A Vision-Based Motion-Detection System Combining MediaPipe with Optical Flow for Long-Term Coma Patient Monitoring

a report submitted by

Verena Grall

IN-2025-B1134-KU

in

IAESTE INDIA

under the supervision of

Dr. D. Pamela

KARUNYA INSTITUTE OF TECHNOLOGY AND SCIENCES

Karunya Nagar, Coimbatore - 641 114

INDIA

(Date: 10/09/2025)

BONAFIDE CERTIFICATE

This is to certify that the project report entitled ***A Vision-Based Motion-Detection System Combining MediaPipe with Optical Flow for Long-Term Coma Patient Monitoring*** is the bonafide work of Ms. **Verena Grall** who carried out the project work under my supervision.

SIGNATURE

Dr. D. Pamela

Head of Department

Department of Biomedical

Karunya Institute of Technology and Sciences

Karunya Nagar, Coimbatore,

Tamil Nadu 641114, India

Abstract

This work presents a real-time, hand-focused motion-detection pipeline aimed at long-duration monitoring where movements are infrequent, brief, and low-amplitude after extended quiescence. The system first localizes a hand-centric region of interest (ROI) using MediaPipe Hands and then computes Farnebäck dense optical flow strictly within that ROI. Motion evidence is formed by thresholding the flow magnitude inside the ROI, denoised via morphological opening, and adjudicated with a temporal debounce requiring ≥3 consecutive motion frames. On a debounced onset, a unified trigger atomically emits an audible alert, appends a timestamped CSV record, and begins video recording with a fixed 2-s pre-roll; recording terminates after 5 s without motion (cooldown), after which the detector re-arms. The implementation is single-threaded (Python/OpenCV/MediaPipe), CPU-only, and portable across common operating systems.

Validation followed engineering-style procedures rather than participant studies. In a scripted-burst file test (five bursts separated by ≥ cooldown), trigger correctness was 100% (one alert/CSV/clip per burst). Pre-roll fidelity matched configuration exactly (60/60 frames at ~30 FPS). Tail accuracy, assessed by wall-clock time $t_{end} - t_{last\ motion}$, passed in 4/5 events with mean 5.055 s (SD 0.011 s); the single failure arose because the source video ended before cooldown completion, not from detector logic. Live-loop tests confirmed reliable re-arming across cycles and correct pause/grace behavior. Realized recording rate during events was ~28–29 FPS on commodity hardware.

Taken together, the results indicate that ROI-restricted dense optical flow combined with debounce and cooldown controls yields synchronized alerts, auditable logs, and context-rich clips with well-defined onsets (pre-roll) and offsets (cooldown tail). The system is intended as an assistive research/engineering tool rather than a medical device; any clinical use requires institutional oversight and appropriate privacy safeguards.

Important note: this system is not a medical device; it is a research/engineering tool to assist caregivers and researchers. It must not be used as the sole basis for medical decision-making.

Table of Contents

Background and Motivation

Monitoring patients with prolonged disorders of consciousness often involves watching for infrequent, brief, and subtle movements that can occur after long periods of quiescence. Clinical literature underscores that detecting and interpreting rare motor phenomena in critical-care contexts is important yet challenging (Hannawi et al., 2016; Hu et al., 2024; Yimer et al., 2024). By contrast, much of the computer-vision work on motion and gesture analysis implicitly assumes abundant or sustained motion—for example, in general foreground/background motion detection, action analysis, or gesture pipelines where movement is continuous or frequent (Sengar and Mukhopadhyay, 2017; Thapa et al., 2014).

The application setting here therefore differs in two key ways: (i) temporal sparsity—long intervals of no movement punctuated by short, low-amplitude events—and (ii) a narrow region of clinical interest (the hand). These characteristics motivate a design that emphasizes precision within a small, semantically relevant region and robustness to brief, subtle displacements, rather than modeling global scene dynamics.

Accordingly, the system first localizes a hand-centric region of interest (ROI) using MediaPipe Hands and then computes dense optical flow (Farnebäck) strictly inside that ROI to quantify inter-frame motion (Farnebäck, 2003; Google Ireland Limited, 2025; Hari et al., 2023; OpenCV, 2025; Zhang et al., 2020). The ROI restriction reduces the search space, focuses computation where clinically meaningful movement is expected, and helps decouple hand motion from unrelated background activity. Dense optical flow supplies per-pixel motion estimates that are sensitive to small displacements, making it suitable when the signal of interest is a short twitch rather than a sustained gesture.

To support long-duration observation without operator burden, the system adds temporal debouncing (require brief persistence before declaring an onset) and a cooldown (define a clean end of event). A pre-roll buffer preserves several seconds of context before the detected onset, facilitating retrospective review in clinical workflows. The overall approach is designed to run on-device and in real time on commodity hardware, favoring determinism and auditability over heavier learned models—an appropriate trade-off for the target setting and the project's time constraints

Objectives and Non-Goals

Objectives

1. Detect hand-localized motion events in real time and atomically trigger: audible alert, CSV logging, and video recording with pre-roll.

2. Ensure repeatable event lifecycle: debounce before onset; cooldown-driven termination; reliable re-arming.
3. Maintain portability (Windows/macOS/Linux) and CPU-only operation at common webcam resolutions (≥720p).
4. Provide auditability through timestamped clip names, a simple CSV schema, and deterministic behavior.

Non-Goals

- Clinical diagnosis or automated clinical decision-making.
- Full-body tracking, multi-patient scenes, or multi-camera fusion.
- Privacy policy tooling beyond basic file hygiene; no cloud upload.
- Advanced learned motion segmentation or training custom models within the current scope.

Requirements

Functional

- Ingest live camera or file; localize a single hand; compute ROI optical flow; fire unified trigger on debounced motion; write clips with pre-roll; stop after cooldown; re-arm; allow pause/resume with grace; render overlays; exit cleanly.

Non-Functional

- **Throughput:** sustain camera FPS with modest ROIs on commodity CPU.
- **Reliability:** no crashes/leaks; robust lifecycle across repeated triggers.
- **Portability:** automatic codec selection; pluggable alert sink per OS.
- **Auditability:** timestamped filenames; CSV log with header auto-creation.

Safety, Privacy, and Ethics

- Local-only storage by default; no protected health information (PHI) beyond video/CSV stored; recommend access controls on the host.
- On-device processing (no network dependency).
- For real patient data, follow institutional policy (consent, storage, retention, deletion).

Detailed Method

This section describes the techniques and algorithms used for real-time, hand-focused motion detection and event recording. A hand region of interest (ROI) is estimated per frame with MediaPipe Hands, and Farnebäck dense optical flow is computed inside that ROI to quantify inter-frame motion. When a debounced motion condition is met, the system atomically (i) plays an audible alert, (ii) logs a timestamped event to CSV, and (iii) starts saving a video clip that includes pre-event context from a ring buffer. Recording stops after a motion-free cooldown, and the detector re-arms.

Hand Detection using Mediapipe

The motion detection should be focused on the hands. Therefore, Mediapipe's hand tracking module was used to detect the hand and extract a region of interest (ROI) around. Mediapipe uses a palm detection model followed by a hand landmark model that outputs 21 landmarks per hand. To detect the hand itself, only the integrated BlazePalm detector would be necessary without the hand landmark model. However, the library does not support that and therefore, the whole Mediapipe library is used. When there are runtime and efficiency issues, the TFLite BlazePalm model, which is derived from the BlazeFace family, could be used and adapted directly. (Bazarevsky et al., 2019; Google Ireland Limited, 2025; Zhang et al., 2020)

First, the captured BGR frames are converted to RGB, as this is required by Mediapipe. (Google Ireland Limited, 2025; Zhang et al., 2020, 2012) When there is a hand present, the landmarks are detected. Then, bounding box coordinates are computed from the min/max of the landmark positions and additionally, a small padding offset (10 pixels) is added to enlarge the ROI slightly and avoid truncating motion near the edges. Finally, the ROI is extracted from the original frame for further analysis. Figure 1: ROI localization. (own representation) representatively shows the blue, hand-centric ROI produced by MediaPipe Hands (padding = 10px). The ROI restricts motion analysis to the clinically relevant area and reduces computation.

Figure 1: ROI localization. (own representation)

In this implementation only one hand (max_num_hands = 1) is tracked per camera. The hand bounding box is computed dynamically using the extremities of the landmark coordinates. This has to be changed when only palm detection is used.

This pre-processing is performed to focus solely on movement in the hand region to minimize false positives from the background. By localizing the ROI dynamically, the system processes only relevant areas, significantly reducing computational cost and false positives.

Motion Detection via Farnebäck Dense Optical Flow

Sparse optical flow and dense optical flow are two types of optical flow. According to Zvoristeanu et.al. (2022) in sparse optical flow only certain features of objects are described by the flow vector. On the other hand, in dense optical flow, the flow vectors of all pixels from the image are calculated. (Beauchemin and Barron, 1995; Horn and Schunck, 1981; Lucas and Kanade, 1981)

Dense optical flow requires more computational power, but it is more accurate and has better performance. For this project different methods and algorithms were tested, but dense optical flow (Farnebäck Method) was most resistant to noise (e.g. shadows). Therefore, this method was chosen. The balance between accuracy and computational power was tried to be found by testing different parameters. The parameters are described further in section Configuration and Parameters.

The dense optical flow method from Farnebäck estimates per-pixel motion between two grayscale frames, enabling robust motion vector computation (Farnebäck, 2003). In this project, dense optical flow is computed between the previous grayscale ROI (prev_gray_roi) and the current grayscale ROI using cv2.calcOpticalFlowFarneback with

standard parameters (pyramid scale and levels, window size, iterations, polynomial neighborhood and smoothing). The flow field is converted from Cartesian components (u,v) to magnitude and angle; only the magnitude is used as a motion signal. (Farnebäck, 2003; Hari et al., 2023; OpenCV, 2025)

To isolate motion, the magnitude image is thresholded at a tunable motion threshold to obtain a binary mask. Then a morphological opening is applied with a 5x5 elliptical kernel to suppress small, isolated activations caused by sensor noise or minor illumination changes. (Soille, 2004)

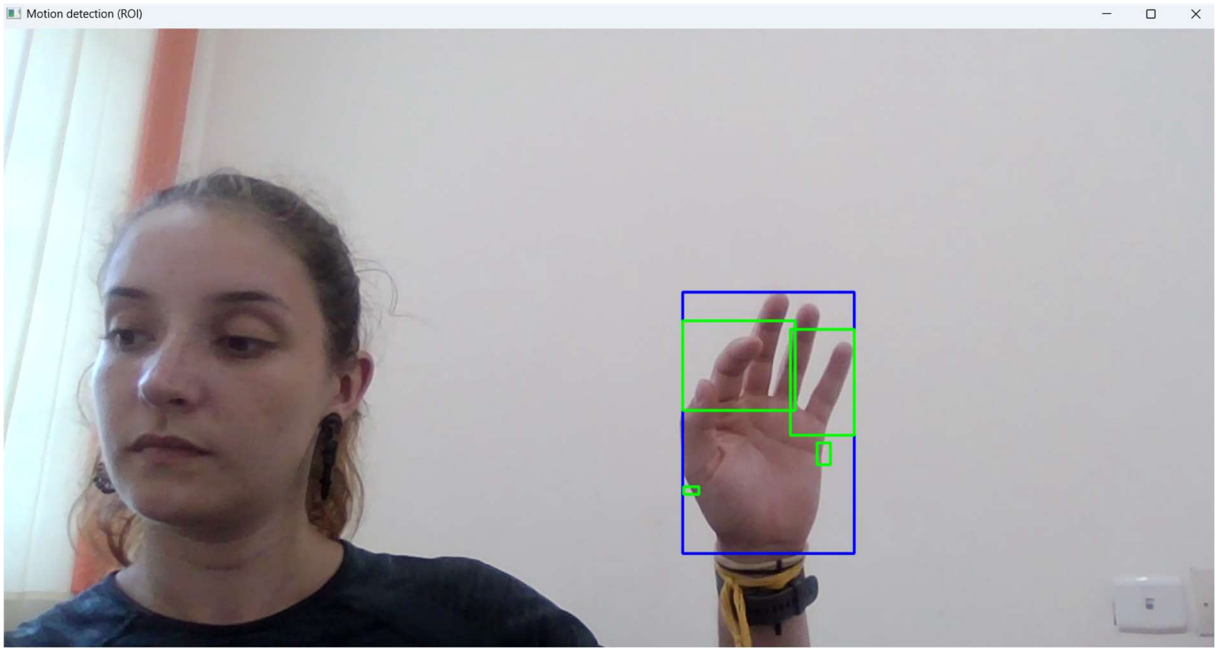

Figure 2: Motion blobs within ROI. (own representation)

Figure 2: Motion blobs within ROI. (own representation) shows frame with green rectangles corresponding to thresholded and morphologically cleaned motion regions inside the blue ROI. The detector uses area fraction and temporal debounce; contours are shown for interpretability only.

Frame-level Motion Decision and Temporal Debounce

For each ROI, the area fraction of active pixels is computed:

$$a_t = \frac{\#\{(x,y): mask(x,y) = 1\}}{ROI\ area}$$

A frame is marked as “motion” if $a_t > 0.01$ (i.e., >1% of ROI pixels exceed the magnitude threshold). To reject one-frame blips, we maintain a counter of consecutive motion frames; the system is considered armed when the count reaches motion_arm_frames (default 3). Only when armed does the system allow a trigger.

For visualization contours are extracted from the motion mask and green rectangles are drawn on the full frame for interpretability. These contours are not used for the decision but are purely for visual.

Unified Event Trigger (Alert + Log + Recording with Pre-roll)

Condition for the triggers to be able to be triggered are when ready == True (armed) and video_writer is None, which means that the system is armed and not currently recording. In this case, an atomic trigger fires:

1. **Timestamp & state:** Set last_motion_time to "now" and a timestamped filename is created.
2. **Recording start:** A cv2.VideoWriter is opened with a platform-appropriate codec (Windows: XVID/AVI; others: MP4V/MP4) at the current frame size and recording FPS.
3. **Pre-roll flush:** All frames in the pre_motion_buffer (last B seconds) are written to the new file in chronological order.
4. **Alert:** An audible cue is emitted (winsound.Beep on Windows; ASCII bell elsewhere).
5. **Log:** A CSV row (Timestamp, Filename, Event) is appended to movement_log.csv (header is auto created on first write).

This coupling ensures the audible cue, the log entry, and clip initiation are synchronized to the same event edge.

Recording Lifecycle, Cooldown, and Re-arming

While recording, each incoming full frame is written to disk. If the current frame exhibits motion, last_motion_time is updated, which extends the session. If the time since the last motion exceeds a cooldown (default 5 seconds), the writer is closed, recording stops, and the detector re-arms by clearing the optical-flow baseline and resetting the debounce counter. This prevents immediate re-triggers from stale flow.

Frame Buffering and Pre-roll

A deque ring buffer maintains the most recent buffer seconds of full frames at the recording FPS. The buffer is appended at every frame, independent of ROI availability, and cleared on resume after a user pause. This guarantees that each saved clip includes pre-event context, improving interpretability and auditability.

System Architecture (summary)

Each iteration performs: frame capture → MediaPipe ROI → baseline management (initialize/resize/refresh, with resume-grace) → Farnebäck flow in ROI → magnitude threshold + morphology → area-fraction decision → temporal debounce → unified trigger (beep+CSV+record with pre-roll) → recording maintenance → cooldown termination → re-

arm. Status overlays are drawn for ROI (blue) and motion regions (green). UI allows pause/resume (with grace) and quit. More detailed information and visualized figures about the primary data flow and the modular architecture can be looked up in section System Architecture at Figure 3: Primary Data Flow (own representation)and Figure 4: Modular Architecture (own representation).

Capture & FPS Adaptation

Frames are captured via cv2.VideoCapture (camera index or file). On the first valid frame, the system records frame size and adopts the camera's reported FPS if available, recomputing the ring buffer length so pre-roll duration remains constant in seconds. A copy (frame_out) is kept for visualization while the original frame is preserved for writing.

Timing, Latency, and Cleanup

Onset latency is approximately (motion_arm_frames – 1) / recording_fps plus per-frame compute. Clip tails are ~cooldown seconds beyond the last motion frame (+-1 frame). On shutdown or error, the capture and writer are released, the MediaPipe object is closed, and windows are destroyed.

UI, Control, and Human-In-the-Loop States

- **Display:** cv2.imshow renders frame_out with overlays; draw_status shows *PAUSED* on *rearming* banners.
- **Input:** ui_step() handles 'p' (pause/resume), 'q'/Esc (quit), and window close.
- **Pause behavior:** Toggling pause sets paused. On resume, the system sets resume_until = now + resume_grace, clears the pre-roll buffer, resets debounce, and drops the optical-flow baseline for a clean re-start.

Logging, Filenaming, and Storage Layout

- **Clips:** Saved under movement_clips/ as movement_YYYY-MM-DD_HH-MM-SS.ext with the platform codec/extension.
- **CSV schema:** Timestamp (string), Filename (string path), Event (string label); one row per trigger.
- **Storage considerations:** Clip size scales with FPS, resolution, and session duration (cooldown + motion). A pruning policy can be added without altering the detection loop.

System Architecture

In this section, the system architecture is described by giving a short overview about the primary data flow, the components and responsibilities, runtime state machine, the timing and performance characteristics, error handling and resource hygiene, and the platform and operational notes.

Overview & Primary Data Flow

The system operates as a single-threaded, event-driven pipeline that processes one video frame per iteration. Each cycle begins by capturing a frame from a camera or file and duplicating it: one copy is retained for disk writing, while the other is used for diagnostic overlays and user interaction. MediaPipe Hands localizes a hand region of interest (ROI) by converting the frame from BGR to RGB, running the palm detector and landmark model, and constructing a tight, axis-aligned bounding box with an additional ten-pixel padding. If a valid ROI is unavailable or too small, motion analysis is skipped for that iteration, and the motion state is reset to prevent stale flow from influencing subsequent decisions.

When a valid ROI exists, it is extracted and converted to grayscale. The system preserves a grayscale baseline of the previous ROI; if the current ROI differs in size, the baseline is resized to match so that optical flow remains well-posed. Immediately after a manual resume, a short grace interval suppresses motion decisions and refreshes the baseline to avoid spurious triggers. Dense optical flow (Farnebäck) is then computed between the previous and current grayscale ROIs. The flow magnitude is thresholded to form a binary motion mask, which is denoised via morphological opening with a small elliptical kernel. The fraction of active pixels within the ROI provides a frame-level motion score; a frame is declared to contain motion when this fraction exceeds 0.01. Temporal stability is enforced by counting consecutive motion frames and declaring the system “ready” only when the count reaches a configurable debounce window.

When the system is ready and not currently recording, a unified trigger fires. It sets the last-motion timestamp, opens a video writer with an operating-system-appropriate codec at the current geometry and frame rate, flushes the pre-motion ring buffer to preserve pre-event context, emits an audible cue, and appends a timestamped row to a CSV log. While recording, each full frame is written to disk; any frame that contains motion extends the session by updating the last-motion time. When the elapsed silence exceeds the cooldown, the writer is closed and the detector re-arms by clearing the optical-flow baseline and the debounce counter.

Figure 3: Primary Data Flow (own representation)

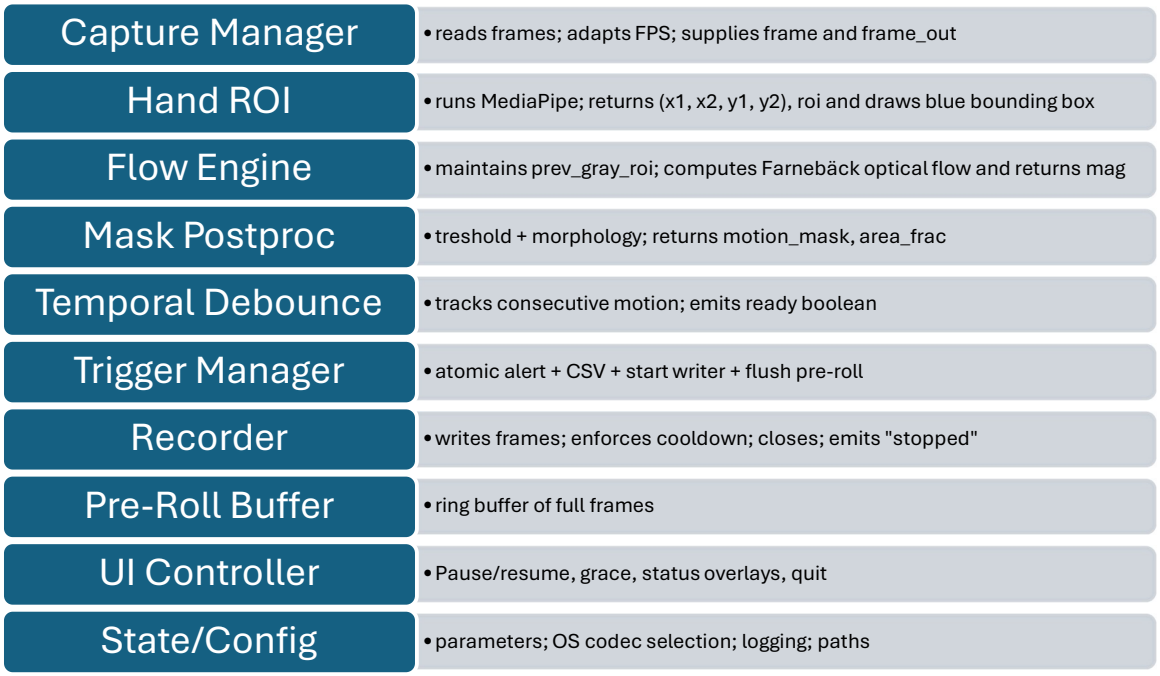

Figure 4: Modular Architecture (own representation)

Components & Responsibilities

The pipeline is composed of small, cooperating components with clear interfaces. The capture component acquires frames and, on the first valid frame, records image geometry and adopts the camera's reported frame rate so that the pre-roll buffer length represents a fixed wall-clock duration. The hand-ROI component executes MediaPipe Hands and renders a blue rectangle over the active ROI. The flow engine maintains the previous grayscale ROI and computes Farnebäck optical flow to obtain a magnitude image suitable for thresholding. The mask processor thresholds the magnitude image, applies morphology, and computes the area fraction used by the decision layer. The temporal debounce component maintains the consecutive-motion counter and produces the readiness signal. The trigger manager couples alerting, CSV logging, writer creation, and pre-roll flushing so that all artifacts align to the same motion edge. The recorder writes frames and enforces cooldown-based termination. A small UI controller renders overlays (ROI in blue, motion blobs in green), handles pause/resume and quit, and reacts to window close events. Finally, the logger manages the CSV audit trail and the file-naming scheme for timestamped clips in a dedicated directory.

Runtime State Machine

Runtime behavior is well captured by a small set of states. In Idle/ROI-missing, no valid ROI is present; the baseline is cleared while the pre-roll buffer continues to accumulate frames. Baseline/Arming maintains the grayscale baseline and evaluates the frame-level motion score without yet being eligible to trigger. Ready (debounced) indicates that the consecutive-motion criterion has been met; the next iteration that finds the system idle will initiate recording. Recording writes frames and implicitly includes the cooldown tail;

the state ends when the silence interval exceeds the cooldown threshold. Paused/Grace is an orthogonal mode entered by user input; during the grace window the system refreshes its baseline and ignores motion, after which it returns to Baseline/Arming.

Timing & Performance Characteristics

The onset latency is determined primarily by the debounce window: it is approximately (motion_arm_frames − 1) / recording_fps plus the per-frame compute time of MediaPipe inference and ROI optical flow. Clip tails are approximately equal to the cooldown beyond the last motion frame, up to a one-frame quantization. Throughput is dominated by MediaPipe and optical flow; however, by restricting analysis to the ROI, the optical-flow cost scales roughly with the ROI-to-frame area ratio, enabling real-time CPU operation at common resolutions with modest ROIs.

Error Handling & Resource Hygiene

Failures and edge cases are handled explicitly. If frame acquisition fails, the loop exits cleanly and a finally block releases the capture device, closes any active writer, shuts down the MediaPipe object, and destroys GUI windows. When an ROI is absent or degenerate, motion processing is skipped for that frame and the motion state is reset to avoid spurious triggers. After resuming, a grace period prevents immediate triggering while a clean baseline is established. Codec selection is abstracted per platform, and a fallback to broadly supported codecs (such as MJPG) can be applied without altering the detection logic. CSV logging creates a header on first use and tolerates I/O errors by reporting them to the console without terminating capture.

Platform & Operational Notes

Two platform concerns are isolated behind small surfaces. Sound alerts use winsound.Beep on Windows and an ASCII-bell fallback elsewhere; if terminal bells are suppressed, the alert sink can be swapped for a tiny WAV playback library without changing trigger semantics. Codecs default to XVID/AVI on Windows and MP4V/MP4 elsewhere; if playback compatibility becomes an issue, switching to MJPG is straightforward. Storage growth is driven by pre-roll duration, event length, and cooldown; deployments that run for long periods benefit from a simple pruning policy or daily caps.

Design Choices

We summarize the most important architectural decisions, the alternatives considered, and their implications.

ROI-restricted motion analysis

Choice. Perform motion estimation only within a hand ROI derived from MediaPipe landmarks.

Alternatives. Full-frame motion analysis; static user-defined ROI; background subtraction over the whole image.

Rationale. Constraining analysis to the hand substantially reduces false positives from torso/background motion and shrinks the pixel domain for dense optical flow. If the ROI contains a fraction r=|ROI/|Frame| of the frame pixels, optical-flow cost scales approximately linearly with *r*, i.e., *O(rWH),* yielding a proportional speedup and lower energy use.

Trade-offs. Misses non-hand events by design; depends on reliable hand detection. Rapid ROI scale/position changes can cause brief baseline misalignment.

Implications. Higher precision in desk scenarios; better real-time behaviour on CPU. Future extensions may union multiple ROIs (e.g., two hands) or fall back to a stable last-seen ROI when detection drops.

Optical flow over frame differencing

Choice. Use dense optical flow (Farnebäck) within the ROI instead of raw frame differencing.

Alternatives. Absolute intensity differencing; temporal running average/background model; sparse (LK) flow; learned motion segmentation.

Rationale. Dense flow is more robust to lighting and sensor-noise variations and captures sub-pixel displacements, improving sensitivity to slow/oblique hand motion. It avoids maintaining a global background model, which is brittle under illumination changes and screen flicker. (Barnich and Van Droogenbroeck, 2011; Stauffer and Grimson, 1999; Zivkovic, 2004)

Trade-offs. Higher per-pixel compute than differencing; sensitive to very abrupt lighting spikes. Sparse flow is cheaper but loses coverage; learned segmentation needs data and adds deployment friction.

Implications. With ROI reduction (Choice 1), dense flow remains real-time on commodity CPUs. Sensitivity is tuned via winsize, levels, iterations, and the magnitude threshold motion_threshold.

Pre-buffered video saving (pre-roll)

Choice. Maintain a ring buffer of the last buffer seconds of full frames and flush it into the file on trigger.

Alternatives. No pre-roll (start at trigger); ROI-only pre-roll.

Rationale. Pre-roll preserves pre-event context (gesture onset, approach) and improves interpretability and auditability of saved clips and logs.

Trade-offs. Memory/disk overhead proportional to buffer_size * frame_size. Full-frame pre-roll is larger than ROI-only but avoids reconstruction artifacts and keeps clips self-contained.

Implications. Users can accurately review what led to a trigger. Storage pressure is governed by buffer, cooldown, and FPS; a pruning policy can be added orthogonally.

Sound alerts (platform detail)

Choice. Use winsound.Beep for Windows; fallback ASCII bell (\a) elsewhere.

Alternatives. Cross-platform audio libraries (e.g., simpleaudio, pygame, playsound), or OS notification APIs.

Rationale. winsound is zero-dependency on Windows and provides immediate operator feedback at trigger time.

Trade-offs. Non-Windows platforms may not emit an audible tone with \a depending on terminal settings; external libraries add dependencies.

Implications. The alert mechanism is pluggable: the trigger point remains the same, and only the alert backend changes by platform.

Implementation Details

The environment and dependencies, how to build and run, information about file and logging layout, data structures and key APIs, cross-platform code paths, and extensibility hooks are described here further.

Environment & Dependencies

The implementation is in Python and depends on OpenCV for video I/O, morphology, and optical flow; MediaPipe for hand detection and landmarks; NumPy for array operations; and standard-library modules for buffering (collections.deque) and logging (csv). The system targets commodity CPUs and standard webcams but also accepts file inputs for deterministic validation.

Build & Run

At startup, the program opens a cv2.VideoCapture on a camera index or a file path. On the first valid frame, it records the frame dimensions and adopts the camera's reported frame rate if available, recomputing the ring-buffer length so that pre-roll remains a fixed number of seconds. The UI window displays diagnostic overlays, and keyboard input supports pause/resume (with grace) and quit. Clips are written to a movement_clips/ directory; the CSV audit log is stored alongside the program.

File & Logging Layout

Clips are named with a timestamp to encode the event start time and are saved with an OS-appropriate extension. The CSV log contains three fields—Timestamp, Filename, and Event—and creates a header on first write. This simple layout enables rapid inspection and downstream scripting without specialized tooling.

Data Structures, Working Set, and Complexity

This subsection summarizes the principal data structures in the pipeline and derives tight, implementation-level bounds for runtime per frame and memory footprint.

Notation

Let the full frame be W x H pixels with $N = W * H$. Let the hand ROI be w x h with $n_{ROI} = w * h$ and area ratio $r = n_{ROI}/N \in (0,1]$. Farnebäck optical flow uses L pyramid levels, I iterations per level, and a window size s (so the per-pixel neighborhood cost is proportional to s^2). The pre-roll buffer holds B full frames, where $B = recording_{fps} * buffer$.

Per-frame runtime bound

Each iteration does four linear passes plus ROI-restricted flow:

- **MediaPipe Hands (full frame):** $\theta(N)$ pixel work per inference (constant factors depend on the model) (Zhang et al., 2020)
- **Farnebäck dense optical flow (ROI):** $\theta(L\ I\ s^2 n_{ROI}) = \theta(L\ I\ s^2\ rN)$ due to multi-scale, iterative, neighborhood operations. (Farnebäck, 2003; OpenCV, 2025)
- **Masking/morphology/contours (ROI):** $\theta(n_{ROI}) = \theta(rN)$.
- **I/O & display (full frame copy/show):** $\theta(N)$.
- **Deque push to pre-roll:** $O(1)$ amortized per frame (append to collections.deque). (Python Software Foundation, 2025a)

Combining terms:

$$T_{frame} = \theta\big(N(1 + LIs^2r)\big)$$

- **Worst case** $r = 1$: $\theta(N(1 + LIs^2))$ (flow dominates).
- **Average case** $r \in [0.05, 0.15]$: flow cost scales linearly with r, which is why ROI restriction is so effective.

Working-set memory

Count only the arrays that are explicitly hold (model weights and library internal omitted):

- **Pre-roll frames (dominant):** $M_{pre} = B * (3N)$ bytes (BGR uint8).
- **Full-frame working copies:** original $3N$ + overlay $3N \Rightarrow 6N$ bytes.
- **ROI grayscale:** current + previous = $2n_{ROI}$ bytes.

- **Flow & masks (ROI):** flow $2xfloat32 \Longrightarrow 8n_{ROI}$; magnitude $float32 \Rightarrow 4n_{ROI}$; binary mask $uint8 \Longrightarrow 1n_{ROI}$.

Total (bytes):

$$M_{total} \approx N(3B + 6 + 15r)$$

Interpretation: pre-roll dominates; ROI-side buffers are comparatively small. So, if memory is tight, reduce buffer (seconds) or the saved FPS. Both shrink B linearly.

Cross-Platform Code Paths

Platform differences are restricted to codec and audio selection. Windows uses XVID/AVI and winsound.Beep; other platforms default to MP4V/MP4 and an ASCII-bell fallback. If either surface proves unsuitable, for example, a muted terminal or a player that lacks the MP4V codec, switching to MJPG for video or to a lightweight WAV playback library for audio does not affect the detection pipeline.

Extensibility Hooks

Several extension points are deliberately left simple. Multiple ROIs can be supported by tracking two hands and either selecting the largest ROI or taking the union; the trigger then fires if either ROI becomes ready. Adaptive thresholds can be derived from scene statistics to account for lighting changes. The alert mechanism can be abstracted behind a small interface so that different audio backends or even OS notifications can be used interchangeably. A log-rotation or pruning policy can be added without touching the core loop. If profiling at higher resolutions reveals bottlenecks, capture, compute, and disk I/O can be pipelined across threads while preserving the unified trigger semantics.

Configuration and Parameters

This section describes the parameters and the default values set. Also a short overview is given about what happens, when these parameters are changed, as well as examples for what to do when issues are faced.

Motion & Recordings

motion_threshold = 3.5

Threshold on optical-flow magnitude (roughly “pixels moved per frame” inside the hand ROI).

- Lower → **more sensitive**; detects small/slow motions but may trigger on noise (sensor grain, compression, flicker).
- Higher → **less sensitive**; ignores small motions and only reacts to larger/faster ones.

- Typical: 2.0–5.0. If raising camera FPS or the subject moves faster, may needing to increase it.

cooldown = 5 (seconds)

After each detected motion, recording continues until there has been no motion for cooldown seconds. Also governs when the system is "armed" again (unified trigger).

- Lower → clips end sooner; might cut off the tail of a motion event.
- Higher → longer clips; fewer files but more dead time at the end.

recording_fps = 20

Target frame rate for the saved video. (Later overwritten with the camera's FPS if it's available and different.)

- Lower ⇒ smaller files, choppier playback, less pre-roll coverage per second.
- Higher ⇒ smoother video, bigger files.
- Note: If the camera's actual FPS is known, the code sets recording_fps to that, and resizes the buffer (see below).

buffer = 2 (seconds)

Pre-motion "pre-roll" length. Frames from the last buffer seconds are saved before the trigger.

- Lower → smaller files but may miss the start of the action.
- Higher → more context before the trigger; bigger files.

buffer_size = int(recording_fps * buffer)

Number of frames held in the pre-motion ring buffer. This updates if the camera FPS is discovered and differs from initial recording_fps.

pre_motion_buffer = deque(maxlen=buffer_size)

Stores those pre-roll frames. Increasing buffer_size increases memory use and file size when a clip starts (since all frames are flushed into the new file).

last_motion_time

Timestamp used to keep a session alive while motion continues. The clip ends when now - last_motion_time > cooldown.

Optical Flow (Farnebäck)

These shape how motion is measured inside the ROI:

pyr_scale = 0.5

Image pyramid scale between levels.

- Lower (e.g., 0.3) → more levels for a given resolution; better large-motion capture, slower.
- Higher (e.g., 0.7) → fewer levels; faster, may miss large motions.

levels = 3

Number of pyramid levels (including the original).

- More → better at capturing large motions; slower.
- Fewer → faster; might miss large/fast movement.

winsize = 15

Window size (px) used for motion estimation.

- Larger (e.g., 21–31) → smoother, more robust to noise; may miss small/local motion, slower.
- Smaller (e.g., 9–13) → more sensitive to small movements; noisier.

iterations = 3

Refinement iterations per level.

- More → more accurate flow, slower.
- Fewer → faster, less stable.

poly_n = 7

Size of the pixel neighborhood for polynomial expansion.

- 5 → sharper sensitivity, more noise.
- 7 → smoother, more robust (good default).

poly_sigma = 1.5

Gaussian smoothing before polynomial expansion.

- Lower (e.g., 1.1) → more detail, more noise.
- Higher (e.g., 1.7–2.0) → smoother, can wash out fine motion.

flags = 0

Special options. You rarely need to change this; OpenCV docs list extra modes (e.g., OPTFLOW_FARNEBACK_GAUSSIAN), which can make flow smoother at the cost of responsiveness.

Motion Mask and Morphology

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))

Used with MORPH_OPEN to clean up the binary motion mask. (Gonzalez and Woods, 2018; Soille, 2004)

- Larger kernel → removes more noise and thin artifacts; may also erode away small/legit motion.
- Smaller kernel → keeps small motion; may leave salt-and-pepper noise.

area_frac > 0.01 (hardcoded threshold)

Triggers "motion_detected" if >1% of ROI pixels exceed motion_threshold.

- Lower (e.g., 0.005) → more sensitive; can trigger on tiny areas/noise.
- Higher (e.g., 0.02–0.05) → requires bigger moving area; fewer false positives.

Contour area filter > 20 (px)

Prevents drawing green rectangles for tiny blobs.

- Lower → more rectangles (noisier visualization).
- Higher → only draws obvious blobs (cleaner view).

Hand ROI (MediaPipe)

These define the region where motion is measured:

static_image_mode = False

- False (streaming) → fast tracking after first detection.
- True → runs detection on every frame (slower) but can re-acquire hands more robustly if tracking is lost.

max_num_hands = 1

Upper bound of hands to track.

- Increase to 2 if two-hand ROI. Also needing to decide how to combine multiple ROIs (union, largest hand, etc.).

min_detection_confidence = 0.5

Confidence threshold for the initial hand detection.

- Lower → more detections (and false positives).
- Higher → fewer detections; more reliable when it fires.

min_tracking_confidence = 0.5

Confidence for tracking the hand over time.

- Lower → tracking sticks more easily (even if uncertain), can drift.
- Higher → stricter; tracking drops more readily when uncertain.

Padding of 10 px around the hand is hardcoded—bigger padding gives more context but may include background motion; smaller padding is tighter but can crop parts of the hand.

Pausing and Arming

Paused

Toggles with the 'P' key. When True, detection and recording stop (UI still updates).

resume_grace = 2.0 (seconds)

After unpausing, the system ignores motion for this long to avoid immediate false triggers (e.g., from touching the keyboard/camera).

- Lower → re-arms faster; more likely to trigger on resume jitters.
- Higher → safer but briefly unresponsive.

resume_until

Timestamp until which motion is ignored after resuming.

motion_arm_frames = 3

Requires N consecutive frames with motion before arming the trigger. Debounce for stability.

- Lower (1–2) → faster triggers; more false positives.
- Higher (4–6) → only sustained motion triggers; might miss quick gestures.

motion_frames

Counts the current consecutive motion frames; reset to 0 on a quiet frame.

Video I/O and Files

cap = cv2.VideoCapture(0)

Default webcam. Swap to a file by uncommenting one of the paths.

frame_width, frame_height

Set lazily from the first good frame. Don't hardcode unless knowing the input.

fourcc / file_ext

Codec differs by OS for compatibility:

- Windows: XVID → **.avi**
- Others: mp4v → **.mp4**
  If seeing unreadable files, try MJPG (large files, broad support) or H264 (needs system codecs).

video_writer

When None, it's not recording. A new writer is created at trigger and closed after the quiet cooldown window.

movement_clips/

Target folder for saved files (auto-created). Clip names are timestamped.

UI and Misc

WINDOW_NAME = "Motion detection (ROI)"

Title of the OpenCV window.

ui_step()

- q / Esc to quit.
- p to pause/resume (triggers the resume_grace ignore window).

Beep in play_alert()

- On Windows: winsound.Beep(1000, 500) (1 kHz, 0.5 s).
- Else: prints ASCII bell (\a); whether you hear it depends on your terminal/OS.

log_movement_csv()

Appends a row per event with Timestamp, Filename, Event. If the file doesn't exist, it writes a header once.

Auto-tuning that happens in loop

Camera FPS adaptation

On the first valid frame, the script reads cam_fps. If it's valid and differs from initial recording_fps, it updates:

- recording_fps = cam_fps

- buffer_size = int(recording_fps * buffer)
  This keeps pre-roll duration accurate in seconds and avoids jerkiness from FPS mismatch.

What to do if …

Too many false triggers (noisy background / low light):

1. Raise motion_threshold a bit (e.g., 3.5 → 4.5).
2. Increase area_frac threshold (edit the code: 0.01 → 0.02).
3. Increase morphology kernel to (7,7) or increase winsize to 21.

Missing subtle/slow movements:

1. Lower motion_threshold (3.5 → 2.5–3.0).
2. Lower area_frac (0.01 → 0.007–0.008).
3. Decrease morphology kernel to (3,3) and/or winsize to 13.

Clips are too choppy or too big:

- Choppy: raise recording_fps (or ensure camera FPS is read).
- Too big: lower recording_fps, reduce buffer, or shorten cooldown.

Triggers happen too easily on quick blips:

- Increase motion_arm_frames (3 → 5).
- Slightly raise motion_threshold or area_frac.

ROI keeps disappearing / re-acquire issues:

- Set static_image_mode=True (slower but re-detects robustly).
- Increase min_detection_confidence a bit (0.5 → 0.6–0.7).

Validation Plan

Given the absence of participant testing, validation uses engineering-style inputs and deterministic procedures to verify functionality and robustness. Configuration is frozen for all runs (see Configuration and Parameters), the detector processes files for determinism in V1, and live input for V2/V3. Event start/end rows are instrumented in event_log.csv, logging float timestamps (t_trigger, t_last_motion, t_end) and counts (pre_roll_frames, frames_written) for automatic checks.

V1. File-based Deterministic Tests

- **Idle baseline (no hand):** 2–3 min of empty scene. **Expected:** 0 triggers, 0 clips, 0 CSV rows beyond startup.
- **Scripted bursts:** 1–2 min clip with 5–10 hand-motion bursts, each ≈3 s, separated by ≥ cooldown + 1 s, hand fully in ROI. **Expected:** 1 alert + 1 CSV + 1 clip per burst, each clip with ~buffer s pre-roll and ~cooldown s tail.

V2. Live-loop Sanity

- **Repeated cycles:** Produce 3–5 motion cycles; wait for cooldown; repeat. **Expected:** re-arming after each cooldown; all three artifacts (alert, CSV, clip) per cycle.
- **Pause/resume:** While paused, move the hand—no triggers. After resume, ignore during grace, then resume normal triggering.

V3. Edge & Stress

- **ROI loss:** Move the hand out of frame mid-recording. **Expected:** clip ends after cooldown; next motion re-arms cleanly.
- *(Not executed in this submission; planned work)* **Background flicker outside ROI** and **parameter sweeps** (vary motion_threshold, motion_arm_frames, morphology kernel) to probe stability; **runtime notes** (FPS/CPU idle vs recording).

Metrics & Acceptance

- **Trigger correctness:** fraction of scripted bursts that produce exactly one alert, CSV, clip. Target ≥95%.
- **Latency:** onset latency ≈ (motion_arm_frames−1)/FPS + compute; verify within ±1 frame.
- **Clip integrity:**
  - Pre-roll (checked by frames): $pre\_roll\_frames \approx buffer * FPS$ within ±1 frame.
  - Tail (checked by time): $tail\_s = t\_end - t\_(last\_motion) \geq cooldown - 0.10s$. (time-based tail is used because realized loop FPS varies during recording.)
- **Stability:** no deadlocks/crashes across ≥1000 frames continuous.

Artifacts to retain: movement_log.csv, event_log.csv, list of produced clips (name, start/stop timestamps, duration), and a brief note on FPS/CPU.

Results

V1. File-based Deterministic Tests

Idle baseline (no hands)

Duration: 2 minutes. **Observed:** 0 triggers, 0 clips, 0 CSV rows beyond startup. – *Meets expectations*.

Scripted bursts

Input contained 5 induced bursts. The detector produced 5 events (1:1 mapping, trigger correctness 100% for this run). Pre-roll matched the configured value in all events; tail-by-time passed in 4/5 events; the single failure occurred because the input video ended before the cooldown completed, i.e., not a detector error.

Table 1: Scripted-bursts integrity (per event) lists the results of the validation tests.

Table 1: Scripted-bursts integrity (per event)

Clip (filename)	fps_nominal	realized_fps	recording_s	pre-roll (exp)	tail_s (min)	pre-roll_ok	tail_ok	PASS
movement_2025-09-10_09-38-04.avi	30.01	28.48	6.81	60 (60)	5.06 (5.00)	True	True	**True**
movement_2025-09-10_09-38-16.avi	30.01	27.92	7.27	60 (60)	5.05 (5.00)	True	True	**True**
movement_2025-09-10_09-38-29.avi	30.01	28.26	8.74	60 (60)	5.07 (5.00)	True	True	**True**
movement_2025-09-10_09-38-43.avi	30.01	28.47	6.74	60 (60)	5.04 (5.00)	True	True	**True**
movement_2025-09-10_09-38-56.avi	30.01	n/a	n/a	60 (60)	3.63 (5.00)	True	**False**	**False**

Aggregate (four valid tails):

- Tail seconds: mean 5.055 s, SD 0.011 s (≈11 ms).
- Realized FPS (post-trigger): mean 28.28 FPS, SD 0.23 FPS.
- Recording duration (post-trigger): mean 7.39 s, SD 0.81 s.
- Pre-roll frames: always 60/60 (+-0).

Interpretation: Clip integrity requirements are met whenever the input continues past the cooldown. The sole FAIL is explained by early input termination (tail_s < cooldown), which is expected and does not indicate a detector defect.

V2. Live-loop Sanity (live runs)

Across 5 cycles, the detector re-armed after each cooldown and produced one alert, one CSV row, and one clip per cycle. During pause, deliberate hand motion generated no events; after resume, motion within the grace interval was ignored, and a single event was produced only after the grace window elapsed. These outcomes confirm that re-arming, pause gating, and grace behavior match the specification.

V3. Edge & Stress (live runs)

- **ROI loss:** When the hand exited the frame mid-recording, the clip was recorded until the cooldown elapsed; a subsequent motion re-armed and triggered a new event.
- *(Not executed in this submission)* **Background flicker outside ROI** and **parameter sweeps**; **runtime notes** (FPS/CPU) are planned.

Discussion

Summary of findings. The system demonstrates deterministic pre-roll fidelity (exactly buffer·FPS frames) and time-accurate cooldown tails when the input continues beyond the last motion. In scripted-burst testing, four of five events passed both integrity checks; the single failure is attributable to early end of the source video, not to the detection or lifecycle logic. Live-loop testing confirms correct re-arming, pause gating, and grace behavior.

Lifecycle correctness and interpretability. The unified trigger aligns the audible alert, CSV entry, and clip onset to the same motion edge, while the pre-roll and cooldown-tail constraints ensure each clip contains both lead-up context and a termination. This is exactly the structure clinicians or reviewers need for retrospective interpretation.

Realized vs nominal FPS. Post-trigger realized loop rates were ~28–29 FPS against a nominal 30 FPS, which is expected given CPU-only MediaPipe inference, ROI optical flow, GUI, and disk I/O. Because tail acceptance is time-based, not frame-based, small FPS deviations do not compromise correctness.

Scope and remaining tests. Within the current submission, V3 tests for background flicker outside ROI and parameter sweeps were not executed; they remain recommended to characterize robustness margins (e.g., how motion_arm_frames, motion_threshold, and morphology kernel trade sensitivity vs stability) and to capture runtime notes (idle vs recording FPS/CPU). These additions would further strengthen external validity without altering the core methodology.

Limitations. The evaluation does not include patient/caregiver usability, diverse lighting/camera conditions, or overnight soak runs; it targets engineering correctness under controlled conditions. The system is designed as an assistive research tool, not a medical device, and should be integrated into clinical workflows with appropriate oversight.

Implications and next steps. Results support the architectural choices of ROI-restricted dense optical flow, temporal debounce, and a cooldown-based session model. If broader deployment or higher resolutions are required, practical refinements include (i) tightening ROIs or reducing flow cost (winsize, iterations) to increase headroom, (ii) adding a storage pruning policy, and (iii) executing the deferred V3 tests to document robustness envelopes.

Deployment and Operations

Environment & Setup

Deployments require Python 3.x with OpenCV, MediaPipe, and NumPy installed. Before first use, verify that the camera device can be opened and that the platform has at least one compatible video codec. As part of commissioning, record a short clip and confirm it plays on the target workstation; if playback fails, switch the writer to a broadly supported codec such as MJPG and retry.

Run-time Operations

To operate the system, launch the program and select either a live camera or a file source. During execution, the display renders diagnostic overlays—an ROI rectangle in blue and motion blobs in green—to support situational awareness. The operator can pause and resume processing with **p** (note that a brief "grace" interval suppresses motion decisions immediately after resuming) and can terminate with **q** or **Esc**. Disk space should be monitored proactively to ensure the movement_clips/ directory retains headroom for new recordings.

Storage and Retention

Expected storage consumption depends on the event rate and the clip length (pre-roll + active motion + cooldown). As a rule of thumb, daily storage can be approximated by $daily\ storage \approx \frac{events}{day} * (buffer + event + cooldown) * recording\_fps * frame\_size * 3bytes$, where frame_size is W×H and the factor 3 accounts for 8-bit BGR frames. When quotas are tight, reduce buffer, shorten cooldown, or lower recording_fps. A simple rotation policy—either a maximum total size or a "days kept" window—should be enforced to prevent growth without bound. For reliable timestamps that align with clinical notes, enable clock synchronization (e.g., NTP) on the host.

Alerting

On Windows, audible cues are produced with winsound.Beep, which provides a reliable, zero-dependency alert mechanism. On Linux and macOS, the terminal bell may be muted; prefer a lightweight WAV playback backend to guarantee audibility, keeping the alert semantics identical across platforms (Python Software Foundation, 2025b). Alerts are intentionally local; if remote notification is required, add a secondary channel—such as an OS notification or message queue—without changing the trigger timing or clip lifecycle.

Security and Privacy

Access to the host filesystem should be restricted to authorized users, with OS-level permissions applied to the movement_clips/ directory. On portable systems, whole-disk encryption is recommended. Retention and deletion procedures must follow institutional policy and be documented. For clinical deployment, adhere to ethics and data-protection requirements: favor de-identification by default (e.g., crop or mask to the hand ROI; avoid storing audio; redact overlays that could reveal identity) and adopt camera-placement practices that minimize incidental capture of faces or third parties (frame the bed/hand area, avoid doorways or whiteboards). Coordination with the IRB or data-protection officer is essential to ensure appropriate consent, signage, retention, and access control for the care setting.

Monitoring & Health

Routine health checks should confirm there is sufficient free space and that clip and log writes are succeeding. A lightweight "heartbeat" entry at startup and after every *N* frames can aid troubleshooting and provide a coarse uptime signal. If desired, a periodic self-test can attempt a short write to a temporary file to verify end-to-end I/O integrity without interrupting normal operation.

Maintenance and Extensibility

Routine Maintenance

- Rotate/prune old clips and CSV logs.
- Pin library versions; re-run the **idle baseline** and **scripted bursts** after upgrades.
- Periodically re-tune motion_threshold and motion_arm_frames if the camera, lighting, or FPS changes.

Extensibility Roadmap

- **Multi-ROI**: track two hands; union or largest-ROI strategy; trigger if either arms.
- **Multi-ROI – add feet and eye-movement:** track also feet movement and/or eye movement for more accurate motion detection.

- **Adaptive thresholds**: rolling quantiles of magnitude to auto-tune sensitivity.
- **AlertSink abstraction**: interchangeable audio backends (WAV/OS notify/silent test).
- **Alert messages**: inform medical staff about motion through e-mail, WhatsApp (WiFi required) or SMS (GSM-Module required).
- **Codec policy**: runtime detection with automatic fallback to MJPG.
- **Threaded pipeline**: decouple capture, compute, and I/O if higher resolutions are required.
- **Metadata sidecar**: JSON per clip for parameters and environment hashes (useful for audits).

Risks and Mitigations

False negatives (missed subtle motion)

- *Risk:* Motion below magnitude/area thresholds or during low-light noise suppression.
- *Mitigation:* Lower motion_threshold, reduce morphology kernel, or increase recording_fps.

False positives (lighting flicker, background)

- *Risk:* Brightness changes within ROI; reflective surfaces.
- *Mitigation:* Increase motion_arm_frames, raise motion_threshold and area fraction; stabilize lighting.

ROI loss or drift

- *Risk:* Hand leaves frame; occlusion.
- *Mitigation:* Use static_image_mode=True temporarily; enlarge padding; consider last-seen ROI fallback.

Storage exhaustion

- *Risk:* High event rates, large cooldowns, long retention.
- *Mitigation:* Prune by age/size; cap daily clip count; monitor free space.

Muted alerts / OS differences

- *Risk:* ASCII bell inaudible.

- *Mitigation:* Replace with WAV playback or OS notifications; add visual popups if required.

Operational misuse (medical interpretation)

- *Risk:* Users over-trust alerts.
- *Mitigation:* Prominent disclaimer; training; pair with clinical workflow rather than replace it.

Conclusion

This work delivers a hand-focused, ROI-restricted motion-detection pipeline that couples dense optical flow with temporal debounce and a unified trigger to produce synchronized alerts, logs, and context-rich clips. Within the present constraints, engineering validation demonstrates reliable event lifecycle behavior, predictable latency, and real-time operation on commodity CPUs. Future extensions target multi-ROI, adaptive sensitivity, and portable alert/codec backends, with a long-term aim of integrating into broader monitoring workflows—always as an assistive tool, not a standalone clinical device.

References

Barnich, O., Van Droogenbroeck, M., 2011. ViBe: A Universal Background Subtraction Algorithm for Video Sequences. IEEE Trans. on Image Process. 20, 1709–1724. https://doi.org/10.1109/TIP.2010.2101613

Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., Grundmann, M., 2019. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. https://doi.org/10.48550/arXiv.1907.05047

Beauchemin, S.S., Barron, J.L., 1995. The computation of optical flow. ACM Comput. Surv. 27, 433–466. https://doi.org/10.1145/212094.212141

Farnebäck, G., 2003. Two-Frame Motion Estimation Based on Polynomial Expansion, in: Bigun, J., Gustavsson, T. (Eds.), Image Analysis, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 363–370. https://doi.org/10.1007/3-540-45103-X_50

Gonzalez, R.C., Woods, R.E., 2018. Digital image processing, Fourth, global edition. ed. Pearson Education, New York, New York.

Google Ireland Limited, 2025. Hand landmarks detection guide [WWW Document]. Hand landmarks detection guide. URL https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker?hl=de (accessed 9.4.25).

Hannawi, Y., Abers, M.S., Geocadin, R.G., Mirski, M.A., 2016. Abnormal movements in critical care patients with brain injury: a diagnostic approach. Crit Care 20, 60. https://doi.org/10.1186/s13054-016-1236-2

Hari, K.C., Shrestha, S., Pokharel, M., 2023. Video Object Motion Tracking using Dense Optical Flow Techniques, in: 2023 Eighth International Conference on Informatics and Computing (ICIC). Presented at the 2023 Eighth International Conference on

Informatics and Computing (ICIC), IEEE, Manado, Indonesia, pp. 1–4. https://doi.org/10.1109/ICIC60109.2023.10381969
Horn, B.K.P., Schunck, B.G., 1981. Determining optical flow. Artificial Intelligence 17, 185–203. https://doi.org/10.1016/0004-3702(81)90024-2
Hu, J., Chen, C., Wu, M., Zhang, J., Meng, F., Li, T., Luo, B., 2024. Assessing consciousness in acute coma using name-evoked responses. Brain Research Bulletin 218, 111091. https://doi.org/10.1016/j.brainresbull.2024.111091
Lucas, B.D., Kanade, T., 1981. An Iterative Image Registration Technique with an Application to Stereo Vision, in: Proceedings of Imaging Understanding Workshop. pp. 121–130.
OpenCV, 2025. OpenCV - Optical Flow [WWW Document]. URL https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html (accessed 9.4.25).
Python Software Foundation, 2025a. python collections — Container datatypes [WWW Document]. URL https://docs.python.org/3/library/collections.html (accessed 9.8.25).
Python Software Foundation, 2025b. winsound — Sound-playing interface for Windows [WWW Document]. URL https://docs.python.org/3/library/winsound.html (accessed 9.10.25).
Sengar, S.S., Mukhopadhyay, S., 2017. Foreground Detection via Background Subtraction and Improved Three-Frame Differencing. Arab J Sci Eng 42, 3621–3633. https://doi.org/10.1007/s13369-017-2672-2
Soille, P., 2004. Morphological Image Analysis: Principles and Applications, Second Edition Corrected second printing. ed. Springer Berlin Heidelberg, Berlin, Heidelberg s.l. https://doi.org/10.1007/978-3-662-05088-0
Stauffer, C., Grimson, W.E.L., 1999. Adaptive background mixture models for real-time tracking, in: Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149). Presented at the Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Comput. Soc, Fort Collins, CO, USA, pp. 246–252. https://doi.org/10.1109/CVPR.1999.784637
Thapa, G., Sharma, K., M.K.Ghose, M.K.G., 2014. Moving Object Detection and Segmentation using Frame Differencing and Summing Technique. IJCA 102, 20–25. https://doi.org/10.5120/17828-8647
Yimer, H.L., Degefa, H.D., Cristani, M., Cunico, F., 2024. IoT-Based Coma Patient Monitoring System. https://doi.org/10.48550/arXiv.2411.13345
Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., Grundmann, M., 2020. MediaPipe Hands: On-device Real-time Hand Tracking. https://doi.org/10.48550/arXiv.2006.10214
Zhang, Y., Wang, X., Qu, B., 2012. Three-Frame Difference Algorithm Research Based on Mathematical Morphology. Procedia Engineering 29, 2705–2709. https://doi.org/10.1016/j.proeng.2012.01.376
Zivkovic, Z., 2004. Improved adaptive Gaussian mixture model for background subtraction, in: Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. Presented at the Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., IEEE, Cambridge, UK, pp. 28-31 Vol.2. https://doi.org/10.1109/ICPR.2004.1333992

Zvorișteanu, O., Caraiman, S., Manta, V.-I., 2022. Speeding Up Semantic Instance Segmentation by Using Motion Information. Mathematics 10, 2365. https://doi.org/10.3390/math10142365