le système intègre une LED standard de 5 mm, utilisée à des fins d'indication visuelle (par exemple, lors de la détection de mouvement). Pour protéger la LED et limiter le courant, une résistance de 1/4 W est ajoutée en série, ce qui est amplement suffisant pour ce type de montage. Le câblage s'effectue de manière classique : l'anode de la LED est reliée à la broche numérique D13 de l'Arduino via la résistance, tandis que la cathode est connectée directement à la masse (GND). Ce montage permet d'utiliser la LED intégrée comme témoin d'état sans surcharge du circuit.

2.2 Logiciels et Outils

2.2.1 Environnement de développement Arduino

2.2.1.1 Configuration spécifique pour YUN :

1. Installer le **package Arduino YUN** via le Board Manager
2. Sélectionner :
   - Carte : "Arduino YUN"
   - Port : COM3 (ou port série approprié)
3. Bibliothèques nécessaires :
   - Bridge (communication AVR↔Linux)
   - Process (exécution de commandes Linux)

2.2.1.2 Fonctionnalités clés :

Le système offre plusieurs **fonctionnalités essentielles** qui facilitent son usage dans des projets embarqués. Il permet une **programmation classique du microcontrôleur** via l'environnement Arduino, tout en bénéficiant d'un **accès SSH intégré** au système Linux (OpenWRT) embarqué,

offrant ainsi une double couche de contrôle. L'**environnement Python** est pleinement exploitable côté Linux, permettant l'exécution de scripts automatisés. De plus, le système prend en charge les **mises à jour OTA (Over-The-Air)**, ce qui simplifie considérablement le déploiement ou la maintenance à distance, sans avoir besoin de rebrancher physiquement l'appareil.

2.2.1.3 Bibliothèques essentielles :

- smtplib : Envoi d'e-mails via SMTP
- email.mime : Formatage des messages
- os : Interaction avec le système

2.2.1.4 Installation sur Arduino YUN :

```
opkg update

opkg install python3 python3-pip

pip3 install smtplib2
```

2.2.1.5 Configuration SMTP typique (Gmail) :

```
server = smtplib.SMTP('smtp.gmail.com', 587)

server.starttls()

server.login('votre@email.com', 'motdepasseapp')
```

2.2.1.6 Outils réseau

SSH (PuTTY/Terminal) :

- Connexion à l'interface Linux :

ssh root@arduino.local

Mot de passe par défaut : "arduino"

2.2.1.7 Gestion des ports série :

Outils recommandés :

- screen (Linux/Mac)
- Putty (Windows)
- IDE Arduino (moniteur série intégré)

2.2.2 Workflow de développement typique

Avant de procéder au montage pratique, certaines étapes techniques sont indispensables. Tout d'abord, il convient d'**écrire le sketch Arduino** dans l'IDE, en veillant à intégrer les instructions nécessaires à l'interaction avec les scripts Python. Ensuite, une phase de **test matériel** via le moniteur série permet de vérifier le bon fonctionnement du capteur PIR et de s'assurer que les signaux sont bien reçus. Une fois ces vérifications faites, il faut **transférer les scripts Python** vers l'Arduino Yun, par exemple à l'aide de la commande scp script.py root@arduino.local:/mnt/sda1/, afin de les placer sur la carte SD intégrée. Enfin, l'**exécution des scripts** se fait directement depuis le sketch Arduino grâce à l'objet Process, permettant de déclencher des actions comme l'envoi de mails ou la capture d'images. Ce chapitre pose ainsi les fondations techniques nécessaires à la mise en œuvre complète du système.

Ce chapitre fournit toutes les bases techniques nécessaires avant d'aborder le montage pratique dans la section suivante.

3. Schéma et Branchements

3.1 Schéma Électrique Détaillé

- **Schéma du montage du circuit**

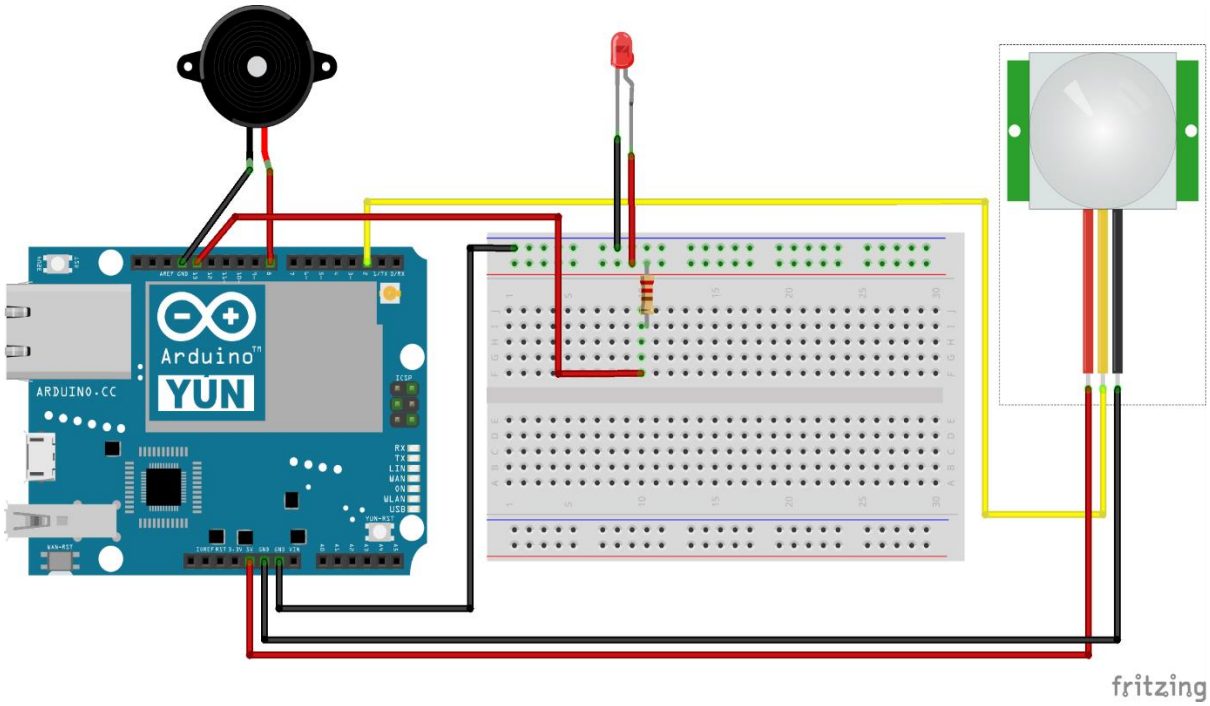


Figure 1: Schéma du montage du circuit

- **Montage du circuit**

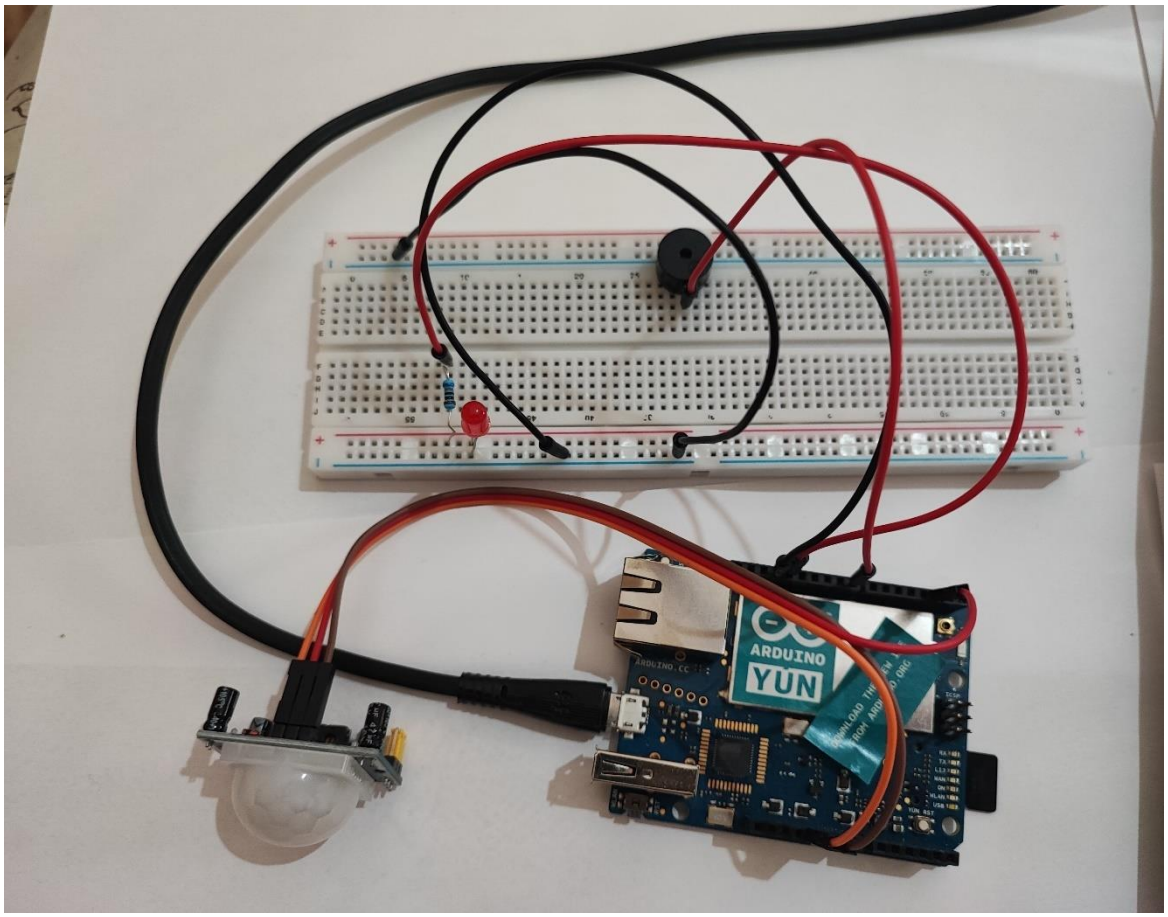


Figure 2 : Montage du circuit

- **Description des Connexions**

Capteur PIR HC-SR501 :

Broche PIR	Connexion Arduino	Couleur Fil (Convention)
VCC	5V	Rouge
OUT	D2	Jaune
GND	GND	Noir

Tableau 1 : Description des connexions

LED :

Élément	Connexion	Remarques
Anode	D13 via résistance	Branchement en série
Cathode	GND	Patte la plus courte

Tableau 2: Descriptions des connexions

Bon à savoir : La LED intégrée de l'Arduino YUN est aussi reliée à D13, ce qui permet de visualiser l'état sans composant externe lors des tests initiaux.

3.2 Explications des Choix de Branchements

3.2.1 Choix de la Broche D2 pour le PIR

Raisons techniques :

- **Broches d'interruption** :
  - L'Arduino YUN (ATmega32u4) dispose de broches supportant les interruptions matérielles : **D2 (INT0)** et **D3 (INT1)** sur cette carte.
  - Avantage : Détection immédiate sans polling logiciel.
- **Exemple de configuration d'interruption :**

attachInterrupt(digitalPinToInterrupt(2), mouvementDetecte, RISING);

  - Détecte le front montant quand le PIR passe à HIGH.
- **Éviter les broches critiques** :
  - D0/D1 utilisées pour la communication série.
  - D13 partagée avec la LED intégrée.

3.2.2 Calcul de la Résistance 220Ω

Application de la loi d'Ohm :

$$R = \frac{Vcc - V_{LED}}{I_{LED}}$$

- **Variables :**
  - Vcc=5V (tension Arduino)
  - V_{LED}≈2V (chute de tension typique pour une LED rouge)
  - I_{LED}=15mA (courant sécuritaire sans endommager la LED)
- **Calcul :**

$$R = \frac{5V - 2V}{0.015A} = 200\Omega$$

Choix de **220Ω** (valeur standard la plus proche).

Vérification de la puissance :

$$P = R \times I^2 = 220 \times (0.015)^2 \approx 0.05W$$

Une résistance **1/4W (0.25W)** convient parfaitement.

3.2.3 Optimisation des Broches

Broche	Alternative	Avantage
D2	D3	Autre broche d'interruption
D13	D12	Évite la LED intégrée si besoin

Tableau 3 : Optimisation des Broches

Schéma alternatif (si D2 indisponible) : const int pirPin = 3; *// Utilisation de D3 à la place*

3.2.4 Validation du Montage

Avant d'utiliser tout le système, il faut tester les composants un par un. Pour la LED, on peut simplement écrire digitalWrite(13, HIGH); dans le code : si la LED est bien branchée, elle s'allumera. Pour le capteur PIR, on vérifie le signal en affichant Serial.println(digitalRead(2)); : si un mouvement est détecté, le moniteur série affichera 1. Si ça ne fonctionne pas, il faut vérifier que le capteur est bien alimenté (certains modèles ont besoin de 5.5V), ou que la LED est bien orientée (la patte courte vers GND) et protégée avec une résistance.

4. Programmation Arduino

4.1 Code Complet Commenté

4.1.1 Structure de Base

```
// Déclaration des constantes
const int pirPin = 2;      // Capteur de mouvement PIR
const int ledPin = 13;     // LED de statut
const int buzzerPin = 8;   // Buzzer
// Variables d'état
bool motionActive = false; // État actuel du mouvement
bool alertSent = false;    // Pour éviter les doublons
unsigned long lastAlertTime = 0;
const unsigned long POST_ALERT_DELAY = 10000;
```

4.1.2 Fonction setup()

```
void setup() {
  // Configuration des broches
pinMode(pirPin, INPUT);
pinMode(ledPin, OUTPUT);
pinMode(buzzerPin, OUTPUT);
// Initialisation communication série
Bridge.begin();
Serial.begin(9600);
while (!Serial);  // Wait for Serial Monitor (Yún)
Serial.println("Motion Alert System Ready");
// Configuration interruption (optionnel)
  attachInterrupt(digitalPinToInterrupt(pirPin), detectionHandler, RISING);
}
```

4.1.3 Fonction loop()

```
void loop() {
  // Détection de mouvement
  if (digitalRead(pirPin) == HIGH) {
    digitalWrite(ledPin, HIGH);// LED ON
    digitalWrite(buzzerPin, HIGH);// Buzzer ON
if (!alertSent && (millis() - lastAlertTime > POST_ALERT_DELAY)) {
      Serial.println("ALERTE : Mouvement détecté !");
      sendAlert();// Envoi d’une requête HTTP au serveur
      alertSent = true;
      lastAlertTime = millis();// Mise à jour du temps de dernière alerte
    }
    motionActive = true;
  } else {
    // Aucun mouvement détecté
    digitalWrite(ledPin, LOW);
```

```
    digitalWrite(buzzerPin, LOW);
    if (motionActive) {
      Serial.println("Fin du mouvement.");
      alertSent = false;// Réinitialisation de l'alerte
    }
    motionActive = false;
  }
  delay(200);// Petite pause pour éviter la surcharge
}
```

4.1.4 Fonctions Auxiliaires

```
// Envoie une alerte HTTP GET vers le serveur Python
void sendAlert() {
  Serial.println("Envoi de la requête HTTP...");


  HttpClient client;
  String url = "http://192.168.162.16:5000/alert";// Adresse IP du
serveur Flask
  client.get(url);// Envoi de la requête GET
  // Attente de réponse (timeout à 5 secondes)
  unsigned long startTime = millis();
  while (!client.available() && millis() - startTime < 5000) {
    delay(100);
  }
  // Affichage de la réponse ou timeout
  if (client.available()) {
    Serial.println("Réponse du serveur :");
    while (client.available()) {
      Serial.print((char)client.read());
    }
    Serial.println();
  } else {
    Serial.println("ERREUR : Pas de réponse (timeout)");
```

```
  }
  delay(2000);  // Pause optionnelle après envoi
}
```

4.2 Fonctionnement Général

Quand un mouvement est détecté, plusieurs actions se déclenchent automatiquement :

1. La LED et le buzzer s'activent pour alerter visuellement et auditivement.
2. L'Arduino envoie une requête HTTP au serveur Python.
3. Le serveur capture une **photo**, enregistre une **vidéo de 5 secondes**, puis envoie un **e-mail** avec ces fichiers en pièce jointe.

Cette approche permet de centraliser la logique avancée (prise de vidéo, envoi d'e-mail) sur un ordinateur via Python, tout en laissant à l'Arduino le rôle de déclencheur réactif.

4.3 Avantages de cette Architecture

- **Modularité** : séparation claire entre la détection (Arduino) et le traitement (Python).
- **Simplicité de l'Arduino** : le code reste léger, sans besoin de gérer les fichiers multimédias ni les protocoles e-mail.
- **Évolutivité** : le serveur Python peut facilement être enrichi (streaming en direct, logs, base de données…).

4.4 Bonnes Pratiques

4.4.1 Gestion des Erreurs

```
void setup() {
  // Test LED
  digitalWrite(ledPin, HIGH);
  delay(200);
  digitalWrite(ledPin, LOW);
  // Test PIR
  if(digitalRead(pirPin) == HIGH) {
    Serial.println("ATTENTION : PIR déjà actif au démarrage");
  }
}
```

4.4.2 Économie d'Énergie

```
// Mode basse consommation
#include <avr/sleep.h>

void enterSleep() {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  sleep_mode();
}
```

4.5 Perspectives

L'implémentation peut être enrichie par :

- La mise en veille de l'Arduino pour économiser l'énergie (via sleep.h).
- La journalisation des événements sur carte SD ou via réseau.
- L'intégration d'une interface web pour configurer les paramètres à distance.

5. Script Python pour l'Envoi d'E-mail

5.1 Configuration SMTP

5.1.1 Activer l'authentification à deux facteurs :

- Aller sur compte Google > Sécurité
- Activer "Validation en 2 étapes"

5.1.2 Créer un mot de passe d'application :

- Même section > "Mots de passe d'application"
- Sélectionner "Autre (Nom personnalisé)" → Ex: "ArduinoYUN"
- Copier le mot de passe généré (16 caractères)

5.2 Paramètres SMTP :

```
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
EMAIL = "votre@gmail.com"
PASSWORD = "votre_mot_de_passe_app"
```

5.3 Code Python (Version de Base avec Gmail)

Ce script Python envoie un email vers la destinataire.

```
import smtplib
from email.mime.text import MIMEText
from email.utils import formatdate


def send_email(subject="Alerte Mouvement", body="Mouvement détecté
!"):
    # Configuration
    from_email = "votre@gmail.com"
    to_email = "destinataire@example.com"
    smtp_server = "smtp.gmail.com"
    smtp_port = 587
    password = "votre_mot_de_passe_app"
    # Création du message
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = from_email
    msg['To'] = to_email
    msg['Date'] = formatdate(localtime=True)
    try:
        # Connexion sécurisée
        with smtplib.SMTP(smtp_server, smtp_port) as server:
            server.starttls()  # Chiffrement TLS
            server.login(from_email, password)
            server.send_message(msg)
        print("Email envoyé avec succès !")
    except Exception as e:
        print(f"Erreur d'envoi : {str(e)}")


if __name__ == "__main__":
    send_email()
```

6. Script Python – Partie Alerte par e-mail avec capture photo et vidéo

Cette section décrit en détail la partie du script Python qui permet de **générer une alerte de sécurité** en capturant une **photo**, une **vidéo**, et en envoyant ces fichiers par **e-mail** avec un lien vers le **flux vidéo en direct**.

6.1 Déclaration des bibliothèques nécessaires

```
import socket
import cv2
from flask import Flask, send_file, Response, render_template
import os
import smtplib
import threading
import time
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.application import MIMEApplication
```

Ces bibliothèques permettent :

- La capture vidéo avec **OpenCV (cv2)**.
- Le serveur web avec **Flask**.
- L’envoi de mail via **smtplib** et les formats MIME pour les pièces jointes.
- La gestion des **fils d’exécution (threading)** et de **l’heure**.
- L'obtention de l'adresse IP locale avec **socket**.

6.2 Initialisation du serveur Flask et des paramètres globaux

```
app = Flask(__name__)
photo_path = 'photo_lowres.jpg'
```

- app = Flask(__name__) initialise le serveur web.
- photo_path est le chemin temporaire pour sauvegarder l'image capturée.

6.3 Paramètres de configuration de l'e-mail

```
EMAIL_SENDER = "..."
EMAIL_PASSWORD = "..."
EMAIL_RECEIVER = "..."
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587
```

Ces variables permettent de se connecter au serveur SMTP de Gmail pour envoyer des e-mails avec **authentification TLS**.

6.4 Gestion de la webcam et thread de capture d'images

```
camera = cv2.VideoCapture(0)
camera.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
latest_frame = None
lock = threading.Lock()
```

- Ouverture de la webcam (/dev/video0 par défaut).
- Résolution réduite pour alléger la charge réseau.
- Utilisation d'un **verrou** pour synchroniser l'accès au dernier frame.

```
def camera_reader():
    global latest_frame
    while True:
        ret, frame = camera.read()
        if ret:
            with lock:
                latest_frame = frame
        time.sleep(0.05)  # environ 20 FPS
```

Cette fonction lit en continu les images de la caméra, toutes les 50 ms, dans un thread séparé.

```
threading.Thread(target=camera_reader, daemon=True).start()
```

On lance la lecture de la caméra dans un **thread démon**, qui s'exécute en arrière-plan.

6.5 Fonction pour enregistrer une vidéo courte

```
def record_video(output_path='clip.mp4', duration=5, fps=20):
    ...
```

Cette fonction :

- Crée un VideoWriter avec codec MP4.
- Lit les frames pendant la durée spécifiée (5 secondes).
- Sauvegarde la vidéo sur le disque (dans clip.mp4).

6.6 Route /alert – Déclenchement de l'alerte

```
@app.route('/alert')
def alert_and_send_email():
```

Quand cette route est appelée :

1. Capture la **photo actuelle** depuis la webcam.
2. Enregistre une **vidéo de 5 secondes**.
3. Génère un **lien local vers le stream** vidéo.
4. Envoie un **e-mail** avec les pièces jointes.

- **- Capture de la photo :**

```
with lock:
    if latest_frame is None:
        return "No frame available", 500
    cv2.imwrite(photo_path, latest_frame,
[int(cv2.IMWRITE_JPEG_QUALITY), 50])
```

L'image est compressée en qualité réduite (50) pour alléger la pièce jointe.

- **- Enregistrement de la vidéo :**

```
record_video(video_path, duration=5, fps=24)
```

- **- Détermination de l'adresse IP locale (Wi-Fi) :**

```
def get_wifi_ip():
    ...
wifi_ip = get_wifi_ip()
stream_url = f"http://{wifi_ip}:5000/stream"
```

Cela permet de construire dynamiquement un lien d'accès local au flux vidéo.

- **- Préparation de l’e-mail :**

```
msg = MIMEMultipart()
msg["Subject"] = "Alerte Sécurité"
msg["From"] = EMAIL_SENDER
msg["To"] = EMAIL_RECEIVER
```

- **- Contenu texte :**

```
body_text = f"""Alerte ! Mouvement détecté 🚨
📷 Une photo est jointe à cet e-mail.
🔴 Live Stream: {stream_url}
"""
msg.attach(MIMEText(body_text, "plain"))
```

- **- Ajout de la photo et de la vidéo :**

```
# Attach photo
        with open(photo_path, "rb") as f:
            image = MIMEApplication(f.read(), _subtype="jpeg")
            image.add_header('Content-Disposition', 'attachment',
filename=os.path.basename(photo_path))
            msg.attach(image)


        # Attach video
        with open(video_path, "rb") as f:
            video = MIMEApplication(f.read(), _subtype="mp4")
            video.add_header('Content-Disposition', 'attachment',
filename=os.path.basename(video_path))
            msg.attach(video)
```

- **- Envoi via SMTP :**

```
server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
server.starttls()
server.login(EMAIL_SENDER, EMAIL_PASSWORD)
server.sendmail(EMAIL_SENDER, EMAIL_RECEIVER, msg.as_string())
```

6.7 Résumé

En résumé, cette section met en œuvre une solution complète d'alerte en cas de mouvement, en combinant la puissance de Python, OpenCV, Flask et le protocole SMTP. Grâce à ce script, chaque détection déclenche automatiquement la capture d'une photo, l'enregistrement d'une courte vidéo, et l'envoi d'un e-mail contenant ces fichiers ainsi qu'un lien vers le flux vidéo en direct. Cela permet de réagir rapidement à toute intrusion, tout en centralisant la gestion des fichiers et des notifications sur un ordinateur ou un serveur local, en parfaite complémentarité avec l'Arduino.

7. Script Python – Partie Serveur de Streaming Vidéo

Cette section décrit la partie du script Python qui permet de diffuser **en temps réel** le flux de la caméra via une page web accessible à l'adresse http://<ip>:5000/stream.

7.1 Route /stream – Page HTML

```
@app.route('/stream')
def stream():
    return render_template('stream.html')
```

Affiche une page web contenant un <img> pointant vers /video_feed.

7.2 Route /video_feed – Diffusion MJPEG

```
@app.route('/video_feed')
def video_feed():
    def generate():
        while True:
            with lock:
                if latest_frame is None:
                    continue
                _, jpeg = cv2.imencode('.jpg', latest_frame)
                frame = jpeg.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame +
b'\r\n\r\n')
            time.sleep(0.05)  # ~20 FPS
    return Response(generate(), mimetype='multipart/x-mixed-replace;
boundary=frame')
```

- generate() encode chaque frame en JPEG.
- Les images sont envoyées en continu avec le format **multipart MJPEG**.
- Ce flux est compatible avec tous les navigateurs.

7.3 Fonctionnement global

La webcam fonctionne en continu grâce à un thread de fond dédié, permettant de capturer des images en temps réel sans bloquer l'exécution principale du script. Chaque image (ou frame) capturée est stockée dans une variable partagée de manière sécurisée à l'aide d'un verrou (lock), garantissant l'intégrité des données lors des lectures et écritures concurrentes. Ce flux vidéo est ensuite diffusé en continu via une route Flask accessible dans un navigateur web, permettant une visualisation en direct. Par ailleurs, le lien vers ce flux est automatiquement inclus dans l'e-mail d'alerte, offrant ainsi un accès immédiat au flux vidéo à distance en cas de détection de mouvement.

7.4 Conclusion

Ce chapitre a présenté en détail la conception d'un système d'alerte intelligent basé sur **Python, Flask et OpenCV**. Lorsqu'un mouvement est détecté, une photo et une vidéo sont capturées automatiquement, puis envoyées par e-mail avec un lien d'accès au **flux vidéo en direct**. La webcam fonctionne en arrière-plan, et le flux est mis à disposition sur une page web grâce au format **MJPEG**, compatible avec tous les navigateurs. Cette intégration assure une **surveillance en temps réel** avec notification immédiate, ce qui renforce considérablement la réactivité du système face à toute activité suspecte.

8. Tests et Validation

Cette section présente les **tests fonctionnels** réalisés pour valider le bon fonctionnement du **système global**, composé de trois modules principaux :

- Le **capteur PIR**, qui détecte les mouvements,
- Le **serveur de streaming vidéo**, qui diffuse en direct les images captées,
- Le **système d'alerte par e-mail**, qui envoie automatiquement une notification avec **photo et vidéo en pièce jointe** lors d'une détection.

Chaque composant a été testé **indépendamment**, puis dans une **configuration intégrée**, afin de garantir une **réaction fiable et rapide** en cas d'intrusion.

8.1 Protocole de Test Complet

8.1.1 Tests Unitaires

Composant	Méthode de Test	Résultat Attendu	Outils
Capteur PIR	Mouvement manuel devant le capteur	LED clignote + Serial.print "1"	Multimètre, Moniteur Série
Envoi Email	Déclenchement manuel du script Python	Réception email sous 10s	Client Email
Latence système	Chronomètre (mouvement → email)	< 5s total	Chronomètre

Tableau 4 : Tests Unitaires

8.1.2 Test du capteur PIR (détection de mouvement)

Le capteur PIR a été relié à la carte Arduino Yun. Un test de détection a été réalisé dans une pièce fermée. Lorsqu'un mouvement était détecté :

- La carte Arduino envoyait une **requête HTTP** au serveur Python hébergé sur le PC.
- Une **LED s'allumait**, signalant la détection d'activité.
- La réponse du serveur confirmait le déclenchement de l'action.

✓ **Résultat :** La détection était instantanée et fiable dans un rayon de 3/5 mètres.

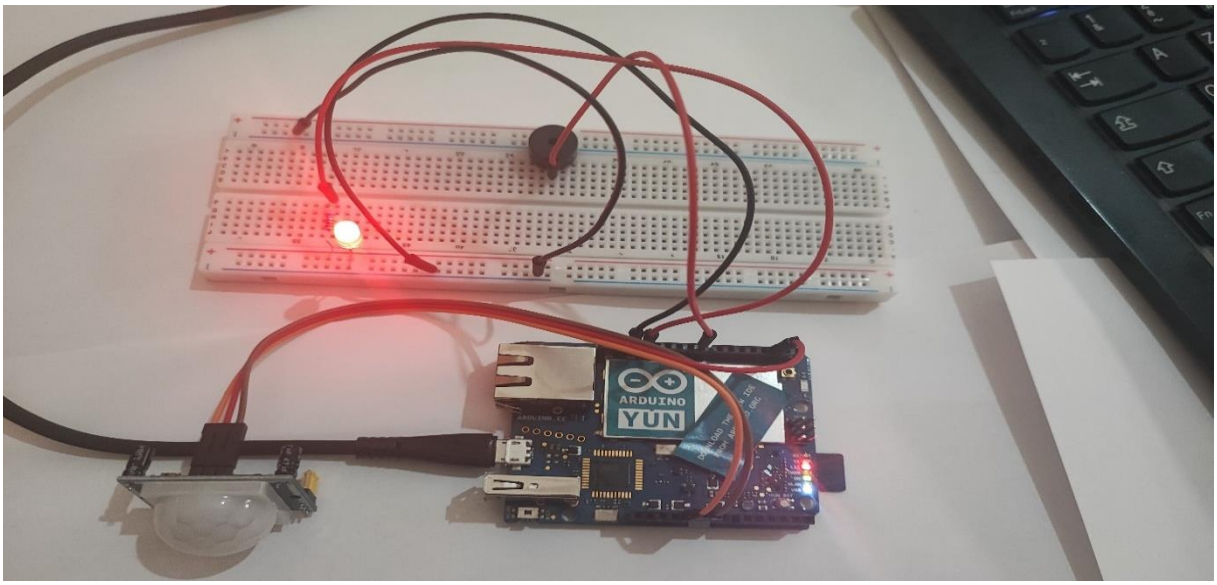


Figure 3 : Test du PIR (Clignoter le LED)

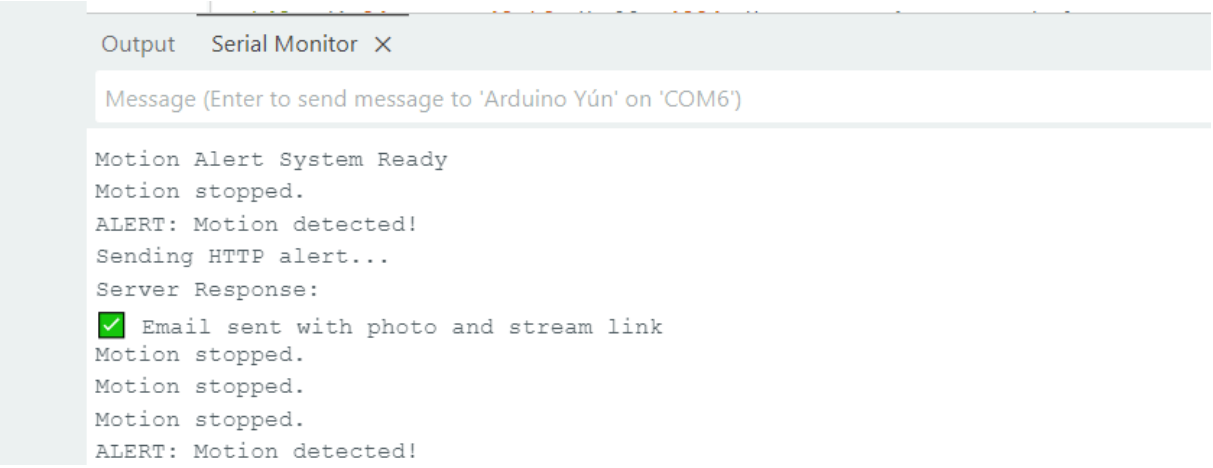


Figure 4 : Test du PIR (Serial Monitor)

8.1.3 Test du serveur de streaming (/stream)

Le module de streaming permet de visualiser en temps réel ce que la webcam du PC capte.

- Le serveur Flask a été lancé localement (ou sur IP Wi-Fi).
- L'URL http://<IP>:5000/stream a été ouverte dans un navigateur (PC et mobile).
- La vidéo s'affichait avec un rafraîchissement fluide (~20 FPS).
- Testé également à distance via un hotspot ou réseau local.

✓ **Résultat :** Le flux était stable et bien lisible. Aucun blocage constaté pendant plus de 30 minutes de test continu.

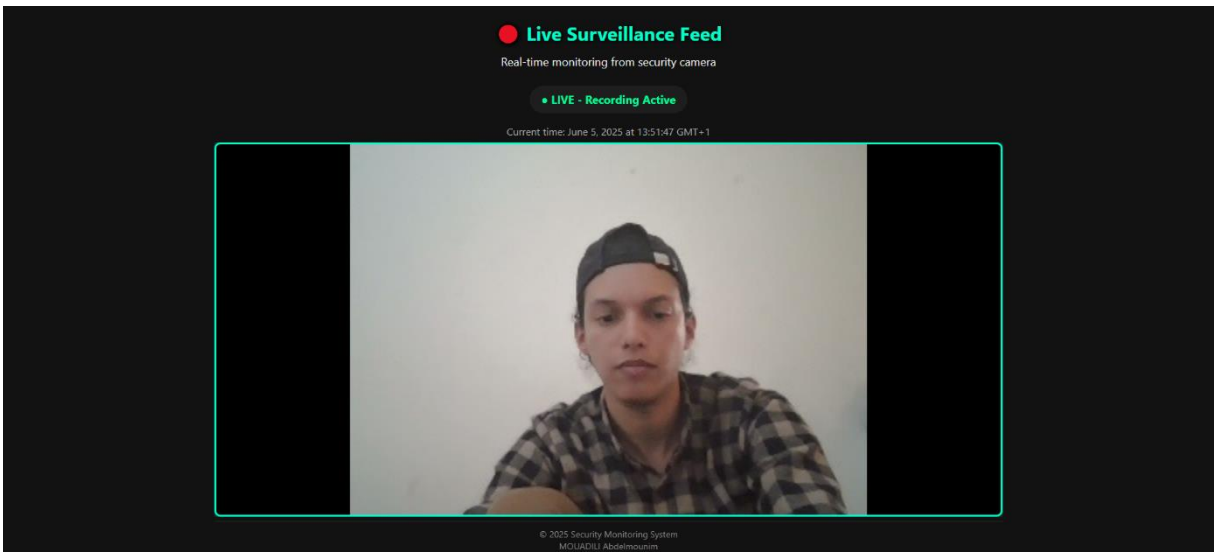


Figure 5 : Test de Stream

8.1.4 3. Test de l'alerte par e-mail (/alert)

Une fois la détection déclenchée :

- Une **photo** de la scène a été capturée.
- Une **vidéo de 5 secondes** a été enregistrée.
- Un **e-mail contenant la photo, la vidéo et le lien vers le flux en direct** a été envoyé à l'adresse de test.
- Réception vérifiée sur Gmail (mobile & PC), avec toutes les pièces jointes.

✓ **Résultat :** L'alerte était reçue en moins de 20 secondes après la détection. Tous les fichiers étaient lisibles et le lien cliquable.

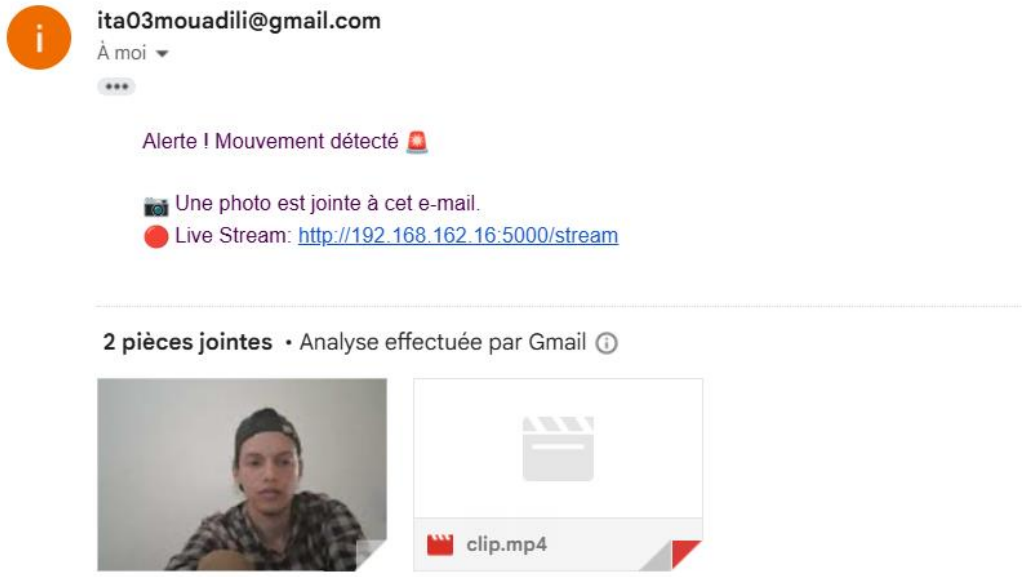


Figure 6 : Test d'envoi de mail

8.2 Validation de l'intégration complète

Enfin, un test complet a été réalisé en simulant un mouvement :

- Le capteur PIR a détecté l'intrus.
- Le serveur a exécuté le script Python : Photo prise, Vidéo enregistrée, E-mail envoyé et le flux accessible en parallèle.

✓ **Résultat final :** Le **système de surveillance est fonctionnel**, réactif, et prêt à être utilisé comme dispositif d'alerte et de contrôle en temps réel.

8.3 Remarques complémentaires

- Le **verrou (threading.Lock)** a permis d'éviter toute concurrence problématique entre la capture vidéo et **l'envoi de fichiers**.
- Des **tests de charge** pourraient être envisagés pour voir la stabilité sur plusieurs heures.
- Un test avec **connexion Internet limitée** montre que l'e-mail peut être ralenti mais non bloqué.

8.4 Conclusion

Ce chapitre a permis de **valider l'efficacité et la cohérence de l'ensemble du système**. Grâce aux différents tests réalisés, nous confirmons que les modules fonctionnent de manière intégrée et harmonieuse, assurant à la fois **la surveillance**, **la détection** et **l'alerte à distance**.

9. Conclusion et Perspectives

9.1 Bilan du Projet

Ce projet a permis de concevoir et de mettre en œuvre un **système de surveillance intelligent** basé sur la détection de mouvements, l'envoi d'alertes par e-mail, et la visualisation en direct via un navigateur web. Grâce à la combinaison de plusieurs technologies (Arduino Yun, Python, Flask, OpenCV, SMTP), nous avons pu :

- Capturer des images et vidéos en temps réel à partir d'un PC.
- Détecter les mouvements via un capteur PIR.
- Déclencher automatiquement une alerte avec pièce jointe (photo, vidéo) et lien de surveillance à distance.
- Garantir une surveillance continue grâce au streaming web intégré.

Ce système offre ainsi une solution **simple, efficace et autonome** pour la sécurité de locaux ou d'espaces personnels.

9.2 Perspectives

Ce travail peut être amélioré et élargi par plusieurs pistes d'évolution :

- **Ajout d'un stockage cloud**
  Les images et vidéos pourraient être stockées automatiquement sur Google Drive, Dropbox ou un serveur FTP sécurisé pour une consultation ultérieure.

- **Reconnaissance d'intrus**
  Intégration de la reconnaissance faciale ou d'objets suspects via un modèle de vision par ordinateur (ex. : YOLO, FaceNet).

- **Machine Learning**
  model.predict(mouvement) *# Filtrage intelligent des faux positifs*

- **Interface mobile**
  Développement d'une application mobile pour recevoir les alertes sous forme de notifications push, en plus de l'e-mail.

- **Historique des événements**
  Mise en place d'une base de données légère (ex : SQLite) pour archiver les alertes avec horodatage et captures associées.

- **Mode nuit & infrarouge**
  Utilisation d'une caméra compatible IR pour une meilleure détection en environnement sombre.

- **Intégration Domotique**

Pour rendre le système plus intelligent et interconnecté, il peut être intégré dans un écosystème de **maison connectée (domotique)** :

- **Interfaçage avec Home Assistant** via des appels API REST, permettant de centraliser la gestion du système dans une plateforme domotique open source.

- **Compatibilité avec Google Home ou Amazon Alexa** en passant par **IFTTT**, pour déclencher des actions vocalement ou automatiquement.

- **Création de scénarios personnalisés** via des fichiers YAML dans Home Assistant, par exemple :

```
# Exemple de scénario Home Assistant
automation:
  - alias: "Alerte Mouvement"
```

```
trigger:
  platform: state
  entity_id: binary_sensor.mouvement_detecte
  to: 'on'
action:
  - service: light.turn_on
    entity_id: light.salon
  - service: notify.mobile_app_monsmartphone
    data:
      message: "Mouvement détecté dans le salon !"
```

9.3 Bilan personnel

Ce projet m'a permis d'approfondir mes compétences en **programmation Python**, en **communication réseau (HTTP, SMTP)**, ainsi qu'en **interaction Arduino-Python**. Il a aussi illustré l'importance de la **synchronisation entre plusieurs composants** pour créer un système cohérent et réactif.

9.4 Applications Industrielles

Le système conçu peut être adapté à des cas d'usage professionnels variés, notamment dans les secteurs de la logistique, de la sécurité et de la gestion de bâtiments intelligents.

L'une des principales applications est la **surveillance d'entrepôts ou de zones de stockage**. En cas d'intrusion, une alerte est déclenchée en temps réel, accompagnée d'une photo et d'une courte vidéo, envoyée automatiquement au responsable de la sécurité. Ce type de déploiement permet une réaction rapide, même à distance, grâce à l'intégration de liens de visualisation directe via **/stream.**

Une autre application envisageable est le **comptage de personnes**, en intégrant des algorithmes de détection et de suivi. Cette fonctionnalité peut alimenter des **statistiques de fréquentation** utiles dans des commerces, musées ou bâtiments publics, en s'intégrant à des plateformes de visualisation de données comme **Power BI** ou Grafana via des APIs.

- **Modèle Économique**

Coût	Détail
Matériel	Moins de 50 € (en version basique)
Développement	Environ 10 heures (version optimisée)
ROI potentiel	6 mois (dans un contexte professionnel)

Tableau 5 : Modèle Economique

Ce modèle est particulièrement pertinent pour les TPE/PME souhaitant améliorer leur sécurité ou optimiser la gestion de leurs espaces sans engager de lourds investissements.

9.5 Recommandations Finales

9.5.1 Pour les Débutants

Pour les personnes découvrant ce type de projet, il est recommandé de **commencer avec une version simple**, sans interactions matérielles complexes. Il est possible de simuler l'alerte avec des déclencheurs manuels ou des boutons, en observant les réactions du système (photo, e-mail, vidéo).

L'utilisation de la **LED intégrée (D13)** permet de vérifier le bon fonctionnement du capteur de mouvement sans dépendre d'un périphérique externe.

Enfin, pour simplifier l'envoi d'e-mails, il est conseillé d'utiliser un compte **Gmail**, qui fonctionne aisément avec le protocole SMTP sécurisé et dispose d'une documentation abondante.

9.5.2 Pour les Experts

Les utilisateurs expérimentés peuvent faire évoluer l'architecture vers un système plus robuste et modulaire. Par exemple, l'adoption de **Mosquitto (MQTT)** couplé à **Node-RED** permet une orchestration d'événements plus fine et une intégration facilitée dans des systèmes domotiques ou industriels.

Il est aussi possible d'ajouter un **système de journalisation centralisé**, à l'aide d'une stack ELK (Elasticsearch, Logstash, Kibana), pour un suivi détaillé des événements et des performances.

Enfin, dans une perspective de cybersécurité, il est vivement recommandé de **mettre en place une authentification forte**, basée sur des **certificats TLS** pour les échanges entre modules ou avec les API externes.

10. Annexes Techniques

Cette section regroupe les détails techniques utiles à la compréhension, la reproduction et l'extension du projet. Elle permet également de centraliser les éléments annexes comme les configurations, extraits de code, et ressources supplémentaires.

10.1 Schéma de câblage

- **Capteur PIR** :
  - VCC → 5V (ou 3.3V selon modèle)
  - OUT → D2 (ou autre pin numérique)
  - GND → GND
- **LED (pour test)** :
  - Anode → D13 (ou autre pin)
  - Cathode → Résistance de 220Ω → GND
- **Buzzer**
  - Anode → D8
  - GND → GND

10.2 Configuration réseau

- Le script Python détecte automatiquement l'adresse IP Wi-Fi locale pour générer dynamiquement l'URL de streaming (/stream).
- Port utilisé par le serveur Flask : **5000**
- Exemple d'URL de stream dans un réseau local :

http://192.168.1.16:5000/stream

10.3 Fichiers principaux

Nom de fichier	Description
app.py	Script principal Flask (stream + alerte)
stream.html	Interface HTML pour visualisation du flux
photo_lowres.jpg	Photo temporairement enregistrée
clip.mp4	Vidéo temporairement enregistrée

Tableau 6 : Fichiers principaux

10.4 Extraits de code utiles

10.4.1 Enregistrement d'une vidéo depuis la webcam

```
def record_video(output_path='clip.mp4', duration=5, fps=20):
    ...
    while time.time() - start_time < duration:
        ret, frame = camera.read()
        if ret:
            out.write(frame)
```

10.4.2 Génération de lien dynamique pour le streaming

```
def get_wifi_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(('10.255.255.255', 1))
    return s.getsockname()[0]
```

10.4.3 Ressources et outils utilisés

- **Langages** : Python 3, HTML (Flask templating)
- **Librairies** : OpenCV, Flask, smtplib, threading, email.mime
- **Matériel** : Arduino Yun, capteur PIR, webcam intégrée ou USB
- **Outils de test** : navigateur, serveur local, Gmail (SMTP)

Ce projet démontre qu'avec :

- **Un budget serré** (< 100€)
- **Des composants accessibles**
- **Une approche méthodique**

Il est possible de développer un système industriellement viable. La suite naturelle serait :

- ✓ **Passage à l'échelle avec une flotte de capteurs**
- ✓ **Intégration à une architecture IoT professionnelle**

(Documentation complète et codes disponibles sur GitHub)

Bibliographie

Arduino. *Arduino Yun Documentation*. Disponible sur : https://www.arduino.cc/en/Guide/ArduinoYun

OpenCV Team. *OpenCV-Python Tutorials Documentation*. OpenCV.org. Disponible sur : https://docs.opencv.org/

Flask Documentation. *Flask – Web Development, One Drop at a Time*. Disponible sur : https://flask.palletsprojects.com/

Python Software Foundation. *Python 3.10 Documentation*. Disponible sur : https://docs.python.org/3/

Simon Monk. *Programming Arduino: Getting Started with Sketches*. McGraw-Hill Education, 2011.

Stack Overflowpour la résolution de problèmes ponctuels liés à Flask, OpenCV et l'envoi d'e-mails en Python.

Dépôt GitHub du projet. *Surveillance System using Arduino Yun and Python Server*. Disponible sur : https://github.com/momonaim/mst-rsi-pfa-motion-alert-system