Web Camera Overwatch and Alert System

Student: Galbaza Alexandru-Mihai

Structure of Computer Systems Project
Technical University of Cluj-Napoca

1 Introduction

1.1 Context

The project is designed to detect unexpected motion in a specific environment, such as an empty room that should remain undisturbed. When motion is detected, an alert system is triggered to notify the user of the event. The system incorporates remote monitoring capabilities, allowing real-time visualization of the environment via a web interface. This can be used in security systems, where the user can monitor a room, such as a warehouse or office, to detect trespassers or unwanted activity.

1.2 Objectives

The main goals of this project are:

- To detect unexpected motion in an image feed.
- To alert the user when motion is detected using sound or other notification methods.
- To allow remote visualization of the monitored environment through a web interface.

2 Bibliographic Research

2.1 OpenCV and Motion Detection Algorithms

For the motion detection aspect of this project, OpenCV was selected as the core library due to its wide use in computer vision applications. OpenCV provides various image processing techniques such as:

- **Background Subtraction**
  What It Is: This method compares each new frame of a video to a background image. By spotting the differences, it detects motion.
  1. Model the Background: Take an average of several frames to create a background model.
  2. Compare Frames: Subtract each new frame from this model.
  3. Spot Differences: Create a black-and-white image where white indicates motion.
  4. Clean Up: Use techniques to reduce noise and fill in gaps.
- **Frame Differencing**
  What It Is: Compares pairs of frames to spot changes.
  1. Grayscale Conversion: Convert each frame to a black-and-white version.

    2. Frame Subtraction: Subtract each frame from the previous one.
    3. Highlight Changes: Convert significant differences into a clearer image of moving objects.
    4. Clean Up: Use noise reduction for a better image.

- **Optical Flow**
  What It Is: Tracks the movement of objects by observing pixel pattern changes between frames.
    1. Calculate Motion: Use algorithms to determine how pixels move from one frame to the next.
    2. Create Motion Vectors: Arrows indicate the direction and speed of movement.
    3. Identify Motion: Highlight significant movements to detect motion.
- **Contour Detection**
  What It Is: Finds the outlines of moving objects.
    1. Spot Differences: Use background subtraction to identify moving parts of the image.
    2. Find Outlines: Detect the contours or edges of these moving parts.
    3. Analyze: Look at these outlines to understand the size and shape of moving objects.
- **Deep Learning Models**
  What It Is: Uses neural networks to recognize motion patterns from large datasets.
    1. Gather Data: Collect examples of moving and non-moving scenes.
    2. Train the Model: Teach a neural network to spot motion using these examples.
    3. Predict Motion: Use the trained model to detect motion in new images or videos.

2.2 Web Development Using Flask

Flask was chosen for its simplicity and ease of use in building web interfaces that can stream live video. This lightweight web framework was essential for creating the remote monitoring feature, allowing users to view the camera feed from any device on the local network.

2.3 Python Libraries for Audio Alerts

The alerting system used in the project was implemented using the `playsound` library, a simple and effective method for triggering sound notifications in Python applications.

3 Bibliography

1. Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media.

2. OpenCV Documentation. (2024). *Motion Analysis and Object Tracking.* Retrieved from `https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html`

3. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python.* O'Reilly Media.

4. Flask Documentation. (2024). *Flask Quickstart.* Retrieved from `https://flask.palletsprojects.com/en/2.x/quickstart/`

5. Python Documentation. (2024). *playsound Library Documentation.* Retrieved from `https://pypi.org/project/playsound/`

4 Analysis

4.1 Project Proposal

The purpose of this project is to develop a motion detection and alerting system using a camera feed, which can detect unexpected movement in a monitored environment. The main goals are to:

- Detect motion in real time by analyzing video frames.
- Trigger a sound or visual alert when significant motion is detected.
- Allow remote monitoring of the video feed over the Internet.

4.2 Project Analysis

To build this system, we considered various methods for motion detection, web streaming, and alerting. The chosen design balances simplicity with functionality, ensuring real-time performance without overloading system resources.

4.2.1 Motion Detection Techniques

Several approaches for motion detection were evaluated:

- **Frame Differencing**: Comparing consecutive frames to detect changes is straightforward and effective for real-time applications, though it may struggle with small movements or sudden lighting changes.
- **Background Subtraction**: This technique maintains a model of the scene's background, detecting motion by identifying deviations. While more accurate, it's computationally heavier and harder to implement in dynamic environments.

Chosen Solution: Frame differencing was selected for its simplicity and efficiency, especially for projects with limited time available.

4.2.2 Web Streaming Options

For remote visualization, Flask was selected to host the video feed. Flask is lightweight, making it a good choice for serving dynamic content without adding unnecessary complexity.

4.2.3 Alert Mechanism

The project aims to notify the user by triggering a sound alert if motion persists for at least 0.5 seconds. This is achieved by monitoring frame differences and timing significant movements.

5 Design

In the Design chapter, we describe the architecture of the system and how each component is structured. This is where we explain the main pieces that make up the motion detection system and how they work together.

5.1 System Architecture

The system comprises three main components:

- **Motion Detection Module**: This module captures and processes frames to identify significant motion using frame differencing and contour analysis.
- **Web Interface Module**: Built using Flask, this module serves the live video feed over the Internet for remote monitoring.
- **Alert Module**: This module triggers an alert when motion is detected consistently for a specified duration.

5.2 Motion Detection Module Design

The motion detection module relies on real-time frame processing to detect movement:

- **Frame Capture**: Frames are captured continuously from the camera.
- **Pre-processing**: Each frame is converted to grayscale and blurred using a Gaussian filter to minimize noise.
- **Frame Differencing**: The absolute difference between consecutive frames is calculated to detect changes. Significant changes trigger the alert mechanism.
- **Contour Detection and Bounding Box**: When motion is detected, contours are drawn around the moving object(s) to indicate where movement occurred. Bounding boxes around contours larger than a certain size (e.g., 500 pixels) highlight the area of significant motion.

5.3 Web Interface Module Design

The web interface module uses Flask to stream the video feed:

- **Live Video Streaming**: Flask continuously captures and serves frames, updating the feed for remote users.
- **Webpage**: A simple HTML page displays the video feed using Flask's Response function with streaming.

5.4 Alert Module Design

This module is responsible for alerting the user when motion is detected:

- **Motion Flag and Timer**: A flag and a timer track motion duration, ensuring that alerts only trigger after sustained motion.
- **Sound Alert**: Once the flag is triggered for 0.5 seconds, an alert sound plays, signaling potential unexpected activity.

6 Implementation

This section covers the detailed steps, code functions, and components involved in implementing the Web Camera Overwatch and Alert System. We'll start with the motion detection logic, integrate it with the Flask server for real-time streaming, and finally cover the asynchronous sound alert feature.

6.1 System Setup and Prerequisites

The system requires the following libraries and tools:

- **Python 3.6+**: Primary programming language for building the application.
- **OpenCV**: Library used for capturing and processing video frames.
- **Flask**: Web framework to host the video stream and provide the web interface.
- **Playsound**: Simple library to handle audio alerts when motion is detected.

Install these libraries with:

```
pip install opencv-python flask playsound
```

6.2 Motion Detection Code

The primary function of this system is to detect motion in a video feed by analyzing changes between consecutive frames. Below is an explanation of the key code segments used in motion detection:

1. **Capturing Video Frames**: The `cv2.VideoCapture()` function is initialized with a specified source (default camera or user-provided URL). Frames are continuously read in a loop, which forms the basis of the live video feed.

   ```
   cap = cv2.VideoCapture(video_source)
   ret, frame = cap.read()
   ```

2. **Frame Differencing for Motion Detection**: Each frame is converted to grayscale and blurred to reduce noise. The difference between consecutive frames is calculated to detect areas of change. If a significant difference is found, it is considered motion.

   ```
   frame_diff = cv2.absdiff(previous_frame, current_gray)
   ```

3. **Contour Detection**: Contours are identified in the binary difference image, and bounding boxes are drawn around these contours to visually highlight the areas of motion in the feed. This is mainly used for debugging purposes and to showcase what part of the motion the program actually "sees".

   ```
   contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
   for contour in contours:
       if cv2.contourArea(contour) < 500:
           continue
       (x, y, w, h) = cv2.boundingRect(contour)
       cv2.rectangle(current_frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
   ```

4. **Persistent Motion Detection**: The system checks if motion continues for a minimum duration (0.5 seconds) before triggering an alert. This helps reduce false positives from brief, insignificant changes.

   ```
   if significant_motion and time.time() - motion_start_time >= 0.5:
       play_sound_async('alert_sound.mp3')
   ```

6.3 Web Interface Using Flask

To make the motion detection feed accessible over a network, we use Flask to serve the video stream.

1. **Web Server Setup**: The `app.py` file initializes the Flask application, setting up the routes to serve the live feed and process user inputs.

   ```
   app = Flask(__name__)
   ```

2. **Dynamic Video Source Input**: The form on the homepage (`index.html`) allows users to enter a custom URL for the video feed. The `/set_feed` route handles this input, updating the `video_source` variable.

   ```
   @app.route('/set_feed', methods=['POST'])
   def set_feed():
       video_source = request.form['feed_url'] if request.form['feed_url'] else 0
   ```

3. **Streaming Video Frames to the Browser**: Frames are encoded as JPEG images and sent to the client in a format the browser can interpret as a continuous stream. This is achieved through the `multipart/x-mixed-replace` MIME type, which keeps updating the displayed image.

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

6.4 Asynchronous Sound Alert

When motion is detected for an extended period, the system plays an alert sound in a non-blocking manner to avoid interruptions in the video processing.

1. **Threading for Asynchronous Sound**: The `play_sound_async` function plays the alert sound in a separate thread. This prevents the main loop from stalling and ensures smooth video streaming.

   ```
   import threading
   from playsound import playsound

   def play_sound_async(sound_file):
       threading.Thread(target=playsound, args=(sound_file,), daemon=True).start()
   ```

7 Testing and Evaluation

To verify the functionality and performance of the Web Camera Overwatch and Alert System, several tests were conducted under various conditions.

7.1 Testing Motion Detection

1. **Accuracy of Motion Detection**: Tested with different object sizes, lighting conditions, and speeds of movement. The system effectively detected and highlighted motion in stable lighting but varied in performance under rapid lighting changes.

2. **False Positive Reduction**: By setting a minimum contour area, the system was able to filter out minor movements, such as shadows, that could otherwise cause false positives.

7.2 Web Interface Testing

1. **Custom Video Feed Input**: Tested with different video URLs, including an IP camera and a sample video URL. The system successfully switched sources and displayed the feed in real time without needing a restart.

2. **Browser Compatibility**: The live feed was tested on multiple browsers (Chrome, Firefox, Edge) to ensure compatibility. The video displayed correctly across all tested browsers with no major issues.

7.3 Alert System Testing

1. **Asynchronous Sound Playback**: Verified that the sound alert plays immediately upon detecting persistent motion, without delaying the video stream. Tested with various durations to ensure sound playback remains non-blocking.

8 Conclusion

The Web Camera Overwatch and Alert System successfully combines motion detection, remote monitoring, and real-time alerts. By utilizing OpenCV for motion analysis, Flask for network streaming, and asynchronous threading for sound alerts, the system is both responsive and adaptable.

8.1 Strengths

- Provides real-time monitoring and motion detection with minimal latency.
- Supports customizable video sources, allowing flexibility in application.
- Asynchronous sound alert system enhances usability without interrupting the video stream.

8.2 Limitations

- Performance may be impacted by low-light conditions, as motion detection may struggle with noise.
- The system's response time depends on the quality and latency of the video feed.

8.3 Future Enhancements

- Integration with additional notification systems, such as email or SMS, to expand alert capabilities.
- Improved noise filtering techniques to enhance accuracy in low-light or dynamic lighting environments.