# Methodology

This section outlines the techniques and algorithms that were used in the project. The script performs real-time motion detection within the regions of interest (ROI) around a detected hand. For hand-localizing to set the ROI, MediaPipe is used, the motion detection inside the ROI is performed by using Farnback dense optical flow. Automated video recording of motion events and CSV logging of the timestamps are performed.

## 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 could be used and adapted directly. (Zhang et al., 2020)

First, the captured BGR frames are converted to RGB, as this is required by MEdiapipe. 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.

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 Farneback Dense Optical Flow

Sparce optical flow and dense optical flow are two types of optical flow. According to Zvoristeanu et.al. (2022) in sparce 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.

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.

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Reason |
| pyr\_scale | 0.5 | Each level is half the resolution → smooths motion detection across scales. Standard and efficient. |
| levels | 3 | Balances performance and accuracy. Tracks motion at multiple resolutions without too much processing. |
| winsize | 15 | Moderate window size — good for detecting smalllocal motion like fingers. |
| iterations | 3 | Enough refinement at each level without adding delay. |
| poly\_n | 7 | Larger neighborhood — adds stability to the flow field, reducing false positives from camera noise. |
| poly\_sigma | 1.5 | Smoother input for polynomial fit → good for slow, small, smooth motion such as finger twitches. |
| flags | 0 | Standard mode. Keeps it simple. |

The dense optical flow method from Farneback estimates per-pixel motion between two grayscale frames, enabling robust motion vector computation.

The key steps include:

* Convert current and previous ROI frames to greyscale
* Apply cv2.calcOpticalFlowFarnback() to estimate flow vectors
* Convert Cartesian flow vectors (dx, dy) to polar coordinates (magnitude, angle)

Then, significant motion events are isolated by applying motion threshold to the magnitude matrix. Binary thresholding followed by morphological opening removes small noisy regions.

## Motion validation and event triggering

To evaluate a motion event, contours are detected in the thresholded magnitude mask. Any contour with an area bigger than 20 pixels is treated as valid motion. When a motion is detected, a beep sound is played to inform the user. Additionally, a timestamped video clip is saved, including frames captured 2 seconds before motion, which were saved in a rolling buffer and are described further in the next section. Also, a CSV log is appended with timestamp and the filename of the captured video. To prevent repeated triggering, a cooldown period (default is 5 seconds) is used.

## Frame buffering and recording

A deque buffer holds pre-motion frames to make sure that the then saved motion-video contains context leading up to the event. When motion is detected, a new cv2.VideoWriter object is created and frames from the buffer and real-time feed are written to disk. The recording stops after the cooldown period elapses.

# System Design

## Architecture Overview

## Functional Modules

|  |  |
| --- | --- |
| Module | Functionality |
| Capture Module | Accesses the webcam or video file and fetches frames |
| Detection Module | Uses Mediapipe to detect hand landmarks and extract ROI |
| Motion Module | Applies optical flow and motion validation within the ROI |
| Alert Module | Plays a beep when motion is detected |
| Recording Module | Saves video from a rolling buffer to disk using cv2.VideoWriter |
| Logging Module | Writes motion events to a CSV log with metadata |
| Display Module | Renders output frame with motion rectangles and annotations using OpenCV GUI |

## Data Flow

Temporal Data:

* Each frame feeds the hand detector and is added to the buffer
* Motion is computed only within the ROI (hand area), reducing noise and computational expensiveness.
* The buffer ensures that also frames before the movement begins are recorded.

Logical Flow Control:

* The system state includes paused, recently\_alerted, and video\_writer.
* A keyboard interface allows toggling between pause/resume.
* Cooldown logic prevents from excessive alerts.

## Design Choices

* ROI-restricted motion analysis: Limits false positives from body/background movement, lowers area for computationally expensive dense optical flow
* Optical flow over frame differencing: more robust to lightning and noise variations
* Pre-buffered video saving: Provides pre-event context
* Sound alerts: Uses winsound for Windows, must be changed when using others