Task: OpenCV Real-Time Streaming and Processing - Week 2

INTRODUCTION

This project implements a real-time processing pipeline for four RTSP streams that performs motion detection and camera integrity checks (blur, coverage/laser). The implementation strives to maintain processing FPS ≥ streaming FPS through a mix of algorithmic choices and engineering trade-offs.

SOURCES CONSULTED

- OpenCV docs: `cv2.createBackgroundSubtractorMOG2`, `cv2.absdiff`, `cv2.putText`, `cv2.rectangle`. https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html
- OpenCV Laplacian tutorial for blur detection. https://docs.opencv.org/4.x/d5/db5/tutorial_laplace_operator.html
- Various Stack Overflow threads for histogram analysis and RTSP reliability.
- YouTube tutorial playlists on OpenCV motion detection.

KEY LEARNINGS AND INSIGHTS

- RTSP stream stability: many consumer cameras reset or drop frames; robust code re-opens capture on failure.
- Motion detection trade-offs: MOG2 is robust to lighting changes but requires tuning of history and thresholds. Frame differencing is simpler but more brittle with noise.
- Real-time optimisation: reducing processing resolution and skipping frames are effective simple levers. Using compiled codecs/hardware accelerations helps on production.

APPROACH AND IMPLEMENTATION

1. Each stream is handled by a `StreamProcessor` instance:
    - Background subtraction for motion detection.
    - Laplacian variance as a blur metric (threshold = 100).
    - Histogram uniformity to detect dominant color/overexposure (dominance threshold = 0.6).
2. A stream is marked as compromised if a heuristic `compromise_percent >=75`.

EXPERIMENTS / PRACTICE ATTEMPTS

- Tested with local webcams (indices `0,1,2,3`).
- Adjusted contour area threshold for motion and Laplacian threshold for blur until reasonable on sample video.

CHALLENGES

- Choosing robust thresholds that generalize across varied cameras and lighting.

- Ensuring OpenCV `VideoCapture` reliably reconnects to failing streams.

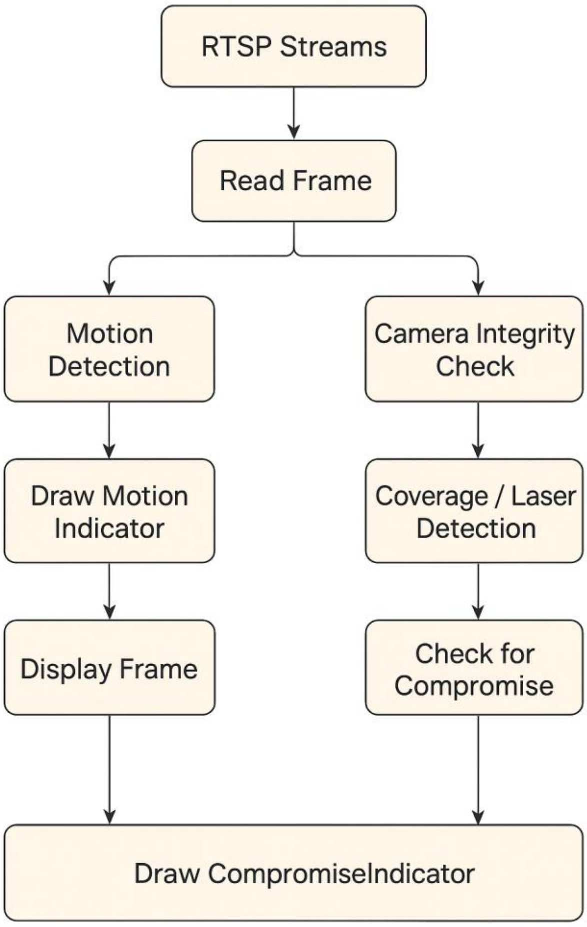
PERFORMANCE METRICS

- Each `StreamProcessor` reports processing FPS in the overlay.
- The main loop prints combined/process FPS on the mosaic window.
- For full profiling, `psutil` can be added to log memory usage over time.

FUTURE IMPROVEMENTS

- Use worker threads/processes or asyncio per stream to fully utilize multicore CPUs.
- Use hardware-accelerated decoders or GStreamer to lower CPU usage.
- Use a small ML model for tamper detection to be more robust than histogram heuristics.

SCREENSHOTS AND DIAGRAMS



CONCLUSION

This implementation demonstrates the requested real-time motion detection and camera integrity checks. With further tuning and production-grade stream handling, it can serve as the basis for a robust multi-camera monitoring system.