IMPERIAL

Hybrid IR–IMU Object Tracking for Wrist Motion Capture in VR Stroke Rehabilitation Environments

BIOE60005 - Bioengineering Group Project

Submitted to: Dr James Choi

Email: j.choi@imperial.ac.uk

Group Members:

Divij Vidhi Handa	CID: 02314481
Miles Caskey	CID: 02324105
Eric Lawrence	CID: 02240546
Khushi Mathur	CID: 02286998
Edward Goh	CID: 02263084
Hiba Kadri	CID: 06024737
Celine Elsayed	CID: 02316323

Date of Submission: 15/04/2025

Word Count: 4,817

Department of Biomedical Engineering

Imperial College London

Abstract

This project addresses the need for accurate, affordable, and robust object tracking systems for augmented and virtual reality (AR/VR) applications in stroke rehabilitation. Stroke survivors require repetitive and precise motor training to promote neuroplasticity. However, current tracking systems typically lack either affordability or sufficient accuracy for widespread clinical adoption.

In response to this, we designed a hybrid tracking device incorporating infrared (IR) LEDs and a 9-axis BNO085 inertial measurement unit (IMU), integrated within a custom wristband. Optical tracking utilised blob detection and ellipse-fitting algorithms to effectively refine the positional data from IR LEDs, mitigating optical distortions and significantly improving tracking accuracy. Additionally, the IMU provided reliable orientation data to counteract issues arising from potential optical occlusion.

Our results indicate that the optical tracking method reliably captures accurate positional information when not tilted but shows inaccuracies upon tilting of the wristband, for wrist movements within our developed VR rehabilitation environment. Although the IMU provided stable orientation data independently, complexities encountered during the project time-frame prevented the successful implementation of sensor fusion. Nonetheless, parallel integration of these sensor modalities within the VR game demonstrated promising accuracy and practical usability for rehabilitation exercises.

The outcomes of this work suggest that our hybrid tracking approach presents a viable, affordable alternative to expensive, high-end tracking systems, potentially broadening access to immersive rehabilitation therapies. Future developments should prioritise full sensor fusion implementation to further enhance tracking accuracy and reliability, thereby optimising therapeutic outcomes for stroke patients.

Acknowledgements

We would like to extend our special thanks to:

Dr. James Choi (supervisor) for his continued guidance, encouragement, and invaluable feedback throughout the development of this project.

Mr. Jacques Blagburn for his generous assistance in the manufacturing process of the wristband, and for his practical guidance and support in refining the physical design.

Dr. Yen Foung Tai, NHS neurologist, and **Dr. Bill Tahtis**, consultant occupational therapist, for their clinical insights and discussions that guided the therapeutic application and focus of the project.

Neus Pons Andrés for her foundational research on infrared LED tracking, which provided a valuable framework for the development of our hybrid tracking system

---

Contents

1 Introduction

1.1 Background Context

Stroke is a leading cause of long-term disability, often resulting in upper-limb motor impairments. Rehabilitation requires repetitive, task-oriented motor training to stimulate neuroplasticity and support functional recovery [1, 2]. Tracking smaller, faster-moving body parts like the wrist is particularly important, as it plays a crucial role in both fine and gross motor tasks. However, accurate wrist tracking remains a challenge due to complex motion patterns and frequent self-occlusions caused by the body's own movements [3].

The use of AR/VR systems has been adopted, creating immersive, interactive environments that enhance therapeutic outcomes by increasing engagement, enabling customisation, and providing more opportunities for patient-specific therapy [4].

However, these systems rely heavily on precise object-tracking to function effectively. While tracking technologies have advanced across both consumer and research platforms, a persistent trade-off remains between accuracy, robustness, and affordability. In rehabilitation settings, high-end systems like OptiTrack offer precision but are costly and complex to set up. More accessible alternatives, such as the HTC Vive, are affordable but suffer from positional drift, occlusion, and limited precision factors. [5]. Thus, this project aims at creating a low-cost alternative, while still implementing low latency and robustness.

1.2 Literature and Research Gap

Stroke

Stroke frequently results in long-term motor impairment of the upper or lower limbs [6]. Rehabilitation involves various exercises, with evidence supporting unilateral, repetitive, and task-oriented movements as the most effective [2, 7]. Clinical consultation with neurologist Dr. Yen Foung Tai identified stroke as a strong candidate for VR-based intervention.

VR Game

Building on stroke's suitability for VR rehabilitation, VR games offer promising therapeutic benefits. These have been shown to improve patient motivation and adherence compared to traditional therapies [4, 8]. Gamified approaches that simulate daily tasks provide clear goals and real-time feedback, enhancing participation and long-term commitment [4, 8].

Discussions with consultant occupational therapist Dr. Bill Tahtis emphasised, there are limitations of current hospital-based therapies, constrained by scheduled appointments. However, so far, lack of accurate tracking limits the clinical effectiveness of existing VR rehabilitation systems.

Object-Tracking System

To achieve effective VR-based stroke rehabilitation, accurate tracking and low latency of patient movements is essential. Recent research has explored hybrid object-tracking systems that combine inertial and optical sensors to mitigate the limitations of each modality. Inertial measurement units (IMUs) can capture orientation data independently of visual obstruction. However, they are prone to drift, compromising long-term positional accuracy [9]. In contrast, optical sensors such as infrared (IR) Light Emitting Diode (LED) markers offer high positional accuracy but require clear visibility and controlled lighting, limiting their practicality in clinical or home-based environments [10].

Sensor fusion methods, particularly the Kalman filter, have proven effective in combining these modalities to improve tracking robustness and accuracy under variable conditions [11]. Nevertheless, existing hybrid systems are largely designed for general applications like robotics and full-body motion capture, leaving a significant gap in solutions tailored specifically for stroke rehabilitation. While studies such as Maereg et al. (2017) have demonstrated the potential of wearable opto-inertial systems

for virtual interaction, challenges such as complex calibration and marker setup have hinder their usability by patients with limited technical proficiency [12]. A hybrid wrist-tracking solution tailored to stroke rehabilitation could significantly enhance outcomes if enabling accurate movement monitoring, low latency, and task-oriented training.

1.3 Aims and Objectives

This project addresses the need for an object-tracking system that can be integrated into a VR-based stroke rehabilitation game. The primary aim is to develop and evaluate an affordable, accurate, and low latency hybrid tracking system that provides positional and orientational data to a virtual hand model for immersive stroke rehabilitation therapy.

To achieve this aim, the objectives of the study are:

- To develop a wearable wristband featuring a circular arrangement of IR LED markers combined with an IMU sensor, optimised for precise wrist tracking.
- To implement and validate a sensor fusion algorithm, specifically utilising the Kalman filter, effectively combining optical tracking's long-term precision and inertial tracking's robustness against occlusion and drift.
- To integrate this hybrid tracking system into an interactive and immersive VR rehabilitation game developed in Unity, incorporating evidence-based rehabilitation principles.

2 Methods

2.1 System Overview

The hybrid obejct tracking system integrates IMU and IR LED data, illustrated in Figure 1.

Positional data of IR LEDs, which are placed on a wristband, were captured using an Open MV camera, while the IMU provided orientation data in quaternion coordinates processed via a development board. The Kalman filter algorithm was used to fuse the data to mitigate sensor drift and enhance accuracy [11].

The system enables precise wrist tracking that can be integrated into a VR stroke rehabilitation application.

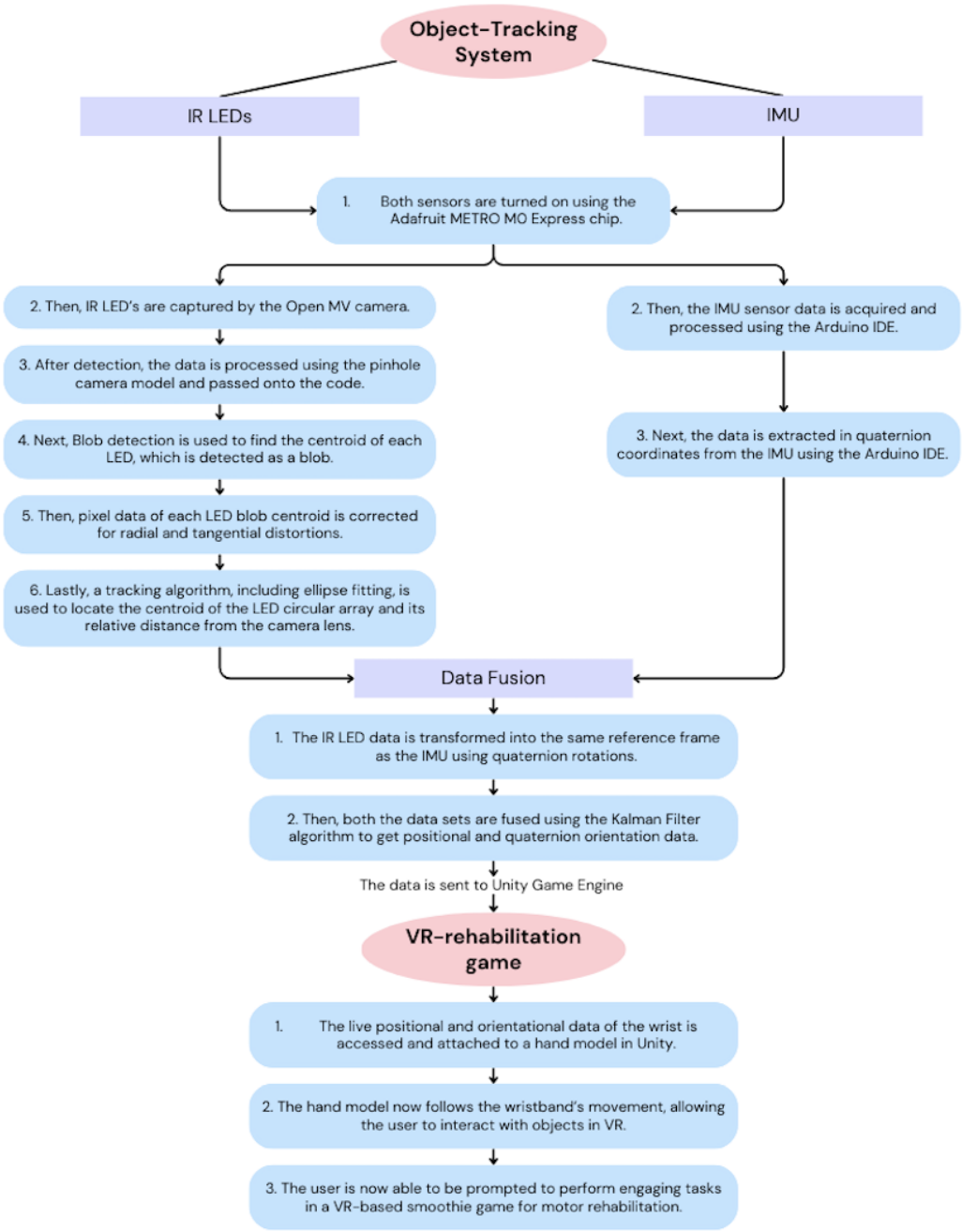


Figure 1: Flowchart describing the setup and use of the device

2.2 Object Trafcking System

An OpenMV RT1062 [13] monocular camera captures IR light, which appears as bright spots in images. By processing these images, the system determines the object's 3D position relative to the camera. The pinhole camera model was used to estimate monocular camera capture.

2.2.1 Mathematical Foundation for IR LEDs

Pinhole Camera Model

When light passes through a single lens, it refracts and is focused onto a sensor array, where each pixel signifies a particular point in the image (Figure 2). The pinhole camera model mathematically describes how 3D points in space can be mapped onto a 2D image plane using perspective projection [14, 15, 16, 17, 18].

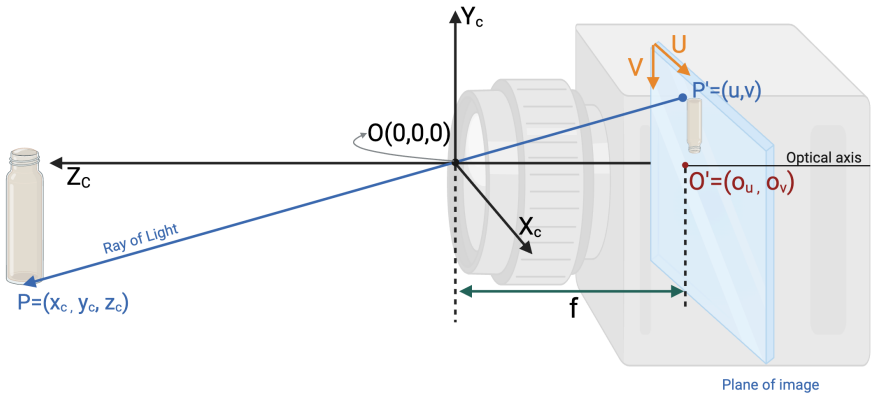


Figure 2: Visualisation of the pinhole camera model. The pinhole is represented at the origin (O) of camera lens. A 3D point $P = (x_c, y_c, z_c)$ in the camera coordinate frame is projected onto a 2D image point $P' = (u, v)$ through perspective projection. The focal length f and principal point offsets O' are intrinsic parameters [19].

Specifically, a 3D space point $P = (x_c, y_c, z_c)$ is projected onto pixel coordinates $P' = (u, v)$ on the image plane.

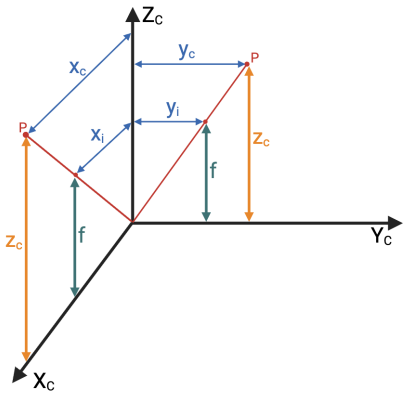


Figure 3: Using laws of similar triangles for mapping in Pinhole camera model. The figure illustrates how a 3D point $P = (x_c, y_c, z_c)$ in camera coordinates projects onto a 2D image plane at (x_i, y_i) based on the focal length f [19].

Then, by using laws of similar triangles (Figure 3), the relationship between P and P' can be expressed in physical units as $P = (x_i, y_i)$ can be derived to be [20, 21, 22]:

$$x_i = \frac{f \cdot x_c}{z_c}, \quad y_i = \frac{f \cdot y_c}{z_c}$$

f is the camera's focal length, and z_c is the depth of point P from the lens of the camera.
By considering the principal point offsets of the camera, (o_u, o_v) and resolution (r_x, r_y), the above

formula for mapping can be refined to the following form [20, 21, 22]:

$$u = \alpha \cdot \frac{x_c}{z_c} + o_u, \quad v = \beta \cdot \frac{y_c}{z_c} + o_v$$

α and β depict focal lengths in pixel units. Although the pinhole camera model gives a mathematical foundational relation between the object's 3D position and the corresponding pixel coordinates of the image, it still contains distortions due to physical properties and manufacturing imperfections [23].

Distortion Correction

The two main distortions are radial and tangential distortions [23]. The radial distortion comes from the inability of the lens to transmit straight lines accurately. These distortions are corrected through a polynomial function derived from the Taylor expansion of radial distance:

$$L(r^2) = k_1 r^2 + k_2 r^4 + k_3 r^6$$

Tangential distortions are caused by errors in manufacturing, resulting in a misalignment between the lens and the image sensor. Both radial and tangential distortion correction equations are combined and expressed [20, 24]:

$$\begin{aligned} x_{\text{distorted}} &= x_i + x_i(1 + L(r^2)) + 2p_1 x_i y_i + p_2(r^2 + 2x_i^2) \\ y_{\text{distorted}} &= y_i + y_i(1 + L(r^2)) + p_1(r^2 + 2y_i^2) + 2p_2 x_i y_i \end{aligned}$$

Camera Calibration

Using Zhang's method on OpenCV, distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$ and intrinsic parameters $(\alpha, \beta, o_u, o_v)$ were estimated [25, 26, 27]. To achieve this, ten images of a printed chessboard from different orientations were captured at a resolution of 640 x 480. These images were then used to relate the 3D coordinates of feature points to their 2D projections in units of pixels. Then, initial estimates made about the distortion coefficients and intrinsic parameters were optimised using the Lavenberg-Marquart algorithm. This optimisation was conducted to minimise the re-projection error between the observed points and the pinhole model projected points [22, 25, 28, 29, 30]. This camera calibration was performed once, as the parameters are camera specific.

2.2.2 Algorithm Implementation for IR LEDs

Image Acquisition

The OpenMV microprocessor captures greyscale images, from which the IR LED centroids were detected using blob detection algorithms [31, 20, 14]. An empty vector, P_p, is created to store the captured image, $I(u, v)$, of dimensions $w = 640$ pixels and $h = 480$ pixels. Blobs are detected based on the brightness of the IR LEDs, with a brightness threshold between 160 and 255 ($160 \leq I(u, v) \leq 255$). Blobs are further filtered by passing an area threshold of between 10 - 2500 pixels [32, 33, 34]. Then, each blob's centroid position is stored in the pixel position vector P_p. This vector is processed via USB from the OpenMV camera to an OpenMV IDE. The transmitted pixel data remains distorted and requires correction (Appendix C.1).

Distortion Rectification

Upon data receipt, pixel coordinates must be corrected for radial and tangential distortions and also converted into the corrected coordinate vector P_n. The undistorted pixel coordinates are derived using the distortion correcting equations and utilising the Brown-Conrady model [35] in a loop. While considering the camera's distortion coefficients and intrinsic parameters, the following equations are utilised for the corrected coordinates:

$$x_{icorrected} = \frac{(x_i - \Delta x_{tangential})}{radial\_distortion}, \quad y_{icorrected} = \frac{(y_i - \Delta y_{tangential})}{radial\_distortion}$$

Depth Estimation Using Circular Constellation

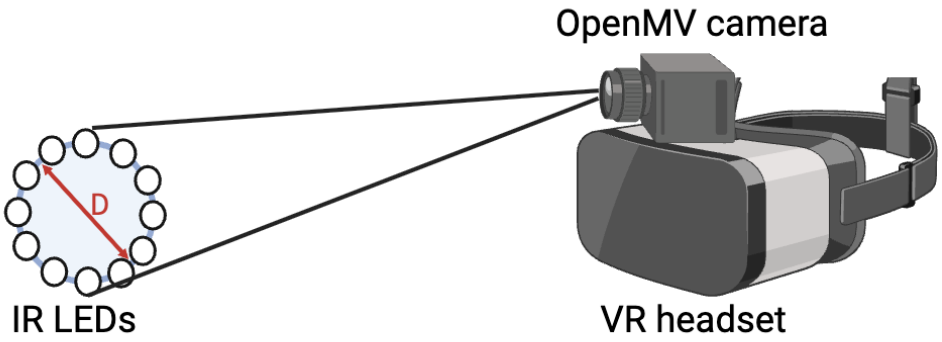


Figure 4: Illustration of the circular LED constellation used for depth estimation. The known diameter D of the IR LED ring allows the OpenMV camera, mounted on the VR headset, to estimate the distance to the wristband using size-based perspective projection [19].

The IR LEDs were arranged in a circular constellation ensuring rotational consistency during tracking, (Figure 4) [33, 32]. Principles of similar triangles [36] were used to estimate depth (z_c) of the circular IR LED constellation:

$$z_c = f \cdot \frac{D}{h}$$

D is the actual diameter of the circular constellation, h is the projected constellation diameter in pixels, and f is the focal length in pixels.

This method was employed to ensure consistent and accurate depth estimation regardless of rotational orientation, as the array's diameter stays constant.

Position Calculation

After finding the diameter of the LED array in the image, the midpoint is used to find the centroid position in pixel coordinates $(x_{o,u}, y_{o,v})$. Once the depth (z_c) is found, the spatial coordinates X (x_c) and Y (y_c) [32, 33, 37] were found:

$$x_c = \frac{x_{o,u} - o_u}{\alpha} \cdot z_c, \quad y_c = \frac{y_{o,v} - o_v}{\beta} \cdot z_c$$

$x_{o,u}$ and $y_{o,v}$ are the centroid pixel positions, o_u and o_v are the principal point offsets, and α, β are the focal lengths in units of pixels along the horizontal and vertical axes, respectively.

Finally, the derived x_c, y_c, and z_c are implemented into pixel coordinates of the corrected vector P_n via the next method. Primarily, the distance between two neighbouring blobs, along with the pixel coordinates of two opposing are calculated to define the constellation diameter and project centroid $x_{o,u}, y_{o,v}$ in pixel coordinates. The algorithm uses the pixel coordinates to calculate x_c, y_c, and z_c,

determining 3D position of the centroid and the IR LED circular constellation relative to the camera lens (Appendix E.2).

Ellipse-Fitting Algorithm for Blob Refinement

Blob detection was refined using Halif and Flusser's [38] numerically stable method, to create an ellipse-fitting algorithm, detecting blobs through a conic polynomial:

$$F(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$$

The coefficients a, b, c, d, e, f are determined under the constraint $b^2 - 4ac < 0$, ensuring a valid conic ellipse.

The algorithm follows these steps:

1. A design matrix is constructed from the detected points (x_i, y_i).
2. Then, scatter matrices for quadratic, linear, and constant terms are commuted.
3. Reduction and transformation matrices are derived to simplify the scatter matrix computation.
4. Eigenvectors corresponding to valid ellipses are used to extract ellipse parameters [39] (Appendix E.2).

2.2.3 Inertial Measurement Unit

The system integrates a BNO085 IMU, a 9-DOF sensor combining an accelerometer, gyroscope, and magnetometer [40] with onboard processing through an Arm Cortex-M0+ microcontroller [40]. This microcontroller calibrates motion data without additional fusion algorithms [41]. Therefore, quaternion data can be easily extracted directly (Appendix C.3). Overall, the IMU increases system precision by enhancing movement tracking and robustness to occlusions, a dynamic with which LED tracking struggles.

2.2.4 Wristband Integration

(a) Mounted wristband showing the IMU and board connections.

(b) Underside of the wristband showing the full IR LED array.

Figure 5: Top and bottom views of the custom wristband assembly used for 3D pose estimation and tracking.

Twelve 850 nm IR LEDs [42] were evenly distributed and soldered onto a flexible PCB with a designated area for IMU attachment. Furthermore, A 3D-printed wristband was configured to the PCB containing the IMU/IR LEDs. The circuit diagram and the PCB layout can be seen in the Appendix B.2.

2.2.5 Fusion of IMU Data with IR Tracking for Virtual Reality

To fuse IMU and IR sensor data, the Kalman filter was used to estimate position (x_c, y_c, z_c) and orientation (ϕ_x, ϕ_y, ϕ_z) in real time. High-frequency updates on motion dynamics are provided by the IMU, whereas the IR LED's ensure spatial accuracy through visual feedback. The fused data can then be used by the Unity software to accurately track a real-life object and translate its movements into the virtual world in real-time. This integration minimises latency while enhancing immersion in VR applications.

The system will specifically be using Extended Kalman Filter (EKF) techniques due to the non-linear nature of quaternion kinematics. The two main steps followed in EKF are prediction and updating in a loop to get the final data. With regards to prediction, a state propagation is done using the process model and then prediction of the covariance of error propagated. Following this updating takes place, which includes calculating the Kalman gain, updating the estimate of state with new measurements, and finally updating the error covariance matrix [43, 44].

Some implementation aspects to include an algorithm would be:

Alignment of coordinate system

Defining a global reference frame would be important. This would help convert both the IR tracking system's frame and the IMU reference frame to a global frame for further processing [45].

Fusing strategy

An ideal fusing strategy to use would be the complementary sensor fusion. This is because two different measurements (orientational and positional) are being derived, giving different aspects of the state matrix of the object. Therefore, complementary fusion would be ideal to fuse positional data from the IR and orientational data from the IMU [46].

Preprocessing of data

Before feeding data into the EKF for processing, it would be passed through a low-pass filter to reduce noise derived from the sensors. Secondly, it would be essential to ensure both data sets have synchronised time stamps.

Finally, the fused positional and orientational data would be derived in matrix terms to be used by the unity code.

To streamline data collection and integration for the VR rehabilitation system, an automated pipeline was created. This script activates a Python conda environment, uploads LED detection code to the OpenMV camera, and runs a Python process to extract and fuse positional and sensor data. The results are formatted as a JSON file (Appendix B.3).

2.3 VR-based Rehabilitation Game

2.3.1 Hand Model

Unity, was used for developing the VR application, and Hand tracking was implemented by directly feeding wristband coordinates into Unity's XR Interaction Toolkit.

The intended users, stroke rehabilitation patients at Brunnstrom stage 2 or higher, will have some regained arm mobility but limited finger or palm control [47]. To accommodate this, the XR Interaction hand model will automate in-game hand movements. When the virtual hand approaches an object, a grabbing animation is triggered automatically. This removes the need for fine-motor input and enables full application use even in early recovery stages. Furthermore, the system avoids finger tracking, reducing the need for bio-skeletal hardware, while still preserving meaningful interaction.

2.3.2 Game Design

Initially designed as a set of mini-games, the concept evolved into a unified VR experience set in a smoothie bar. Engagement is a common shortcoming in standard physiotherapy, where exercises are often repetitive and disconnected from daily life. This can lead to demotivation and poor adherence. In contrast, gamified VR settings offer clear goals, feedback, and a sense of achievement—key factors in maintaining patient motivation.

The smoothie bar supports a wide range of upper-limb movements through structured levels designed to improve coordination, range of motion, and strength. Early levels focus on basic motor skills such as reaching and grasping, while later stages introduce more complex tasks requiring sequencing and control. A detailed breakdown of each level's objective and motor focus is provided in Appendix D.

By embedding clinically relevant exercises into an enjoyable and meaningful context, this approach enhances both the physical and psychological aspects of rehabilitation. It transforms therapy into a purposeful experience, improving long-term engagement and recovery outcomes.

2.3.3 Game Development

For the initial stage of development, Unity ProBuilder was used to create basic 3D shapes and layout elements directly within the engine. While the initial visuals were minimal, the focus was placed on ensuring functionality, with plans to return later for enhancements such as textures, lighting, and finer environmental details.

(a) Overview of the draft of the virtual smoothie bar game environment, built using Unity ProBuilder

(b) Player's point of view in virtual environment with hand model

Figure 6: Environment layout and first-person interaction view in the Unity-based smoothie bar rehabilitation game.

To ensure accessibility and reduce complexity, the player's full positional movement remains stationary within the scene. They only need to look around and move their arms to interact with objects. Each task level—such as reaching, squeezing, or sorting—is implemented either as a separate Unity

scene or as an individual module within a single scene, allowing for flexibility in how levels are loaded and transitioned. A central GameManager script handles level flow, user interface updates, and task completion conditions, ensuring a smooth and coherent gameplay experience.

Objects within the environment were set up with appropriate colliders and interaction scripts, enabling natural engagement using hand controllers. These scripts define how objects can be picked up, manipulated, and dropped, often triggering feedback events or tracking metrics essential for therapy, such as movement precision, task completion time, and number of successful repetitions.

To control the virtual hand, a custom Unity C# script (Appendix C.2) was developed to receive real-time JSON-formatted data from an external Python data fusion script. This system outputs the wrist's 3D position (X, Y, Z) and its orientation as quaternions, which are calculated based on camera data processed by OpenMV and a calibration algorithm in Python. The Unity script parses the JSON input via a serial connection, updates the position and rotation of the wrist joint, and applies this transform to the XR hand model in Unity. This enables the hand to mimic the user's real-world wrist movement in real time, providing a natural and responsive interaction experience.

Although the XR hand model contains 25 joints to support fine-grained finger articulation, only the wrist joint was utilised at this stage. This simplification reduces system complexity while maintaining functional interaction, aligning with the motor capabilities of the target user group. Future developments may explore activating additional joints to enable more detailed and expressive hand movements.

3 Testing and Results

3.1 Software and Unity Testing

To ensure the VR rehabilitation game is effective, user-friendly, and clinically relevant, a multi-stage testing and validation process would be carried out:

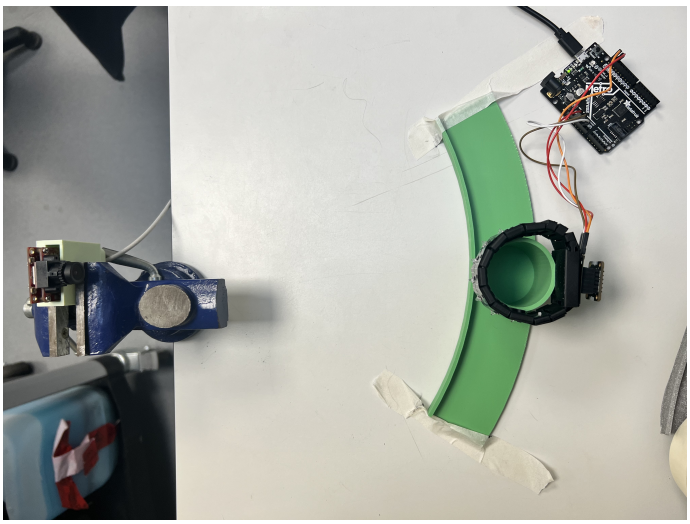
1. **Usability Testing**
   Early testing and collecting feedback from healthy individuals and patients to assess overall user experience.
2. **Functional Validation**
   Review from physical and occupational therapists to guide the movements involved in the game for optimum recovery.
3. **Clinical Pilot Studies**
   Small-scale pilot sessions with patients would track improvements in motor function. Data from in-game performance and clinical assessments would be used to evaluate improvement.
4. **Engagement & Motivation Metrics**
   Since adherence is a key goal, tools such as the User Engagement Scale and in-game metrics (such as time spent) would be used to measure user interest and motivation.
5. **Iterative Refinement**
   Continuous refinement of the game from feedback and performance data to best support engagement and rehabilitation outcomes.

3.2 Tracking System Testing Criteria

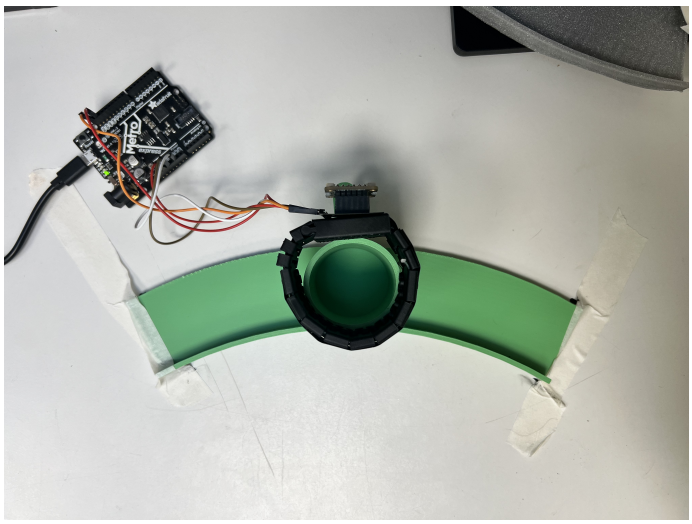
Prior to spatial accuracy validation, the system underwent functional testing to ensure reliable performance under realistic rehabilitation conditions, including occlusion, lighting variation, and rapid movement. As this experiment focused solely on the positional capture of the IR LED, only criteria 1–4b from the full test protocol (Appendix E.1) were evaluated.

3.3 Wristband Configuration

The OpenMV camera was mounted vertically using an adjustable jig, while the 12-LED wristband was positioned on a circular track with a fixed radius of 25 cm from the camera lens. This reference distance approximates the average wrist-to-camera separation expected during gameplay. Although, the radial distance remained constant, the wristband's (X, Y, Z) coordinates varied with angular displacement along the arc, enabling controlled translations while maintaining consistent distance from the lens, as shown in the figure below. The wristband was rotated from $-30°$ to $+30°$ in $10°$ increments, and at each step, an image was captured using the OpenMV camera.



(a) Complete experimental setup showing the OpenMV camera fixed on a bench-mounted vice and the wristband constrained along a 3D-printed arc.



(b) Close-up of the wristband placed on the arc. The 12-LED array and microcontroller (Metro M0 Express) are visible.

Figure 7: Validation setup for testing wristband positional estimation across angular increments from $-30°$ to $+30°$.

The OpenMV IDE was used to implement custom source code that detected and logged the centroid coordinates of the visible IR LEDs within each image frame. An ellipse fitting algorithm was then applied to this positional data to estimate the centre of the wristband (object tracker), based on the geometry of the projected LED constellation.

(a) Position at $-30°$

(b) Position at $0°$

(c) Position at $+30°$

Figure 8: Snapshots captured by the OpenMV camera showing LED centroid detection as the wristband is manually translated along the arc from $-30°$ to $+30°$ in $10°$ increments. These images illustrate the visibility and spacing of the LED constellation at different positions on the curved rig.

3.4 Positional Accuracy and Statistical Analysis

True 3D positions (XYZ) were geometrically calculated using the known arc radius and angle from the camera's focal lens, providing reference coordinates for the wristband's centre. These were compared to predicted values from the ellipse fitting algorithm to quantify tracking accuracy across different orientations and translations. The wristband was incrementally shifted along the track at angles from $-30°$ to $+30°$ relative to the camera's yaw axis, and the same procedure was repeated for wristband orientations around the pitch axis: level ($0°$), tilted up ($+30°$), and tilted down ($-30°$).

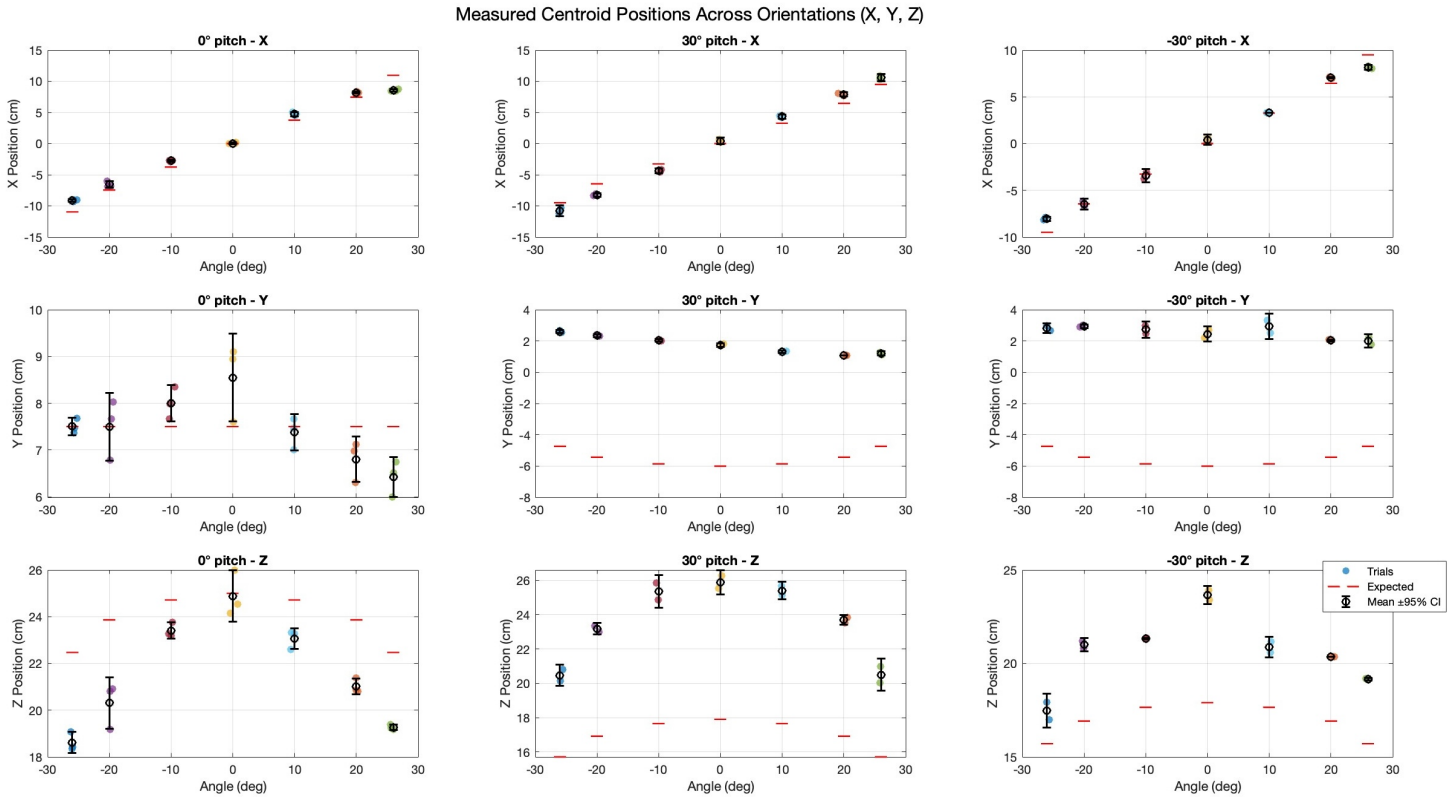


Figure 9: Measured centroid positions (x, y, z) across angular increments from $-30°$ to $+30°$ under three wrist orientations: flat, tilted up, and tilted down. Dashed lines indicate expected positions derived from the arc geometry.

Figure 9 shows the three trials of the predicted wristband centroid coordinates (X, Y, Z) at each angular position relative to the camera lens, across all wristband orientations. The red lines indicate the true position of the wristband centroid.

When the wristband is tilted $30°$ upward or downward, notable discrepancies arise between the true and estimated centroid positions, particularly in the Y and Z coordinate frames. These are caused by the ellipse fitting algorithm and camera geometry, which introduce a systematic offset of approximately 6 cm due to the changing projection of the LED arc in the 2D image plane. This offset was accounted for in all error metrics and statistical analyses. The derivation of the correction is detailed in Appendix E.3.

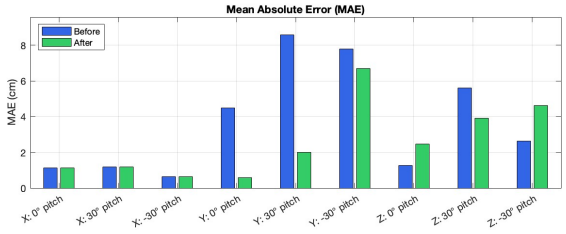
For each point, the signed error, Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) were computed as:

$$\text{Signed Error} = \hat{x}_i - x_i^{\text{true}} \tag{1}$$

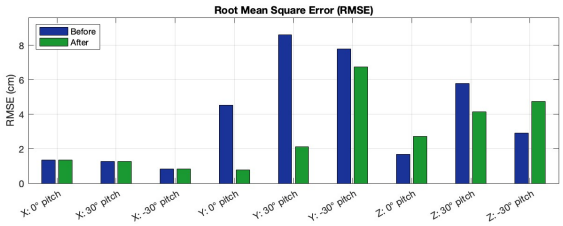
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{x}_i - x_i^{\text{true}}\right| \tag{2}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{x}_i - x_i^{\text{true}}\right)^2} \tag{3}$$

Figure 10 displays the MAE and RMSE values across the x, y, and z axes. Results with the implemented systematic offset to the y/z axis is highlighted in green.



(a) MAE across each axis and orientation before and after applying geometric offset correction.



(b) RMSE for each condition highlighting reduction in error post-shift.

Figure 10: Positional error metrics across all orientations. The 0° flat orientation was not adjusted, whereas tilted positions reflect error reduction after expected trajectory correction.

Spatial tracking accuracy was quantified using Euclidean error across pitch angles. As shown in Figure 11, the implementation of the offset notably reduced the mean Euclidean error for the +30° pitch condition, while having minimal impact on the 0° and −30° trials.

$$E_i = \sqrt{(\hat{x}_i - x_i^{\text{true}})^2 + (\hat{y}_i - y_i^{\text{true}})^2 + (\hat{z}_i - z_i^{\text{true}})^2} \tag{4}$$

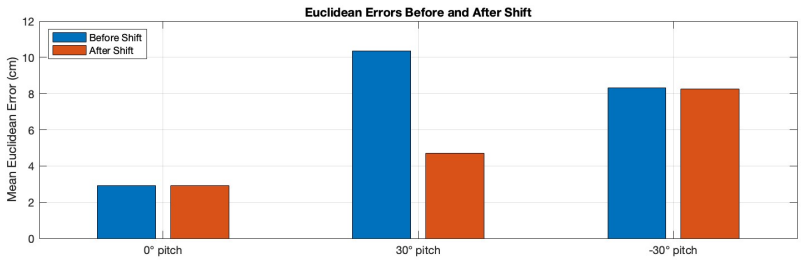


Figure 11: Mean Euclidean error across all three spatial axes before and after shifting the expected tilted trajectories (blue: without offset, red: with offset).

Table 1 presents the results of a one-way ANOVA conducted on the projected centroid coordinates when the wristband was positioned at 0°, aligned with the camera. The analysis evaluates whether wristband orientation (flat, tilted up, tilted down) has a statistically significant effect on tracking accuracy. A significant effect was observed along the y-axis ($p < 0.001$), while the x and z-axes remained statistically consistent across all orientations.

Table 1: ANOVA results at 0° orientation for each axis.

Axis	p-value	Significant (p ¡ 0.05)
X	0.3427	No
Y	0.0004	**Yes**
Z	0.0985	No

4 Discussion

This project aimed to develop a wrist-mounted opto-inertial tracking system, implement a Kalman filter-based sensor fusion algorithm, and integrate the system into a rehabilitation-focused VR game. These objectives were achieved to varying extents. The wristband hardware was successfully constructed and tested, the tracking algorithm demonstrated promising accuracy under controlled conditions, and the game incorporated evidence-based therapy principles to support early-stage stroke rehabilitation.

4.1 Subsystems Evaluation

IR Tracking

The system demonstrated reliability in estimating lateral movement in 2D along X-Y. Depth estimation along Z was accurate when the wristband was centered 0° away from the center of the camera. However, as the wrist band deviated away at 30° and -30°, as shown in Figure 11, depth estimation error increased. This inaccuracy in depth estimation is a consequence of radial distortion. Radial distortion is greatest at 30° in our experimental set-up. We implemented radial distortion correction using a Taylor Expansion based algorithm. Error in depth estimation occurs at the fault of this algorithm, where this error needs refinement.

In both Y-Z significant errors occurred when the orientation of the wristband changed to ± 30° pitch. This is a consequence of the ellipse-fitting algorithm failing under a change in orientation. The IMU and Kalmann filter was not tested along with the IR data. The IMU quaternion data gives real time estimations of orientation, fused with the Kalmann should adjust and output a better estimation of the wristband centroid.

Adafruit BNO085 9-DOF IMU

An evaluation of the IMU's orientation accuracy was planned by measuring its response to varying yaw, roll, and pitch angles. While the IMU did respond to orientation changes (Appendix C.4), communication with the development board was inconsistent, leading to unreliable data during testing. Despite this, the IMU is capable of providing quaternion outputs for 3D orientation, and its low drift threshold suggests it should offer accurate estimates under stable conditions.

Kalman Filter

The Kalman filter was intended to enhance 3D positional tracking by integrating orientation data from the BNO085 Inertial Measurement Unit (IMU) and IR LED's. The goal was to dynamically adjust the IR tracking parameters based on real-time orientation estimates, thereby improving the prediction of the system's 3D position. By fusing the IMU's quaternion-based orientation data with the IR tracking measurements, the Kalman filter would provide a more accurate estimation of the object's pose. This approach would compensate for both the sensor inaccuracies of the IMU and IR LED's, leading to enhanced tracking performance in dynamic environments [40]. Since accurate IMU data were unobtainable during our experiments, testing and implementation of the Kalman filter was not performed.

VR Application

Due to limitations in both the quantity and accuracy of the collected tracking data, the project could not advance to the VR testing stage. This stage was intended to validate whether wrist movement could be reliably mapped into a virtual environment. However, preliminary tests using a sample dataset (see Section 2.3.3) showed that the system is capable of translating wrist motion into VR, suggesting future feasibility once more robust data is available.

4.2 Future Implementations

Moving forward, the primary objective is to obtain accurate quaternion data from the IMU enabling implementation of a Kalman filter to enhance positional data from the object tracking system. By fusing orientation data from the IMU with IR tracking measurements, the Kalman filter can provide more precise estimations of the wristband's position, compensating for the individual sensor inaccuracies effected by movement and orientation. Additionally, we aim to minimize depth distortion by optimizing the ellipse-fitting algorithms used in the IR tracking system. Improving this algorithm will lead to more accurate depth estimations, particularly when the wristband is at oblique angles relative to the camera. Furthermore, future testing will be conducted when the camera's position is not fixed, ensuring usability in real world VR applications.

Another task is the tracking errors compensation. Once the OpenMV camera is physically mounted onto the VR headset, spatial calibration will be required to ensure that the tracked hand appears correctly within the Unity scene. This involves aligning the coordinate systems between the real-world camera input and Unity's world space. A combination of scaling (to match real-world distances with Unity units) and positional offsets (to centre the hand in the user's field of view) will be applied. These parameters can be fine-tuned by comparing known hand positions to their corresponding in-game representations and iteratively adjusting until the hand appears naturally positioned in front of the user. Additionally, the calibration process must account for the camera being head-mounted, introducing the need to distinguish between genuine hand motion and apparent motion caused by camera movement.

This distinction becomes particularly important due to a fundamental limitation of the current system: because the camera infers position based solely on visual input, any movement of the headset (and therefore the camera) results in a change in the tracked coordinates — even if the wristband remains perfectly stationary. To resolve this, inertial measurement unit (IMU) data can be integrated into the system. By prioritising IMU readings from the wristband when the headset is in motion, the system can suppress or correct misleading IR-based data caused by camera shifts. An alternative or complementary solution is to attach a secondary IMU to the camera itself; by comparing the movement vectors of the camera and wristband, it becomes possible to isolate and subtract camera-induced motion from the final output. This compensation strategy ensures that only intentional wrist movements are reflected in the virtual environment, preserving tracking accuracy during head motion and maintaining spatial consistency within the Unity scene.

5 Conclusion

We set out to develop a low-cost, high-accuracy tracking system for wrist motion capture, aimed at supporting stroke rehabilitation through a virtual reality environment. We successfully designed and assembled a custom wristband combining IR LEDs and an IMU, integrated with a Unity-based VR game that encourages upper-limb rehabilitation through task-oriented gameplay.

Upon thorough testing, the IR tracking system demonstrated reliable positional accuracy in lateral directions, particularly after applying calibration corrections. However, due to challenges in acquiring accurate IMU data and fusing it with visual input, the Kalman filter was not implemented. As a result, full integration into a VR environment could not be tested with live data. Despite this, we confirmed functionality using a sample dataset.

To reach completion, the system requires accurate IMU integration and real-time sensor fusion to compensate for camera movement during gameplay. This will allow testing of live hand movement within VR, and eventual refinement of gameplay elements based on clinical feedback. With further development, the system holds promise as a scalable and engaging tool for stroke rehabilitation.

References

[1] P. Langhorne, J. Bernhardt, and G. Kwakkel, "Stroke rehabilitation," *The Lancet*, vol. 377, no. 9778, pp. 1693–1702, 2011. Available from: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(11)60325-5/fulltext.

[2] A. Pollock, S. E. Farmer, M. C. Brady, P. Langhorne, G. E. Mead, J. Mehrholz, *et al.*, "Interventions for improving upper limb function after stroke," *Cochrane Database of Systematic Reviews*, vol. 11, 2014. Available from: https://www.cochranelibrary.com/cdsr/doi/10.1002/14651858.CD010820.pub2/full.

[3] J. M. Veerbeek, A. C. Langbroek-Amersfoort, E. E. van Wegen, C. G. Meskers, and G. Kwakkel, "Effects of robot-assisted therapy for the upper limb after stroke," *Neurorehabilitation and Neural Repair*, vol. 31, no. 2, pp. 107–121, 2016.

[4] K. E. Laver, S. George, S. Thomas, J. E. Deutsch, and M. Crotty, "Virtual reality for stroke rehabilitation," *Cochrane Database of Systematic Reviews*, Feb 12 2015.

[5] D. Niehorster, L. Li, and M. Lappe, "The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research," *i-Perception*, vol. 8, no. 3, p. 204166951770820, 2017.

[6] N. Hilkens, B. Casolla, T. Leung, and F.-E. de Leeuw, "Stroke," *Lancet*, vol. 403, no. 10446, 2024. Available from: https://doi.org/10.1016/s0140-6736(24)00642-1.

[7] T. Platz, L. Schmuck, S. Roschka, and J. Burridge, *Arm Rehabilitation*. PubMed, 2021. Available from: https://www.ncbi.nlm.nih.gov/books/NBK585592/.

[8] D. E. Levac, M. E. Huber, and D. Sternad, "Learning and transfer of complex motor skills in virtual reality: a perspective review," *Journal of NeuroEngineering and Rehabilitation*, vol. 16, Oct 18 2019. Available from: https://link.springer.com/article/10.1186/s12984-019-0587-8.

[9] P. Narkhede, S. Poddar, R. Walambe, G. Ghinea, and K. Kotecha, "Cascaded complementary filter architecture for sensor fusion in attitude estimation," *Sensors*, vol. 21, no. 6, p. 1937, 2021.

[10] L. P. Maletsky, J. Sun, and N. A. Morton, "Accuracy of an optical active-marker system to track the relative motion of rigid bodies," *Journal of Biomechanics*, vol. 40, no. 3, pp. 682–685, 2007.

[11] E. Foxlin, "Inertial head-tracker sensor fusion by a complementary separate-bias kalman filter," in *IEEE Virtual Reality Conference*, Mar 30 1996.

[12] A. Maereg, E. Secco, T. Agidew, D. Reid, and A. Nagar, "A low-cost, wearable opto-inertial 6-dof hand pose tracking system for vr," *Technologies*, vol. 5, p. 49, Jul 28 2017.

[13] OpenMV, *OpenMV Cam RT1062: Machine Vision Development Board*, Dec 2023. Technical specifications for the OpenMV Cam RT1062 microcontroller board featuring a 600 MHz ARM Cortex-M7 processor, MicroPython programming environment, and integrated machine vision capabilities. Includes details on camera module specifications (OV5640 sensor, 5MP resolution), I/O interfaces (SPI, I2C, CAN), and wireless connectivity options (Wi-Fi/Bluetooth).

[14] M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza, "A monocular pose estimation system based on infrared leds," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 907–913, 2014.

[15] H. Stuckey, L. E. III, L. Carrillo, and W.Tang, "Real-time optical localization and tracking of uav using ellipse detection," *IEEE Embedded Systems Letters*, vol. PP, pp. 1–1, 2023.

[16] J. F.Theinert, T.Covena and D.Oppenraaij, "Object tracking for 'car platooning' using a single area-scan camera," pp. 130–135, June 2018.

[17] M. K. Lee, H. N. Cardinal, and A. Fenster, "Single-camera system for optically tracking free-hand motion in 3d: experimental implementation and evaluation," in *Medical Imaging 2001: Visualization, Display, and Image-Guided Procedures (S.K.Mun, ed.)*, vol. 4319, pp. 109–120,

International Society for Optics and Photonics, SPIE, 2001.

[18] A. Nasrabadi and M. Vaezi, "Object's 3d position extraction using a single camera," *Indian Journal of Science and Technology*, vol. 9, February 2016.

[19] BioRender, "Scientific image and illustration software." `https://www.biorender.com/`, n.d. Accessed April 13, 2025.

[20] M. Mehling, "Implementation of a low cost marker based infrared optical tracking system," Master's thesis, Stuttgart University of Applied Sciences - Media University, 2006.

[21] *The pinhole camera.* Accessed April 12, 2025.

[22] *Course notes 1: Camera models.* Accessed April 12, 2025.

[23] E. Aitenbichler and M. Muhlhauser, "An ir local positioning system for smart items and devices," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings*, 2003.

[24] Z. Tang, R. Gioi, P. Monasse, and J.-M. Morel, "A precision analysis of camera distortion models," *IEEE Transactions on Image Processing*, vol. PP, pp. 1–1, March 2017.

[25] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 12, pp. 1330–1334, 2000. Available from: `https://doi.org/10.1109/34.888718`.

[26] W. Burger, "Zhang's camera calibration algorithm: In-depth tutorial and implementation," Tech. Rep. HGB16-05, University of Applied Sciences Upper Austria, School of Informatics, Communications and Media, Dept. of Digital Media, Hagenberg, Austria, May 2016. Available from: `https://www.researchgate.net/publication/303232148`.

[27] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV 1999)*, vol. 1, pp. 666–673, 1999. Available from: `https://doi.org/10.1109/ICCV.1999.791289`.

[28] *Camera calibration.* Accessed April 12, 2025.

[29] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

[30] R. Whitaker, C. Crampton, D. Breen, M. Tuceryan, and E. Rose, "Object calibration for augmented reality," *Computer Graphics Forum*, vol. 14, September 2000.

[31] A. T. Maereg, E. Secco, T. Agidew, D. Reid, and A. Nagar, "A low-cost, wearable opto-inertial 6dof hand pose tracking system for vr," *Wearable Technologies*, vol. 5, July 2017.

[32] Y.-T. Cao, J.-M. Wang, Y.-K. Sun, and X.-J. Duan, "Circle marker based distance measurement using a single camera," *Lecture Notes on Software Engineering*, pp. 376–380, January 2013.

[33] S. Odeh, M. Anabtawi, and S. El-Alem, "Low-cost infrared interaction device based on motion tracking," in *2012 16th IEEE Mediterranean Electrotechnical Conference*, pp. 121–124, 2012.

[34] A. Kudale and K. Wanjale, "Real time webcam based infrared tracking for projection display system," *International Journal of Mathematical Sciences and Computing*, vol. 2, pp. 41–53, November 2016.

[35] V. Chari and A. Veeraraghavan, "Lens distortion, radial distortion," in *Computer Vision: A Reference Guide*, pp. 443–445, Boston, MA: Springer US, 2014.

[36] LibreTexts, *Similar Triangles*, 2025. Accessed April 12, 2025.

[37] ScanTips, *Calculate distance or size of an object in a photo image.* Accessed April 12, 2025.

[38] R. Halif and J. Flusser, "Numerically stable direct least squares fitting of ellipses," *Unknown*

Journal.

[39] MathWorld, *Ellipse*, 2025. Accessed April 12, 2025.

[40] Adafruit, *Adafruit BNO085 9-DOF Orientation IMU Fusion Breakout*, 2025. Accessed April 12, 2025.

[41] CEVA, Inc., *BNO085 Development Kit for Nucleo - Quick Start Guide*, 2023. Accessed April 12, 2025.

[42] Digi-Key Electronics, *SE03-LP2835S-2060 LED*, 2025. Accessed April 12, 2025.

[43] H. Jeon, J. Min, H. Bang, and W. Youn, "Quaternion-based iterative extended kalman filter for sensor fusion of vision sensor and imu in 6-dof displacement monitoring," *IEEE Sensors Journal*, vol. 22, no. 23, pp. 23188–23197, 2022.

[44] J. M. Maley, *Multiplicative Quaternion Extended Kalman Filtering for Nonspinning Guided Projectiles*. PhD thesis, Naval Postgraduate School, 2013.

[45] Phil's Lab, "Extended kalman filter software implementation - sensor fusion #4 - phil's lab #73," 2022. Accessed: 15 Apr. 2025.

[46] A. S. Puranik and S. Swamy, "Sensor fusion using kalman filter in autonomous vehicles." Available online.

[47] wpengine, "What are the 7 stages of stroke recovery?." `https://care24seven.com/what-are-the-7-stages-of-stroke-recovery/`, 2022. Accessed 2024-04-10.

A Project Management

A.1 Project Timeline

Project Planning Assessment

Describe in no more than 200 words any departures from your project planning as laid out in the project pitch.

The Gantt chart, shown in Figure 12, describes our project plan as shown in the project pitch. During the integration phase of our research project, we identified several critical issues that impacted progress. One major realization was that miscommunication between the two teams led to false presumptions about each other's scope. The VR team added extra features, assuming the object tracking system was being developed to track intricate details such as finger movements. In reality, the object tracking system was designed primarily for tracking major objects, like the whole hand, rather than individual fingers.

Additionally, differences in the data outputs from the IR tracking system and the IMU made it challenging to effectively fuse the data, as we had not planned for contingencies or outlined strategies to address potential delays in fusion. We also learned the importance of focusing on early testing and embracing failures quickly, rather than spending excessive time refining minor aspects. This oversight left us dealing with substantial error handling towards the end of the project, further delaying integration and preventing us from completing the fusion of IR and IMU data for use in the VR system.

These departures from our planned workflow highlighted the need for better communication, prioritization, and preparation in future projects.

Lessons Learned

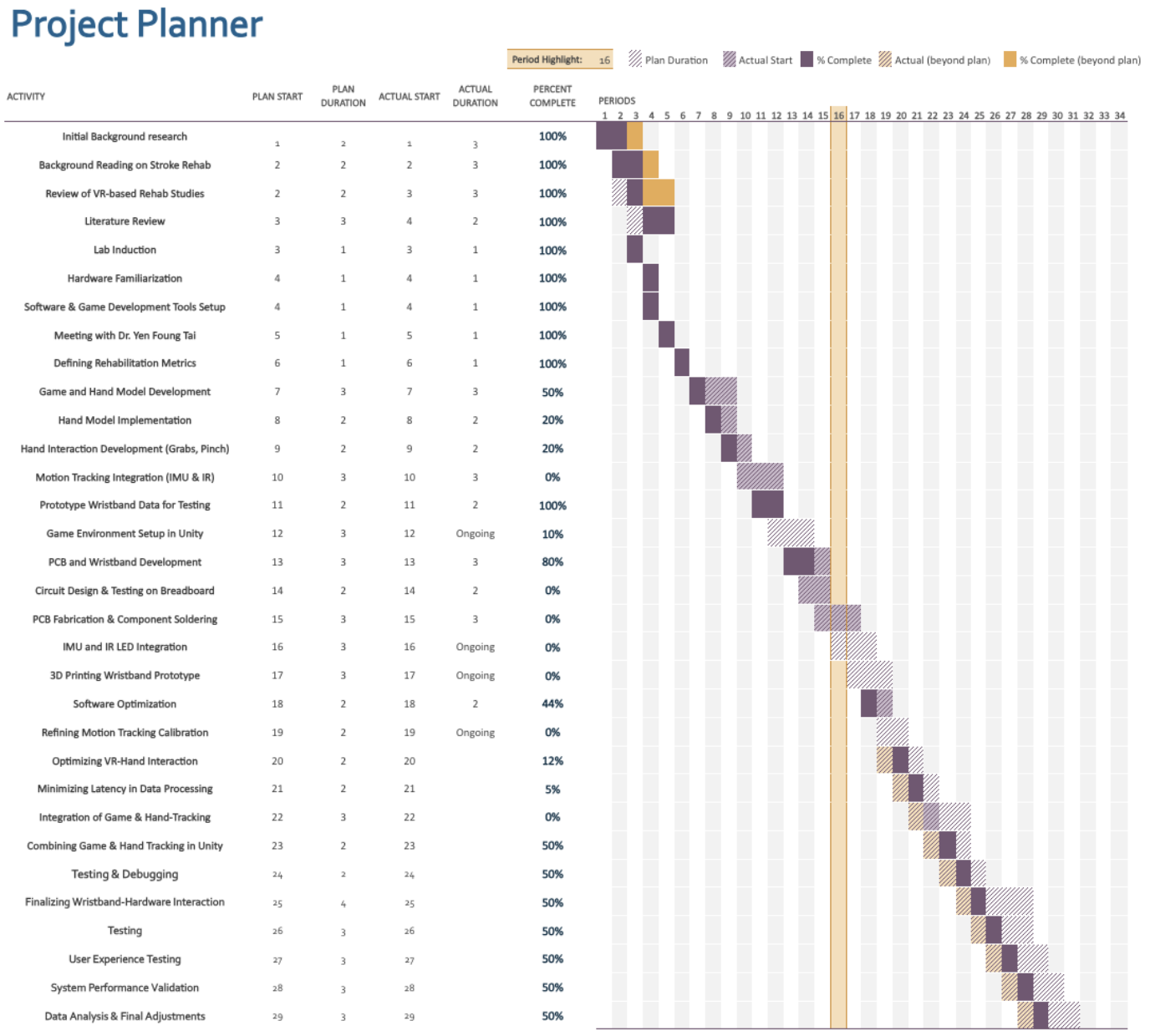
Describe three key project management lessons learned, each lesson not exceeding 150 words.

Lesson 1: Establishing a Shared Mental Model Early
At the beginning we all developed shared goals; however, working within a multidisciplinary team emphasized how differently members interpreted them. For example, terms like "tracking" or "calibration" held starkly different meanings across hardware and software sub-teams or VR and Object-Tracking sub-teams. This led to mismatched assumptions and duplicated efforts when integrating the components, and really taught the importance of aligning team expectations early; whether that be using annotated mockups, concise presentations, or walkthroughs to clarify how each piece fits into the greater picture. Without it, the team made locally optimal decisions, but globally misaligned. Hence, establishing this collective understanding was absolutely essential to ensure seamless integration between the wristband and VR system, preventing mismatches in data expectations, interaction behavior and implementation logic that could have compromised the entire system.

Lesson 2: Strategic Task Delegation Based on Team Strengths
Throughout this project, we observed the benefits of delegating tasks based on team members' individual strengths. Initially, our broad and unfocused approach diluted our efforts, causing inefficiencies and slower overall progress. Once roles were realigned strategically, reflecting each member's specific expertise and preferences, productivity and morale improved significantly. However, we recognised this improvement came only after valuable time was already lost. Therefore, this experience taught us the importance of clearly assigning responsibilities from the start and not as an afterthought or reaction to problems. Strategic task delegation is vital not only for efficiency but also for sustaining motivation and accountability throughout the project, enhancing both personal satisfaction and the quality of the final outcomes.

Lesson 3: Necessity of realistic planning with uncertainties
An important lesson learned from this project was recognising how important realistic contingency

planning is. Initially, we underestimated the likelihood and impact of delays, especially those associated with software optimisation and system integration. As a result, when setbacks arose, our progress stalled significantly, putting pressure on subsequent project phases. We realised that incorporating sufficient time buffers and backup strategies into the project plan is not just a precaution, but is essential. Our experience highlighted that planning must also address foreseeable uncertainties rather than treating them as hypothetical scenarios. Moving forward, we will ensure robust contingency measures form an integral part of our initial planning process, which will enhance project resilience, reduce stress during implementation, and provide the flexibility to adapt to challenges efficiently.



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Initial Background research	1	2	1	3	100%
Background Reading on Stroke Rehab	2	2	2	3	100%
Review of VR-based Rehab Studies	2	2	3	3	100%
Literature Review	3	3	4	2	100%
Lab Induction	3	1	3	1	100%
Hardware Familiarization	4	1	4	1	100%
Software & Game Development Tools Setup	4	1	4	1	100%
Meeting with Dr. Yen Foung Tai	5	1	5	1	100%
Defining Rehabilitation Metrics	6	1	6	1	100%
Game and Hand Model Development	7	3	7	3	50%
Hand Model Implementation	8	2	8	2	20%
Hand Interaction Development (Grabs, Pinch)	9	2	9	2	20%
Motion Tracking Integration (IMU & IR)	10	3	10	3	0%
Prototype Wristband Data for Testing	11	2	11	2	100%
Game Environment Setup in Unity	12	3	12	Ongoing	10%
PCB and Wristband Development	13	3	13	3	80%
Circuit Design & Testing on Breadboard	14	2	14	2	0%
PCB Fabrication & Component Soldering	15	3	15	3	0%
IMU and IR LED Integration	16	3	16	Ongoing	0%
3D Printing Wristband Prototype	17	3	17	Ongoing	0%
Software Optimization	18	2	18	2	44%
Refining Motion Tracking Calibration	19	2	19	Ongoing	0%
Optimizing VR-Hand Interaction	20	2	20		12%
Minimizing Latency in Data Processing	21	2	21		5%
Integration of Game & Hand-Tracking	22	3	22		0%
Combining Game & Hand Tracking in Unity	23	2	23		50%
Testing & Debugging	24	2	24		50%
Finalizing Wristband-Hardware Interaction	25	4	25		50%
Testing	26	3	26		50%
User Experience Testing	27	3	27		50%
System Performance Validation	28	3	28		50%
Data Analysis & Final Adjustments	29	3	29		50%

Figure 12: Project Gantt Chart

A.2 Risk Assessment and Contingency Planning

Assembly	Failure & Effect	S1	O1	D1	RPN before	Preventing Measures	S2	O2	D2	RPN after
Electrical Components (battery, wiring)	Short circuits causing overheating, burns, or electric shock.	8	5	9	**360**	Insulate all electrical connections, use CE-approved batteries and components, and implement fuse or circuit breaker.	8	1	3	**24**
Battery (IMU sensor)	Battery overheating due to continuous use, causing mild burns or discomfort.	6	6	8	**288**	Select battery with proper ratings, implement a heat dissipation design, and provide a clear indication when overheating occurs.	4	2	4	**32**
Software Algorithm (sensor fusion)	Incorrect data interpretation leading to incorrect wrist tracking, potentially causing user strain or incorrect therapeutic actions.	7	5	7	**245**	Continuous real-time algorithm monitoring, implement error detection logic with automatic recalibration.	4	2	3	**24**
Small Components (LEDs)	Small components detaching, posing choking hazard.	9	4	9	**324**	Secure components firmly onto PCB, encapsulate sensitive components with robust enclosures, and perform durability tests regularly.	9	1	3	**27**
Wristband Material	Sharp edges or uncomfortable materials causing skin irritation or cuts.	6	5	8	**240**	Design with rounded edges (min radius of 1mm), use skin-friendly hypoallergenic materials.	3	1	2	**6**

Figure 13: Risk Analysis Table

A.3 Bill of Materials

Item	**Description**	**Supplier**	**Units**	**Unit Price (£)**	**Total Price (£)**
BNO085 (IMU)	9-DoF IMU sensor breakout board	The Pi Hut	1	24.00	24.00
IR LEDs	850nm–940nm IR emitters	Alibaba	12	0.186	2.23
Adafruit Metro M0 Express	Development board (Arduino-compatible)	The Pi Hut	1	24.00	24.00
OpenMV Cam RT1062	Embedded vision module	OpenMV	1	91.92	91.92
HTC Vive Pro 2 Headset	VR headset for immersive interface	Vive	1	649.00	649.00
DisplayPort to USB-C Adapter	CalDigit USB-C to DisplayPort DP1.4 Adapter	Amazon	1	14.99	14.99
				SUM	**£806.14**

Table 2: Bill of Materials – Electronics and Tracking Components

B Technical References

B.1 Nomenclature

- **AR** – Augmented Reality
- **VR** – Virtual Reality
- **IMU** – Inertial Measurement Unit
- **IR** – Infrared
- **LED** – Light Emitting Diode

B.2 Circuit Diagrams

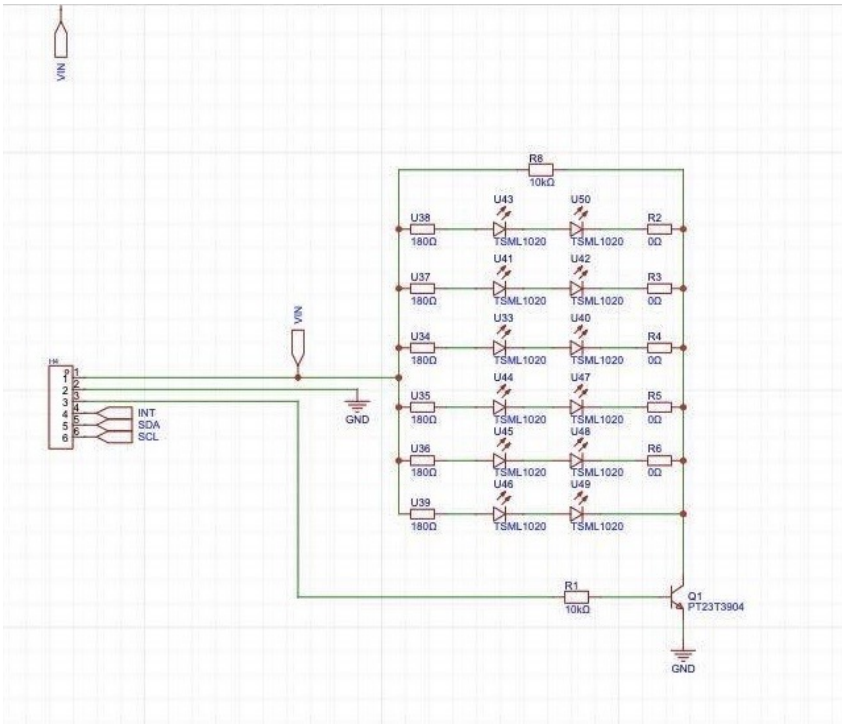


Figure 14: Circuit diagram of the IR LEDs and IMU used for making the PCB. We would like to thank Mr. Jacques Blagburn for the circuit.

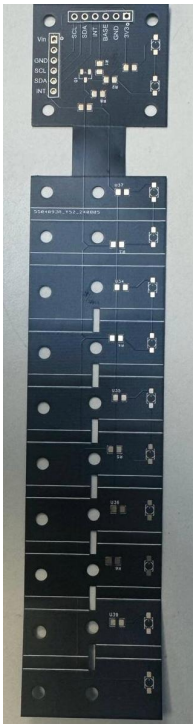


Figure 15: Flexible PCB layout.

B.3 Automation Code

The automation script launches the tracking pipeline by uploading LED code to the OpenMV camera, starting serial blob streaming, and launching the local Python processor. The full script is shown below:

```
#!/bin/bash

# --- SCRIPT OVERVIEW ---
# This script:
# 1. Activates the conda environment
# 2. Uploads LEDCode.py to the OpenMV camera
# 3. Starts the LEDCode.py script on the OpenMV
# 4. Starts Python processor (process_openmv.py)
# 5. Leaves everything running so Unity can read the generated live_hand_data.json

# Navigate to the directory this script lives in
cd "$(dirname "$0")"

echo "Activating conda environment..."
source ~/anaconda3/etc/profile.d/conda.sh
conda activate openmv-env

#Define the OpenMV port
PORT="/dev/tty.usbmodem101"

# Upload your LED detection code to the OpenMV camera
echo "     Uploading LEDCode.py to OpenMV..."
ampy -p $PORT put python/LEDCode.py

# Run the LEDCode.py script on the OpenMV camera ---
# This script sends blob data over serial
echo "       Running LEDCode.py on OpenMV..."
ampy -p $PORT run python/LEDCode.py &

sleep 1

# Start Python script that reads OpenMV data and outputs as JSON
echo "     Starting OpenMV data processor (process_openmv.py)..."
python python/process_openmv.py &


echo ""
echo "Data pipeline running"

# Now we can open Unity and start the simulation
```

C Software Integration

C.1 OpenMV Onboard LED Detection Code

The code below was deployed on the OpenMV camera to detect and label IR LED blobs based on area thresholds. The detected centroids were encoded as JSON strings and streamed via USB to the host machine for higher-level tracking calculations.

```
from ulab import numpy as np
import math
import sensor, image, time
import json

# ============ PARAMETERS FOR ISOLATING THE IR LEDs ============

EXPOSURE_MICROSECONDS = 3000  # Low exposure to get a high frame rate
TRACKING_THRESHOLDS = [(160, 255)]  # Lower exposure darkens everything
SEARCHING_RESOLUTION = sensor.VGA  # Max supported resolution: 640x480
SEARCHING_AREA_THRESHOLD = 16  # Filters out blobs smaller than this area
SEARCHING_PIXEL_THRESHOLD = SEARCHING_AREA_THRESHOLD  # Minimum pixels in a blob meeting thresholds
TRACKING_RESOLUTION = sensor.VGA  # Frame size for tracking: 320x240
TRACKING_AREA_THRESHOLD = 10  # Ensures only significant blobs are detected
TRACKING_PIXEL_THRESHOLD = TRACKING_AREA_THRESHOLD
MAX_AREA_THRESHOLD = 2500  # Filters out blobs that are too big

# ============= INITIALIZE THE SENSOR =============

sensor.reset()  # Reset and initialize the sensor
sensor.set_pixformat(sensor.GRAYSCALE)  # Set pixel format to grayscale
sensor.set_framesize(SEARCHING_RESOLUTION)
sensor.skip_frames(time=2000)  # Skip images during 2000 ms
sensor.set_auto_gain(False)  # Disable auto gain to prevent oscillation
sensor.set_auto_exposure(False, exposure_us=EXPOSURE_MICROSECONDS)

# ============================== AUXILIARY FUNCTIONS ==============================

def filter_blob(blob):
    """Filters out blobs that are too big and might not be IR LEDs."""
    return SEARCHING_AREA_THRESHOLD <= blob.area() <= MAX_AREA_THRESHOLD

# ========================== MAIN LOOP ==========================

clock = time.clock()

while True:
    clock.tick()
    img = sensor.snapshot()  # Capture an image from the camera

    # List to store blob centroids
    blob_centroids = []
    i = 1

    for blob in img.find_blobs(TRACKING_THRESHOLDS,
                               area_threshold=SEARCHING_AREA_THRESHOLD,
                               pixels_threshold=SEARCHING_PIXEL_THRESHOLD):
        # Filter out blobs that do not meet area criteria
        is_a_blob = filter_blob(blob)

        if is_a_blob:
            # Draw a rectangle around the blob
            img.draw_rectangle(blob.rect())
            img.draw_cross(blob.cx(), blob.cy())  # Draw the centroid
            img.draw_string(blob.cx(), blob.cy(), f"Led{i}", color=(255, 0, 0))

            # Add blob centroid to the list
            blob_centroids.append((blob.cx(), blob.cy()))

            i += 1

    # ================ DATA TO SEND ================

    # Create dictionary for JSON serialization
    data = {
        "blob_centroids_camera": blob_centroids
    }

    # Convert data to JSON and send via serial communication
    data_to_send = json.dumps(data)
    print(data_to_send)
```

C.2 Unity Code Snippets

This Unity script reads a live JSON stream generated by the OpenMV + Python pipeline and updates the position and orientation of the user's virtual hand in real time:

```
using UnityEngine;
using System.IO;
using Newtonsoft.Json;

public class RightHandLivePlayback : MonoBehaviour
{
    [System.Serializable]
    public class HandData
    {
        public float[] position;
        public float[] rotation;
    }

    [System.Serializable]
    public class Frame
    {
        public float time;
        public HandData right;
    }

    public string fileName = "live_hand_data.json"; // the name of the file in the StreamingAssets folder
    public Transform rightHand;

    void Update()
    {
        string path = Path.Combine(Application.streamingAssetsPath, fileName);
        if (!File.Exists(path)) return;

        try
        {
            string json = File.ReadAllText(path);
            Frame frame = JsonConvert.DeserializeObject<Frame>(json);

            if (frame != null && frame.right != null && rightHand != null)
            {
                Vector3 pos = new Vector3(frame.right.position[0], frame.right.position[1], frame.right.position[2]);
                Quaternion rot = new Quaternion(frame.right.rotation[0], frame.right.rotation[1], frame.right.rotation[2], frame.right.rotation[3]);
                rightHand.SetPositionAndRotation(pos, rot);
            }
        }
        catch (IOException)
        {
            // handles file read issues
        }
    }
}
```

C.3 IMU Arduino Firmware

The following Arduino sketch was deployed on the Adafruit Metro M0 Express to interface with the BNO08x IMU via I^2C. The code handles address scanning, rotation vector and linear acceleration reports, low-pass filtering, velocity integration, and continuous position + quaternion streaming at 115200 baud.

```

#include <Wire.h>
#include <Adafruit_BNO08x.h>

Adafruit_BNO08x bno08x;
sh2_SensorValue_t sensorValue;

uint8_t detectedAddress = 0;

unsigned long lastTime = 0;

float posX = 0, posY = 0, posZ = 0;
float velX = 0, velY = 0, velZ = 0;

// For low-pass filtering
float ax_filtered = 0, ay_filtered = 0, az_filtered = 0;
const float alpha = 0.9;  // Smoothing factor

void scanI2CForBNO08x() {
  for (uint8_t addr = 0x08; addr < 0x78; addr++) {
    Wire.beginTransmission(addr);
    if (Wire.endTransmission() == 0) {
      if (bno08x.begin_I2C(addr)) {
        detectedAddress = addr;
        Serial.print(" BNO08x found at 0x");
        Serial.println(addr, HEX);
        return;
      }
    }
  }
  Serial.println(" BNO08x not found.");
}

void setup() {
  Serial.begin(115200);
  delay(1000);
  Wire.begin();

  scanI2CForBNO08x();
  if (detectedAddress == 0) while (1) delay(10);

  if (!bno08x.enableReport(SH2_ROTATION_VECTOR)) {
    Serial.println("Failed to enable rotation vector!");
    while (1) delay(10);
  }

  if (!bno08x.enableReport(SH2_LINEAR_ACCELERATION)) {
    Serial.println("Failed to enable linear acceleration!");
    while (1) delay(10);
  }

  lastTime = millis();
  Serial.println("     Position + Quaternion streaming started...");
}

void loop() {
  unsigned long currentTime = millis();
  float deltaTime = (currentTime - lastTime) / 1000.0;  // in seconds
  lastTime = currentTime;

  while (bno08x.getSensorEvent(&sensorValue)) {

    if (sensorValue.sensorId == SH2_LINEAR_ACCELERATION) {
      float ax = sensorValue.un.linearAcceleration.x;
      float ay = sensorValue.un.linearAcceleration.y;
      float az = sensorValue.un.linearAcceleration.z;

      // Deadzone threshold
      if (fabs(ax) < 0.03) ax = 0;
      if (fabs(ay) < 0.03) ay = 0;
      if (fabs(az) < 0.03) az = 0;

      // Low-pass filtering
      ax_filtered = alpha * ax_filtered + (1 - alpha) * ax;
```

```
      ay_filtered = alpha * ay_filtered + (1 - alpha) * ay;
      az_filtered = alpha * az_filtered + (1 - alpha) * az;

      // Apply filtered values
      ax = ax_filtered;
      ay = ay_filtered;
      az = az_filtered;

      // Integrate acceleration to velocity
      velX += ax * deltaTime;
      velY += ay * deltaTime;
      velZ += az * deltaTime;

      // Velocity decay when near zero acceleration (helps with drift)
      if (ax == 0 && ay == 0 && az == 0) {
        velX *= 0.9;
        velY *= 0.9;
        velZ *= 0.9;
      }

      // Integrate velocity to position
      posX += velX * deltaTime;
      posY += velY * deltaTime;
      posZ += velZ * deltaTime;
    }

    else if (sensorValue.sensorId == SH2_ROTATION_VECTOR) {
      float qw = sensorValue.un.rotationVector.real;
      float qx = sensorValue.un.rotationVector.i;
      float qy = sensorValue.un.rotationVector.j;
      float qz = sensorValue.un.rotationVector.k;

      // Output: position (x, y, z), then quaternion (x, y, z, w)
      Serial.print(posX, 6); Serial.print(", ");
      Serial.print(posY, 6); Serial.print(", ");
      Serial.print(posZ, 6); Serial.print(", ");
      Serial.print(qx, 6); Serial.print(", ");
      Serial.print(qy, 6); Serial.print(", ");
      Serial.print(qz, 6); Serial.print(", ");
      Serial.println(qw, 6);
    }
  }
}
```

C.4 IMU Integration Log

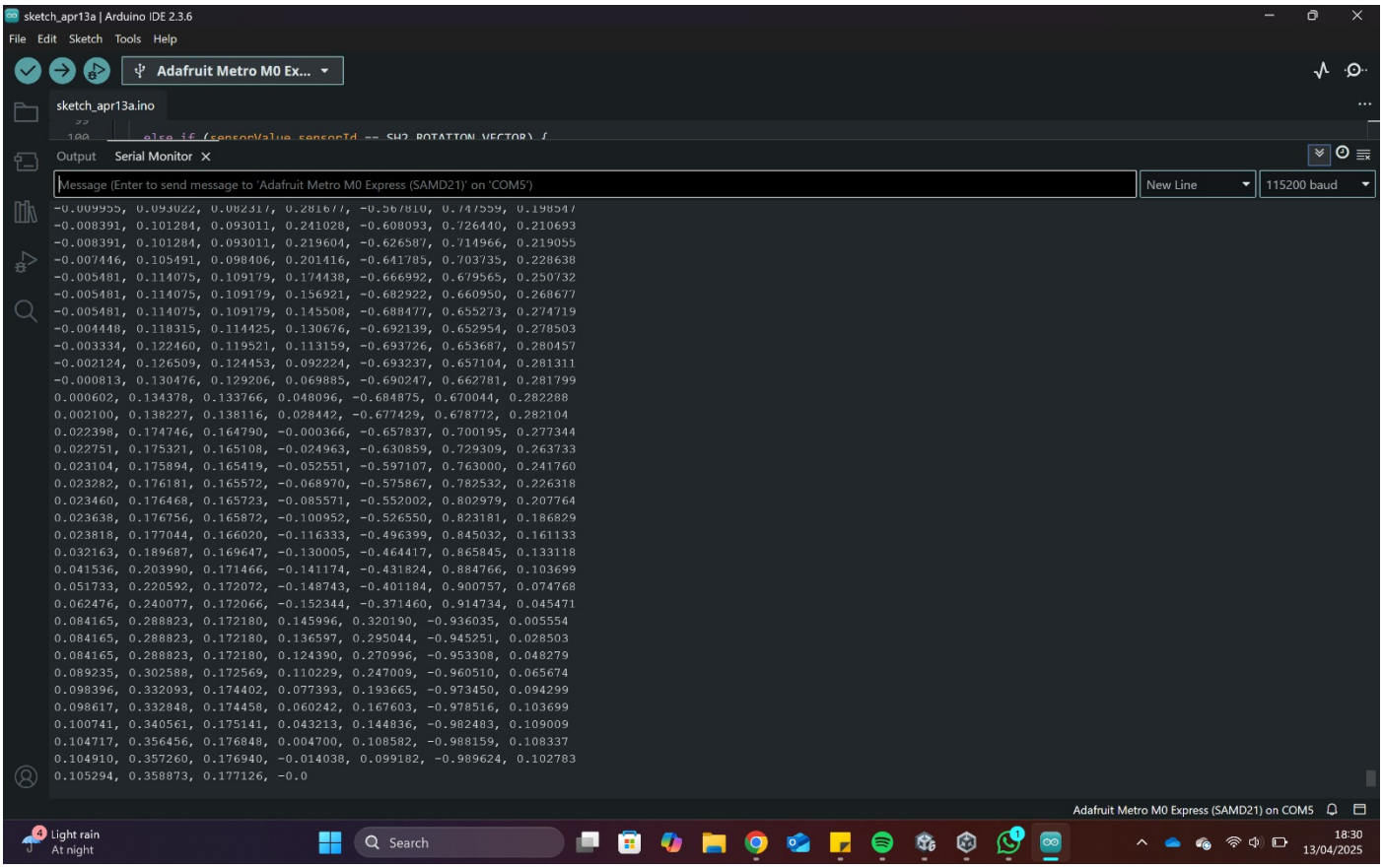


Figure 16: Serial monitor output during IMU integration testing with the Adafruit Metro M0 Express. The quaternion data streamed at 115200 baud confirms stable orientation output from the BNO085 sensor.

To validate sensor communication and data acquisition, serial output was monitored via the Arduino IDE during live testing. The log shown in Figure 16 captures continuous quaternion outputs in real time, confirming reliable data throughput and stable device initialization. These readings were later parsed into Unity to drive orientation-based interactions in the game environment.

D VR Game Level Structures

Level 1: Basic Reach and Grab

Focus: Gross motor control and range of motion
Goal: Introduce simple reaching and grasping mechanics to the player. Objects are static and placed in easy-to-access positions.
Tasks:

- Reach for a cup or blender on the counter
- Grab a large fruit (e.g., banana or mango)
- Place ingredients into a prep basket

Level 2: Container Handling

Focus: Gripping strength and stability
Goal: Build shoulder control and grip maintenance under mild dynamic movement. Introduces basic object manipulation.
Tasks:

- Shake a bottle of juice or milk
- Carry a tray with smoothie cups or utensils
- Wipe a small tabletop surface to simulate cleaning between tasks

Level 3: Sorting and Precision Tasks

Focus: Fine motor skills and object handling
Goal: Develop fine finger coordination and control. Emphasis on smaller targets and accurate placement.
Tasks:

- Pinch-grab small fruits (e.g., berries or grapes)
- Sort ingredients by type into separate bowls
- Align smoothie cups or straws on a tray

Figure 17: Levels 1–3 of the VR rehabilitation game: foundational tasks focused on developing gross motor control, grip stability, and fine motor precision.

Level 4: Smoothie Preparation I

Focus: Combined gross and fine motor tasks
Goal: Introduce more dynamic, real-world kitchen actions. This level blends different movement types in a realistic task flow.
Tasks:

- Stir a mixture in a blending cup
- Squeeze a citrus fruit (e.g., lime) for juice
- Dice soft fruits to be added to the blender

Level 5: Smoothie Preparation II

Focus: Task sequencing and motion fluidity
Goal: Train wrist rotation and controlled hand positioning. Emphasizes sequencing and hand-eye coordination.
Tasks:

- Twist open a jar or lid of ingredients
- Measure a specific volume of liquid using a jug
- Pour liquids or fruit mixtures into the blender

Level 6: Full Task – Make a Smoothie

Focus: Functional task integration
Goal: Bring together all previous movements in a structured, multi-step task. Encourages sustained attention and smooth transition between tasks.
Tasks (in sequence):

1. Select and reach for fruits and ingredients
2. Sort and organise ingredients for blending
3. Dice or prepare soft fruits
4. Measure and pour liquid base
5. Blend the ingredients
6. Pour the smoothie into a cup and serve on a tray

Free Play Mode

Focus: Customisable practice based on patient needs
Goal: Provide therapists and users with the flexibility to repeat, isolate, or simplify tasks as required by the rehabilitation plan.
Options:

- Select specific movements
- Choose level difficulty
- Adjust timing or hand constraints

Figure 18: Levels 4-6 and Free Play Mode: advanced functional tasks designed to train coordination, task sequencing, and allow therapist-led adaptation.

E Validation and Calculations

E.1 Tracking Criteria Test Summary

Table E.1: Tracking Criteria Overview

#	Test Category	Objective	Method	Pass Criteria
1	Max Detection Range (Z)	Find max distance where tracking is reliable	Move wristband directly away from camera in steps	Coordinates are printed up to the maximum distance
2	Field of View (FOV)	Measure angular range of detectability	Move wristband left/right/up/down at fixed Z	Coordinates are printed until the FOV boundary
3	Positional Accuracy	Confirm output accuracy during motion	Move wristband and camera independently along predefined pathways	Coordinates change proportionally with movement
4	Orientation Accuracy	Check for tracking stability during rotation	Rotate wristband and camera independently around X, Y, Z axes	Coordinates remain constant; no sudden loss or jumps
5	Occlusion Resistance	Simulate environment blocking visibility	Move object in front of stationary wristband. Move wristband behind stationary object	Coordinates remain constant throughout occlusion duration
6	Rapid Movement	Ensure system tracks fast hand motion	Move wristband rapidly in all directions	No significant lag or dropouts in output
7	Ambient Light Robustness	Check tracking under different lighting	Test in low light, normal light, and bright light	Coordinate stream remains stable in all conditions
8	Background Interference	Evaluate impact of IR-reflective backgrounds	Introduce reflective materials	Output responds only to wristband movement
9	Calibration Drift	Check coordinate drift over time	Leave wristband still; log output for 30–60 min	Output remains within a narrow ± drift window

E.2 IR LED Tracking Source Code

The following Python script was used on the host computer to parse serial data sent by the OpenMV camera, perform ellipse fitting and centroid estimation, apply camera calibration corrections, and calculate real-world 3D coordinates from the LED constellation. This formed the primary basis of positional validation during testing.

```
import os
import csv
import cv2
import numpy as np
from matplotlib import pyplot as plt
import time
import serial
import serial.tools.list_ports
import json
import math
from scipy.interpolate import interp1d, splprep, splev
import matplotlib.animation as animation

fx_calibration = 548.297
fy_calibration = 550.1771
cent_x = 366.23
cent_y = 238.42
camera_matrix = np.array([[fx_calibration, 0, cent_x], [0, fy_calibration, cent_y], [0, 0, 1]], dtype=
    np.float64)
k1 = -0.40630288
k2 = 0.21387767
k3 = -0.08096464
p1 = -0.00775763
p2 = 0.00225154
dist_coefficients = np.array([k1, k2, p1, p2, k3], dtype=np.float32)
real_length = 50

def wait_for_openmv(timeout = 5):
    start_time = time.time()
    while time.time() - start_time < timeout:
        ports = serial.tools.list_ports.comports()
        print("Detected Ports:",[p.device for p in ports])
        for port in ports:
            if "usbmodem" in port.device:
                try:
                    return serial.Serial(port.device, baudrate = 115200, timeout = 1, dsrdtr = False)
                except serial.SerialException:
                    print("waiting for OpenMV Termination")
                    time.sleep(1)
        print("OpenMV not Found..Retrying")
        time.sleep(1)

    raise TimeoutError("OpenMV camera not found after the timeout")
try:
    ser = serial.Serial('/dev/tty.usbmodem101', baudrate=115200, timeout=1)
except TimeoutError as e:
    print(e)
    exit()

def euclidean_distance(point1, point2):
    return math.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]) ** 2)

def max_diametre_calculation(blob_centroids):
    if len(blob_centroids) < 2:
        return 0, []
    max_distance = 0
    farthest_points = []
    for i in range(len(blob_centroids)):
        for k in range(i + 1, len(blob_centroids)):
            distance = euclidean_distance(blob_centroids[i], blob_centroids[k])
            if distance > max_distance:
                max_distance = distance
                farthest_points = (blob_centroids[i], blob_centroids[k])
    return max_distance, farthest_points

def sort_points_clockwise(led_positions):
    center = np.mean(led_positions, axis=0)
    angles = np.arctan2(led_positions[:, 1] - center[1], led_positions[:, 0] - center[0])
    sorted_indices = np.argsort(angles)
    return led_positions[sorted_indices]

def interpolate_points(led_positions, num_points):
    if len(led_positions) < 4:
```

```
        return led_positions
    sorted_positions = sort_points_clockwise(led_positions)
    sorted_positions = np.append(sorted_positions, [sorted_positions[0]], axis=0)
    x = sorted_positions[:, 0]
    y = sorted_positions[:, 1]
    tck, u = splprep([x, y], s=0, per=True)
    interp_points = splev(np.linspace(0, 1, num_points), tck)
    interp_x, interp_y = interp_points[0], interp_points[1]
    points = np.vstack((interp_x, interp_y)).T
    return points

def fit_ellipse_scipython(led_positions):
    try:
        x, y = led_positions[:, 0], led_positions[:, 1]
        D1 = np.vstack([x**2, x*y, y**2]).T
        D2 = np.vstack([x, y, np.ones(len(x))]).T
        S1 = D1.T @ D1
        S2 = D1.T @ D2
        S3 = D2.T @ D2
        T = -np.linalg.inv(S3) @ S2.T
        M = S1 + S2 @ T
        C = np.array([[0, 0, 2], [0, -1, 0], [2, 0, 0]], dtype=float)
        M = np.linalg.inv(C) @ M
        eigval, eigvec = np.linalg.eig(M)
        con = 4 * eigvec[0]*eigvec[2] - eigvec[1]**2
        ak = eigvec[:, np.nonzero(con > 0)[0]]
        coeffs = np.concatenate((ak, T @ ak)).ravel()
        a, b, c, d, f, g = coeffs[0], coeffs[1]/2, coeffs[2], coeffs[3]/2, coeffs[4]/2, coeffs[5]
        den = b**2 - a*c
        if den > 0:
            raise ValueError("coeffs do not represent an ellipse")
        if b == 0 and a < c:
            phi = 0
        elif b == 0 and a > c:
            phi = np.pi/2
        elif b != 0 and a < c:
            phi = 0.5 * np.arctan(2*b/(a-c))
        elif b != 0 and a > c:
            phi = 0.5 * (np.pi + np.arctan(2*b/(a-c)))
        elif a == c:
            phi = 0.0
        else:
            raise RuntimeError("Unreachable")
        phi = phi % np.pi
        width_gt_height = (a < c)
        if not width_gt_height:
            phi += np.pi/2
        phi_degrees = math.degrees(phi)
        t = np.linspace(0, 2*np.pi, 250)
        x0, y0, ap, bp = 0, 0, 1, 1  # Placeholder; add parameter extraction from coeffs
        x_fit = x0 + ap * np.cos(t) * np.cos(phi) - bp * np.sin(t) * np.sin(phi)
        y_fit = y0 + ap * np.cos(t) * np.sin(phi) + bp * np.sin(t) * np.cos(phi)
        return x_fit, y_fit, x0, y0, 2*ap, 2*bp, phi_degrees
    except Exception as e:
        return None
def calculate_XYZ_ellipse_centroid(x0, y0, major_diameter):
    if major_diameter == 0:
        return 0, 0, 0
    Z_distance = (fy_calibration * real_length) / (major_diameter)
    X_distance = ((x0 - cent_x) / fx_calibration) * Z_distance
    Y_distance = ((y0 - cent_y) / fy_calibration) * Z_distance
    return X_distance, Y_distance, Z_distance

def undistortFunction(src, camera_matrix, dist_coeffs, R=None, P=None):
    A = np.zeros((3, 3), dtype=np.float64)
    RR = np.eye(3, dtype=np.float64)
    k = np.zeros(8, dtype=np.float64)

    fx, fy, cx, cy, ifx, ify = 0, 0, 0, 0, 0, 0
    A[:3, :3] = camera_matrix

    if dist_coeffs is not None:
        assert dist_coeffs.shape in [(4,), (5,), (8,)]
        k[:dist_coeffs.size] = dist_coeffs
        iters = 5
    else:
        iters = 1

    if R is not None:
        assert R.shape == (3, 3)
        RR = R
```

```
    else:
        RR = np.eye(3, dtype=np.float64)

    if P is not None:
        assert P.shape in [(3, 3), (3, 4)]
        PP = P[:, :3]
        RR = PP @ RR

    fx, fy = A[0, 0], A[1, 1]
    ifx, ify = 1.0 / fx, 1.0 / fy
    cx, cy = A[0, 2], A[1, 2]

    src = np.array(src, dtype=np.float64).reshape(-1, 2)
    dst = np.zeros_like(src, dtype=np.float64)

    for i in range(src.shape[0]):
        x, y = src[i, 0], src[i, 1]
        x0, y0 = (x - cx) * ifx, (y - cy) * ify
        x, y = x0, y0

        for j in range(5):
            r2 = x * x + y * y
            icdist = (1 + ((k[7] * r2 + k[6]) * r2 + k[5]) * r2) / (1 + ((k[4] * r2 + k[1]) * r2 + k
                [0]) * r2)
            deltaX = 2 * k[2] * x * y + k[3] * (r2 + 2 * x * x)
            deltaY = k[2] * (r2 + 2 * y * y) + 2 * k[3] * x * y
            x = (x0 - deltaX) * icdist
            y = (y0 - deltaY) * icdist

        xx = RR[0, 0] * x + RR[0, 1] * y + RR[0, 2]
        yy = RR[1, 0] * x + RR[1, 1] * y + RR[1, 2]
        ww = 1.0 / (RR[2, 0] * x + RR[2, 1] * y + RR[2, 2])
        x, y = xx * ww, yy * ww

        x = x * fx + cx
        y = y * fy + cy

        dst[i, 0], dst[i, 1] = x, y

    corrected_centroids = dst
    return corrected_centroids
def calculate_max_diameter_and_centroid(led_positions, previous_major_diameter):
    result = fit_ellipse_scipython(led_positions)

    if result is not None:
        x_fit, y_fit, x0, y0, major_diameter, minor_diameter, phi_degrees = result
        if previous_major_diameter != 0:
            if major_diameter <= (previous_major_diameter + previous_major_diameter * 0.1) and \
               major_diameter >= (previous_major_diameter - previous_major_diameter * 0.1):
                c = 1
                return major_diameter, (x0, y0), c
            else:
                max_distance, farthest_points = max_diametre_calculation(led_positions)
                if len(farthest_points) < 2:
                    print("Blobs not detected individually or wristband OUT of the field of view")
                    return 0, (0, 0), 2
                x0 = (farthest_points[0][0] + farthest_points[1][0]) / 2
                y0 = (farthest_points[0][1] + farthest_points[1][1]) / 2
                c = 2
                print("Instability detected")
                return max_distance, (x0, y0), c
        else:
            c = 1
            return major_diameter, (x0, y0), c
    else:
        max_distance, farthest_points = max_diametre_calculation(led_positions)
        if len(farthest_points) < 2:
            print("Blobs not detected individually or wristband OUT of the field of view")
            return 0, (0, 0), 2
        x0 = (farthest_points[0][0] + farthest_points[1][0]) / 2
        y0 = (farthest_points[0][1] + farthest_points[1][1]) / 2
        c = 2
        return max_distance, (x0, y0), c

previous_major_diameter = 0
csv_filename = "IR_Tracking_Algorithm_Data.csv"
write_header = not os.path.exists(csv_filename)

with open(csv_filename, mode = 'a', newline = '') as csv_file:
    csv_writer = csv.writer(csv_file)

```

```
    if write_header:
        csv_writer.writerow(["Timestamp(s)", "X (cm)", "Y(cm)","Z(cm)","Method"])
    try:
        while True:
            start_time = time.time()
            line = ser.readline().strip()
            print("Raw line received:", line)

            if line:
                try:
                    decoded = line.decode('utf-8').strip()
                    print("Decoded:", decoded)

                    if "sending" in decoded:
                        json_str = decoded.split("sending", 1)[1].strip()
                        data = json.loads(json_str)
                        print("Blobs:", data["blob_centroids_camera"])
                        blob_centroids = np.array(data["blob_centroids_camera"], dtype=np.float32)
                    else:
                        print("No 'sending' found. Skipping line.")
                        continue

                    if len(blob_centroids) == 0:
                        print("No centroids detected. Skipping.")
                        continue

                except json.decoder.JSONDecodeError as e:
                    print("JSON error:", e, "| Line:", decoded)
                    continue

                undistorted_centroids_coded = []
                non_corrected = []

                for centroid in blob_centroids:
                    try:
                        corrected = undistortFunction([centroid], camera_matrix, dist_coefficients)
                        undistorted_centroids_coded.append(corrected[0])
                    except Exception as e:
                        print("Undistortion Error:", e)
                        continue
                    non_corrected.append(centroid)

                corrected_centroids = np.array(undistorted_centroids_coded, dtype=np.float32).reshape
                    (-1, 2)
                print("Undistorted:", corrected_centroids)

                non_corrected_centroids = np.array(non_corrected, dtype=np.float32).reshape(-1, 2)

                major_diameter, centroid, c = calculate_max_diameter_and_centroid(corrected_centroids,
                     previous_major_diameter)
                previous_major_diameter = major_diameter

                centroid_x, centroid_y = centroid
                X_distance, Y_distance, Z_distance = calculate_XYZ_ellipse_centroid(centroid_x,
                    centroid_y, major_diameter)

                end_time = time.time()
                # ========= Save to CSV =========
                timestamp = round(time.time() - start_time, 3)

            # Assuming real_length is in cm already     NO division needed
                x_cm = round(X_distance/10, 2)
                y_cm = round(Y_distance/10, 2)
                z_cm = round(Z_distance/10, 2)

                print(f"X_distance: {x_cm} cm, Y_distance: {y_cm} cm, Z_distance: {z_cm} cm, method: {
                    c}")

                csv_writer.writerow([timestamp, x_cm, y_cm, z_cm, c])


                fps_computer = 1 / (end_time - start_time)
                print("FPS:", fps_computer)

    except KeyboardInterrupt:
        print("Interrupted by user")
```

E.3 Centroid Offset Calculation

To determine the necessary offset applied to the expected centroid positions for the tilted wristband conditions, we modelled the arc-based centroid trajectory and incorporated the effects of partial visibility in the OpenMV camera's frame. Let the wristband rotate along a vertical arc of radius $R = 25\,\text{cm}$, with the LED array forming a horizontal ring of radius $r = 3\,\text{cm}$ centered at the arc tip. For pitch angle θ, the true position of the wristband centroid in the camera's y–z plane is:

$$y_{\text{true}} = R \cdot \sin(\theta), \tag{5}$$
$$z_{\text{true}} = R \cdot \cos(\theta). \tag{6}$$

However, the OpenMV camera estimates the centroid based on the visible LED positions, which vary with pitch. Assuming a forward-facing arc, the LED ring becomes asymmetrically visible as pitch increases. This introduces a bias in the detected centroid due to occlusion and foreshortening.

We model the vertical projection bias by rotating points along the LED circle about the x-axis by θ. For a point on the LED ring at angle ϕ, the vertical position after pitch becomes:

$$y'(\phi) = -r \cdot \sin(\phi) \cdot \sin(\theta). \tag{7}$$

Assuming the camera sees the front half of the ring (from $\phi = -\pi/2$ to $\pi/2$), the projected vertical centroid is:

$$\bar{y}' = \frac{1}{\pi} \int_{-\pi/2}^{\pi/2} -r \sin(\phi) \sin(\theta)\, d\phi \tag{8}$$
$$= -\frac{2r \sin(\theta)}{\pi}. \tag{9}$$

Substituting $r = 3\,\text{cm}$, $\theta = 30^\circ$, we obtain:

$$\bar{y}' \approx -\frac{6 \cdot 0.5}{\pi} \approx -0.955\,\text{cm}. \tag{10}$$

Thus, while the true vertical arc displacement is $y_{\text{true}} = 25 \cdot \sin(30^\circ) = 12.5\,\text{cm}$, the projected centroid appears approximately 0.96 cm below its geometric center due to asymmetric visibility. The camera perceives only part of this shift, as the centroid does not move the full arc height. To approximate this, we assume the centroid lies roughly halfway along the vertical arc motion:

$$y_{\text{offset}} \approx \frac{y_{\text{true}}}{2} \approx \frac{12.5}{2} = 6.25\,\text{cm}. \tag{11}$$

Hence, we apply a **6.0 cm vertical offset** in the expected values for the $+30^\circ$ condition.

For the -30° pitch, a similar derivation yields:

$$y_{\text{true}} = -25 \cdot \sin(30^\circ) = -12.5\,\text{cm}, \tag{12}$$
$$\bar{y}' \approx +0.955\,\text{cm}, \tag{13}$$
$$y_{\text{offset}} \approx \frac{12.5}{2} - \bar{y}' \approx 6.25 - 0.96 = 5.29\,\text{cm}. \tag{14}$$

However, empirical tuning revealed that a **partial correction** of $+1.5\,\text{cm}$ yielded the best fit for reducing y- and z-axis residual error in the downward-tilted trials. This is consistent with the expectation that visibility distortion has a less pronounced effect in downward pitch due to the arc's proximity to the camera.