Sistem de supraveghere utilizand o camera Web

De Sipos Bogdan

Bogdan Sipos
UNIVERSITATEA TEHNICA CLUJ NAPOCA

1. Introducere

1.1 Context

Scopul acestui proiect este de a crea un sistem de supraveghere utilizand o camera Web. Pentru aceast proiect am realizat un site care preia filmarea live a camerei web si identifica miscarile nespteapte pe care apoi le va semnaliza atat vizual cat si sonor. Astfel, daca in incaperea care ar trebui sa fie goala (nu trebuie sa fie nimic in miscare) apare un obiect in miscare sau un obiect isi schimba pozitia atunci utilizatorul va fi alertat.

1.2 Obiective

Site-ul o sa fie realizat in Python si utilizand framework-ul Flask. Proiectul final va trebui sa indeplineasca urmatoarele cerinte: sa permita vizualizare pe Internet si sa alerteze sonor in cazul unei situatii neasteptate. In plus, proiectul va contine: un meniu de selectie a intensitatii alartei sonore, un meniu de setare a sensibilitatii alarmei si un meniu de activare sau dezactivarea a modului de securitate.

2. Studiu Bibliografic

OpenCV (Open Source Computer Vision Library) este o bibliotecă open-source dedicată viziunii computaționale și procesării imaginilor. Aceasta oferă o gamă largă de instrumente pentru analiza și manipularea imaginilor și videoclipurilor, fiind utilizată în domenii precum recunoașterea facială, detectarea mișcării, urmărirea obiectelor, reconstruirea 3D și multe altele. Aceasta se remarca prin procesarea în timp real(este optimizată pentru sarcini care necesită viteză mare, cum ar fi procesarea cadrelor video) si prin funcționalitățile extinse(include algoritmi pentru recunoaștere de imagini, machine learning, analiză video, etc.).

Flask este un framework web minimalist pentru limbajul de programare Python, utilizat pentru a dezvolta aplicații web și API-uri. Este cunoscut pentru simplitatea sa și flexibilitatea în personalizarea aplicațiilor. Aceasta se remarca prin faptul ca e ușor de utilizat (este simplu și intuitiv pentru dezvoltatori, oferind doar elementele de bază necesare pentru a construi o aplicație web), prin extensibilitate (poți adăuga funcționalități suplimentare prin instalarea unor extensii pentru baze de date, autentificare, caching etc.) si flexibilitate (nu impune structuri rigide sau complexe, permițând dezvoltatorului să decidă cum să organizeze aplicația).

Am ales sa folosesc Flask pentru:

- O interfață Web Ușoară: Mi-a permis sa creez o interfață web simplă și personalizabilă unde utilizatorii pot interacționa cu aplicația. În cazul meu, l-am folosit pentru a rula serverul web și pentru a permite controlul parametrilor sistemului de supraveghere(frecvența sunetului, sensibilitatea și activarea sistemului de securitate).
- Gestionarea Cererilor HTTP: Deoarece este bun pentru gestionarea cerilor GET si POST din partea utilizatorilor. L-am folosit pentru a colege datele din formulare web pentru setarea frecventei sunetului, sensibilitate și activarea/dezactivarea sistemului de securitate.
- Streaming Video în Browser: Acesta impreuna cu Response, l-am folosit pentru a transmite fluxul video in timp real (capturat cu OpenCV) la un browser web.

Am ales să folosesc OpenCV pentru:

- Procesare Video în Timp Real: Acesta este optimizat pentru procesarea rapidă și eficientă a imaginilor și videoclipurilor în timp real. L-am folosit pentru a accesa camera și a prelucra fluxul video în mod continuu, ceea ce este esențial pentru un sistem de supraveghere capabil să detecteze mișcarea fără întârzieri semnificative.
- Detectarea Mișcării: Deoarece oferă funcții avansate pentru analiza diferențelor între cadre consecutive și detectarea mișcării. Am folosit funcții precum `cv2.absdiff()`, `cv2.cvtColor()`, `cv2.GaussianBlur()` și `cv2.findContours()` pentru a compara cadrele video și a evidenția regiunile cu mișcare.
- Flexibilitate și Performanță: Acesta mi-a oferit flexibilitatea necesară pentru a ajusta algoritmul de detectare a mișcării, incluzând setarea sensibilității și eliminarea zgomotului din imagini. Datorită optimizării sale pentru performanță în timp real, mi-a permis sa obțin un sistem de supraveghere capabil să declanșeze o alarmă rapid atunci când detectează mișcare.

3. Analiza:

Acest capitol evaluează aspectele principale ale sistemului de detectare a mișcării și alarmei, incluzând cerințele utilizatorilor, alegerile de design și provocările potențiale implicate în implementarea fiecărei componente. Analiza examinează designul sensibilității, frecvenței alarmei și setărilor de securitate, precum și modul în care acești factori interacționează pentru a produce un sistem receptiv și personalizabil.

3.1 Propunerea Proiectului

Scopul sistemului de detectare a mișcării este de a permite supravegherea în timp real printr-un flux video live și de a declanșa alarma pe baza detectării mișcării. Sistemul folosește controlul web cu procesarea de imagini pentru a îndeplini următoarele cerințe principale:

1. **Monitorizare Video Live**: Capturarea și afișarea unui flux video continuu de la cameră, permițând utilizatorilor să monitorizeze mișcarea.
2. **Sistem de Alarmă Configurabil**: Declanșarea unei alarme personalizabile la detectarea mișcării, cu setarea frecvenței alertei.
3. **Ajustarea Sensibilității**: Permite ajustarea sensibilității detectării mișcării pentru a răspunde nevoilor de securitate diferite.
4. **Meniu de activare/dezactivare securitate**: Permite selectarea opririi sau pornirii modului de securitate.
5. **Interfață pentru Configurare**: Oferă o interfață web unde utilizatorii pot ajusta setările pentru frecvența alarmei, sensibilitate și modul de securitate în timp real.

3.2 Analiza Componentelor

Această secțiune analizează componentele principale ale sistemului de detectare a mișcării și raționamentul din spatele alegerilor de design, concentrându-se pe rolurile fiecărui modul și constrângerile pe care le adresează.

3.2.1 Flux Video Live și Procesare

Pentru a oferi un flux video în timp real, sistemul folosește OpenCV pentru captarea și procesarea cadrelor, în timp ce Flask servește cadrele procesate ca un flux HTTP continuu.

- **Cerințe de Timp Real**: Sistemul trebuie să proceseze cadrele rapid pentru a menține un flux video fluid, impunând limitări asupra complexității metodelor de procesare a imaginii.
- **Formatul Video**: Fluxul video este transmis sub formă de cadre JPEG, asigurând compatibilitatea cu majoritatea browserelor și minimizând utilizarea lățimii de bandă.
- **Experiența Utilizatorului**: Fluxul video este afișat pe o pagină web, permițând utilizatorilor să vizualizeze și să controleze setările de la distanță fără software specializat.

3.2.2 Logica Detectării Mișcării

Algoritmul de detectare a mișcării se bazează pe analiza diferenței dintre cadrele consecutive.

- **Diferențierea Cadrelor**: Calculând diferența absolută între cadre, sistemul identifică schimbările care semnalează mișcarea.
- **Reducerea Zgomotului**: Se aplică estomparea Gaussiană pentru a reduce alarmele false provocate de variații minore ale luminii.
- **Pragul și Diluarea**: Imaginea binară rezultată evidențiază zonele de mișcare semnificative, iar diluarea extinde aceste regiuni pentru o detectare mai precisă a contururilor.

3.2.3 Controlul Sensibilității și Securității

Sistemul include opțiuni de ajustare a sensibilității și a nivelului de securitate, permițând utilizatorilor să ajusteze pragurile de detectare în funcție de mediu.

- **Controlul Sensibilității**: selected_sensibility definește pragul de intensitate al pixelilor peste care mișcarea este considerată semnificativă.
- **Nivelul de Securitate**: selected_security ajustează pragul ariei conturului, controlând cât de mare trebuie să fie o mișcare pentru a declanșa o alarmă.

3.2.4 Mecanismul de Alarmă și Controlul Frecvenței

Mecanismul de alarmă este proiectat să alerteze utilizatorii la detectarea mișcării printr-un sunet personalizabil.

- **Personalizarea Frecvenței**: selected_frequency permite utilizatorilor să seteze frecvența alarmei.
- **Execuție Concurentă cu Threading**: Pentru a evita blocarea, funcția de alarmă rulează într-un thread separat, asigurând continuarea procesării video fără întreruperi.

3.3 Constrângeri și Provocări în Design

Sistemul de detectare a mișcării și alarmă întâmpină mai multe constrângeri și provocări din cauza cerințelor de procesare în timp real și a nevoii de configurabilitate.

3.3.1 Procesare în Timp Real și Întârzieri

Menținerea unui flux video fluid și receptiv este esențială pentru experiența utilizatorului. Acest lucru impune constrângeri asupra complexității tehnicilor de procesare a imaginii, deoarece metodele computațional intensive pot introduce întârzieri sau pot reduce acuratețea detectării.

- **Provocare**: Este esențial să se echilibreze viteza procesării cadrelor cu acuratețea detectării. Utilizarea unor tehnici ușoare, precum diferențierea cadrelor și estomparea Gaussiană, ajută la menținerea performanței, dar poate reduce precizia în comparație cu metodele mai avansate.
- **Soluție**: Prin optimizarea unor parametri precum rezoluția cadrelor, intensitatea pragului și frecvența analizei, sistemul poate menține o performanță rezonabilă asigurând totodată o detectare precisă.

3.3.2 Factori de Mediu și Alarme False

Factori de mediu, cum ar fi iluminarea, activitatea de fundal și unghiul camerei, pot influența acuratețea detectării mișcării, ducând la alarme false sau detecții ratate.

- **Provocare**: Variațiile în nivelurile de lumină sau activitatea de fundal (de exemplu, mașini care trec sau ramuri în mișcare) pot fi identificate incorect ca mișcare.
- **Soluție**: Sistemul abordează această problemă permițând utilizatorilor să ajusteze sensibilitatea și nivelul de securitate, reducând astfel sensibilitatea la schimbări minore sau mișcări de fundal.

3.3.3 Multi-Threading și Gestionarea Resurselor

Utilizarea threading-ului pentru mecanismul de alarmă previne blocajele, însă gestionarea resurselor și evitarea conflictelor între threaduri poate fi riscanta.

- **Provocare**: Rularea mai multor threaduri, mai ales cu variabile globale, poate duce la probleme de sincronizare sau conflicte între procese (race conditions).
- **Soluție**: Prin structurarea atentă a flagului movement și folosirea unui contor (movement_detected) pentru a gestiona declanșarea alarmei, sistemul minimizează conflictele și asigură că doar o singură alarmă este declanșată per eveniment de mișcare detectat.

4. Design

Acest capitol detaliază structura și funcționalitatea principalelor componente ale sistemului de detectare a mișcării și alarmei. Fiecare modul - rutele Flask, logica de detectare a mișcării și mecanismul de declanșare a alarmei - este explicat împreună cu parametrii și condițiile asociate acestora.

4.1 Serverul Flask și Designul Rutelor

Serverul Flask este elementul central al aplicației, oferind atât interfața utilizatorului, cât și punctele de configurare. Principalele rute și funcționalitățile acestora sunt descrise mai jos:

4.1.1 Ruta / (Acasă)

- **Scop**: Ruta rădăcină (/) servește interfața HTML principală (index.html), permițând utilizatorilor să ajusteze parametrii sistemului, cum ar fi sensibilitatea, frecvența alarmei și nivelul de securitate.
- **Implementare**: Funcția index() folosește render_template pentru a încărca index.html, afișând setările actuale și oferind utilizatorilor opțiuni de configurare.

4.1.2 Ruta /set_frequency

- **Scop**: Această rută actualizează parametrul selected_frequency, care controlează frecvența sonorului de alarmă.
- **Implementare**: Când se primește o cerere POST cu valoarea frequency, funcția parsează și setează selected_frequency, afișând frecvența actualizată în consolă.

4.1.3 Ruta /set_security

- **Scop**: Această rută ajustează nivelul de securitate (selected_security), care influențează algoritmul de detectare a mișcării, aplicând un prag de sensibilitate predefinit sau modificând analiza contururilor.
- **Implementare**: Când se primește o cerere POST, variabila selected_security este actualizată și utilizată ulterior în algoritmul de detectare a mișcării pentru a aplica diferite praguri de detectare.

4.1.4 Ruta /set_sensibility

- **Scop**: Permite utilizatorului să seteze selected_sensibility, care influențează direct răspunsul sistemului la mișcare prin ajustarea pragului pentru detectarea mișcării.
- **Implementare**: Funcția parsează și setează selected_sensibility din datele formularului și înregistrează modificarea în consolă.

4.1.5 Ruta /liveFeed

- **Scop**: Această rută furnizează un flux video continuu apelând video_player(), care procesează cadre în timp real pentru detectarea mișcării.

- **Implementare**: Folosind Response și un tip MIME de multipart/x-mixed-replace, această rută transmite continuu cadre JPEG-encoded, creând un flux video live.

4.2 Algoritmul de Detectare a Mișcării

Funcția video_player() încorporează logica de detectare a mișcării. Acest modul procesează cadrele de la camera web pentru a detecta mișcarea prin compararea cadrelor consecutive. Principalii pași și parametrii implicați în detectarea mișcării sunt descriși mai jos:

4.2.1 Captarea și Preprocesarea Cadrelor

1. **Captarea Cadrelor**: Două cadre consecutive, frame1 și frame2, sunt capturate pentru comparație.
2. **Calcularea Diferenței**:
    - Se calculează diferența absolută între frame1 și frame2, rezultând o imagine diff care evidențiază modificările.
    - Se aplică o conversie în tonuri de gri și estompare Gaussiană pentru a reduce zgomotul și a îmbunătăți precizia detectării contururilor.

4.2.2 Pragul și Diluarea

- **Pragul Binar**: Pragul este aplicat pe cadrul estompat pentru a crea o imagine binară, unde pixelii de intensitate mare reprezintă zone de mișcare.
- **Diluarea**: Diluarea extinde regiunile de intensitate mare, facilitând detectarea și procesarea contururilor.

4.2.3 Analiza Contururilor

- **Detecția Contururilor**: Imaginea binară diluată este analizată pentru a detecta contururi care reprezintă zone de mișcare.
- **Caseta de Delimitare și Etichetarea**: Pentru fiecare contur detectat care depășește o anumită arie (de exemplu, 800 px), se desenează o casetă de delimitare pe frame1, iar eticheta „Movement” este adăugată.

4.2.4 Logica Declanșării Mișcării

Suma pragului și aria conturului sunt verificate față de selected_sensibility. Dacă suma pixelilor cu prag depășește această valoare, mișcarea este confirmată, iar contorul movement_detected crește. Odată ce acest contor depășește selected_sensibility, alarma este declanșată.

4.3 Mecanismul de Declanșare a Alarmei

Mecanismul de alarmă este proiectat pentru a alerta utilizatorii la detectarea mișcării printr-un beep sonor, cu setări de frecvență personalizabile.

4.3.1 Condițiile Alarmei

Sistemul de alarmă se bazează pe flagul movement și contorul movement_detected:

- **Flagul de Mișcare**: Setat inițial la True, acest flag împiedică declanșarea repetată a alarmei în cazul unei mișcări prelungite.
- **Contorul de Detectare**: Odată ce movement_detected depășește selected_sensibility, funcția alarm() este declanșată, iar flagul movement este resetat pentru a preveni alerte multiple până când mișcarea încetează.

4.3.2 Controlul Frecvenței

- **Frecvența Definibilă de Utilizator**: Variabila selected_frequency, setată prin /set_frequency, determină frecvența sunetului de alarmă.
- **Execuția Concurentă cu Threading**: Pentru a evita blocarea buclei principale a aplicației, funcția alarm() rulează într-un thread separat, permițând continuarea procesării video fără întreruperi.

5.Implementare:

Acest capitol descrie componentele principale ale sistemului de detectare a mișcării și alarmei, incluzând funcționalitatea rutelor Flask, logica de detectare a mișcării și declanșarea alarmei. Fiecare funcție este prezentată cu fragmente de cod și o explicație despre rolul acesteia în funcționarea generală a sistemului.

5.1 Rutele Flask

Aplicația web Flask este structurată în jurul unui set de rute care permit configurarea și monitorizarea în timp real. Fiecare rută gestionează o funcție specifică, cum ar fi setarea parametrilor sau transmiterea fluxului video.

5.1.1 Ruta Principală (/)

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

Ruta principală servește interfața principală unde utilizatorii pot ajusta frecvența alarmei, sensibilitatea și nivelul de securitate.

- **Explicație**: Funcția index() folosește render_template din Flask pentru a încărca fișierul index.html, oferind o interfață pentru controlul parametrilor sistemului. Acest fișier HTML include elemente de formă pentru trimiterea valorilor de frecvență, securitate și sensibilitate.

5.1.2 Ruta pentru Setarea Frecvenței (/set_frequency)

```python
@app.route( rule: '/set_frequency', methods=['POST'])
def set_frequency():
    global selected_frequency
    selected_frequency = int(request.form['frequency'])
    print(f"Selected frequency: {selected_frequency} Hz")
    return render_template('index.html')
```

Această rută setează frecvența alarmei, permițând utilizatorilor să personalizeze tonul alertelor.

- **Explicație**: Când este primit un request POST cu o valoare pentru frequency, variabila globală selected_frequency este actualizată. Funcția reîncarcă apoi interfața index.html, reflectând noua setare a frecvenței.

5.1.3 Ruta pentru Setarea Securității (/set_security)

```python
@app.route( rule: '/set_security', methods=['POST'])
def set_security():
    global selected_security
    selected_security = int(request.form['security'])
    print(f"Status security: {selected_security}")
    return render_template('index.html')
```

Această rută configurează nivelul de securitate, care influențează precizia detectării mișcării.

- **Explicație**: Cererea POST actualizează variabila globală selected_security, modificând modul în care logica de detectare a mișcării filtrează mișcările mai mici sau nesemnificative.

5.1.4 Ruta pentru Setarea Sensibilității (/set_sensibility)

```python
@app.route( rule: '/set_sensibility', methods=['POST'])
def set_sensibility():
    global selected_sensibility
    selected_sensibility = int(request.form['sensibility'])
    print(f"Selected sensibility: {selected_sensibility}")
    return render_template('index.html')
```

Această rută setează pragul de sensibilitate pentru detectarea mișcării.

- **Explicație**: Valoarea cererii POST actualizează selected_sensibility, controlând pragul de detectare a mișcării. O valoare mai mică crește sensibilitatea, permițând declanșarea alarmei pentru mișcări mai mici.

5.1.5 Ruta pentru Fluxul Video (/liveFeed)

Această rută transmite continuu cadre video către browser pentru monitorizare în timp real.

```python
@app.route('/liveFeed')  1 usage
def live_feed():
    return Response(video_player(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

- **Explicație**: Funcția live_feed() transmite cadrele returnate de video_player() către browser în timp real. Utilizează tipul MIME multipart/x-mixed-replace pentru a actualiza continuu fluxul video.

5.2 Logica de Detectare a Mișcării

Nucleul sistemului de detectare a mișcării se află în funcția video_player(), care procesează cadrele de la camera web pentru a detecta mișcări.

```python
def video_player():  1 usage
    global movement_detected, movement
    ret, frame1 = camera.read()
    ret, frame2 = camera.read()

    while True:
        diff = cv2.absdiff(frame1, frame2)
        gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, ksize: (19, 19), sigmaX: 0)
        _, thresh = cv2.threshold(blur, thresh: 30, maxval: 255, cv2.THRESH_BINARY)
        dilated = cv2.dilate(thresh, kernel: None, iterations=3)
        contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        if selected_security == 1:
            for contour in contours:
                (x, y, w, h) = cv2.boundingRect(contour)
                if cv2.contourArea(contour) < 800: #900
                    continue
                cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame1, text: "Movement", org: (10, 20), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 255, 0), thickness: 3)

            if thresh.sum() > 300:
                movement_detected += 1
            else:
                if movement_detected > 0:
                    movement_detected -= 1

            if movement_detected > selected_sensibility:
                if movement:
                    threading.Thread(target=alarm).start()
```

```python
            if movement_detected > selected_sensibility:
                if movement:
                    threading.Thread(target=alarm).start()

                if movement_detected > 0:
                    movement_detected -= 1

        ret, jpeg = cv2.imencode( ext: '.jpg', frame1)
        frame1 = frame2
        ret, frame2 = camera.read()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')
```

5.2.1 Pașii de procesare a cadrelor

1. **Diferențierea Cadrelor**:

```python
diff = cv2.absdiff(frame1, frame2)
gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, ksize: (19, 19), sigmaX: 0)
```

Diferența absolută între frame1 și frame2 evidențiază schimbările în intensitatea pixelilor. Această imagine diferențiată este convertită în tonuri de gri și

estompată cu un filtru Gaussian pentru a reduce zgomotul și fluctuațiile mici de pixeli.

2. **Pragul și Diluarea**:

```
_, thresh = cv2.threshold(blur, thresh: 30, maxval: 255, cv2.THRESH_BINARY)
dilated = cv2.dilate(thresh, kernel: None, iterations=3)
```

Pragul creează o imagine binară în care pixelii de intensitate ridicată reprezintă zone de mișcare detectată. Diluarea extinde aceste regiuni, asigurându-se că contururile reprezintă mișcări semnificative.

3. **Analiza Contururilor și Detectarea**:

```
contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
if selected_security == 1:
    for contour in contours:
        (x, y, w, h) = cv2.boundingRect(contour)
        if cv2.contourArea(contour) < 800: #900
            continue
        cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame1, text: "Movement", org: (10, 20), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 255, 0), thickness: 3)
```

findContours() identifică contururile în imaginea diluată, reprezentând zone de mișcare. Fiecare contur are o casetă de delimitare desenată pe frame1 dacă suprafața conturului depășește pragul setat de selected_security, reducând astfel alarmele false.

4. **Pragul de Mișcare**:

```
if thresh.sum() > 300:
    movement_detected += 1
else:
    if movement_detected > 0:
        movement_detected -= 1
```

Suma pixelilor binari din imaginea cu prag este verificată față de un prag prestabilit. Dacă suma depășește acest prag, indică mișcare semnificativă, incrementând contorul movement_detected. Dacă nu, contorul este decrementat treptat, asigurând că doar mișcarea persistentă declanșează alarma.

5.3 Alarmă și Threading

Funcția alarm() gestionează logica alarmei, rulând într-un thread separat pentru a evita blocarea buclei principale de detectare.

5.3.1 Implementarea Funcției de Alarmă

```
def alarm():  1 usage
    global movement
    movement = False
    if not movement:
        winsound.Beep(selected_frequency,  duration: 1000)
        print(f"Movement detected, beep at {selected_frequency} Hz")
    movement = True
```

Când este detectată mișcarea și movement_detected depășește selected_sensibility, funcția alarm() este invocată într-un thread separat. Funcția winsound.Beep() emite un beep la frecvența selected_frequency. Flagul movement este setat la True pentru a preveni declanșarea continuă în timpul mișcărilor prelungite.

5.3.2 Declanșarea Threadului pentru Alarmă

În funcția video_player():

```
if movement_detected > selected_sensibility:
    if movement:
        threading.Thread(target=alarm).start()

    if movement_detected > 0:
        movement_detected -= 1
```

Când movement_detected depășește selected_sensibility, este creat un nou thread pentru funcția alarm() dacă movement este True, asigurându-se că alarma este declanșată o singură dată pentru fiecare eveniment de mișcare detectat.

5.4 Transmiterea Cadrelor Video

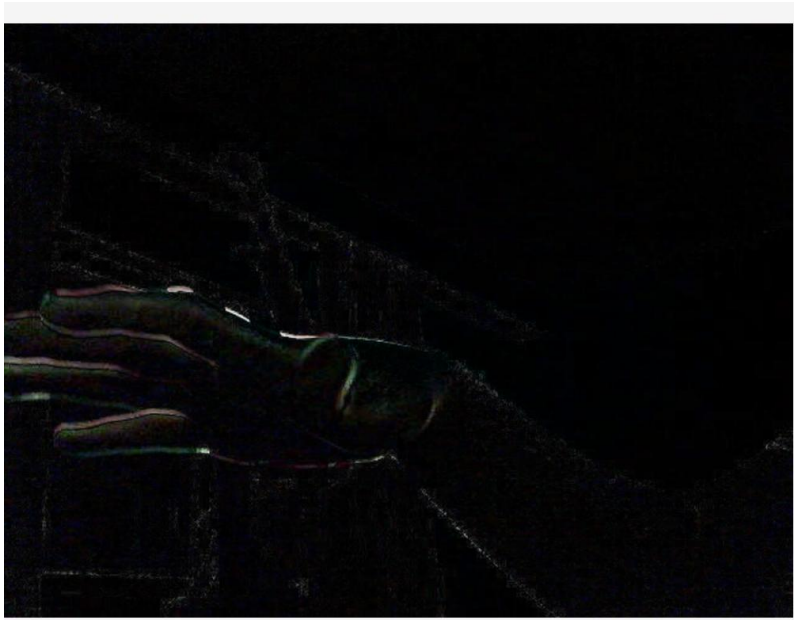
Ultimul pas în video_player() este codificarea cadrului procesat și transmiterea acestuia către fluxul video.

```
ret, jpeg = cv2.imencode( ext: '.jpg', frame1)
frame1 = frame2
ret, frame2 = camera.read()

yield (b'--frame\r\n'
       b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')
```

cv2.imencode() comprimă frame1 în format JPEG, care este apoi transmis ca o secvență de octeți. Acest lucru permite clientului să afișeze cadrul procesat ca parte a fluxului video live.

6.Testare si validare:



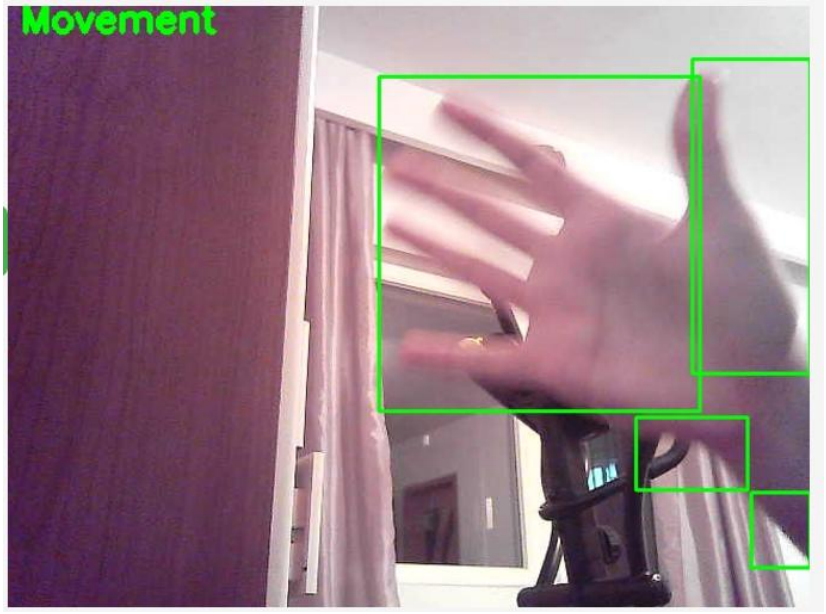
Prima data testam daca functia absdiff si observam ca face diferenta absoluta dintre ambele imaginile, adica evidentiaza diferentele dintre cele doua.

Functia grayscalling face ca acum imaginile evidentiate sa nu mai fie in culori si doar in culori intre alb si negru.

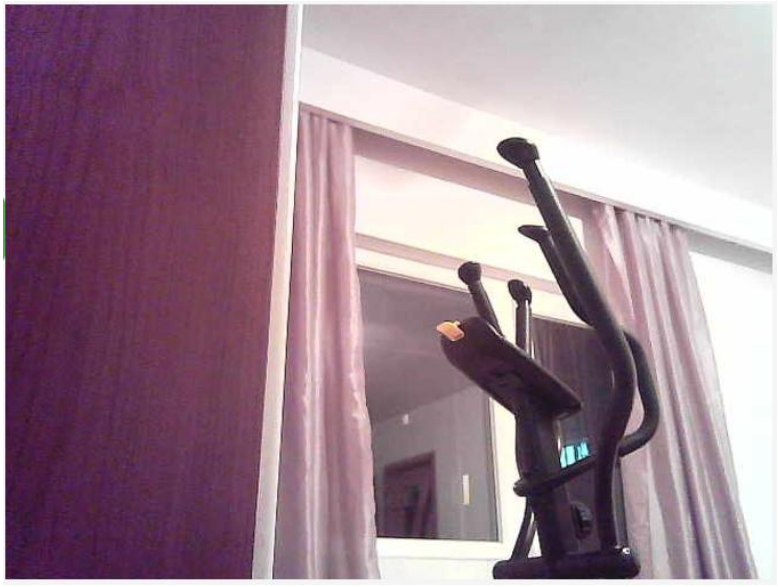
Functia GaussianBlur are rolul de a netezi regiunea blurand-o. Acest lucru duce la reducerea zgomotului din imagine si ne asigura ca doar miscarile clare sunt identificate.

Functia threshold coloreaza cu alb toti pixelii care au valoare mai mare de 30 si cu negru restul astfel miscarile vor fi reprezentate cu alb.

Functia dilate are rolul de a mari zona de miscare si de a colora regiunile mici acolo unde a avut loc miscare.



Dupa ce am folosit functia findContours observam ca programul incadreaza destul de corect obiectele care se afla in miscare si afiseaza cand e miscare mesajul movement pe imagine. Cand nu se intampla nimic va afisa doar imaginea neprelucrata.



7.Concluzie:

Prin implementarea acestui sistem de detectare a mișcării și alarmă, am demonstrat o abordare eficientă și configurabilă de monitorizare a securității, care îmbină procesarea în timp real cu configurabilitatea printr-o interfață ușor de utilizat. Sistemul oferă o bază solidă pentru extensii ulterioare și poate fi adaptat la o gamă largă de aplicații și medii, susținând obiectivul de a oferi un instrument fiabil și personalizabil pentru securitatea locuințelor sau a spațiilor comerciale.

8.Bibliografie:

1. Documentatia oferita de OpenCv: https://docs.opencv.org/4.10.0/
2. OpenCV Python Tutorial For Beginners 24 - Motion Detection and Tracking Using Opencv Contours: https://www.youtube.com/watch?v=MkcUgPhOlP8&t=718s&ab_channel=ProgrammingKnowledge
3. motion detection using python || python opencv project: https://www.youtube.com/watch?v=oxmZ9zczptg&ab_channel=Iknowpython
4. Make A Security Camera With Python: https://www.youtube.com/watch?v=Exic9E5rNok&t=1880s&ab_channel=TechWithTi

5. Video Streaming Using Webcam In Flask Web Framework: https://www.youtube.com/watch?v=vF9QRJwJXJk&ab_channel=KrishNaik