**OpenCV Real-Time Streaming and Processing - Week 2**

**Introduction**

This task focused on building a real-time video pipeline using OpenCV. Drones, CCTV cameras and many other gadgets have the capabilities to see the world wherever they are. It could detect different things in the frame along with motion detection. This task consists of working on RSTP. RTSP, or Real-Time Streaming Protocol, is a network control protocol used to manage multimedia streams, primarily audio and video, over networks. The goal was to handle multiple RTSP together, detect motion, and monitor camera health while keeping fps stable. The tasks combined were: showing 4 streams in one window, detecting motion in them, and checking if any camera is compromised (blur, blocked etc).

**Setup:**

1. Create a new folder and open VS code in it.
2. Put in the terminal: git clone https://github.com/Ishan1819/opencv-week2.git
3. Install dependencies by

pip install -r requirements.txt

1. To run the code, put the below command in the terminal

python [main.py](http://main.py)

### **Objective:**

* To develop a real-time multi-stream video viewer capable of monitoring multiple video feeds simultaneously.
* To detect motion in each stream using background subtraction and highlight moving areas.
* To check camera integrity by identifying blurry or blocked frames and raise warnings for compromised cameras.
* To ensure robust, thread-safe, and continuous monitoring with clear visual feedback on all streams.

**Deliverables:**

* Multi-Stream Viewer (2x2): Four streams displayed in a grid with basic controls.
* Motion Detection: Background subtractor used to highlight movement when threshold is crossed.
* Camera Check: Blurriness, brightness, and darkness analyzed. If too many faulty frames, a “Camera Compromised” warning appears.
* Repo + Report: Includes code, screenshots, and a note on issues like RTSP lag and fps trade-offs.

### **Approach**

### **a) Multi-Stream Viewer**

* Used cv2.VideoCapture to fetch video streams. It can capture from HTTP streams, RTSP streams, or a local camera.
* Chose 4 public sample MP4 videos because RTSP streams were not reliable and often failed to connect.
* Each stream runs on its own thread using threading.Thread, allowing all streams to run in parallel without blocking each other.
* Frames from each stream are resized to 640x480 using cv2.resize to ensure consistent display size.
* Combine all frames into a 2x2 grid using numpy.hstack (horizontal stacking) and numpy.vstack (vertical stacking).
* Displayed the combined output using cv2.imshow, which opens a window to show the live multi-stream feed.
* Added frame number and stream info using cv2.putText for easy monitoring.
* Thread-safe updates ensured using a threading.Lock when writing frames to the shared frames list.
* Stream status maintained in stream\_status array, showing real-time info such as "Connecting" or current frame number.

### **b) Motion Detection**

* Used cv2.createBackgroundSubtractorMOG2 for background subtraction and motion detection.
* Parameters explained:
  + history=200 - The number of frames used to model the background. Larger value is slower adaptation.
  + varThreshold=50 - Pixel sensitivity threshold. Pixels changing more than this from background are considered moving.
  + detectShadows=True - Detects shadows and marks them gray instead of counting them as motion.
* Counted moving pixels using cv2.countNonZero(fg\_mask).
* If moving pixels exceed motion\_threshold=5000, the frame is flagged as "Motion Detected" and a red rectangle is drawn on the frame using cv2.rectangle.
* Motion detection is updated frame by frame in real time.

### **c) Camera Integrity Check**

* Ensures the camera is not compromised by detecting blurry or covered frames.
* Blur detection:
  + Converted frame to grayscale.
  + Computed variance of Laplacian.
  + If variance < threshold=100, frame is considered blurry.
* **Covered/Blocked detection:**
  + Converted frame to grayscale.
  + Counted dark pixels (gray < dark\_thresh=40) and bright pixels (gray > bright\_thresh=220).
  + If more than 75% of the frame is too dark or too bright, the frame is considered blocked.
* History of last 30 frames stored using collections.deque(maxlen=30).
* If more than 75% of recent frames are compromised, displayed "Camera Compromised" warning on the frame.
* Updated compromised status frame-by-frame, ensuring real-time monitoring.

### **d) Additional Features in Code**

* Added stream error handling: if a stream fails to open, a black error frame is displayed with “Stream FAILED” text.
* Shows runtime of viewer in seconds using time.time() - start\_time.
* Press 'q' to quit viewer, 's' to print current stream status for all streams.
* Ensures thread safety for shared resources (frames and stream\_status) using a lock.

### **Key Learnings**

1. **Multi-Threading in Python**
   * Learned how to run multiple streams in parallel using threading.Thread.
   * Understood thread safety and why a threading.Lock is needed when multiple threads access shared resources (frames list and stream\_status).
2. **Video Processing with OpenCV**
   * Explored cv2.VideoCapture for reading video from both local files and HTTP streams.
   * Learned how to resize frames (cv2.resize) and combine multiple frames into a grid layout using numpy.hstack and numpy.vstack.
   * Gained experience in using cv2.imshow for real-time video display.
3. **Motion Detection**
   * Implemented motion detection using background subtraction (cv2.createBackgroundSubtractorMOG2).
   * Learned the role of parameters: history, varThreshold, and detectShadows.  
     Understood how to detect motion pixel by pixel and use a threshold on total moving pixels to avoid false positives.
4. **Camera Integrity Monitoring**
   * Learned techniques for blur detection using the variance of Laplacian.
   * Implemented covered/blocked frame detection using pixel brightness thresholds and percentage calculations.
   * Maintained a frame history with deque to identify if a camera is consistently compromised.
5. **Error Handling and Robustness**
   * Learned to handle cases where a video stream fails to open and show an error frame.
   * Gained experience in restarting streams automatically if the video ends or fails.
6. **Real-Time Frame Annotation**
   * Practiced using cv2.putText and cv2.rectangle to display motion alerts, frame count, stream info, and warnings directly on the video frames.
7. **Performance Optimization**
   * Learned to control playback speed using time.sleep to prevent high CPU usage.
   * Observed the effect of frame resizing and threshold tuning on real-time performance.
8. **Combining Computer Vision Techniques**
   * Integrated motion detection and camera integrity checks in a single pipeline.
   * Learned how to visualize and interpret results for multiple streams simultaneously.
9. **Practical Exposure**
   * Hands-on experience with debugging real-time video streams.
   * Learned challenges of handling public sample streams vs. RTSP streams.
10. **Parameter Tuning**
    * Gained insight into tuning parameters (varThreshold, motion\_threshold, blur threshold, brightness limits) for reliable motion and integrity detection.

### **Challenges Faced**

1. RTSP Stream Stability – Streams often disconnected or buffered; used public MP4 links and auto-restart to handle failures.
2. Multi-Threading Synchronization – Needed threading.Lock to safely update shared frames and statuses.
3. Performance vs Real-Time Processing – Motion and integrity checks are CPU-intensive, resized frames and controlled playback speed to maintain real-time display.
4. Stream Errors and Visualization – Handled video end/errors gracefully and displayed clear annotations without clutter.

### **7. Results & Outputs**

* **2x2 Stream Window** – All four streams displayed simultaneously in a combined grid using numpy.hstack and numpy.vstack.
* **Motion Detection Alerts** – Moving objects highlighted with **“**MotionDetected**”** text and a redrectangle on frames.
* **Compromised Camera Warning** – Frames flagged as blurry or blocked show **“**BlurDetected**” / “**Covered/Blocked**” / “**CameraCompromised**”** messages.
* **Status Logs** – Real-time stream status printed in console (e.g., “Playing 50/100 frames”, “FAILED TO CONNECT”).
* **Runtime Display** – Viewer shows elapsed runtime in seconds on the video window.
* **Error Handling** – Failed streams display a black frame with errormessage, ensuring continuous monitoring.

### **8. Conclusion**

* **Summary of Key Learnings:**
  + Implemented multi-threaded video processing and frame synchronization.
  + Applied motion detection using backgroundsubtraction and tuned parameters for accuracy.
  + Developed camera integrity checks (blur and coverage) with historical frame tracking.
  + Handled stream errors, frame resizing, and real-time display effectively.
* **Real-World Applications:**
  + **Traffic Monitoring** – Detecting moving vehicles or congestion.
  + **Security Surveillance** – Detecting motion in CCTV feeds and compromised cameras.
  + **CCTV Analytics** – Automated monitoring, alert generation, and video summarization.
* **Possible Future Improvements:**
  + Integrate objectdetection (e.g., detecting humans, vehicles).
  + Support cloudstreaming for remote monitoring of multiple cameras.
  + Optimize performance using GPU acceleration or multiprocessing for higher resolution streams.