A Low-Cost and Low-Complexity Motion Sensor System Using IMU and Video-Based Tracking

CM2015 HT25-1 Project Carrier Course for Medical Engineers, part 1 15.0 credits

Jack Hutton (hutton@kth.se)

Alexander Jonsson (alejon2@kth.se)

Lisa Kanehl (kanehl@kth.se)

Sahand Khodabandehloo (sahandkh@kth.se)

Lucia Perez Saez (luciaps@kth.se)

Kungliga Tekniska högskolan
KTH Royal Institute of Technology

2025-12-18

Abstract

Motion tracking is widely used in healthcare, sports, rehabilitation, and human–computer interaction, but many commercial systems are costly and rely on specialized hardware. This project investigates a low-cost, low-complexity alternative that combines inertial measurement unit (IMU) signals with video-based pose estimation in a standard web browser. We implemented a browser application that imports IMU recordings, estimates 2D body keypoints from live or uploaded video, and visualizes both modalities side by side. A simple event-based synchronization method aligns video time with IMU time using manually marked reference points. The resulting prototype enables basic inspection and annotation of synchronized motion data without dedicated equipment.

1 Introduction

1.1 Motivation

Motion tracking refers to the use of sensing technologies to capture, record, and quantify human movement. Depending on the application, this can be done with camera-based systems, wearable sensors, or a combination of both, producing measurable quantities such as joint positions, velocities, and movement trajectories. By representing movement as numerical data, motion tracking enables systematic analysis beyond subjective visual assessment.

Objective movement measurement is especially relevant in healthcare, sports science, and rehabilitation, where movement quality, efficiency, and consistency directly influence clinical decisions and training outcomes. Quantitative motion data can support the detection of movement deficits, track progress over time, and provide feedback for technique improvement in a reproducible way. In practice, combining video observations with IMU signals can also support technique assessment in sports and progress monitoring in rehabilitation, and it can provide additional context for exercises performed in home-based or remote settings.

Despite their potential, many established motion tracking solutions remain difficult to deploy in practice. Commercial systems are often expensive, require specialized hardware, and depend on controlled laboratory or studio environments. These barriers limit the availability of objective motion analysis in real-world clinical and training settings.

1.2 Research Objective

The objective of this project is to design and implement a low-cost, low-complexity motion tracking prototype that runs in a standard web browser. The system combines IMU recordings with video-based pose estimation and provides a practical method to synchronize both modalities in time, enabling side-by-side visualization and basic annotation of motion data without dedicated equipment.

1.3 Aim and Research Question

The aim of this project is to develop a motion-tracking prototype that is low-cost, easy to use, and accessible without specialized equipment. To achieve this, we combine two data sources: (1) inertial measurement unit (IMU) recordings and (2) video-based pose estimation in the browser. A key challenge is that both modalities have different time bases, so the system must also provide a practical method to synchronize IMU signals with the corresponding video frames.
This leads to the following research questions: How can IMU data and video-based pose estimation be integrated into one browser application for motion analysis?

1.4 Requirements

The system should be low-cost and avoid specialized motion capture equipment. It should run in a standard web browser without complex installation. The interface should support an easy workflow for importing IMU recordings, analyzing uploaded videos (and optionally live webcam video), visualizing both data sources, and synchronizing them in time. Basic interaction should include video playback with a pose overlay, IMU plots, and simple synchronization markers.

1.5 Report Structure

The report first introduces the motivation, objectives, and background on IMU's, pose estimation, and time synchronization. It then presents the system design and methods, followed by implementation details of the web application. Finally, the results are demonstrated and discussed, including limitations, ethical considerations, and directions for future work.

2 Background

2.1 Inertial Measurement Unit (IMU)

An Inertial Measurement Unit (IMU) is a sensing device that captures motion using three-axis (*x*-*y*-*z*) sensors: an accelerometer for linear acceleration, a gyroscope for angular velocity, and a magnetometer for orientation relative to the Earth's magnetic field. In this project, Movesense IMUs were used. Although the hardware provides additional functions (e.g., heart-rate features and ruggedized design), this study considered only their motion sensors as well as wireless data transfer and onboard storage for logging measurements.

2.2 Video Pose Estimation

Video pose estimation is a computer-vision method that detects a set of human body keypoints (e.g., shoulders, elbows, hips, and knees) in each video frame. A convolutional neural network (CNN) processes an RGB image and outputs keypoint coordinates together with confidence scores, which can then be drawn as an overlay ("skeleton") on top of the video. In TenserFlow.js, PoseNet and MoveNet both follow this idea and return 17 standard body points.

2.2.1 PoseNet

Posenet was one of the first widely used pose-estimation models available directly in the browser vis TenserFlow.js. The TensorFlow.js PoseNet release emphasized accessibility (run locally in the browser, no server needed) and an API that can be used with only a few lines of JavaScript.

A practical advantage of PoseNet is that developers can tune parameters that trade speed for accuracy, such as the image scale factor and output stride (lower stride → higher accuracy but slower).

PoseNet is well suited for initial prototyping because it is widely documented within the TensorFlow.js ecosystem and provides configurable inference parameters that allow systematic tuning of the speed–accuracy trade-off.

2.2.2 MoveNet

MoveNet is a newer pose detector released with the TensorFlow.js pose-detection API. It is designed to deliver a strong speed–accuracy balance and to run fully client-side in the browser after the initial model load.

Compared to PoseNet, MoveNet is engineered for high frame rates and stable tracking in dynamic movements: the official TensorFlow blog describes optimisations such as intelligent cropping and temporal filtering to reduce jitter while keeping responsiveness.

With the API, createDetector loads and configures the pose model once. Then estimatePoses is called for each video/webcam frame: it runs the model on the current image and returns the detected pose(s) as 2D keypoints (x,y) with confidence scores.

MoveNet provides two single-person variants: Lightning and Thunder. Lightning targets latency-critical use and Thunder targets higher accuracy. A key technical difference is the fixed model input resolution: the TensorFlow team reports 192×192 inputs for Lightning and 256×256 for Thunder, which is part of the speed/quality trade-off. [1]

2.3 Synchronization

When combining video-based motion tracking with IMU recordings, both modalities typically follow independent time bases.

Synchronization therefore refers to establishing a consistent mapping between the video timeline and the IMU timeline such that corresponding samples represent the same physical instant. A common and practical approach is event-based synchronization, where a distinct, clearly identifiable motion event (e.g., a clap, jump landing, or sudden knee flexion) is used as a reference point in both signals.

Let t_{Video} denote the timestamp of the reference event in the video and t_{IMU} the corresponding timestamp in the IMU recording. The temporal alignment can then be expressed as a constant time offset $\Delta t = t_{Video} - t_{IMU}$. Using this offset, any IMU sample at time t_{IMU} can be mapped to the video time. This assumption is appropriate when the dominant discrepancy between both data streams is a fixed start-time difference; more complex effects such as clock drift would require additional reference points or a more advanced alignment model.

Once synchronized, the IMU signals can be interpreted directly in the context of the observed movement in the video. This improves interpretability by linking quantitative measurements (e.g., accelerations and angular velocities, and potentially derived kinematic variables) to the corresponding visual motion, which supports clearer analysis and reduces ambiguity during inspection and annotation.

3 Methods

3.1 System Overview

The system processes two inputs: a video recording and an IMU recording imported from a CSV file. From the video, it estimates 2D body keypoints and renders them as a skeleton overlay during playback. From the IMU file, it generates timestamped sensor channels and visualizes them as interactive time-series charts. The application also stores user-defined synchronization markers and annotations and supports export/import to save and reload complete analysis sessions.
Internally, the workflow consists of a video module (frame decoding and pose estimation), an IMU module (CSV parsing and chart visualization), a synchronization module that aligns both timelines via a constant time offset derived from reference events, and a session module for persistence. The main use case is uploaded-video analysis for reliable inspection of recordings; a live webcam mode is included as an optional feature for real-time demonstration.

3.2 Development Approach

The system was developed iteratively. First, two separate browser prototypes were implemented in parallel: one focusing on video-based pose tracking and skeleton visualization, and one focusing on importing and plotting IMU CSV data. After both pipelines were validated independently, they were merged into a single application to enable joint inspection. In subsequent iterations, an event-based synchronization mechanism was added, followed by session export/import to preserve analysis state, and finally an annotation feature to store timestamps, events, and notes within a session.

3.2.1 IMU Subsystem

This project uses two Movesense HR2 IMU sensors. Connection with these devices and data extraction was possible with the mobile Movesense Showcase app, however for ease of use and overall simplicity it was decided to directly connect the sensors to a laptop device through a custom python application.
For motion analysis, we focus on the tri-axial accelerometer, gyroscope, and magnetometer signals, while other device features are not considered. Recordings are stored and transferred wirelessly and are provided to the web application as CSV files for offline analysis.
In early iterations, a lightweight python Prototype was used to validate Bluetooth connectivity, live streaming, and CSV logging before migrating the processing pipeline to the browser. The code from this app was used as the basis for the initial IMU Sensor WebApp, coded with HTML and JavaScript. Initially, this contained the same features as the python app, with the addition of a video player, seen in Figure 4.

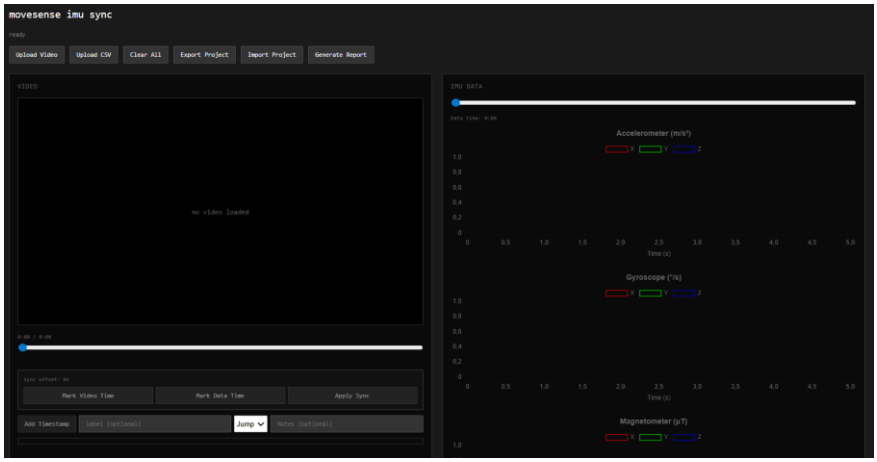

Figure 1. First version of the IMU Video WebApp

This WebApp was iterated several times as new features were added, and the project scope shifted. When the IMU data visualization was complete, the code was merged into the Skeleton Tracking webapp to allow for simultaneous Video and IMU data viewing.

3.2.2 Video Subsystem: Pose Estimation

The video subsystem performs single-person 2D pose estimation directly in the browser. Frames are obtained from an HTML video element, and the estimated pose is rendered on a synchronized canvas overlay. The primary operating mode is offline analysis of uploaded videos, while a live webcam mode is provided as an optional real-time variant.

Pose estimation is implemented with the TensorFlow.js pose-detection API using two MoveNet detectors: MoveNet Thunder is used for uploaded-video analysis, and MoveNet Lightning is used for live tracking. The detector is initialized once and pose inference is performed by calling estimatePoses(...), which returns 2D keypoints with confidence scores.

To improve usability on consumer devices, the upload pipeline applies frame skipping: pose estimation is executed only every Nth frame (with N=2 in the current configuration) and the most recent pose is reused for intermediate frames. In addition, uploaded video playback is set to a reduced rate (0.75) to maintain stable rendering and inference under limited computational resources. In live mode, inference is performed every frame to prioritise responsiveness. Keypoint quality is controlled via confidence thresholds: low-confidence keypoints are visually deemphasized and skeleton connections are omitted if either endpoint falls below the minimum confidence.

3.2.3 Integrated Browser App for Side-by-Side IMU–Video Inspection

After the IMU visualization and the video-based skeleton tracking were developed in parallel, both were integrated into a single browser application to enable side-by-side inspection of an uploaded video together with the corresponding IMU recording. In upload mode, users load a video and an IMU CSV file; the CSV is parsed into sensor channels with a time axis derived

from the fixed sampling rate and displayed as three stacked time-series charts with an interactive “data-time” slider.
Temporal alignment is performed via event-based synchronization. The user marks a reference event in the video and the corresponding event in the IMU view. When “Apply sync” is pressed, the system computes and stores a global syncOffset from the two markers and persists it as part of the project state.
During coupled inspection, the stored offset is applied whenever the video time changes: the IMU view is updated by converting the current video time to the corresponding IMU time (adjustedTime = videoTime − syncOffset) and displaying a sliding time window with a vertical marker at the aligned instant. This update runs both when the video plays without active pose tracking and during tracking.
The approach assumes a constant start-time offset between both modalities (no explicit drift model is applied).

3.3 Session management

3.3.1 Storage, Export and Import

To support reproducible analysis across sessions, the application maintains a per-video project state consisting of the synchronization offset, timestamp annotations, and associated metadata. This state is persisted in the browser via localStorage using a key derived from the active video file (filename and file size), enabling automatic restoration when the same video is loaded again.
For portability and sharing, the project can additionally be exported as a ZIP archive containing a project.json file (sync offset, timestamps, sampling rate, creation time, and notes) and, if available, the uploaded video and IMU CSV stored under dedicated subfolders. Import supports restoring the full ZIP project by reloading both files and metadata into the application state; alternatively, a metadata-only JSON import is supported, in which case video and CSV must be provided separately.
The following figures show the workflow for saving video and CSV files.

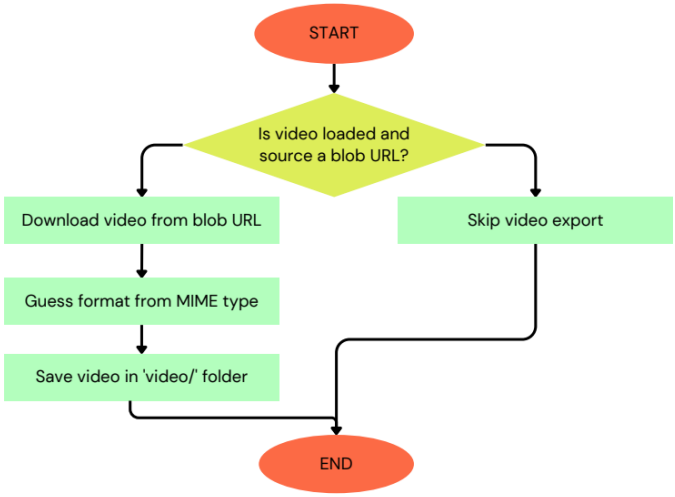

Figure 2. Workflow for exporting the video file to the video/ folder.

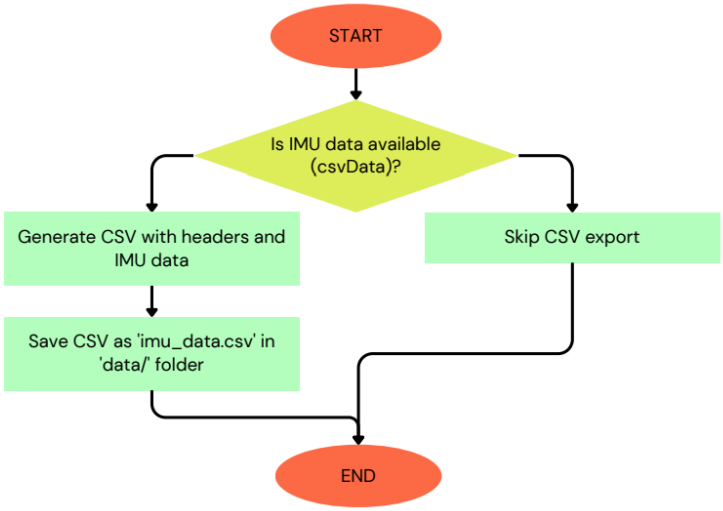

Figure 3. Workflow for exporting the IMU CSV file to the data/ folder.

3.3.2 Annotation model: timestamps, events, and notes

Annotations are represented as an ordered list of timestamp objects with the fields {time, label, eventType, notes}. The stored time value refers to the video timeline, which ensures that annotations remain stable with respect to the visual reference and can be mapped to IMU time through the synchronization offset. The event type is selected from a small, predefined set (Clap/Jump/Other), and optional free-text notes can be attached to each entry.

Selecting an annotation seeks the video to the stored time and updates the IMU view accordingly, allowing qualitative inspection of the corresponding motion together with the quantitative sensor signals.

3.3.3 Report generation

A lightweight reporting function serialises the current project state into a human-readable TXT file. The total recording duration is derived from the IMU time axis (using the last value of csvTimesSeconds), and the number of stored annotations is taken from the current timestamps list.

Each annotation is written as an indexed entry containing the label, the stored event time (kept in video time via video.currentTime), the event type, and optional free-text notes. The report text is then generated fully client-side by creating a UTF-8 Blob and triggering a download through a temporary object URL, avoiding any server-side processing.

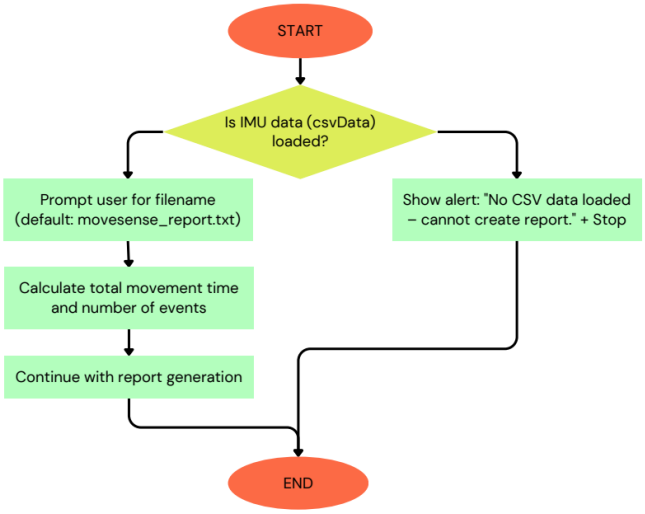

Figure 4. Workflow for creating the IMU report .txt file from the loaded CSV.

3.3.4 Key implementation decisions and limitations

Several implementation choices were made to maintain usability on consumer hardware. For uploaded video analysis, pose inference is executed with frame skipping (UPLOAD_FRAME_SKIP = 2) and playback is slightly slowed down (playbackRate = 0.75) to reduce compute load while preserving a stable overlay; in live mode playback remains real-time. Keypoint rendering is filtered using a minimum confidence threshold (MIN_PART_CONFIDENCE = 0.2).

On the IMU side, timestamps are derived from a fixed sampling rate (SAMPLE_RATE_HZ = 104 Hz) and displayed using a moving time window around the current aligned time; chart updates are throttled to avoid excessive redraw overhead during playback.

The main limitations are the use of 2D pose keypoints (sensitive to occlusion and camera viewpoint), performance variability across devices, and the synchronization model assuming a constant time offset without drift compensation. In addition, IMU timing accuracy depends on the correctness of the assumed sampling rate and stable logging behavior.

3.4 Summary

This section described the methods used to build and operate the browser-based IMU–video motion analysis prototype. It outlined the IMU processing pipeline (CSV import, timestamp reconstruction, and chart visualization) and the video pose-estimation pipeline (MoveNet for uploaded videos with an optional live mode). It also introduced the event-based synchronization approach, together with session persistence (local storage and export/import) and the annotation and report-generation features. The following section presents the resulting prototype behaviour and evaluates its functionality, usability, and practical limitations.

4 Results

4.1 Test Setup and Data

To evaluate the prototype, we carried out functional tests primarily in the upload workflow, using short prerecorded videos together with recorded IMU CSV data. The optional live tracking mode was also tested briefly using the device webcam (configured to 640×480).

The uploaded test videos were limited to a maximum duration of 20 s and contained simple, clearly identifiable movements such as claps and short dynamic actions. During playback, the application processed video frames and rendered the estimated 2D skeleton overlay on a canvas aligned with the video element.

IMU data were collected with a single Movesense HR2 sensor held in the hand and exported as CSV files containing tri-axial accelerometer, gyroscope, and magnetometer channels. The IMU time axis was reconstructed from the sample index using a fixed sampling rate of 104 Hz, which is the sampling rate assumed by the application.

Synchronization between modalities was performed manually by selecting a distinctive reference event visible in both the video and the IMU signals.

All tests were executed locally in a standard web browser on a consumer laptop. The prototype runs fully client-side in the browser, using TensorFlow.js for pose estimation and Chart.js for IMU visualization.

4.2 Functional results

The prototype successfully supports uploaded-video playback with a 2D skeleton overlay, using MoveNet Thunder as the detector for this mode. To keep the overlay responsive on consumer hardware, the implementation applies performance optimizations (pose updates on a subset of frames and a reduced playback rate), which stabilized rendering during our tests.

A webcam-based live mode is implemented and runs pose estimation with MoveNet Lightning for real-time operation, using fixed camera constraints (640×480).

IMU recordings can be imported from CSV, converted into a time axis, and visualized as three stacked charts (accelerometer, gyroscope, magnetometer) with a data-time slider for navigation.

Manual event markers produce a stored syncOffset, and after applying synchronization the IMU view follows the current video time during playback and scrubbing by updating a sliding chart window and aligned marker.

Project state (including syncOffset and stored timestamp annotations) can be exported as a ZIP project containing project.json and re-imported to restore both data and metadata. Users can create timestamp annotations with labels, event types, and notes, and a TXT summary report can be generated from the loaded IMU data and the stored event list.

5 Discussion

The results indicate that a low-cost, browser-based workflow can integrate video pose estimation with IMU recordings and support basic inspection, synchronization, and annotation in a single interface. The upload-focused design is particularly suitable for the project goals, because offline processing reduces dependence on real-time compute performance and improves stability compared to a purely live system. Overall, the prototype is best interpreted as an exploratory analysis tool rather than a validated measurement system for professional use.

5.1 Feasible but limited functionality

The prototype demonstrates that IMU recordings and video-based pose estimates can be combined in a lightweight browser workflow, but the current scope remains intentionally limited. In its present form, the system supports comparative inspection of two modalities rather than performing true sensor fusion or producing calibrated, physically validated kinematic quantities. Consequently, the outputs are most appropriate for exploratory analysis and documentation, while professional deployment would require stronger calibration, robustness measures, and validation.

5.1.2 Functional validation vs. numerical validation

The evaluation primarily addresses whether the workflow behaves consistently and supports the intended analysis steps, but it does not provide a rigorous assessment of measurement accuracy. Without ground-truth reference data, it is not possible to quantify systematic errors or precision in absolute terms, and comparisons to high-end motion-capture systems remain conceptual. In addition, the pose component is inherently constrained by 2D, single-person keypoint estimation, which limits the range of motions and scenarios for which strong conclusions can be drawn.

5.1.3 Consistency and stability of outputs

From a methodological perspective, stability depends on both algorithmic choices and available computational resources. Any browser-based, client-side implementation must balance responsiveness against model complexity, and performance can degrade on less capable devices or in demanding scenes. Measures that increase stability (e.g., reducing inference frequency) can introduce trade-offs such as lower temporal resolution, which is relevant when analysing fast or short-duration events.

5.1.4 Qualitative plausibility of motion data

Qualitative inspection suggests that the produced trajectories and sensor patterns are interpretable, but visual plausibility is not equivalent to quantitative correctness. Pose estimation quality can vary with viewpoint, occlusion, and lighting, and IMU signals can be affected by placement and magnetic interference. As a result, the most defensible interpretation is that the prototype provides a coherent visual-and-signal context for motion events, while numerical interpretation should be treated cautiously.

5.1.5 Synchronization outcomes

The manual, event-based alignment method improves interpretability by connecting visible events to sensor peaks, but its accuracy is bounded by the user's event selection and by the assumptions of the alignment model. In particular, a constant-offset approach does not account for clock drift or variable latency, which can accumulate error over time. More robust alignment would require additional reference events or a drift-aware model.

5.1.6 Performance and usability results

The browser-based design lowers deployment barriers and supports rapid iteration, which is valuable for prototyping and low-cost settings. However, usability is coupled to performance: responsiveness, smooth playback, and stable overlays are all influenced by device capability and environmental conditions. For broader applicability, future work should therefore treat performance and validation as first-class goals, alongside the existing emphasis on simplicity and accessibility.

5.2 Ethical considerations

An ethical framework is essential for ensuring user safety and public trust. Since this project involves the collection and processing of human motion data derived from video recordings and IMU sensors, it raises several ethical considerations related to privacy, informed consent (*respect for autonomy*), data handling, and potential misuse.

First, privacy and data protection are central concerns, particularly because video data may contain identifiable information about individuals. To mitigate the risk, the system is designed to operate locally in the user's web browser, meaning no video recordings or motion data are transmitted to or processed on external servers; processing is performed locally in the user's browser. Project metadata may be stored locally in the browser and can optionally be exported/imported by the user. Thus, users retain full control over their data from start to finish. This design choice aligns with principles of data minimization and reduces the risk of unauthorized access or data breaches, which further ensures public trust.

Second, informed consent, or *respect for autonomy*, from the bioethical framework of principlism — i.e., the right of an individual to make a self-determined choice, given that they are fully informed in the context at hand — is essential when other people, namely the users, are involved. In this project, all testing was conducted voluntarily and for educational and prototype purposes only. The testing of the system was conducted by the group members —

not external participants. For any future advancements or deployment in clinical, rehabilitation, or sports contexts, explicit consent procedures and clear user information would be required.

Third, there is a risk of misinterpretation or over-reliance on system output. Due to the limited accuracy and lack of clinical validation — since the motion data was only extracted from the IMUs, not compared or validated with other data — the system should not be used for diagnosis or treatment decisions, especially in medical or rehabilitation settings. Instead, it should be used as a supportive or exploratory tool. Consequently, the project raises questions about equity and accessibility . While the low-cost and browser-based design improves accessibility and makes it less hard to operate compared to high-end motion capture systems, it may also lead to use by non-expert users without sufficient understanding of its limitations. Therefore, clear disclaimers and usage guidelines would be necessary in any real-world application to ensure ethical consistency.

Finally, ethical concerns due to the use of third-party and open-source code are relevant. A significant portion of the underlying code and system functionality is based on existing implementations made publicly available by other developers in open-source platforms (e.g., GitHub) and online technical resources. This approach was intentionally chosen, given a recommendation by the project supervisor, as it reflects common practice in modern software engineering and enables efficient development within the constraints of a course project. All reused code was adapted, integrated, and understood within the context of the system. In line with this approach, the project also makes the developed source code publicly available, in GitHub, enabling transparency, reproducibility, and further development by others. This allows future students, researchers, or developers to reuse, evaluate, and improve the system, thereby contributing back to the same open-source ecosystem from which this project built on.

5.3 Future improvements

Several enhancements could be considered to extend the system's capabilities. Implementing sensor fusion algorithms (e.g., complementary or Kalman filters) could improve the integration of IMU and video-derived information and support more robust motion interpretation. Motion tracking accuracy could be further optimised through systematic benchmarking of pose-model configurations (e.g., MoveNet variants and processing settings) and through improved handling of challenging conditions such as occlusions and lighting. Additional built-in analysis tools would enable more detailed post-processing, and controlled experiments with reference measurements could help quantify performance.

A particularly relevant extension concerns synchronization. The current workflow relies on manual event marking and assumes a constant offset; future versions could reduce user effort and improve repeatability by introducing automatic alignment. One practical approach is a predefined calibration event (e.g., a clap) at the beginning of the recording. On the video side, a clap can be detected from the 2D pose keypoints by monitoring the distance between the left and right wrist (or hand) keypoints and triggering when this distance falls below a threshold.

On the IMU side, the same event can be detected by thresholding a prominent peak (e.g., acceleration magnitude) in the sensor signal. The first matched event in both modalities can then be used to compute the initial offset automatically. Beyond single-event alignment, synchronization could be improved by detecting multiple events and fitting a model that also accounts for potential clock drift, or by using cross-correlation between motion-energy features extracted from video (e.g., keypoint velocity magnitude) and IMU-derived motion intensity signals to estimate alignment over longer segments.

Further extensions include tracking multiple people simultaneously, supporting real-time operation across a wider range of hardware constraints, importing multiple sensor streams, and integrating real-time IMU input for fully synchronized motion capture.

5.4 Summary

In summary, the discussion suggests that the proposed prototype is not intended to replace high-end motion capture systems, but to provide a low-cost and accessible alternative for exploratory motion analysis in non-laboratory settings. The identified limitations and trade-offs outline clear directions for future work and reflect practical constraints that influence real-world deployment.

5 Conclusion

This project demonstrated the feasibility of a low-cost, low-complexity motion analysis prototype that runs entirely in a standard web browser. The implemented system combines video-based 2D pose estimation with imported IMU recordings, enabling side-by-side visualization, manual event-based synchronization via a constant time offset, and basic annotation of motion events. In addition, the prototype supports session persistence through local storage and portable export/import, and it can generate a plain-text summary report from the recorded IMU timeline and stored annotations.

While the system is functional and supports exploratory inspection and documentation, it is not intended as a replacement for high-end motion capture solutions. Key limitations include the use of 2D keypoints (sensitive to occlusions and viewing conditions), the reliance on manual synchronization without drift compensation, and the absence of ground-truth validation for numerical accuracy. Future work should therefore prioritize quantitative evaluation, drift-aware or automated synchronization, and deeper integration of both modalities (e.g., sensor fusion) to improve robustness and measurement reliability.

Appendix A: Development History of the IMU Module

To reduce implementation risk, the IMU pipeline was validated in several iterations before being integrated into the final browser-based system. Initial connectivity and data acquisition were tested using a lightweight Bluetooth Low Energy (BLE) prototype based on an existing template (using the bleak library) to confirm stable pairing, streaming, and access to the relevant sensor signals. As illustrated in the following figure, the initial Bluetooth application is displayed.

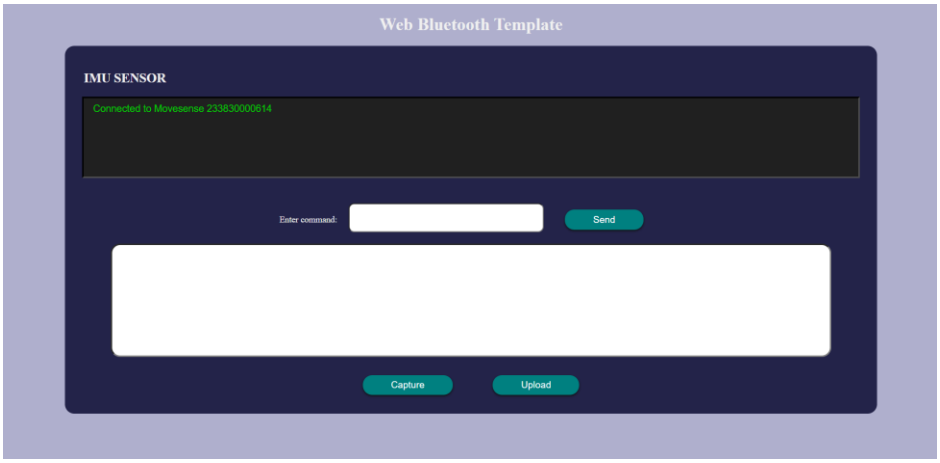

Figure 5. Initial BLE-based IMU web application prototype.

After basic connectivity was verified, a small Python application (seen in the following figure) was developed to visualize live accelerometer, gyroscope, and magnetometer data and to support recording and saving measurements as CSV files. This step served as a functional prototype to validate the end-to-end IMU workflow (acquisition → logging → reloading → plotting) independently of the web interface.

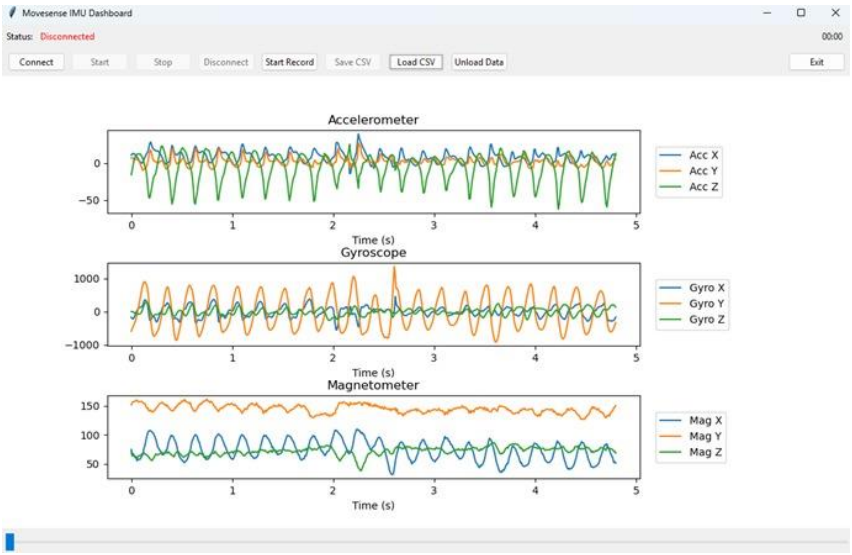

Figure 6. Python-based BLE IMU dashboard used to visualize live sensor data and record CSV logs.

Appendix B: User Guide – Web Application Workflow

B.1 Video Upload and Skeleton overlay

When opening the web application, the first step is to load a video file. Click the “Select video” button and a pop-up will appear, allowing you to choose a pre-recorded file from your device’s file explorer.
Once the video is loaded, the user must click "Start" to activate skeleton tracking.
A slider above the video enables navigation through the recording. When "Start" is active, the skeleton tracking continues to run as the user moves the slider; when "Pause" is pressed, the tracking is paused. All changes to the project should be made while the "Pause" button is active.

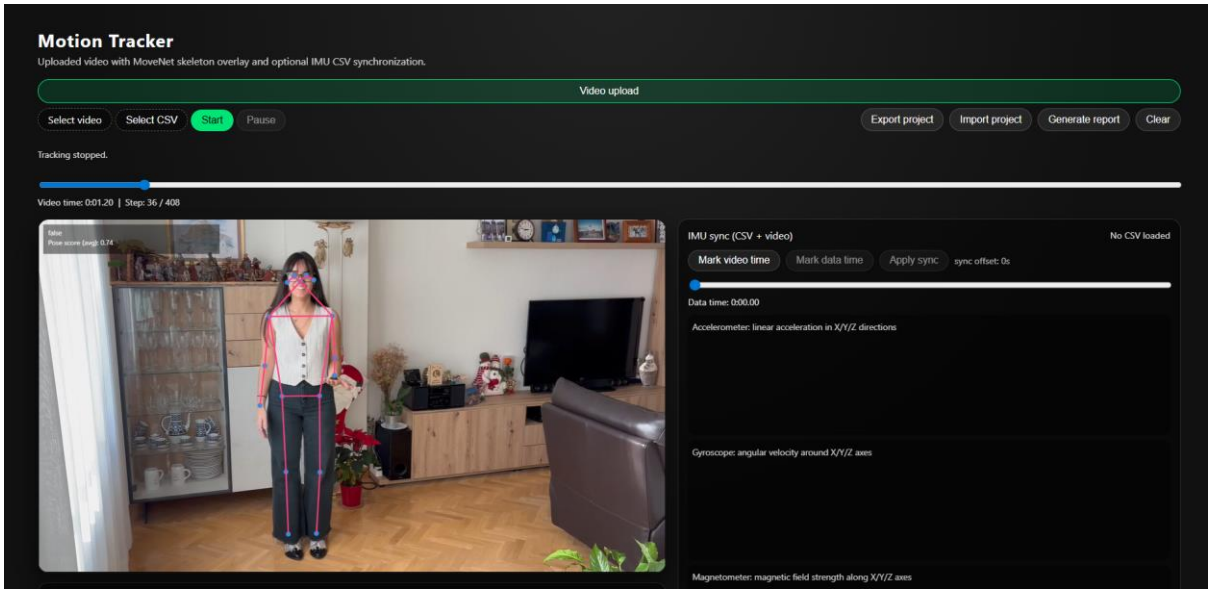

Figure 7. Motion Tracker interface for video upload and skeleton overlay.

B.2 IMU CSV upload and visualization

After uploading the video, the user proceeds to upload the IMU data by clicking the "Select CSV" button. A pop-up window appears, prompting the user to select the .csv file from the device's file explorer. Once uploaded, the user can navigate through the sensor data using a slider. The application displays three real-time graphs for the accelerometer, gyroscope, and magnetometer data, as previously described.

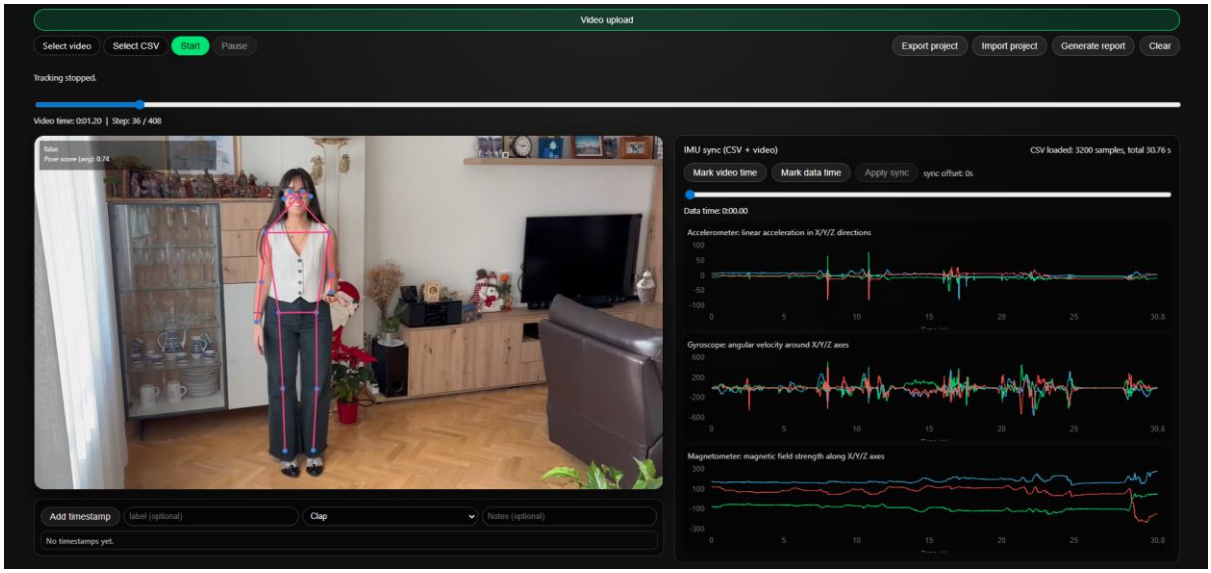

Figure 8. Motion Tracker interface for IMU CSV upload and signal visualization.

B.3 Manual synchronization (video ↔ IMU)

To align both modalities, use the synchronization controls after both files are loaded and while the Pause button is selected. Navigate to a distinctive motion event in the video (e.g., a clap or jump landing) and mark the time using the “Mark video time” button. Then navigate to the corresponding event in the IMU plots and mark it using the “Mark data time” button. Click “Apply sync” to align the time axes so that video playback and IMU inspection refer to the same physical moment.

From then on, the slider above the video controls both the video and the graphs in a synchronized way.

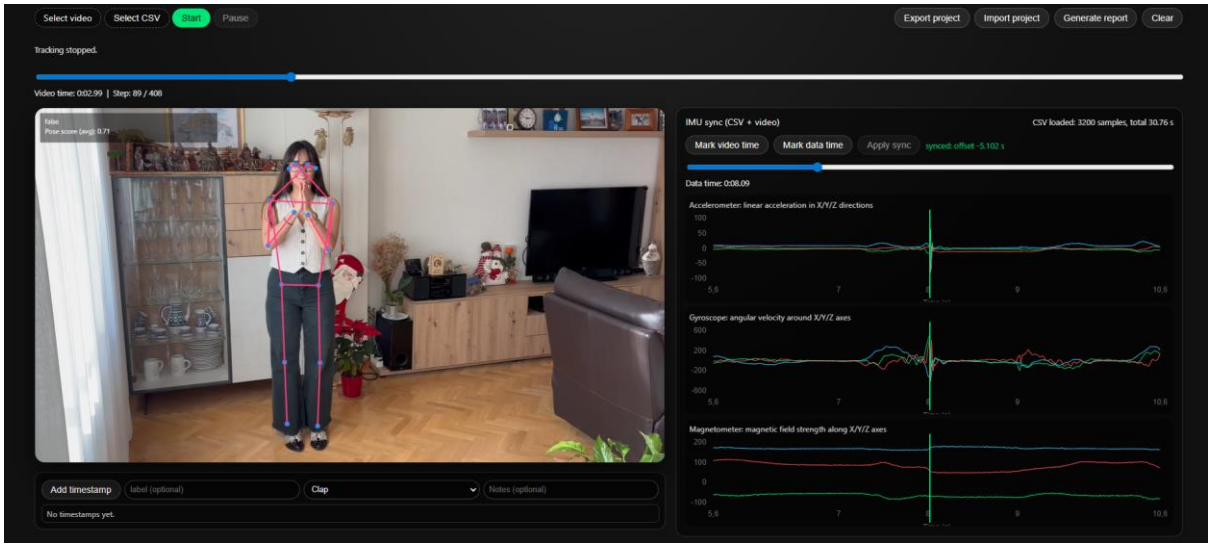

Figure 9. Manual video–IMU synchronization using the Mark time controls.

B. 4 Annotation and classification

After synchronization, relevant moments can be annotated. Navigate to the desired video time and enter:

- Label (a short name for the event)
- Classifier (prototype categories: Jump, Clap, Other)
- Note (optional free-text description)
- Press Add Timestamp to store the annotation. Stored timestamps can be used to document events of interest and support later review.

For example, if the user wants to highlight a moment where the subject is jumping, they can navigate to that frame, enter a label (e.g., "First Jump"), and classify the action using the dropdown menu (e.g., "Jump," "Clap," "Other"). The note input box allows the user to add descriptive text, such as "Lucia's first jump during training session number 13, first try of this new exercise, analyzing if her performance has improved compared to previous sessions." After completing these fields, the user clicks "Add Timestamp" to store the annotation.

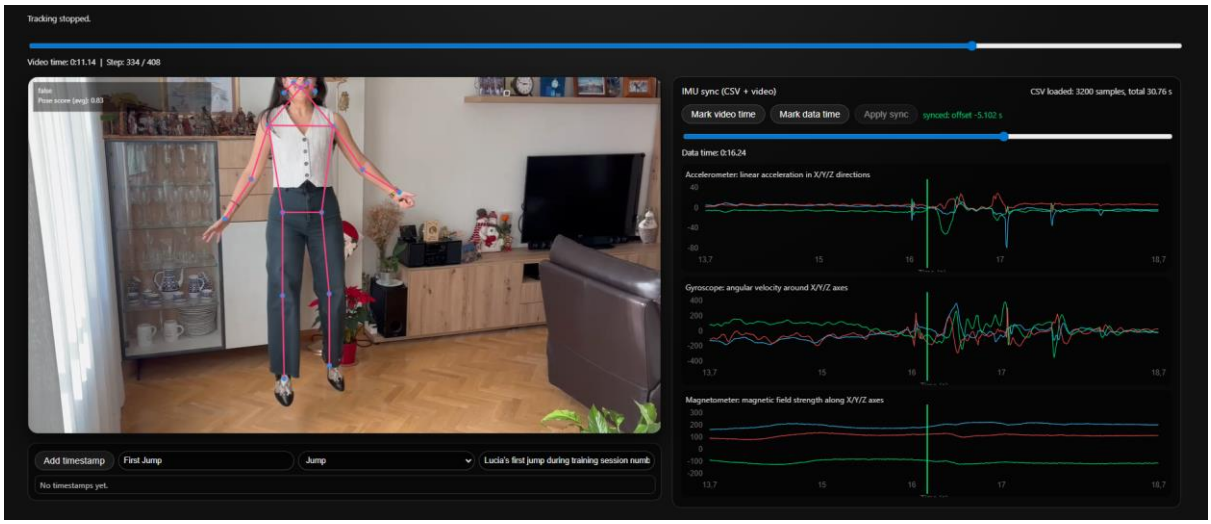

Figure 10. Annotation interface for adding labeled timestamps to the recording.

Once the timestamp is stored, it will appear in a list of timestamps below. The user can navigate through them, and when one is selected, the view will jump to that moment and display the stored information.

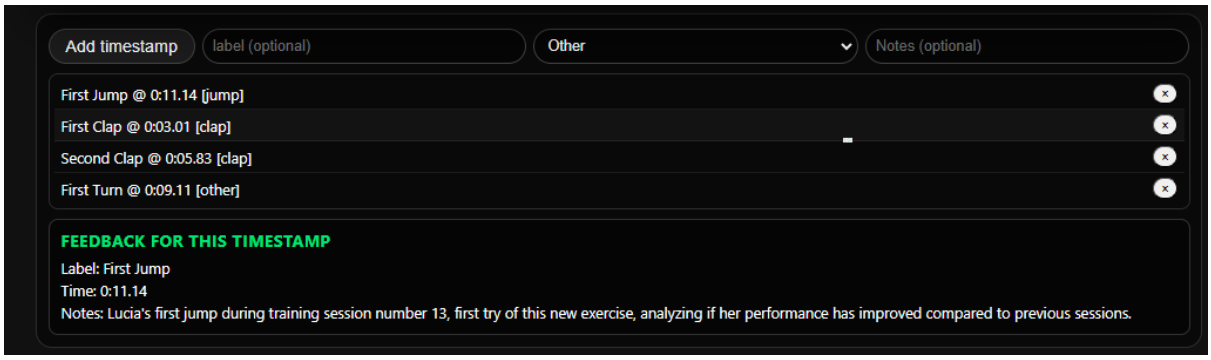

Figure 11. Timestamp list and detailed feedback view for a selected event.

B.5 Import and export

Currently, the project data is managed in the browser using LocalStorage, which stores only metadata (not the actual video or .csv). When you upload a video and/or a .csv file and change anything—apply a time sync, add timestamps, labels, or notes—that information is saved in LocalStorage and linked to the names of the video and .csv files. If you clear the page or close the browser, this metadata is removed, but when you upload the same video and .csv again, all your changes are restored: the video and IMU stay synchronized and your timestamps and notes appear again.

To make a project reusable, portable, and shareable between users (researchers, patients, colleagues) and to avoid losing work when changing computers or having system issues, use the "Import Project" and "Export Project" buttons.

- **Export Project**

When you click "Export Project" a pop-up asks you for a filename for the ZIP. The app creates a ZIP with this structure, containing your data:

```
movesense_project.zip
├── project.json
├── video/
│   └── video_from_project.mp4
└── data/
    └── imu_data.csv
```

The file project.json stores your session metadata:

- syncOffset: synchronization offset between video and IMU.
- timestamps: list of timestamps and events.
- sampleRate: IMU sampling rate.
- createdAt: date and time when the project was created.
- notes: any additional notes.
- videoFileName: generic video filename (for example, video_from_project.mp4).
- csvFileName: generic CSV filename (for example, imu_data.csv).

- **Import Project**

When you want to import a project, click the "Import Project" button and select a ZIP that was exported previously. The app reads the ZIP, loads the video and the .csv, and applies all metadata from project.json.

The project is restored exactly as before: video and IMU are synchronized, timestamps, labels, and notes are back, and you can continue working without repeating setup steps.

B.6 Generate Report (TXT summary)

To generate a simple summary, use Generate Report. The application produces a plain-text report that includes the total recording duration, the number of stored events, and a list of all

annotations (index, label, timestamp, event type, and optional notes). The TXT file is downloaded automatically and can be used for documentation or sharing results.

Contributions

References

Acknowledgements

This project was carried out as part of the course CM2015 HT25-1 Project Carrier Course for Medical Engineers, part 1 (15 credits) at KTH Royal Institute of Technology, where the work was conducted under the supervision of Jonas Willén, lecturer and program director of MSc in

Sports Technology. The supervisor provided guidance on the project scope, methodological approach, and application context, ensuring that the system design aligned with both academic objectives and practical use cases.

[1] https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html