Real-Time Streaming and Processing

Objective :

The goal of this task was to:

- Stream multiple RTSP feeds simultaneously using OpenCV.
- Display them in a synchronized 2x2 grid (multi-camera viewer).
- Implement real-time motion detection on each stream.
- Add camera integrity checks to identify if a feed is covered, blurred, or blocked.
- Provide a clean, real-time interface with motion alerts and compromised camera warnings.

Key Learning :

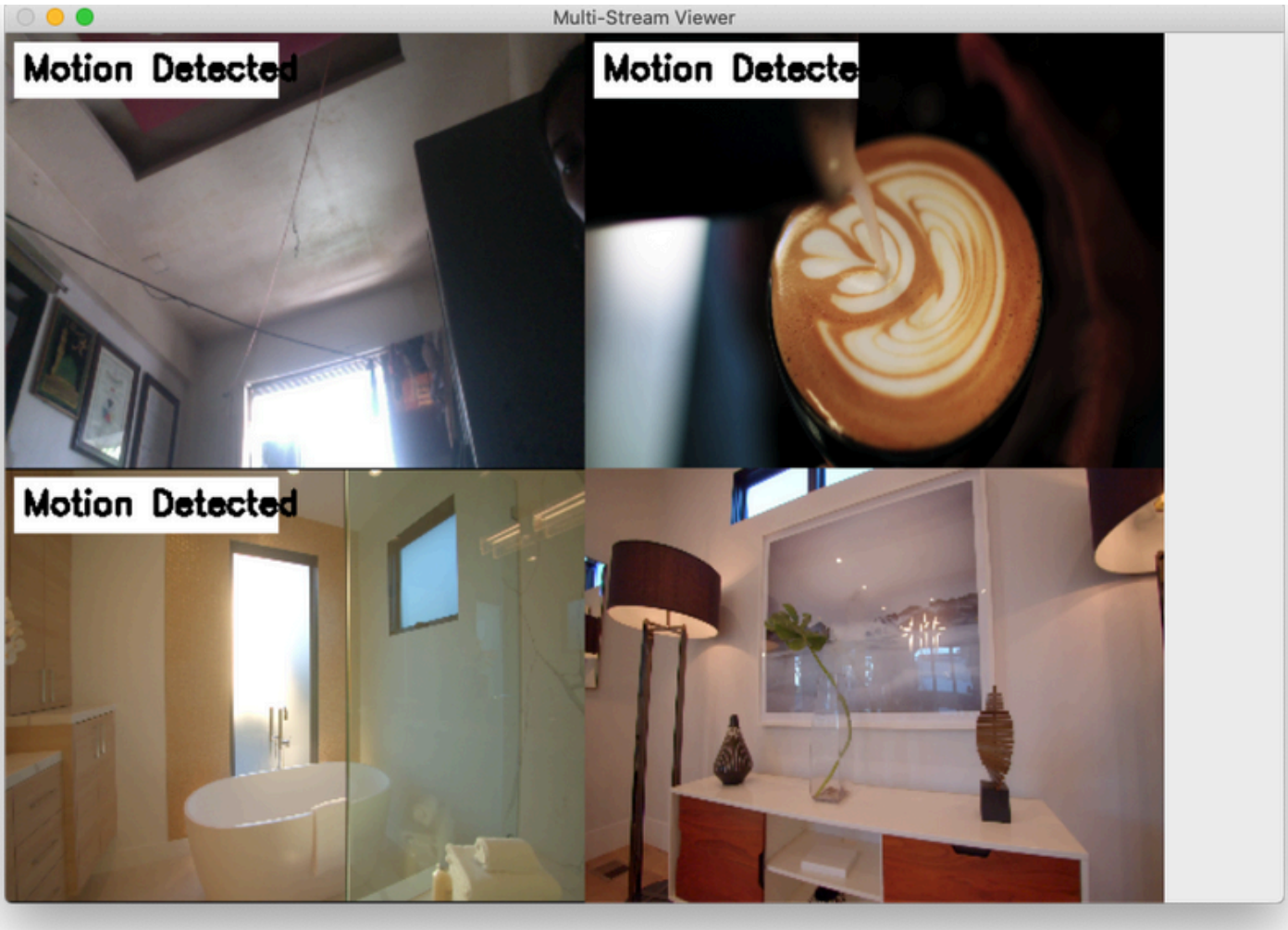
- Learned how to use multithreading to process multiple streams in real-time.
- Understood the importance of frame resizing before arranging in a grid.
- Motion detection works best with background subtraction (MOG2) instead of only frame differencing.
- The variance of Laplacian is a reliable measure for detecting blur or camera tampering.
- Combining frames with numpy.hstack() / vstack() and handling FPS profiling ensures smoother display.

Challenges :

- RTSP Stream Availability: Some public streams were unstable or required credentials, so local laptop cameras were used as substitutes.
- Synchronization: Threads fetching frames at different FPS sometimes caused mismatched grids.
- Real-time Performance: High-resolution feeds slowed processing; resizing helped achieve real-time FPS.
- False Motion Alerts: Small lighting changes or shadows sometimes triggered motion detection.
- Camera Integrity Check: Balancing thresholds (blur variance, histogram uniformity) was tricky to avoid false positives.

Results and Screenshot :

- Successfully created a 2x2 live camera grid with motion detection labels.
- Cameras showed "Motion Detected" alerts when activity was present.
- When a camera was blocked or blurred, the system displayed “No Signal”.



Conclusion :

This project demonstrated a real-time computer vision system capable of:

- Handling multiple live streams.
- Detecting motion events per stream.
- Identifying camera tampering or compromise.

The implementation provides a foundation for surveillance systems where multiple CCTV feeds must be monitored efficiently. Future improvements could include:

- Saving motion events to disk.
- Adding audio alerts for compromised cameras.
- Deploying with GPU acceleration for better performance.