# Multi-Sensor Synchronized Recording System Architecture

## System Overview and Goals

This architecture supports **synchronized multi-modal recording** using two Android smartphones (Samsung S22), each paired with a Topdon thermal camera (models TC001 and TC001 Plus), two Logitech Brio 4K USB webcams, one or more Shimmer3 GSR+ sensing devices, and a controlling Windows PC. The goal is to capture high-quality 4K RGB video (with optional RAW Bayer frames) alongside thermal imagery and physiological signals, all aligned in time with a centrally presented stimulus (e.g. emotion-evoking videos with audio). The PC serves as the **master controller** to configure devices, trigger simultaneous recording start/stop, manage calibration routines, display live previews, and play stimulus media in sync with data collection[[1]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Smartphones%20,Phone%20App%20Capabilities)[[2]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Experiment%20Control%20%2F%20Stimulus%20Presentation%3A,ends%20or%20the%20operator%20clicks). Each Android phone operates as an intelligent capture unit, recording locally but obeying the PC’s commands, and streaming low-latency previews and status updates to the PC. The design emphasizes **robustness** (data should continue recording even if connections drop), **precision timing**, and ease of use for researchers.

## Hardware Architecture and Layout

**Devices and Connections:** The system’s hardware layout is depicted below (textually):

* **Android Phones (2×):** Each Samsung S22 phone is mounted (e.g. on tripods) focusing on the subject or area of interest. Via USB-C OTG, each phone attaches to a **Topdon Thermal Camera** (TC001 on one, TC001 Plus on the other), giving a paired RGB + IR capture unit[[3]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=IR%20Camera%3A%20Capture%20infrared%20video,OTG%2C%20depending%20on%20the%20hardware). The phones connect to the PC over a Wi-Fi network for primary communications. (Optionally, a USB cable can tether phones to the PC to provide a network interface or ADB connection for more reliable high-bandwidth links[[4]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=USB%20Direct%20Connection%3A%20If%20mobility,mobility%20and%20adds%20cable%20constraints).) Each phone also has Bluetooth for connecting to Shimmer sensors if needed. To maintain power, phones should be connected to chargers or power banks during operation, as continuous 4K recording and wireless streaming are power-intensive[[5]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Additional%20Suggestions%20and%20Considerations). Ensure the phone’s USB-C port can handle OTG + charging simultaneously (if not, an OTG hub with power injection may be used).
* **Thermal Cameras (Topdon TC001/TC001 Plus ×2):** These attach to the phones and draw power from them. They capture infrared thermal video (e.g. 25–30 FPS at their native resolution, typically 256×192 or similar). The thermal sensors may produce heat; thus ensure adequate ventilation or small heatsinks if necessary. The phones and thermal units might be co-mounted so that their fields of view overlap for the same scene (to enable later pixel-aligned analysis after calibration). Each Topdon uses the manufacturer’s SDK on Android for image acquisition[[6]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L189-L197). USB drivers are handled on the phone side via the SDK (no direct PC connection for these cameras).
* **Logitech Brio 4K Webcams (×2):** These high-resolution USB cameras plug directly into the Windows PC. They can be positioned to capture additional angles (e.g. a face view of the participant, or a wide shot of the environment). Each Brio provides up to 4K30 video. For best performance, connect each to a separate USB 3.0 port/controller to avoid bandwidth bottlenecks (4K uncompressed streaming is heavy). If needed, use powered USB hubs and ensure the PC’s USB ports can supply sufficient power. The webcams will be controlled and recorded by the PC’s software (likely via a dedicated capture module using OpenCV, DirectShow or Media Foundation).
* **Shimmer3 GSR+ Sensors:** These wearable sensors (possibly worn by the participant on the fingers for GSR and with a PPG ear clip or similar) connect via Bluetooth. They can stream physiological data at high rate (e.g. 128–1024 Hz)[[7]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L90-L98). In this setup, we allow either the **PC or the phones** to be the data receiver. Ideally, the PC will pair with and collect data from the Shimmer(s) for central logging. However, as a fail-safe, the phone apps are also capable of connecting to Shimmer and recording the data[[8]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Shimmer%20Integration%3A%20Utilize%20the%20Shimmer,time%20%28if%20live)[[9]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20app,with%20too%20many%20small%20packets). This ensures at least one copy of the sensor data is captured if, say, the PC’s Bluetooth is unreliable. Only one device will actively stream from a given Shimmer at a time (since the Shimmer typically allows one BT master). Which device is master can be configured in software (with the PC instructing a phone to take over if needed). All Shimmer units and phones should be within BT range (~10m) of the PC to allow flexibility.
* **Windows PC:** A desktop or laptop with sufficient performance (dedicated GPU recommended for video decoding/encoding) runs the **Python-based controller application**. It communicates with the phones over Wi-Fi (or USB tether), interfaces with the Logitech webcams over USB, and presents stimuli (video with audio) via its display. If possible, use a dual-screen setup: one monitor for the operator UI and another (or a projector) for full-screen stimulus playback to the participant[[10]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=bindings%2C%20or%20opencv%20VideoCapture%20%2B,knows%20when%20each%20video%20starts%2Fends)[[11]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=ideally%20not%20show%20any%20of,in%20Qt%20by%20specifying%20screen). The PC should have a large SSD for storing multi-stream video files and at least USB3.0 ports for the cameras. Wired Ethernet or a dedicated Wi-Fi router for the devices can improve network reliability. The PC can also host an **NTP server** or use internet time so all devices sync clocks (discussed later).

**Power and Thermal Management:** Keep all devices powered throughout sessions. **Phones:** Running 4K recording plus Wi-Fi streaming will heat the phones significantly[[12]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Thermal%2FPerformance%3A%20Recording%204K%20on%20a,processes%20or%20using%20performance%20modes). Use external cooling if possible: e.g., attach small fans or heat-dissipating mounts to phones, especially if ambient temperature is high. Allow the phones to cool between recording sessions (or implement automated cooldown periods if doing back-to-back trials)[[12]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Thermal%2FPerformance%3A%20Recording%204K%20on%20a,processes%20or%20using%20performance%20modes). Running the phones on “High Performance” modes (if available) can help maintain frame rate but will increase heat – test to find a stable setting. **PC:** Ensure proper ventilation for the PC as it will be handling video encoding/decoding and possibly real-time preview from multiple sources. **Webcams:** Typically bus-powered, they can heat up when streaming 4K; ensure they have open air around them. **Shimmer:** Confirm the Shimmer’s battery is charged; if long experiments are planned, have spare batteries or connect it to a charger (Shimmer can sometimes be used while plugged in).

All components should be laid out so that the participant can comfortably view the stimulus screen and interact naturally, without tripping on wires. Mount cameras securely to avoid jitter. Mark all cables and use cable ties to keep the setup tidy, reducing the risk of disconnection mid-experiment.

## Software Architecture

### Android Phone Application (Kotlin)

Each phone runs an identical custom **Android app** responsible for locally capturing and buffering sensor data, while being remote-controllable. The app is built with modern Android frameworks (Camera2/CameraX, Kotlin, possibly Jetpack libraries) and uses a **service-oriented architecture** so that recording can continue in the background[[13]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L219-L223)[[14]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L220-L223). Key software components on the phone include:

* **Camera Manager (RGB Video + RAW):** Using the Camera2 API at the **FULL/Level\_3** capability, the app opens the main rear camera in 4K mode and configures a CameraCaptureSession with multiple output streams[[15]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20Capture%20,RAW%20images%20at%20intervals%20during)[[16]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=bandwidth%20%E2%80%93%20phones%20with%20Camera2,tested%20on%20the%20specific%20hardware). Likely streams: (1) a high-resolution 4K stream to an ImageReader or MediaRecorder for encoding the video to MP4, (2) a **RAW\_SENSOR** stream (Bayer RAW) to an ImageReader for capturing raw frames, and (3) a lower-resolution preview stream for local display or sending to PC[[16]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=bandwidth%20%E2%80%93%20phones%20with%20Camera2,tested%20on%20the%20specific%20hardware)[[17]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=processing,Every%20Camera2%20device%20can%20support). Many devices (especially Level\_3 devices like the S22) support 2–3 concurrent streams in one session[[16]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=bandwidth%20%E2%80%93%20phones%20with%20Camera2,tested%20on%20the%20specific%20hardware). This allows simultaneous 4K recording and periodic RAW image capture. The app will record 4K video using hardware H.264/H.265 encoder for efficiency[[18]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20RGB%20camera%E2%80%99s%20output%20will,RAW%20still%20frames%20for%20later), and can capture RAW frames at chosen intervals or on events (since continuous 30fps RAW would be impractical). If continuous RAW is absolutely required, the app might instead record a sequence of DNG images, but this will generate enormous data (hundreds of MB/s)[[19]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=video%20as%20an%20MP4%20%28e,feed%20as%20a%20secondary%20video). A more balanced approach is to record the compressed video and **occasionally capture RAW stills** (e.g. one every few seconds or on key moments) for later analysis[[20]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=enormous%20data%20,that%20IR%20and%20RGB%20can). The app should handle camera autofocus/exposure as needed or allow manual control if required (with API support for locking exposure, focus, etc., possibly triggered from PC). Audio from the phone’s microphone can also be recorded if needed, but since the primary audio (stimulus) is on the PC, phone audio might only capture participant reactions (optional).
* **Thermal Camera Module:** Using the Topdon/InfiRay **SDK** (provided in the app as a .aar or native library[[6]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L189-L197)), the app interfaces with the USB thermal camera. This likely provides a stream of thermal frames (either raw sensor counts or temperature values per pixel) at ~25–30 FPS[[21]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L74-L81). The module opens the device through the SDK (ensuring the user grants USB permission) and starts the IR frame capture thread. Thermal frames can be processed to overlay on the RGB preview (for visualization) and are also saved. The app can **record the thermal video** in parallel – e.g., by encoding an MP4 of the thermal feed, or by storing each frame as an image with timestamps. Given the relatively low resolution of thermal images, storing them as a sequence of PNGs or a Motion-JPEG video is feasible. Alternatively, the app might package thermal data into a timestamped binary file or even stream it live via a library like LSL (Lab Streaming Layer)[[22]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L199-L204). In this design, we assume the app will at least store the thermal video locally (so that nothing is lost if network issues occur). The **ThermalRepository** in the app manages enabling/disabling the IR camera and provides frames to other components[[23]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L22-L28). Thermal calibration routines (like shutter correction, dead pixel removal) provided by the SDK can be invoked at startup[[22]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L199-L204).
* **Shimmer Sensor Manager:** The app integrates the **Shimmer Android API**[[24]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=%5B1%5D%20GitHub%20) to handle Bluetooth communication with the Shimmer3 GSR+ sensor. This manager scans for the Shimmer device (by name/ID), connects, and subscribes to its data streams (GSR, PPG, accelerometer etc., configurable)[[8]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Shimmer%20Integration%3A%20Utilize%20the%20Shimmer,time%20%28if%20live). It sets the sampling rate (e.g. 512 Hz) and GSR range as needed[[25]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L156-L164)[[26]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L94-L98). Once streaming, the manager timestamps each incoming sample (using the phone’s clock or the Shimmer’s timestamp if available) and buffers it. The Shimmer data can be logged locally (e.g. to a CSV file) and/or forwarded in real-time to the PC. To avoid overwhelming the network with tiny data packets at high rate, the phone can batch the sensor readings and send periodic updates (e.g. 10–20 samples at a time)[[9]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20app,with%20too%20many%20small%20packets). If the Shimmer connection drops, the manager will attempt reconnection and notify the PC of the status. The app exposes a **ShimmerRepository** for higher-level use, which tracks connection state, battery level, and provides live data via observable streams[[27]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L11-L19)[[28]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L14-L18).
* **Local Preview & UI:** On each phone, a minimal **user interface** is provided primarily for setup and redundancy. The UI (likely a simple Activity) shows the RGB camera preview and possibly the thermal overlay or a toggle to view either[[29]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Local%20UI%20%26%20Previews%3A%20The,here%20to%20capture%20checkerboard%20images). This is useful when positioning the phone so both the RGB and IR cameras have the subject in frame. The UI can also show status indicators (e.g., “Connected to PC”, Shimmer connection state, battery)[[30]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=positioning%20the%20cameras%20,PC%20will%20be%20handling%20monitoring). Basic controls like a manual Record/Stop button (as a backup if PC control fails) and a Calibration trigger may be included. However, most control is intended to come remotely, so the UI remains clean – perhaps a single screen with the two camera previews and a few icons. The preview uses TextureView/SurfaceView for each camera feed. When the app is recording under PC control, the phone’s screen can optionally dim or turn off to save power (the recording can continue in a background service with a wake lock)[[13]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L219-L223).
* **Remote Control Service:** A background **RemoteControlService** listens for commands from the PC[[14]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L220-L223). This service manages a communication socket (or similar) and parses incoming messages (like “START”, “STOP”, etc., defined in the Protocol section below). On receiving a command, it coordinates the other managers: e.g., a START triggers the Camera Manager to start recording files, the Shimmer Manager to start logging data, etc. The service then sends acknowledgements or status updates back to the PC (e.g., “STARTED” with a timestamp, or “ERROR: camera failure”). It also ensures that if the app goes to background, these commands can still be received (using a foreground service or by utilizing Firebase Data messages or Bluetooth if using BT control). In addition, the service can handle **status queries** (responding with battery level, free storage, current recording duration, etc.) and **configuration updates** from the PC.
* **Data Handling on Phone:** The phone app writes data to local storage in an organized manner. Likely, each recording session on the phone creates a folder (with a session ID or timestamp). Within, it saves:
* RGB\_video.mp4 – the 4K camera video (with embedded timestamp track if possible).
* Thermal\_video.mp4 (or .mj2/image sequence) – thermal camera recording.
* raw\_frames/ – a directory of raw image files (e.g. DNG or PNG) if any were captured.
* shimmer.csv – timeline of sensor readings (with columns like timestamp, GSR, PPG, etc.), if the phone handled Shimmer logging.
* log.txt – optional log of events (start/stop times, errors).  
  Each phone keeps its local data until the PC retrieves it (post-session, the PC can command a transfer or the user can manually copy). This local-first approach means even if the network fails mid-recording, the phones still have the data safely on device[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch).

**Android app internal design** follows a clean architecture: for instance, using a Repository pattern for data sources (Camera, Thermal, Shimmer, LSL) and ViewModels or similar for UI state[[32]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L29-L37)[[33]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L65-L73). This separates concerns and makes it easier to maintain. Dependency injection (e.g. Hilt) is used to inject singletons like the Shimmer manager or ThermalCameraWrapper[[34]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L141-L149). The recording itself likely runs in a foreground service with high priority to avoid being killed by the OS. All heavy processing (encoding, file I/O) is done on background threads or via asynchronous coroutines so the UI thread remains responsive[[35]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L283-L287). The app should handle Android lifecycle events (pausing the preview if UI goes away, releasing cameras properly on stop, etc.). It also needs to request and handle **permissions**: Camera, Microphone (if audio), Write Storage, Bluetooth, and USB host permissions[[36]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L316-L325)[[37]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L257-L265), prompting the user as needed.

### PC Controller Application (Python)

The Windows PC runs a **Python-based controller** that provides the user interface and orchestrates all devices. The PC application has several responsibilities: 1. Provide a **GUI** for the researcher to configure the system, monitor status, and control recording/stimuli. 2. Manage **communications** with the two phones (and potentially with Shimmer if directly connected). 3. Handle **video capture** from the two Logitech webcams. 4. Execute the **stimulus presentation** (play the correct videos with precise timing). 5. Perform **calibration computations** (camera intrinsics/extrinsics) using images from phones. 6. Coordinate **data synchronization** and logging (time-stamping events, possibly merging streams post hoc).

To achieve performance and responsiveness, the PC software is designed with a hybrid approach: a Python GUI + controller layer, and a high-performance capture backend (for webcam video) possibly written in C++ or using optimized libraries[[38]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L14-L22)[[39]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L18-L26). The architecture can be outlined as follows[[39]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L18-L26)[[40]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L22-L30):

* **Graphical User Interface (GUI):** Built with a framework like **PyQt5** (Qt for Python) for a modern, responsive interface[[41]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L19-L27). The GUI runs in the main Python thread and includes windows/tabs for device status, calibration, and experiment control. It uses Qt’s capabilities to display video (via QLabel or QPixmap for frames, or QMediaPlayer for playback) and to draw simple graphs (e.g. real-time GSR plot using PyQtGraph or matplotlib)[[42]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L78-L86)[[43]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L80-L88). The UI elements include buttons (Connect, Start, Stop, Calibrate, etc.), text fields or spinners for configuration parameters, and status indicators (e.g., icons that turn green when a phone is connected, or a red “REC” indicator during recording)[[44]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Keep%20the%20UI%20layout%20intuitive%3A,video%20selection%20and%20start%2Fstop%20controls)[[45]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Provide%20logging%20or%20console%20output,research%20setting%20to%20diagnose%20issues). A console/log text box is also provided to show real-time messages and debug info (e.g. “Phone1 ACK start” or error warnings)[[45]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Provide%20logging%20or%20console%20output,research%20setting%20to%20diagnose%20issues).
* **Device Connection Manager:** This component (in Python) handles network communication with the Android phones. It either opens server sockets to listen for phone connections, or it initiates connections to known phone IPs (depending on the chosen protocol, see Communication section). For each phone, a separate thread or asynchronous task is used to send/receive messages without blocking the GUI[[46]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Device%20Connection%20Management%3A%20The%20PC,the%20PC%20can%20handle%20two). Upon a new connection, a handshake identifies the device (e.g., phone sends its ID/name). The manager updates the GUI’s device list (e.g., marking Phone A as “connected”)[[47]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=listens%20for%20incoming%20connections%20from,two%20threads%20with%20blocking%20sockets). It also manages **heartbeat** messages to detect if a phone goes offline. If a connection drops, it will alert the user and attempt reconnection. The manager funnels control commands from the GUI to the phones and routes incoming status or preview data to the appropriate handlers. It ensures thread-safe communication, using locks or Qt signals/slots to update UI from background threads[[43]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L80-L88)[[48]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L94-L101).
* **Webcam Capture Module:** To record the two Logitech 4K webcams with minimal frame drop, the PC app uses either a highly optimized Python approach (e.g., leveraging OpenCV with multithreading) or an embedded **C++ capture engine**. One strategy is to spawn a separate **C++ process** that grabs frames from the webcams (using OpenCV VideoCapture or the Media Foundation API) and writes them to video files on disk, or pipes frames to the Python app for preview[[49]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L20-L28)[[50]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L34-L42). For instance, a C++ program could open both cameras (each on its own thread for parallel capture) and start writing to two MP4 files (using ffmpeg libraries or OpenCV’s VideoWriter). This program could be launched by the Python app at “Start Recording” time, and controlled via IPC (inter-process communication) like stdout messages or a socket[[50]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L34-L42)[[51]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L54-L61). The Python side reads the C++ process output (e.g., “WEBcam1 FRAME timestamp=...”) to know it’s running[[52]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L58-L66). Alternatively, if performance allows, the Python app can use cv2.VideoCapture on two threads to pull frames and encode them. But 4K30 two streams may tax pure Python GIL, so offloading to C++ is safer. The PC ensures these webcam videos are started almost simultaneously with the phone recordings. The resulting files (e.g., webcam1.mp4, webcam2.mp4) are saved to the session folder. Optionally, a low-res preview of the webcams can be shown in the GUI (small thumbnails) to ensure they are pointing correctly. If using the C++ approach, shared memory or writing JPEGs to disk that Python loads can be used to display frames in the UI[[53]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L88-L96)[[54]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L98-L102). The architecture prioritizes **synchronization**: both webcam threads/process and the phone triggers are coordinated to minimize offset (discussed under Sync Strategy).
* **Stimulus Player:** The PC app includes an **Experiment/Stimulus module** that reads a configured playlist of stimuli (video files with audio) and plays them for the participant[[2]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Experiment%20Control%20%2F%20Stimulus%20Presentation%3A,ends%20or%20the%20operator%20clicks)[[55]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Loading%20Stimuli%3A%20Based%20on%20a,second%20monitor%20dedicated%20to%20the). This can be implemented with QtMultimedia (QMediaPlayer), which supports video with audio and can output to a designated screen[[56]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=prepare%20a%20playlist%20of%20videos,knows%20when%20each%20video%20starts%2Fends). Alternatively, integration with a robust player like **VLC** via python-vlc bindings can provide reliable playback of various video formats[[56]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=prepare%20a%20playlist%20of%20videos,knows%20when%20each%20video%20starts%2Fends)[[57]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Video%20decoding%20on%20PC%3A%20OpenCV,with%20recording%20isn%E2%80%99t%20critical%2C%20the). The player module is triggered when recording starts and can either automatically sequence through a list of videos or wait for user prompts between stimuli. It also hooks into events: for example, it emits a signal (or callback) when a video starts or ends, so the controller can log these times and possibly send markers to devices[[58]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Stimulus%20Sequence%20Control%3A%20After%20one,that%20can%20be%20handled%20too). If a “prep time” is needed, the module can display a blank or “Get Ready...” screen for a second before playing the video to ensure cameras have started[[59]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=camera%20start%20delay%29,whole%20video%20from%20the%20beginning). The stimuli configuration might be a JSON file listing each video file path and an ID or description. The PC app allows loading this file and displays the list in the UI. During playback, it might show a progress bar or timer for the operator’s reference. For best sync, if the app needs to coordinate precisely, it can delay actual playback start until it receives confirmation that phone recordings have started (or simply include a short lead-in as mentioned)[[60]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Start%20Recording%3A%20When%20the%20experimenter,Alternatively%2C%20as%20mentioned)[[59]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=camera%20start%20delay%29,whole%20video%20from%20the%20beginning). After each video, if the experiment has rest periods or questionnaires, the operator can pause or proceed as needed. The Stimulus module ensures the participant view is full-screen and free of distractions (all OS notifications disabled, etc.)[[61]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Test%20the%20playback%20for%20potential,in%20Qt%20by%20specifying%20screen).
* **Calibration Processor:** The PC software includes a **Calibration tool** (likely accessible via a “Calibration” tab in the UI). This uses **OpenCV** in Python to perform camera calibration. The process: when the user clicks “Start Calibration,” the PC sends a command to each phone to enter calibration mode[[62]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Workflow%20in%20PC%20App%3A,if). The user is prompted to hold the checkerboard target in view of the cameras. The PC can either request a frame capture from the phones on-demand (e.g., user clicks “Capture Frame” and PC sends a CAPTURE\_CALIB command)[[63]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Mode%3A%20When%20the%20PC,sure%20to%20chunk%20it%20properly), or the phone can autonomously detect a checkerboard and send frames. In a simple implementation, for each capture request, **each phone app captures a high-res still image from the RGB and IR cameras simultaneously**, then transmits those images to the PC[[63]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Mode%3A%20When%20the%20PC,sure%20to%20chunk%20it%20properly)[[64]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=captured%20images%20,we%20can%20map%20IR%20image). The PC receives pairs of images (RGB and IR). Using OpenCV, it runs findChessboardCorners on both. It may display the corners on the preview for feedback to ensure detection[[65]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20user%20to%20show%20the,the%20calibration%20results%20back%20to). The user should move the board around and capture multiple viewpoints (at least 10–20 images per camera) for a robust calibration[[66]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=visible%20image%29,when%20calibration%20quality%20is%20sufficient)[[67]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20calibration%3A%20Using%20OpenCV%E2%80%99s%20calibration,3). Once enough data is collected, the PC runs calibrateCamera() for the RGB camera (yielding intrinsic matrix K\_rgb and distortion coefficients) and for the thermal camera (K\_ir, etc.)[[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on). Then it runs stereoCalibrate() on the two sets of corner points to get the extrinsic transform (rotation R and translation T that maps the IR camera coordinate system to the RGB’s)[[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on). The resulting calibration parameters – intrinsics for each lens and the extrinsic R, T – are saved to a file (e.g., calibration\_phone1.yaml and calibration\_phone2.yaml)[[69]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=camera%E2%80%99s%20set%20and%20cv,if%20doing%20augmented%20feedback%2C%20etc)[[66]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=visible%20image%29,when%20calibration%20quality%20is%20sufficient). The PC can also send these results back to the phones (the phone app might store its intrinsics if it wants to do on-device overlay or undistortion)[[70]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=applicable%29,if%20doing%20augmented%20feedback%2C%20etc). The calibration tab in the UI can show the **reprojection error** to inform the user of calibration quality[[71]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=for%20later%20use%20in%20analysis,when%20calibration%20quality%20is%20sufficient). For thermal cameras, note that a normal chessboard may not be visible; the calibration target must have a thermal contrast pattern (e.g., a heated board with cooler chess squares or an emissive material pattern)[[72]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20specific%20hardware)[[73]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Use%20an%20external%20IR%20camera,to%20integrate%20such%20cameras). This is addressed in documentation and references[[74]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20Capture%20,does%20support%20it%2C%20you%20can)[[75]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=10,3). The PC software might integrate any special calibration procedures (like using an **infrared calibration toolkit** if needed) but standard OpenCV works if the pattern is detectable[[75]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=10,3).
* **Data Management and Logging:** The PC application coordinates the creation of a **session directory** where all data will be collected. When a new session/trial is started (e.g. by entering a participant ID and clicking “New Session”), the software creates a folder, e.g. Session\_2025-07-27\_16-30-00\_Ppt01/. During recording, it knows the filenames that the phones will produce (the phone can send the actual names or they are pre-defined conventions). It also knows its own files (webcam videos, etc.). The PC logs a **metadata file** (JSON or CSV) that contains key information: session ID, start time (PC clock), phone A start time (according to PC, or the offset), phone B start time, which files belong to this session (with checksums maybe), any notable event timestamps (stimulus events, etc.), and config settings used[[76]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Synchronization%20and%20Merging%3A%20In,This%20helps%20in%20later%20analysis). This metadata greatly aids post-hoc data alignment and ensures traceability. After “Stop Recording,” the PC can automatically **fetch files** from the phones. For example, the phones might start an HTTP server or use an RPC call to send the files. Or the PC could use ADB (if phones are USB connected) to pull the files[[77][78]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=overwhelm%20the%20network%29,after%20capture%20would%20be%20useful). If network bandwidth allows, the PC software could initiate a transfer of the just-recorded videos; progress is shown in the UI (given 4K video sizes, this might take some time)[[79]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=%E2%80%9CStop%E2%80%9D,clearly%20indicate%20recording%20has%20stopped)[[80]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Post,at). If immediate transfer isn’t feasible (too slow), the PC at least pulls critical data (like the sensor log or a lower-res copy of video) and the rest can be copied manually later[[80]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Post,at). The UI will reflect transfer status or advise the user to retrieve the files. Additionally, the PC app might integrate a feature to automatically push the collected data to a backup location (network drive or cloud) after each session, given the volume of data (to avoid loss and help manage storage)[[81]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Management%3A%204K%20video%20files,world%20workflow).
* **Error Handling and Recovery:** The PC software is built to handle common error scenarios gracefully. For instance, if one phone fails to acknowledge a Start command, the UI will alert the operator and perhaps give the option to retry or continue with partial data. If a phone disconnects during recording, the PC will log the event; the phone (by design) should continue recording on its own[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch). The PC may attempt to reconnect; if it succeeds, it can send a Stop command at the end. If not, the phone user may have to stop it manually. All such events (and any errors like “low disk space” or “camera failure”) are recorded in the PC log console and maybe written to a session\_log.txt. The UI should highlight faults (e.g., a warning icon next to a device that disconnects). **Fault tolerance measures:** if the Wi-Fi link is unstable, the system can optionally fall back to an alternative path (for example, if a phone also has Bluetooth paired to the PC as backup, or if USB tethering is available, the operator could switch to that). These are manual interventions unless automated link redundancy is built in. For Shimmer, if the PC is primary but loses connection, one of the phones can detect this (PC not sending heartbeat) and automatically attempt to connect to the Shimmer to continue data capture – ensuring “at least one always succeeds” in logging GSR[[82]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20impact%20of%20network%20latency,data%20by%20the%20shared%20start)[[83]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=triggers,by%20the%20shared%20start%20time). Such logic can be built into the coordination: the Shimmer could be *dual-paired* (in memory) with both PC and phone, though only one actively streams at a time. The phone app could start streaming if it notices PC dropped. The data from the phone’s portion and PC’s portion would later be merged. This is complex but provides redundancy. At minimum, **if Shimmer fails on one device, the operator can manually start it on another device mid-session** if possible. The architecture also uses timeouts – e.g., if a phone does not respond within X seconds to a command, the PC notifies the user. Conversely, if the PC app crashes or the PC itself fails, the phones (which are recording locally) will continue until a predefined timeout or until storage fills. For safety, the phone app could have a *max recording duration* that it honors (configured before start) – for instance, if no stop command comes after 30 minutes, it stops and saves file to prevent endless recording[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch). Overall, no single point of failure should lead to data loss: each critical sensor has a local recording backup.
* **Security Considerations:** If the data is sensitive (which it may be, containing physiological responses and video of participants), communications should be kept local (no external servers). The PC can act as a local router. We can also use encrypted channels – e.g., running the command/control protocol over TLS (SSL) or using SSH port forwarding. If using MQTT, set it up with authentication on a closed network. Ensure all data files are stored securely and consider encryption if needed. These measures are outside core functionality but recommended[[84]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Security%3A%20If%20this%20is%20used,case%20the%20network%20setup%20changes).

### Communication Protocol (PC–Phone Synchronization)

A **reliable, low-latency communication protocol** is essential for synchronizing start/stop commands and streaming previews/status. We have several options, but the chosen design uses **Wi-Fi TCP sockets with a custom message protocol**, plus fallback channels if needed:

* **Connection Setup:** On launch, the PC app opens a listening socket (e.g., TCP on a specific port) for each phone to connect, or a single port that can accept multiple clients. The phones, upon app start, attempt to connect to the PC’s IP (which can be configured in the app or discovered via multicast DNS or QR code scanning). If using USB tethering, the PC will have a known IP (e.g., 192.168.42.1) to connect to. Optionally, if using WebSockets or HTTP, the phone could make a WebSocket connection to a server in the PC app[[85]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Network%20Sockets%20or%20HTTP%3A%20For,frames%20as%20they%20become%20available). Another robust option is **MQTT**: the PC runs an MQTT broker and each phone connects as a client (topics for commands, etc.)[[86]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=MQTT%20or%20Pub%2FSub%3A%20Alternatively%2C%20use,the%20broker). MQTT adds overhead and single point of failure (the broker), so a direct socket is simpler here. We assume a direct socket with our own protocol.
* **Message Format:** Use a text-based JSON message format for readability, or a lightweight binary if performance dictates. For clarity, JSON is fine for command/control since frequency is low. For example, a “Start Recording” command could be:
* {"cmd": "START\_RECORD", "start\_time": "<iso-timestamp>", "config": {...}}
* The phone responds with {"resp":"ACK", "cmd":"START\_RECORD", "phone\_time":123456789} indicating it will start. Status updates might be: {"event":"status", "battery":85, "free\_space": "12GB", "recording": true, "elapsed": 5.2}. Preview frames might be sent as separate binary messages or via a secondary channel (discussed below). By using JSON, it’s easy to extend (e.g., adding a “mode” or file info). WebSocket could carry these JSON messages neatly if used[[85]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Network%20Sockets%20or%20HTTP%3A%20For,frames%20as%20they%20become%20available). Over TCP, messages should be delimited (e.g., newline) or length-prefixed to allow the receiver to parse streams correctly.
* **Command Types:**
* **CONFIGURE:** Sent from PC to phone to set parameters (resolution, frame rate, etc.) before recording[[87]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Configuration%20Interface%3A%20Provide%20controls%20to,This%20could%20include)[[88]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=After%20the%20user%20sets%20these,UI%20with%20the%20confirmed%20settings). Payload includes settings like {"camera\_res":"3840x2160","fps":30,"thermal\_on":true,"shimmer\_on":false}. Phone applies and replies ACK or error if unsupported[[88]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=After%20the%20user%20sets%20these,UI%20with%20the%20confirmed%20settings).
* **START\_RECORD:** Instructs phone to begin recording. May include a start\_time in the future (see Sync below)[[89]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=an%20NTP%20server%20or%20use,Shimmer%E2%80%99s%20documentation%2C%20their%20software%20supports)[[90]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=start%20timestamp%20a%20couple%20of,a%20marker%20in%20the%20data). Phone replies ACK immediately if ready (or NACK if not ready).
* **STOP\_RECORD:** Instructs to stop recording. Phone replies when done (and perhaps provides filenames or file sizes).
* **CAPTURE\_FRAME:** (Calibration) Tell phone to capture a still image (or a pair from both cameras) for calibration[[63]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Mode%3A%20When%20the%20PC,sure%20to%20chunk%20it%20properly). Phone responds with the image data or indicates it’s sending via a separate channel.
* **STATUS\_REQUEST:** Poll for status. Phone can respond with current status (battery, storage, recording state, last error). Alternatively, phones can send periodic **STATUS** messages without request.
* **HEARTBEAT:** Simple keepalive pings to ensure connection (could be just a periodic “ping” message from either side).
* **PREVIEW\_ON/OFF:** If we want to enable/disable live preview streaming from the phone to save bandwidth during certain times.
* **FILE\_LIST / FILE\_TRANSFER:** After recording, PC could request a directory listing or specific file from phone. Or phone could initiate sending. This could also be done outside the main socket (e.g., phone starts an HTTP server for file download).
* **Preview Streaming:** For real-time monitoring, each phone will transmit a video preview stream. As sending raw 4K frames is infeasible over Wi-Fi, the phone should downsample/compress this preview[[91]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Streams%3A%20For%20the%20preview,frame%20rate%20preview%20just%20for)[[92]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=sending%20full%204K%20raw%20frames,the%20operator%20sees%20a%20near). Two approaches:
* **JPEG frames over socket:** The phone periodically (say 5–10 fps) takes a preview frame, compresses it to JPEG (e.g., 720p resolution), and sends the byte array with a header (length). The PC receives it, decodes to an image (with OpenCV or PIL), and displays in the GUI[[93]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Preview%20Display%3A%20For%20each%20phone%2C,to%20embed%20a%20video%20player)[[94]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=streams%20in%20one%20app%20might,to%20save). This is simpler to implement. The frame rate can be adjusted to balance latency vs. load.
* **Real video stream:** The phone uses MediaCodec to encode a low-bitrate H.264 stream of the preview. It could either open an RTSP server (using something like libVLC or an Android streaming library)[[95]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=can%20send%20compressed%20low,running%20the%20preview%20stream%20in), or directly packetize the H.264 stream over the socket. The PC would then use a decoder (ffmpeg, OpenCV VideoCapture with RTSP URL, or a GStreamer pipeline) to play the stream. This provides smoother preview at the cost of complexity. Given our requirements, **JPEG-over-socket** at ~2-5 fps per camera is often sufficient to confirm framing and focus[[96]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=streams%20in%20one%20app%20might,during%20actual%20recording%20if%20needed), since we don’t need full frame rate monitoring. We will implement a toggle in the UI to turn previews on/off to save bandwidth during critical recording if needed[[96]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=streams%20in%20one%20app%20might,during%20actual%20recording%20if%20needed). Each preview frame can carry a timestamp so we know when it was captured, but slight lag in preview doesn’t affect data sync.
* **Resilience of Comms:** The TCP socket approach ensures reliable delivery of commands. If using WebSocket, built-in keepalive and reconnection strategies can be used. We will implement a heartbeat: e.g., every 2 seconds the PC sends a small ping and expects a pong from each phone. If missed for, say, >5 seconds, the PC flags that connection is lost. The phone similarly can monitor if it hasn’t heard from PC and attempt to reconnect. In case of connection loss: if during an active recording, the phone continues unaffected. The PC will try to reconnect in the background (perhaps with exponential backoff). If reconnected, the PC may query the phone’s status (“are you still recording?”) and update UI accordingly[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch).
* **Alternate Channels:** In addition to Wi-Fi, we have **ADB** and **Bluetooth** as backup control channels:
* *ADB:* If the phones are connected via USB (and developer mode enabled), the PC can use adb shell commands to start/stop the recording service as a fail-safe[[97]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L149-L157). For example, if the network socket fails right when we need to start, the PC app could invoke an ADB command: adb shell am startservice -n com.gsr.dualvideostream/.DualCaptureService --ez start 1 as given in the app documentation[[98]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L150-L158). This is not primary but is a useful backdoor for development and emergencies.
* *Bluetooth SPP:* The phones also have a Bluetooth Remote Control protocol (simple serial commands)[[99]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L160-L168). The PC could connect to the phone over Bluetooth (if paired) and send “START” or “STOP” ASCII commands as per the app’s implementation[[99]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L160-L168). This is slower and less convenient than Wi-Fi, but it’s another layer of redundancy in case Wi-Fi is completely down. However, using BT for both Shimmer and control might crowd the BT radio, so this is optional.

In summary, the PC and phones maintain an **always-on communication link** (Wi-Fi socket), with clearly defined message flows. The system is designed such that even if that link breaks, each device continues its critical task and no data is lost; the link primarily coordinates and provides quality-of-life feedback.

### Time Synchronization Strategy

Accurate time sync between all streams is vital. Our strategy involves both **synchronized start triggers** and post-recording **timestamp alignment**:

* **Shared Clock Reference:** Ideally, all devices should share a common time base. In practice, the PC’s clock can act as the master reference (since it triggers events). We consider using NTP: if the phones and PC are online, they could sync to a time server before experiments. However, a simpler method is a **direct sync handshake** with the PC. For example, upon connection, each phone can request the current PC time; the PC sends its timestamp, and the phone compares it to its own clock, deriving an offset. (This is essentially what NTP does; doing it multiple times and averaging helps account for network latency.) The phone can then log all its data in PC-time (by applying the offset to its SystemClock readings). Even if it doesn’t adjust its internal clock, it can tag each recorded frame with the PC time. Alternatively, the PC’s “START\_RECORD” command can include a scheduled start time slightly in the future[[89]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=an%20NTP%20server%20or%20use,Shimmer%E2%80%99s%20documentation%2C%20their%20software%20supports)[[90]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=start%20timestamp%20a%20couple%20of,a%20marker%20in%20the%20data). E.g., PC says “START at time = 12:00:05.000”. Each phone receives this and knows how far in the future that is relative to its clock, and prepares to trigger exactly then. This method can yield sub-50ms accuracy in start time alignment, limited mostly by clock drift and command propagation delay.
* **Command Propagation Delay:** With Wi-Fi, typical latencies are a few milliseconds on a local network. Including a future start time of even 2 seconds later ensures both phones have the command well ahead of the actual start, nullifying network jitter issues[[89]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=an%20NTP%20server%20or%20use,Shimmer%E2%80%99s%20documentation%2C%20their%20software%20supports)[[90]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=start%20timestamp%20a%20couple%20of,a%20marker%20in%20the%20data). Both phones then hit “record” at their local time that corresponds to the agreed start. They also each note the exact moment in their local clock and perhaps include that in their metadata. The PC does the same (notes when it thinks start happened). Using these, we can later verify the offsets.
* **Ongoing Sync & Drift:** If recordings run for long durations, clock drift could introduce offset over time. The phones’ internal clocks might drift relative to PC by a few milliseconds per minute. Since we cannot easily enforce constant sync (unless using specialized protocols or continuous NTP sync, which is complex), the post-hoc alignment is important. The phones timestamp each video frame (e.g., camera timestamps from Sensor API or simply the system time on frame capture) – these can be stored in the video file metadata or a sidecar file. The Shimmer data likewise gets timestamps (either from PC if PC is logging it, or from phone’s clock if phone logs). Later, during analysis, one can adjust those timestamps by the known offset (if the phone clock was X ms behind/ahead of PC). The **Lab Streaming Layer (LSL)** approach is an alternative: our system includes LSL integration stubs, which when activated, allow all devices to contribute to a common time-synchronized stream network[[100]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L205-L213)[[101]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L215-L219). LSL performs its own clock offset estimation between streams. We could use LSL even just for time sync: e.g., have the PC and phones each create a small LSL outlet and subscribe to each other – LSL’s internal timing could then be used to correlate clocks (this is advanced and beyond current scope; the simpler timestamp exchange should suffice).
* **Stimulus Sync:** The PC, being the source of stimuli, inherently knows the timeline of events (video start/stop times). The moment the PC starts playing the first stimulus, that is effectively time zero of the “experiment timeline.” We ensure recordings include this point. If using the scheduled-start approach, we might align the stimulus start exactly with the recording start. If instead we do a manual short delay, we can start recordings slightly before playing the video. In practice, one approach: PC sends START, waits 0.5 seconds, then plays video[[59]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=camera%20start%20delay%29,whole%20video%20from%20the%20beginning). This way the cameras are already rolling when the video begins, capturing the full stimulus from frame one. The PC logs “Video1 started at PC\_time T0”. Because the phones can translate their frame timestamps to PC\_time, we can later find which video frame corresponds to stimulus start. For additional precision, we could embed a **sync flash or sound** at the start of the stimulus that all sensors capture[[102]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=needed%2C%20consider%20adding%20a%20sync,will%20be%20sufficient%20for%20most). For example, a brief 50ms beep or a single white frame flash on screen at t=0. This would appear in the audio track (if any audio recorded) and possibly the video (the flash illuminating the scene), and could even cause a tiny response in GSR. In post-analysis, this provides a clear alignment marker across modalities[[102]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=needed%2C%20consider%20adding%20a%20sync,will%20be%20sufficient%20for%20most). This is optional; often the software timestamp alignment is enough for small (<100ms) accuracy needs.
* **Across Devices (Two Phones & PC & Shimmer):** We measure initial sync by a test: e.g., clapping in view of both phone cameras or using a LED visible to both[[103]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Synchronization%20Between%20Two%20Phones%3A%20Starting,it%20in%20software%20if%20needed). By checking the frame timestamps or audio of the clap, we can quantify any offset between phone A and B recordings and adjust in software. If a consistent bias is found (say Phone1 frames always 50ms later than Phone2), we can incorporate that offset in the analysis or even adjust one phone’s start slightly earlier next time. The system can be calibrated for sync by doing such tests and storing a sync offset parameter.
* **Shimmer Synchronization:** If the Shimmer is logged on the PC, it inherently shares the PC’s timeline (e.g., PC starts logging at time T0 exactly). If logged on a phone, we rely on the phone’s timestamp alignment. The phone would note when it started recording relative to the PC command. For example, phone gets “START at 12:00:05”, it starts its Shimmer log with an entry “Start marker = 12:00:05 PC time (which was 8:00:00 phone time)”. The Shimmer samples then come with phone times, but can be converted. The Shimmer’s internal clock might also produce timestamps for each sample (some Shimmer firmware do) – but those might be relative to the connection start, not absolute. We can simply treat the first sample as t=0 of the recording and align that to PC time of start. For multi-shimmer setups, Shimmer has its own sync mechanism (their Android “Multi Shimmer Sync” achieves ~1ms accuracy between multiple sensors and external triggers by using a common reference or trigger pulse[[104]](https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/#:~:text=,Camera%20based%20motion%20capture%20systems)). We won’t reach 1ms alignment between video and GSR without specialized hardware trigger, but our goal is within ~20-50ms which is fine for physiological responses. If needed, one could plug a Shimmer’s digital input to an external sync pulse (like a microcontroller that flashes an LED and sends a trigger to Shimmer)[[105]](https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/#:~:text=The%20application%20will%20allow%E2%80%A6)[[106]](https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/#:~:text=accelerometer%20range%20,settings%20into%20the%20application%20when) – but that complicates the hardware. Our software approach, combined with possibly an audio/visual sync mark, suffices[[102]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=needed%2C%20consider%20adding%20a%20sync,will%20be%20sufficient%20for%20most).

In summary, **synchronization** is handled by: synchronized start via scheduled commands, continuous timestamp logging, and optional calibration of offsets via test signals. The architecture logs all relevant timing info (e.g., the exact PC time a command was sent and phone’s acknowledgment with its time) so that any drift or latency can later be accounted for in data analysis[[107]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Timing%20%26%20Sync%3A%20It%E2%80%99s%20crucial,some)[[82]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20impact%20of%20network%20latency,data%20by%20the%20shared%20start).

### Data Storage, Formats, and Session Management

Each recording session produces a wealth of data from multiple sources. Organizing and managing these files systematically is crucial:

* **Session Folder Structure:** For each session or participant trial, the PC designates a folder (as mentioned, e.g., Session\_<timestamp>\_<ID>). Within it, sub-folders or naming conventions separate data by device and sensor:
* **Phone1\_main.mp4:** RGB video from phone 1 (4K compressed video). Ideally named with phone ID and maybe camera (e.g., phone1\_rgb.mp4). If audio was captured by phone, it will be in this file’s audio track.
* **Phone1\_ir.mp4:** Thermal video from phone 1. Possibly saved as an MP4 with a fixed palette. If radiometric data is needed, an alternative is to save a CSV of temperature per frame or a binary dump. However, given storage and ease, we opt to save an MP4 where pixel intensity correlates to temperature (with a known formula). Additionally, a calibration file (maybe in the metadata JSON) can store the mapping of grayscale to temperature for that session.
* **Phone1\_raw/**: directory of RAW image frames (if any). Could contain .DNG files or .jpg/png from full-res stills. If only a few captured, they can be standalone files. If a high-frequency raw sequence was recorded, consider saving as a video (some formats allow raw, or even a HEIF sequence).
* **Phone1\_shimmer.csv:** If phone1 was capturing Shimmer, a CSV (or .edf or .mat) file containing the sensor readings. Columns: timestamp, GSR, PPG, etc. Timestamp could either be absolute (epoch or ISO) or relative to start=0. We will include enough info to map it to the unified timeline.
* **Phone2\_rgb.mp4, Phone2\_ir.mp4, Phone2\_raw/, Phone2\_shimmer.csv:** similarly for phone 2.
* **Webcam1.mp4, Webcam2.mp4:** videos from the Logitech webcams recorded by PC.
* **stimuli/** (optional): copies of the stimulus videos or references to them, if needed for analysis (or at least filenames listed in metadata).
* **calibration\_phone1.yaml, calibration\_phone2.yaml:** saved calibration parameters (intrinsic/extrinsic) for each phone’s cameras[[66]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=visible%20image%29,when%20calibration%20quality%20is%20sufficient)[[108]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=stereo%20rotation%2Ftranslation,when%20calibration%20quality%20is%20sufficient). Possibly also include webcam intrinsics if those were calibrated (one might calibrate the webcams with a standard method if using their data for any measurement).
* **session\_log.csv / metadata.json:** a master log file capturing key events and info[[76]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Synchronization%20and%20Merging%3A%20In,This%20helps%20in%20later%20analysis). For example, a JSON might look like:
* {  
   "session\_id": "P01\_trial1\_2025-07-27\_16-30",  
   "pc\_start\_time\_utc": "2025-07-27T15:30:00.123Z",  
   "phones": {  
   "phone1": {  
   "device\_name": "S22\_1",  
   "start\_time\_phone\_clock": 1234567890.000,  
   "start\_time\_pc\_clock": 1627380000.120,  
   "file\_rgb": "phone1\_rgb.mp4",  
   "file\_ir": "phone1\_ir.mp4",  
   "file\_shimmer": "phone1\_shimmer.csv",  
   "calibration\_file": "calibration\_phone1.yaml"  
   },  
   "phone2": { ... similar ... }  
   },  
   "webcams": {  
   "webcam1\_file": "webcam1.mp4",  
   "webcam2\_file": "webcam2.mp4"  
   },  
   "stimuli\_sequence": [  
   {"video": "stim1\_happy.mp4", "started\_pc\_time": 1627380005.500},  
   {"video": "stim2\_sad.mp4", "started\_pc\_time": 1627380100.000}  
   ]  
  }
* This metadata allows any post-processing script to ingest all data and synchronize via the provided timestamps[[76]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Synchronization%20and%20Merging%3A%20In,This%20helps%20in%20later%20analysis)[[109]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=analysis%20script,This%20helps%20in%20later%20analysis). It also serves as documentation for that session.
* **File Formats:** We choose standardized, widely compatible formats:
* Video files in **MP4 (MPEG-4)** container with H.264 or H.265 codec for compression[[18]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20RGB%20camera%E2%80%99s%20output%20will,RAW%20still%20frames%20for%20later). H.264 is universally readable; H.265 offers better compression but ensure the lab has tools to read it. Each MP4 will have a continuous timestamp track (implicitly from start, with frame timestamps). If possible, we can use metadata to store the real start time (e.g., in MP4 user data or filenames). However, since we have metadata externally, not strictly needed.
* Thermal video can be MP4 with H.264 as well. If colorized, that’s fine for qualitative analysis. If quantitative pixel values are needed, one could record the 16-bit raw frames. One approach is encoding thermal frames as a series of images (PNG supports 16-bit grayscale) – but that would be hundreds of images per second. Alternatively, consider using a format like **MAT** or **HDF5** to store the entire thermal sequence as an array per frame with timestamp. This architecture will initially use MP4 for simplicity, understanding that any precision loss in conversion can be calibrated out. (The references note advanced calibration methods for IR/RGB which assume raw thermal data[[75]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=10,3), but for synchronization architecture, MP4 is acceptable.)
* Sensor data in **CSV** (comma-separated) with header row. Easy for researchers to load into MATLAB, Excel, or analysis scripts. If high-frequency, CSVs can be large, but still manageable. Optionally, we could use an EDF (European Data Format) or MAT file if we had multiple biosignals, but here one CSV per device is fine.
* Calibration in **YAML or JSON** – OpenCV can output camera calibration data as YAML which is human-readable and easily parsed by software for future use[[66]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=visible%20image%29,when%20calibration%20quality%20is%20sufficient).
* **Thumbnails or snapshots:** The PC app might also save a couple of preview snapshots for quick reference (like phone1\_sample.jpg just to quickly see content without opening 4K video).
* If needed, **audio** recordings (not explicitly mentioned, but if we wanted to record subject’s voice reactions, the PC or phone could record audio via a microphone). The PC’s webcam videos could include audio if those webcams have mics, or an external microphone could be recorded on PC with an audio file and synced. This design doesn’t focus on that, but it’s straightforward to add.
* **Data Volume and Management:** Two 4K videos (phones) plus two 4K webcam videos plus sensor logs per session – this is a lot of data. At 30fps 4K H.264, each video might be ~200-400 MB per minute (depending on scene complexity). For a 10-minute session, that’s on the order of 4× (phones+webcams) \* ~2–3 GB = ~8–12 GB, plus small logs. Ensure the PC has ample disk space and the phones have enough internal storage (we might configure the app to store to SD card if available). We should implement checks: before start, query free space on each phone and PC and warn the user if insufficient (e.g., less than X GB)[[110]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Storage%3A%204K%20video%20will%20produce,4%20fragmentation)[[111]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=space%20or%20consider%20writing%20to,4%20fragmentation). The PC could also manage archiving: for example, compress older session folders or move them after completion. Regular backups are advised (the system could integrate with a cloud or external drive backup after each session)[[81]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Management%3A%204K%20video%20files,world%20workflow).
* **Post-Experiment Merging:** Although not part of the real-time architecture, note that after data collection, one might merge the data for analysis. Our architecture’s careful time-stamping allows creation of a unified timeline. For instance, using the metadata, one can take Phone1 video and Phone2 video and play them side by side in sync, with an overlay of GSR signal. Tools like Python’s OpenCV, or custom analysis scripts, will use the timestamps we provided. The Lab Streaming Layer integration (if used live) could even allow recording a combined XDF file with all streams[[112]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L31-L39)[[100]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L205-L213), but since we focus on local recording, we rely on offline merging.

### Calibration Workflow

*(Covered partly in PC architecture above, here we summarize workflow steps with hardware integration.)*

Before any experimental session, **intrinsic and extrinsic calibration** must be performed for each phone’s camera pair (RGB & thermal). The steps are:

1. **Preparation:** Use a suitable calibration target. For the RGB camera, a standard black-and-white checkerboard pattern (e.g., 7×9 grid) printed on paper works. For the thermal camera (which sees heat differences), either **heat the same checkerboard** (so black squares warmer than white, etc.), or use an **infrared calibration kit** (special textured patterns visible in IR)[[72]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20specific%20hardware)[[75]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=10,3). Alternatively, a piece of cardboard with some cut-out shapes that can be heated or a pattern of tape with different thermal emissivities can create a checker pattern visible to thermal. Ensure the pattern is large enough to be clearly seen by the cameras at the working distance.
2. **Initiation:** Launch the PC app and phone apps. Connect all devices. In the PC UI, go to the “Calibration” tab. Place the checkerboard in view of Phone1’s cameras. Ideally, both the RGB and thermal on that phone should see it simultaneously. Press “Capture Calibration Frame” for Phone1. The PC sends the command, Phone1 captures nearly-simultaneous images from its RGB and IR cameras[[63]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Mode%3A%20When%20the%20PC,sure%20to%20chunk%20it%20properly) (if needed, it may alternate quickly if it cannot grab both at the exact same time, but ideally uses two camera streams to snap both). These images are sent to PC (perhaps JPEG or PNG format uncompressed for accuracy). The PC displays them and tries to detect the checkerboard. It might outline the detected corners on the preview for user confirmation.
3. **Capture Multiple Views:** The user moves the checkerboard to different positions (covering different parts of the frame, at various angles). For each, capture a frame (or the system could auto-capture if it detects a stable board). Do this, say, 15 times. Ensure some captures cover edges and corners of the FOV to calibrate lens distortion thoroughly[[67]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20calibration%3A%20Using%20OpenCV%E2%80%99s%20calibration,3). Repeat the process for Phone2’s cameras. (This can be done in an interleaved way or one phone after the other.) If both phones can see the same physical checkerboard at once, you *could* capture simultaneously, but it might be simpler to do sequentially to avoid confusion.
4. **Calibration Computation:** Once sufficient images are collected, the PC computes intrinsics for each camera:
5. For each camera (e.g., Phone1\_RGB), compile object points (the real 3D positions of checker corners, assuming Z=0 plane) and image points (2D pixel coords from each image). Run OpenCV’s calibrateCamera to get camera matrix (focal length fx, fy, principal point cx, cy) and lens distortion coefficients[[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on). Also get reprojection error.
6. Do the same for Phone1\_Thermal.
7. Then run stereoCalibrate with matched image points from RGB and Thermal (for each view where both saw the board) to get rotation R and translation T for Phone1\_Thermal relative to Phone1\_RGB[[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on).
8. Repeat for Phone2 similarly.
9. If needed, one could also calibrate the two *RGB cameras of Phone1 vs Phone2* if those needed to be related for some reason (for instance, if doing 3D reconstruction between phones). That would require them seeing the same scene/board as well – typically not needed unless doing multi-camera 3D, which is beyond scope.
10. **Result Storage:** Save the results to files (as mentioned, YAML/JSON). Also show the user summary: e.g., “Phone1 RGB reprojection error: 0.4 px, Phone1 Thermal error: 0.7 px, Stereo RMS error: 1.2 px”. If errors are too high, advise capturing more images or check if the pattern was detected properly. Perhaps display one of the thermal images undistorted using the calibration to give user confidence.
11. **Usage of Calibration:** These parameters can be used later to **undistort** images, or to overlay thermal over RGB accurately. For example, knowing R, T, one can project the thermal image onto the RGB image coordinates (if their fields overlap). This is outside of recording, but important for data analysis. The phone app itself might use intrinsics to correct lens distortion on the preview or to do an on-phone overlay of thermal and visible images during monitoring[[113]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L134-L143)[[114]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L130-L138). If that’s a feature, the PC should send intrinsics back to phone (the design anticipated storing calibration in the app’s repository)[[33]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L65-L73). Otherwise, the calibration is mostly for offline processing.
12. **Webcam Calibration (optional):** If the precise field of view of the Logitech webcams matters, calibrate them too using a standard checkerboard and OpenCV. Intrinsics for the webcams can be saved. We might not strictly need it unless doing quantitative analysis (e.g., measuring head movement in mm from video), but it could be done for completeness.
13. **Thermal-Visible Alignment Validation:** After calibration, it’s wise to do a quick check: overlay a thermal image on its corresponding RGB (the system can apply the homography from calibration) to see if a known feature aligns. This ensures calibration worked. The calibration tab could have a “Validate” button for this – it might, for instance, use the last captured checkerboard image to draw reprojected corners and show how well they line up.

The calibration step is essential before the first data capture of the day or whenever the relative positioning of thermal and RGB cameras changes (e.g., if the thermal camera was removed/re-attached). We store calibration per session to account for slight shifts each time. If the setup is fixed and reproducible, calibration could be reused across sessions, but doing it each time ensures maximum accuracy.

### Start/Stop Recording Workflow (Putting It Together)

*(This section illustrates how the above components interact during an actual experiment run, ensuring clarity on sequence.)*

* **Pre-experiment:** Operator checks all devices are on, connected, and calibrated. The PC UI shows Phone1 & Phone2 “Connected” and Shimmer status ready. The operator enters participant ID or any meta info, verifies settings (e.g., resolution = 4K, frame rate 30, etc.), and that previews from both phones are visible and well-framed[[115]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Preview%20Monitoring%3A%20Receive%20live%20previews,to%20switch%20between%20IR%2FRGB%20preview). The Shimmer signal quality can be quickly checked (maybe a small live readout or just an indication that data is coming).
* **Begin Recording:** Operator hits the **“Start Recording/Experiment”** button. Behind the scenes, the PC sends out the START\_RECORD command to both phones (with an included start timestamp perhaps). Almost simultaneously, it initializes the recording of the two webcams (starts capture threads/process)[[50]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L34-L42). It also prepares the first stimulus video. Once acknowledgments from phones are received (or after a tiny delay as planned), the PC begins playback of the stimulus video (e.g., on a second screen)[[60]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Start%20Recording%3A%20When%20the%20experimenter,Alternatively%2C%20as%20mentioned). The PC log notes “Recording started at 16:30:05.500, Stimulus1 started at 16:30:06.000”. The UI now indicates status: e.g., a timer starts counting recording duration, a red dot “REC” is shown, and perhaps phone status icons turn red as well to denote active recording.
* **During Recording:** The phones record their videos locally and stream preview frames. The PC displays these previews (if not turned off) so the operator can glance if everything is still in frame[[93]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Preview%20Display%3A%20For%20each%20phone%2C,to%20embed%20a%20video%20player). The PC also could show a small plot of Shimmer data in real-time if that data is being forwarded (for example, heart rate or GSR level trend)[[116]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=During%20Recording%3A%20The%20PC%20app,implementing%20a%20live). This is not strictly necessary but is a nice confidence check that physiological data is coming through (e.g., a flat line would warn the operator of a sensor issue)[[117]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=elapsed%20time%29,feasible%20at%20low%20data%20rates). The operator generally will not intervene unless something goes wrong. The system should require no manual sync actions now – everything is automatically aligned via the initial trigger. If multiple stimuli videos are queued, the PC will either play them back-to-back or wait and then play the next as configured (with possibly a rest interval). Each transition can be logged or a sync marker sent if needed[[58]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Stimulus%20Sequence%20Control%3A%20After%20one,that%20can%20be%20handled%20too). The webcams record continuously through the whole session (they don’t stop between stimuli unless we explicitly wanted to segment videos per stimulus; usually keeping one file is easier). If any phone or device drops, the operator might be alerted but likely will continue the session and address it after.
* **Stop Recording:** After the last stimulus or when the operator clicks “Stop”, the PC sends STOP\_RECORD to both phones[[118]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Stop%20Recording%3A%20At%20the%20end,clearly%20indicate%20recording%20has%20stopped). Each phone stops its recordings (finalizing the MP4 files) and responds with a completion message. The PC also stops the webcam recording (joining threads or stopping the C++ process) resulting in finalized webcam video files. The total recorded time is noted. The UI now turns off the REC indicators. Immediately, the PC can request the phones to upload their files. Alternatively, the phones proactively start sending files once stopped[[79]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=%E2%80%9CStop%E2%80%9D,clearly%20indicate%20recording%20has%20stopped). Depending on file sizes and network speed, this could take a while – if the videos are tens of GBs, it might be faster to plug in via USB and copy, but the architecture can attempt network transfer for convenience[[78]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=overwhelm%20the%20network%29,after%20capture%20would%20be%20useful)[[80]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Post,at). During transfer, the UI shows progress bars for each file. If the user doesn’t want to wait, they can cancel and copy manually later (the data remains on phones regardless). The Shimmer data (if PC-collected) is already on PC; if phone-collected, that CSV can be transferred quickly since it’s small.
* **Post-session Wrap Up:** The PC then finalizes the metadata file for the session, listing all the files and timing info as described. It may prompt the user to enter any notes (e.g., “Participant was agitated during stimulus 2”) that get saved in a notes field. Finally, the operator can start the next session or close the app. If starting a new session, it might disconnect and reconnect to flush any state and open a new folder.

Throughout, **user interface cues** ensure the operator knows what’s happening: e.g., “Phone1: Recording… (10s elapsed)”, “Phone2: Waiting for response…”, “Webcams: recording”, etc., and after stopping, “Transferring file phone1\_rgb.mp4: 45%” etc. This feedback loop helps users trust the system.

### Fault Tolerance and Fallback Logic

We have touched on many fault scenarios, here we summarize specific strategies:

* **Phone Connection Loss:** If a phone’s Wi-Fi drops mid-session, the phone app will continue recording locally uninterrupted[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch). The PC will detect the loss (heartbeat fail) and warn the operator (“Warning: Lost contact with Phone1”). The operator might decide to stop the experiment if critical, or continue knowing data is still being recorded offline. The phone will still stop recording either on its own (if a pre-set duration was given) or the user can manually stop it via the phone UI. When the phone reconnects (maybe Wi-Fi comes back), it can notify the PC that it is still recording or has completed. At that point, the PC could even retrieve the partial data. The architecture should ensure that a **temporary network issue doesn’t ruin data collection**[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch).
* **PC Application Crash:** In case the PC app crashes or the PC itself reboots mid-session, the phones again should keep recording (they don’t depend on continuous commands once started). The webcams, however, would stop because the PC was capturing them – if PC dies, webcam data collection is lost for that session. To mitigate, one could run the webcam capture in a separate process that might continue even if the GUI crashes (though if PC reboots, nothing can help). For critical captures, a backup device (another camera or a hardware recorder) could be used for redundancy, but that’s beyond scope. The focus is to save phone and shimmer data at least. Upon PC restarting, the operator could manually stop the phones via their UI. It’s not ideal but data is not lost. Therefore, **stability testing of the PC app** is important to avoid crashes (use try/except around threads, etc., so that one device error doesn’t crash the whole app). Logging to file each step can help recover what happened in case of failure.
* **Phone App Crash:** If a phone app crashes (or phone battery dies), obviously its recording stops and that data might be incomplete. The PC will detect it (no heartbeats). Not much can be done mid-session except possibly using the remaining devices to continue. The architecture should minimize this risk by handling exceptions in the phone code (e.g., catching camera errors or out-of-memory issues). Ensuring phones have enough storage and are not overheating will prevent many crashes. Also using **Android’s foreground service with restart** flags can attempt to restart a crashed service, but if it crashes during recording, continuity is broken. Thus, testing on the actual Samsung S22 hardware with the full load is crucial to tune performance (perhaps limit preview frame rate to reduce app strain, etc.)[[12]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Thermal%2FPerformance%3A%20Recording%204K%20on%20a,processes%20or%20using%20performance%20modes).
* **Shimmer Failures:** Shimmer sensors can drop connection (if radio interference or out of range). If the Shimmer disconnects, the phone/PC should try to reconnect automatically a few times[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch). If it can’t, it logs the event. Data loss from that period is inevitable, but at least it’s known. If using multiple Shimmers as backup, maybe having two on the participant could be a fallback (but then each would need separate connection and combining their data). Typically, one ensures the Shimmer is secure (fresh battery, within range). In multi-subject setups, each subject might have one Shimmer and one phone to simplify.
* **Storage Issues:** If a phone runs out of disk space during recording, ideally the app should handle it (MediaRecorder usually stops gracefully when out of space). The app should have already warned if low space before start[[110]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Storage%3A%204K%20video%20will%20produce,4%20fragmentation). But if it happens, the phone should stop recording and send an error (“Disk Full – stopped”). The PC then stops that stream (and maybe stops the whole session if needed). If a file exceeds 4GB on certain filesystems, Android’s MediaRecorder might split it (some devices enforce a split at 4GB even on internal storage); our app should account for that by either using a modern format or capturing multiple segments (could happen for very long recordings)[[110]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Storage%3A%204K%20video%20will%20produce,4%20fragmentation)[[111]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=space%20or%20consider%20writing%20to,4%20fragmentation). The PC should then handle multiple segments if present (we can design file naming such that phone will output part1, part2, etc., and list them in metadata).
* **Thermal Camera Issues:** Thermal cameras sometimes need to recalibrate (shutter closes for a moment to recalibrate, causing a frame drop). The app/SDK usually handles this with a callback. The design should simply log if frames were dropped and perhaps mark them so it’s not mistaken for a sync issue. If a thermal camera disconnects (e.g., loose cable), the phone app can attempt to reinitialize it. If it fails, it should notify PC but continue the rest (RGB, etc.).
* **Fallback to Secondary Networks:** We have multiple ways to control devices (Wi-Fi, BT, ADB). The system could theoretically detect “Wi-Fi down, trying Bluetooth...” and send a STOP via Bluetooth. This is complex and may not be needed if we trust Wi-Fi or use USB tether which is robust. In practice, having a simple manual backup (like the phone UI Stop button or an ADB script) is sufficient. The operator is part of the fault tolerance loop.
* **Monitoring and Alerts:** The PC UI should clearly alert if any component is in a bad state (e.g., battery low, or connection lost). For example, if phone battery is <20% and not charging, show a red battery icon so the operator can plug it in before starting[[119]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Status%20Monitoring%3A%20The%20PC%20can,in%20before%20starting%20the%20experiment)[[120]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=app%29,in%20before%20starting%20the%20experiment). If the Shimmer battery level is accessible, show that too. By addressing issues proactively, many faults can be prevented.

In essence, the architecture does not rely on perfect conditions – each device can operate independently (especially the phones recording to themselves) to preserve data integrity[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch). Coordination enhances functionality but is not a single point of failure for critical data capture. Testing various failure scenarios (drop network, kill PC app, etc.) should be part of development to ensure graceful handling.

### Stimulus Presentation and Synchronization

The stimulus presentation subsystem is a key part of the experiment, responsible for delivering the psychological triggers (videos with audio) and marking the timeline for data synchronization. The design integrates this as follows:

* **Stimulus Configuration:** Researchers can prepare a **config file** (JSON, CSV, or custom) listing the sequence of stimuli. For example, a JSON array of objects:
* [  
   {"file": "stimulus1.mp4", "name": "HappyVideo", "duration\_s": 60},  
   {"file": "stimulus2.mp4", "name": "SadVideo", "duration\_s": 60}  
  ]
* The PC app provides a UI to load this list. It might show each item with a label (so the operator knows what’s coming). The config could also specify if there are breaks or questions between videos, etc. In a simpler form, it might just be a text file with file paths in order.
* **Playback Engine:** Using PyQt5’s **QMediaPlayer** or the VLC Python bindings, the PC can play the video files one after another[[56]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=prepare%20a%20playlist%20of%20videos,knows%20when%20each%20video%20starts%2Fends). We choose QMediaPlayer here for integration with the Qt GUI (it can easily render video to a widget or full screen on another monitor). The PC app will create a QMediaPlayer instance and set its output to a full-screen video window. If a second monitor is present (preferred), that window is placed on it (covering it entirely) so the participant only sees the video, not the control UI[[61]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Test%20the%20playback%20for%20potential,in%20Qt%20by%20specifying%20screen). If only one screen is present, the app can temporarily cover the UI with the video (though experimenters usually have at least a laptop screen + external display). The audio is played through speakers for the participant. It’s important to ensure the audio is loud enough and the environment is controlled (and that phone microphones potentially recording audio don’t inadvertently cause feedback – usually they won’t since speakers are away from phones near participant).
* **Sync Hooks:** The PC knows exactly when it starts each video (because it issues the play command). QMediaPlayer has signals like mediaStatusChanged or positionChanged. We can connect to a signal that fires when playback actually starts (or rely on the fact that our Start button triggers playback at a known time)[[60]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Start%20Recording%3A%20When%20the%20experimenter,Alternatively%2C%20as%20mentioned). The PC will log Stimulus "HappyVideo" START at PC\_time 12:00:10.000. If needed, it could immediately send an “EVENT: Stimulus1\_start” message to the phones (the phones could record this in a log or insert as a marker in Shimmer data stream)[[83]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=triggers,by%20the%20shared%20start%20time). This isn’t strictly necessary since we have the PC timeline, but it could be useful for redundancy. Similarly, when the video ends, we log Stimulus "HappyVideo" END at PC\_time 12:01:10.000 and potentially notify devices or mark the data[[58]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Stimulus%20Sequence%20Control%3A%20After%20one,that%20can%20be%20handled%20too). If the experiment is automated, the PC then automatically loads the next video and plays it after a short gap (or immediately). If it’s manual, it might wait for the operator to press “Next”.
* **Ensuring Smooth Playback:** We must ensure that playing the video doesn’t freeze the PC app or cause frame drops. QMediaPlayer runs in its own thread internally and is efficient, but we should test with the video resolutions used (if they are HD or 4K videos). If any performance issues, using VLC’s optimized playback might be better[[56]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=prepare%20a%20playlist%20of%20videos,knows%20when%20each%20video%20starts%2Fends)[[57]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Video%20decoding%20on%20PC%3A%20OpenCV,with%20recording%20isn%E2%80%99t%20critical%2C%20the). The videos should ideally be present on the PC (not streaming online, to avoid any network unpredictability). We also note the potential small delay when starting a video (decoding can have a ~100ms delay). That’s why we either incorporate a “1-2-3-go” approach or a tiny pre-countdown before actual content. Another solution is to **preload** the videos: QMediaPlayer can be buffered with the video paused at the first frame, ready to go. The PC could do that for the first stimulus while waiting for user to hit Start. This reduces the delay on start. For subsequent videos, if automatically chaining, we might load the next file in background shortly before the previous ends (not too early to avoid high memory usage, but a second or two before).
* **Participant Display:** The system should ensure the participant only sees the stimuli and not any technical overlay. So any OS pop-ups, notifications, or mouse cursor should be hidden on that display. The PC app can enforce this by using Qt to hide the cursor on the fullscreen window, and advise the experimenter to use “Do Not Disturb” mode on the PC.
* **Stimulus Markers in Data:** In addition to logging times in metadata, if extremely precise correlation is needed, one could embed an actual marker in the media. For example, include a single black frame or an audio tone at the very start of each stimulus. The phones’ camera might catch a faint flash if the screen goes bright, or a microphone (if used) might catch the tone. But since our system time-sync is already handling alignment, these are secondary measures[[102]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=needed%2C%20consider%20adding%20a%20sync,will%20be%20sufficient%20for%20most). We document them as an option (maybe a footnote in the user manual, e.g., “Video files include a 50ms flash frame at start for alignment purposes”).
* **Inter-stimulus Interactions:** If the experiment calls for e.g. asking the participant something between videos, the operator can pause. The Start/Stop of recording can either span the whole sequence (probably easier, one continuous recording that covers all stimuli and breaks) or stop between each stimulus. We chose continuous to avoid sync complexity from multiple segments. If needed, the PC could still send event markers for “pause” times that can be later cut out in analysis. The UI can have a “Next Stimulus” button if manual progression is desired.
* **Flexibility:** The experiment tab could allow manual override, like skip a video or replay, etc., but usually in a study they follow the planned sequence. If an emergency stop is needed (participant wants to stop), the operator clicks “Stop” which stops everything – recording and playback immediately.
* **Data Alignment with Stimuli:** In the final data, we will have videos from phones and webcams that include the content of the stimuli (e.g., if a camera is pointed at participant’s face, you’ll see their reactions as the video plays in front of them). We also have the exact timestamps of what was shown when. This allows correlating GSR peaks to specific moments in the stimulus. The architecture ensures that these timestamps are accurate by aligning to the same clock.
* **Testing Sync:** One can test the whole chain by filming a test pattern. For example, a test video with known time-coded flashes can be used, and you’d check the phone recordings to see if those flashes appear at the expected times. Given our approach, we expect ~tens of milliseconds accuracy which is generally sufficient for human physiological response studies (where reactions are on the order of seconds). If higher precision is required, hardware triggers as mentioned can tighten sync, but those complicate the setup.

In sum, the PC’s stimulus engine centralizes the experimental stimulus timeline and hooks into the recording control so that all sensor data is tied to the stimulus events. This design fulfills the requirement that stimuli are “precisely synchronized with all sensor streams”[[121]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=window%20that%20the%20participant%20can,is%20playing%2C%20time%20elapsed%2C%20etc)[[122]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=presentation,is%20playing%2C%20time%20elapsed%2C%20etc).

### User Interface Design (PC Controller)

The PC controller’s UI should be user-friendly and tailored for quick operation during research sessions. Here we outline the UI layout and features, incorporating best practices:

* **Main Window Layout:** Use a **tabbed interface** or a multi-section window with clear separation of concerns[[44]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Keep%20the%20UI%20layout%20intuitive%3A,video%20selection%20and%20start%2Fstop%20controls):
* **Devices / Status Tab:** Shows the connected devices and their statuses. For each phone, display an icon or name (“Phone 1” and “Phone 2”). Next to each: a colored status indicator (green for connected idle, red for recording, yellow for error), battery level (maybe an icon with %), storage remaining, and Shimmer status (if the phone is handling Shimmer, show sensor connected and battery; if PC is handling Shimmer, show it globally). There could be a “Connect” or “Refresh” button if manual connection is needed, but phones likely auto-connect. Possibly include a “Configure” or “Settings” sub-section here: where the user can set parameters like resolution, etc., for each device, then hit “Apply” to send configurations[[87]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Configuration%20Interface%3A%20Provide%20controls%20to,This%20could%20include)[[88]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=After%20the%20user%20sets%20these,UI%20with%20the%20confirmed%20settings). If any config is invalid or adjusted by the phone (e.g., you requested 60fps but phone says only 30fps available), display that info to user. This tab also can have the **live preview thumbnails** from each phone (two small video panels) – or those could be in a separate “Monitoring” tab/pop-up. Having them visible helps adjust framing initially[[115]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Preview%20Monitoring%3A%20Receive%20live%20previews,to%20switch%20between%20IR%2FRGB%20preview). If screen space is an issue, maybe one at a time with a toggle to switch between phone1 and phone2 preview. Additionally, show Shimmer signals if desired – perhaps a tiny chart of GSR values updating (if PC is reading them).
* **Calibration Tab:** This provides controls and feedback for camera calibration. It might have a text area with instructions (“Show the checkerboard to Camera, then press Capture. Repeat for different angles.”). There’s a “Capture Frame” button (and maybe a dropdown to choose which phone or both). If we can automate detection, we could also have a “Auto-Capture when detected” checkbox. As images come in, the UI might show a count of captures and an average reprojection error so far (if doing incremental calibration). After clicking “Compute Calibration,” it displays the results (focal lengths, etc., or just “Calibration successful, error X px”). Possibly, a button to “Save & Send to Phones” if we want to update phones with intrinsics[[70]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=applicable%29,if%20doing%20augmented%20feedback%2C%20etc). Since calibration might be done rarely, the UI doesn’t need to be extremely fancy, but clear feedback (like highlighting if chessboard not found in an image) helps. This tab can also allow selecting which pattern size (if not hardcoded).
* **Experiment / Stimulus Tab:** This is where the operator runs the actual experiment. There should be a way to load/select the stimulus configuration. For example, a “Load Playlist” button that opens a file dialog for the JSON/CSV. Once loaded, display the list of stimuli in order (maybe a listbox with each entry “1. HappyVideo.mp4 – 60s”). The operator can optionally reorder or skip if needed (but typically fixed). Key controls here: **Start** and **Stop** big buttons. Perhaps also “Pause” if we allowed pausing the recording/stimulus, but pause is tricky across devices (likely not used; better to just stop fully if needed). If automatic progression is used, the UI can highlight the current video playing. If manual, perhaps a “Next” button appears when one stimulus ends (or the Start button becomes “Next”). There can also be a countdown timer showing how much time is left in the current video (if that info is available) – QMediaPlayer can give duration and current position, so we can display that (“Video 1 – 00:45 / 01:00 remaining”). During recording, also show the **overall recording time** elapsed.
* **Live Data Display:** Optionally integrated in either the Devices tab or Experiment tab, a small panel can show live sensor readings. For GSR, a simple numeric value or a small scrolling chart would do. If the project includes interesting AI analytics (the docs mention emotion or stress AI analysis on the phone)[[123]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L80-L88), that could also be shown, but that’s beyond the core requirements. We focus on raw data monitoring. This helps troubleshoot (e.g., if GSR flatlines, the operator might adjust the sensor on participant).
* **Logging Console:** At the bottom or side of the window, have a multiline text box that logs system messages[[45]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Provide%20logging%20or%20console%20output,research%20setting%20to%20diagnose%20issues). Each line could be timestamped or just sequential: e.g., “12:00:05.120 – Phone1 ACK start”, “12:00:05.130 – Phone2 ACK start”, “12:00:05.600 – Recording begun”, “12:00:06.000 – Playing HappyVideo.mp4”, “12:00:06.500 – Phone1 preview lost (reconnecting…)” etc. Use color coding for severity (red for errors). This console is mainly for the operator/developer to see what’s happening internally, which is valuable for debugging during initial deployment and even during sessions if something subtle is off. The user should not be overwhelmed by it, but it’s there if needed. It can be auto-scrolling, with perhaps an option to save the log to a file each session.
* **Usability Considerations:** Keep controls **intuitive and minimal**. For instance, one Start button triggers everything – the user shouldn’t have to separately start phones and webcams; the system does that in one go. Disable the Start button if prerequisites aren’t met (e.g., a phone is not connected) to prevent mistakes[[124]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Indicate%20clearly%20if%20any%20device,disabled%20or%20warn%20the%20user). Conversely, if only one phone is connected and the experimenter intentionally wants to run with one (maybe the other failed but they proceed), allow override but warn. The Stop button should be prominent and perhaps confirmation if clicked accidentally (maybe not, because if you need to stop quickly, don’t ask twice – this is a design tradeoff).
* **UI State Management:** Use clear signals of state. For example, when recording, maybe the border of the window turns red or a big “RECORDING” label appears, so it’s obvious. When idle, it could be green or neutral. Between sessions, ensure fields reset appropriately (or when starting a new session, auto-generate new folder name). Also, block any config changes while recording to avoid inconsistent states (you don’t want user toggling resolution mid-record).
* **Phone App UI:** While the PC is the primary interface, the phone app UI should also be designed for ease of use during setup. It might just show “Waiting for PC command… (Connected)” and allow a manual start if needed. Possibly it can show the IP it’s trying to connect to, to help debugging connectivity. Once recording via PC, it could display “REC” indicator on phone too, in case someone glances at the device. But the phone will often be mounted out of reach, so it’s secondary. In case the PC is not used (the app should still be usable standalone), the phone UI would then need the ability to start/stop recording, choose modes etc., which it presumably has. But for our integrated operation, we assume minimal local interaction.
* **User Training:** As part of deliverables, a user manual will accompany the system, but the UI itself should be self-explanatory enough that someone with basic training can run it. Including tool-tips (small pop-up text on hover) for buttons can be helpful – e.g., hovering “Start” could show “Begin recording on both phones and start stimulus playback (F5 hotkey)” if we add hotkeys. Hotkeys could indeed be useful so the operator can start/stop without moving the mouse (e.g., spacebar to stop, etc.), but careful to not allow accidental presses.
* **Aesthetics:** Follow a clean design (e.g., Material Design or just simple flat design). Use adequate font sizes for readability in a lab setting. Possibly include the lab or project logo on the UI for professionalism. Colors for status (green, yellow, red) should be used consistently for good/bad states (accessible also in terms of color-blindness, maybe include icons or text). If using PyQt, Qt Designer can be used to craft this UI and then we hook up functionality in code.

Overall, the UI is the control panel for a complex system, so it must present lots of info in an organized way. The above design, with segregated tabs and real-time visual feedback, will make operation manageable and help ensure nothing is overlooked (e.g., noticing a battery low icon in time to charge the phone). It also contributes to **fault tolerance** by enabling the operator to catch and fix issues (like plugging in a charger) before they cause failure.

## Recommended Libraries and SDKs

To implement the system efficiently, we leverage existing libraries and SDKs for each component:

* **Android (Kotlin) Side:**
* **Camera2 API / CameraX:** Use Camera2 low-level API for full control (especially to access RAW and concurrent streams)[[15]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20Capture%20,RAW%20images%20at%20intervals%20during). CameraX could be used for easier preview and lifecycle handling, but ensure it supports concurrent camera usage if needed. Given the complexity (RAW + video), Camera2 is likely the main API[[74]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20Capture%20,does%20support%20it%2C%20you%20can).
* **Topdon Thermal SDK:** The manufacturer’s SDK (InfiRay SDK v1.3.7 as referenced) provides FastIRCamera and callbacks for thermal frames[[6]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L189-L197). Include that in the project (likely as a .aar or .so library). Follow their documentation for initialization and image retrieval. If Topdon SDK also provides radiometric conversion, use it to get temperature values if needed.
* **Shimmer Android API:** Use the official Shimmer library (from ShimmerEngineering on GitHub or their site)[[24]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=%5B1%5D%20GitHub%20). According to documentation, include ShimmerBluetoothLibrary.jar and related jars (as noted in the project setup snippet)[[125]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L282-L291)[[126]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L288-L297). This gives access to Shimmer device scanning, connection, and data streams. The API likely provides an ObjectCluster object with sensor readings; use that in the Shimmer manager to parse GSR, PPG values[[25]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L156-L164)[[127]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L170-L173).
* **OpenCV (Android):** Not strictly needed on the phone, since we’re doing calibration on PC. If we wanted to do any on-phone computer vision (like automatic checkerboard detection to auto-capture), we could embed OpenCV in the app. But that’s optional complexity. We might just send frames to PC for detection.
* **Networking:** On Android, use either Java Sockets or higher-level libraries. For a simple TCP client, the java.net.Socket in a background thread is fine. Alternatively, OkHttp could be used if we wanted a WebSocket client or HTTP calls. If using MQTT, the Eclipse Paho MQTT Android client is available[[128]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Networking%3A%20Python%20%E2%80%93%20socket%20,like%20Eclipse%20Paho%20for%20Android). But the simplest: a thread reading/writing to a Socket with JSON using Kotlin serialization or GSON for parsing. Ensure to handle reconnect logic if socket breaks (maybe use a Foreground Service with onTaskRemoved to restart attempts).
* **JSON parsing:** Kotlin has kotlinx.serialization or one can use org.json. Since messages are not huge, org.json is fine for quick parsing of commands.
* **Other Android libraries:** Jetpack components like LiveData/StateFlow to communicate between the service and UI (so UI can update status indicators reactively when, say, Shimmer connects). Also use **Hilt (Dagger)** for dependency injection to manage singletons like the Repositories elegantly[[34]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L141-L149). Use WorkManager if any background tasks needed (though not obvious need here). Use **AndroidX** libraries for permissions (Activity Result API) to simplify runtime permission handling.
* **PC (Python) Side:**
* **PyQt5 / PySide2:** For the GUI and event loop. PyQt5 is robust and has a designer tool for UI layout[[41]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L19-L27)[[129]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Tech%20Stack%3A%20Use%20a%20GUI,a%20desktop%20GUI%20is%20fine). PySide2 is the open-source equivalent; either is fine. We’ll get widgets for buttons, labels, image displays, etc., and the signals/slots mechanism which is useful for threading (we can emit signals from worker threads to update UI in the main thread safely)[[48]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L94-L101)[[130]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L86-L94).
* **OpenCV (cv2):** Used for decoding images (like preview JPEGs) and especially for camera calibration computations[[65]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20user%20to%20show%20the,the%20calibration%20results%20back%20to)[[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on). OpenCV’s Python bindings will be used for findChessboardCorners, calibrateCamera, etc. Also possibly for reading from RTSP if we went that route for streaming (cv2.VideoCapture can open an RTSP URL). Also can be used to capture from webcams, though for high perf we might go to C++.
* **NumPy:** Indispensable for image and data processing – OpenCV gives images as NumPy arrays. Also useful for any signal processing if needed (e.g., smoothing GSR for live display).
* **PySerial or Bluetooth library:** If the PC were to connect to Shimmer via Bluetooth, we might use PyBluez or even treat Shimmer as a serial port. However, since we offload to phone or use Shimmer’s own software, we might skip directly connecting PC to Shimmer. If we did, Shimmer has a Python API? Not sure. Possibly easier to let phones handle it. So maybe no BT library needed on PC, keeping PC simpler.
* **paho-mqtt (optional):** If we choose MQTT for comm, the paho-mqtt Python library is great[[128]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Networking%3A%20Python%20%E2%80%93%20socket%20,like%20Eclipse%20Paho%20for%20Android). It handles reconnects, etc. But if we do direct sockets, we use Python’s socket library and maybe select or asyncio for async operation. asyncio could elegantly manage multiple client connections in one thread, but mixing asyncio with PyQt (which has its own loop) can be tricky. Many integrate them or just use QTcpSocket from Qt. Another approach: use Qt’s networking classes (QTcpServer) so everything stays in Qt’s event loop. That’s an option – PyQt can indeed use QTcpServer, which on new connection emits a signal etc. That might simplify integration and thread handling. But Python socket on separate threads is also fine.
* **ffmpeg / GStreamer (for webcam capture):** If we implement the webcam capture in Python, using OpenCV’s VideoWriter might suffice but it could be heavy. For better performance, we could use an ffmpeg command launched via subprocess to record each webcam. E.g., use FFmpeg CLI to capture DirectShow device and encode to file (that might actually be simplest: one command per camera). The PC app can spawn those processes at start (like ffmpeg -f dshow -i video="Logitech Brio" -r 30 -vcodec libx264 -preset ultrafast output.mp4). This offloads everything to ffmpeg which is highly optimized. We just have to ensure it starts and stops at the right times (stop by killing the process). This method is quite viable. Alternatively, use GStreamer with Python (like gst-launch from code) – but ffmpeg CLI is straightforward. If coding fully, OpenCV will use ffmpeg internally anyway to encode, but doing it externally may avoid GIL issues. So, **FFmpeg** is recommended for reliability in recording webcam streams.
* **VLC (python-vlc bindings):** If QMediaPlayer fails to provide needed control or format support for stimuli, python-vlc can be used to play videos easily and send events on start/stop[[56]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=prepare%20a%20playlist%20of%20videos,knows%20when%20each%20video%20starts%2Fends)[[57]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Video%20decoding%20on%20PC%3A%20OpenCV,with%20recording%20isn%E2%80%99t%20critical%2C%20the). This would require installing VLC on the system and the pip python-vlc which is lightweight binding.
* **Matplotlib/PyQtGraph:** For plotting live sensor data. **PyQtGraph** is preferred for real-time plotting in PyQt (it’s fast and integrates with Qt’s event loop). We can embed a PyQtGraph plot widget in the UI for the GSR signal if desired[[131]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=OpenCV%20,after%20recording%20as%20a%20visualization)[[132]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=composite%20of%20IR%20over%20RGB,after%20recording%20as%20a%20visualization).
* **Lab Streaming Layer (LSL):** If we want to stream data to other lab software in real-time, we could integrate pylsl. The Android code has an LSLRepository stub[[100]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L205-L213), meaning future integration. If we foresee connecting this system to other LSL consumers (like BioSemi EEG or such), adding LSL outlets on PC for each data stream could be beneficial. But if our focus is local recording, LSL is optional. We mention it as a capability: the codebase planned streams for thermal, GSR, etc. at defined rates[[133]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L221-L228). Implementation would involve using pylsl to create a StreamInfo and StreamOutlet for each and pushing samples. It’s something to consider if live analysis or multi-system integration is needed.
* **Others:** JSON handling via Python’s json module for metadata. Possibly numpy.linalg if doing any manual math (OpenCV covers calibrations though). **Threading** or **asyncio** for managing parallel tasks (depending on design; PyQt’s thread with signals or Python threads with locks).
* **Development Tools:** Use an environment like Anaconda for easy package management, especially OpenCV and PyQt. Version control via Git (ensuring we manage large binary files carefully). Test on Windows since that’s target (some library behaviors differ on Windows, e.g., ffmpeg device names).
* **General Recommendations:** Keep all software updated to latest stable versions (Android API level 33+ for modern camera features, Python 3.10+ for best library support). Use 64-bit build on Android for performance. Employ logging (Android Logcat and Python logging) generously for debugging. Also, unit test components where possible (e.g., simulate receiving commands in the phone service to test state changes, or test the PC’s calibration routine with known inputs).

By using these well-supported libraries and SDKs, we minimize the need to “reinvent the wheel” for low-level tasks, and we ensure the system is built on reliable foundations. For example, using the official Shimmer SDK avoids dealing with raw Bluetooth protocols for sensor data[[25]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L156-L164), and using OpenCV for calibration leverages proven algorithms[[67]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20calibration%3A%20Using%20OpenCV%E2%80%99s%20calibration,3). This allows the development effort to focus on integrating these components into a cohesive system.

## Conclusion and Next Steps

The proposed architecture provides a **comprehensive, implementation-ready design** for the synchronized multi-sensor recording platform. It addresses hardware setup, software components on each platform, data flow, synchronization, and user interaction, fulfilling all the requirements:

* Two Android phones capturing **4K video and thermal data in sync**, with support for RAW frames[[134]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=RGB%20Camera%20%28Main%29%3A%20Capture%20high,for%20highest%20fidelity)[[20]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=enormous%20data%20,that%20IR%20and%20RGB%20can).
* Integration of **Shimmer3 GSR+** sensor data via Bluetooth, ensuring physiological signals are recorded and time-aligned[[7]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L90-L98)[[83]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=triggers,by%20the%20shared%20start%20time).
* A Python PC application that centrally controls start/stop, runs the **stimulus presentation** with precise timing, and coordinates calibration and data management[[2]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Experiment%20Control%20%2F%20Stimulus%20Presentation%3A,ends%20or%20the%20operator%20clicks)[[60]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Start%20Recording%3A%20When%20the%20experimenter,Alternatively%2C%20as%20mentioned).
* Communication protocols over Wi-Fi (with optional USB tethering) that are robust to delays and dropouts, including acknowledgments and health monitoring[[135]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Network%20Sockets%20or%20HTTP%3A%20For,frames%20as%20they%20become%20available)[[91]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Streams%3A%20For%20the%20preview,frame%20rate%20preview%20just%20for).
* A clearly defined **file management and session structure**, so that multi-modal data can be easily merged after recording.
* A calibration procedure using OpenCV to achieve accurate intrinsics/extrinsics for RGB-thermal alignment[[136]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=high,to%20compute%20the%20intrinsic)[[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on).
* Use of appropriate **SDKs and libraries** at every layer, from Android Camera2 and Shimmer API to PyQt, OpenCV, and ffmpeg on PC[[128]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Networking%3A%20Python%20%E2%80%93%20socket%20,like%20Eclipse%20Paho%20for%20Android)[[137]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Video%20decoding%20on%20PC%3A%20OpenCV,integrating%20it%20gives%20more%20control).
* Strategies for **synchronization** (timing offsets, scheduled triggers) that ensure all streams (video, thermal, GSR, audio if any) can be correlated within a small error margin[[89]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=an%20NTP%20server%20or%20use,Shimmer%E2%80%99s%20documentation%2C%20their%20software%20supports)[[90]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=start%20timestamp%20a%20couple%20of,a%20marker%20in%20the%20data).
* **Fault tolerance and fallback** considerations that allow the system to handle real-world issues (network drop, device crash) without losing critical data[[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch)[[102]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=needed%2C%20consider%20adding%20a%20sync,will%20be%20sufficient%20for%20most).
* A user-friendly **GUI design** enabling researchers to operate the system confidently and see real-time feedback from all sensors[[44]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Keep%20the%20UI%20layout%20intuitive%3A,video%20selection%20and%20start%2Fstop%20controls)[[45]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Provide%20logging%20or%20console%20output,research%20setting%20to%20diagnose%20issues).

With this architecture in hand, the development team can proceed to implementation. The immediate next steps would include: 1. **Prototype Communication**: Implement the basic PC-phone socket connection and send a dummy command to confirm connectivity (maybe start with simple Wi-Fi commands and an ADB backup). 2. **Camera Testing on Phone**: Write a small app to record 4K video and capture a RAW image to ensure the Samsung S22 supports the dual output as expected, adjusting the Camera2 configurations accordingly[[138]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=processing,5%5D.%29%20Use%20cases). 3. **Thermal SDK Integration**: Integrate the Topdon SDK in a test app to fetch an IR frame and display it, confirming the SDK works on S22 (and measuring frame rate). 4. **Shimmer Integration**: Using the ShimmerAndroidAPI, connect to the sensor and log data for a minute to verify throughput and latency. 5. **Webcam Capture Dry-run**: On the PC, use ffmpeg or OpenCV to record a short 4K clip from one Brio to gauge performance and CPU usage. 6. **GUI Construction**: Build the PyQt UI layout (without full functionality) to validate the design ergonomics on screen. 7. **End-to-End Small Test**: With one phone and one webcam, attempt a simplified end-to-end run (start command, record 10s, stop, file transfer) to iron out the control flow. 8. **Calibration Validation**: Simulate or actually perform a calibration using a printed pattern to ensure OpenCV code yields reasonable results.

Finally, thorough testing with all components simultaneously is crucial. The entire system should be tested in a rehearsal as suggested[[139]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Once%20Setup%20is%20Done%3A%20After,It%E2%80%99s%20much): run a mock session, then inspect all recorded files for sync and quality. This architecture was informed by both the requirements and best practices from related projects and documentation[[140]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L48-L56)[[141]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L50-L58). By following it closely, the development team will create a powerful and reliable multi-modal recording platform that significantly advances the research capabilities for which it is intended. All critical design choices are grounded in known frameworks or prior implementations, and we have cited references (Android docs, Shimmer guides, etc.) to support our decisions and ensure that the approach is validated by existing knowledge. The system is now ready to be built, tested, and iterated upon to achieve the final product.

[[1]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Smartphones%20,Phone%20App%20Capabilities) [[2]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Experiment%20Control%20%2F%20Stimulus%20Presentation%3A,ends%20or%20the%20operator%20clicks) [[3]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=IR%20Camera%3A%20Capture%20infrared%20video,OTG%2C%20depending%20on%20the%20hardware) [[4]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=USB%20Direct%20Connection%3A%20If%20mobility,mobility%20and%20adds%20cable%20constraints) [[5]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Additional%20Suggestions%20and%20Considerations) [[8]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Shimmer%20Integration%3A%20Utilize%20the%20Shimmer,time%20%28if%20live) [[9]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20app,with%20too%20many%20small%20packets) [[10]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=bindings%2C%20or%20opencv%20VideoCapture%20%2B,knows%20when%20each%20video%20starts%2Fends) [[11]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=ideally%20not%20show%20any%20of,in%20Qt%20by%20specifying%20screen) [[12]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Thermal%2FPerformance%3A%20Recording%204K%20on%20a,processes%20or%20using%20performance%20modes) [[15]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20Capture%20,RAW%20images%20at%20intervals%20during) [[16]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=bandwidth%20%E2%80%93%20phones%20with%20Camera2,tested%20on%20the%20specific%20hardware) [[17]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=processing,Every%20Camera2%20device%20can%20support) [[18]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20RGB%20camera%E2%80%99s%20output%20will,RAW%20still%20frames%20for%20later) [[19]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=video%20as%20an%20MP4%20%28e,feed%20as%20a%20secondary%20video) [[20]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=enormous%20data%20,that%20IR%20and%20RGB%20can) [[24]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=%5B1%5D%20GitHub%20) [[29]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Local%20UI%20%26%20Previews%3A%20The,here%20to%20capture%20checkerboard%20images) [[30]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=positioning%20the%20cameras%20,PC%20will%20be%20handling%20monitoring) [[31]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Error%20Handling%20%26%20Resilience%3A%20The,made%20to%20reconnect%20and%20fetch) [[44]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Keep%20the%20UI%20layout%20intuitive%3A,video%20selection%20and%20start%2Fstop%20controls) [[45]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Provide%20logging%20or%20console%20output,research%20setting%20to%20diagnose%20issues) [[46]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Device%20Connection%20Management%3A%20The%20PC,the%20PC%20can%20handle%20two) [[47]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=listens%20for%20incoming%20connections%20from,two%20threads%20with%20blocking%20sockets) [[55]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Loading%20Stimuli%3A%20Based%20on%20a,second%20monitor%20dedicated%20to%20the) [[56]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=prepare%20a%20playlist%20of%20videos,knows%20when%20each%20video%20starts%2Fends) [[57]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Video%20decoding%20on%20PC%3A%20OpenCV,with%20recording%20isn%E2%80%99t%20critical%2C%20the) [[58]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Stimulus%20Sequence%20Control%3A%20After%20one,that%20can%20be%20handled%20too) [[59]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=camera%20start%20delay%29,whole%20video%20from%20the%20beginning) [[60]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Start%20Recording%3A%20When%20the%20experimenter,Alternatively%2C%20as%20mentioned) [[61]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Test%20the%20playback%20for%20potential,in%20Qt%20by%20specifying%20screen) [[62]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Workflow%20in%20PC%20App%3A,if) [[63]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Calibration%20Mode%3A%20When%20the%20PC,sure%20to%20chunk%20it%20properly) [[64]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=captured%20images%20,we%20can%20map%20IR%20image) [[65]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20user%20to%20show%20the,the%20calibration%20results%20back%20to) [[66]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=visible%20image%29,when%20calibration%20quality%20is%20sufficient) [[67]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20calibration%3A%20Using%20OpenCV%E2%80%99s%20calibration,3) [[68]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=receives%20them%3B%20runs%20corner%20detection,Still%2C%20saving%20intrinsics%20on) [[69]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=camera%E2%80%99s%20set%20and%20cv,if%20doing%20augmented%20feedback%2C%20etc) [[70]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=applicable%29,if%20doing%20augmented%20feedback%2C%20etc) [[71]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=for%20later%20use%20in%20analysis,when%20calibration%20quality%20is%20sufficient) [[72]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20specific%20hardware) [[73]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Use%20an%20external%20IR%20camera,to%20integrate%20such%20cameras) [[74]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Camera%20Capture%20,does%20support%20it%2C%20you%20can) [[75]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=10,3) [[76]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Synchronization%20and%20Merging%3A%20In,This%20helps%20in%20later%20analysis) [[77]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=overwhelm%20the%20network%29,after%20capture%20would%20be%20useful) [[78]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=overwhelm%20the%20network%29,after%20capture%20would%20be%20useful) [[79]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=%E2%80%9CStop%E2%80%9D,clearly%20indicate%20recording%20has%20stopped) [[80]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Post,at) [[81]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Management%3A%204K%20video%20files,world%20workflow) [[82]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=the%20impact%20of%20network%20latency,data%20by%20the%20shared%20start) [[83]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=triggers,by%20the%20shared%20start%20time) [[84]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Security%3A%20If%20this%20is%20used,case%20the%20network%20setup%20changes) [[85]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Network%20Sockets%20or%20HTTP%3A%20For,frames%20as%20they%20become%20available) [[86]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=MQTT%20or%20Pub%2FSub%3A%20Alternatively%2C%20use,the%20broker) [[87]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Configuration%20Interface%3A%20Provide%20controls%20to,This%20could%20include) [[88]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=After%20the%20user%20sets%20these,UI%20with%20the%20confirmed%20settings) [[89]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=an%20NTP%20server%20or%20use,Shimmer%E2%80%99s%20documentation%2C%20their%20software%20supports) [[90]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=start%20timestamp%20a%20couple%20of,a%20marker%20in%20the%20data) [[91]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Data%20Streams%3A%20For%20the%20preview,frame%20rate%20preview%20just%20for) [[92]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=sending%20full%204K%20raw%20frames,the%20operator%20sees%20a%20near) [[93]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Preview%20Display%3A%20For%20each%20phone%2C,to%20embed%20a%20video%20player) [[94]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=streams%20in%20one%20app%20might,to%20save) [[95]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=can%20send%20compressed%20low,running%20the%20preview%20stream%20in) [[96]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=streams%20in%20one%20app%20might,during%20actual%20recording%20if%20needed) [[102]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=needed%2C%20consider%20adding%20a%20sync,will%20be%20sufficient%20for%20most) [[103]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Synchronization%20Between%20Two%20Phones%3A%20Starting,it%20in%20software%20if%20needed) [[107]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Timing%20%26%20Sync%3A%20It%E2%80%99s%20crucial,some) [[108]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=stereo%20rotation%2Ftranslation,when%20calibration%20quality%20is%20sufficient) [[109]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=analysis%20script,This%20helps%20in%20later%20analysis) [[110]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Storage%3A%204K%20video%20will%20produce,4%20fragmentation) [[111]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=space%20or%20consider%20writing%20to,4%20fragmentation) [[115]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Preview%20Monitoring%3A%20Receive%20live%20previews,to%20switch%20between%20IR%2FRGB%20preview) [[116]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=During%20Recording%3A%20The%20PC%20app,implementing%20a%20live) [[117]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=elapsed%20time%29,feasible%20at%20low%20data%20rates) [[118]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Stop%20Recording%3A%20At%20the%20end,clearly%20indicate%20recording%20has%20stopped) [[119]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Status%20Monitoring%3A%20The%20PC%20can,in%20before%20starting%20the%20experiment) [[120]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=app%29,in%20before%20starting%20the%20experiment) [[121]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=window%20that%20the%20participant%20can,is%20playing%2C%20time%20elapsed%2C%20etc) [[122]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=presentation,is%20playing%2C%20time%20elapsed%2C%20etc) [[124]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Indicate%20clearly%20if%20any%20device,disabled%20or%20warn%20the%20user) [[128]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Networking%3A%20Python%20%E2%80%93%20socket%20,like%20Eclipse%20Paho%20for%20Android) [[129]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Tech%20Stack%3A%20Use%20a%20GUI,a%20desktop%20GUI%20is%20fine) [[131]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=OpenCV%20,after%20recording%20as%20a%20visualization) [[132]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=composite%20of%20IR%20over%20RGB,after%20recording%20as%20a%20visualization) [[134]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=RGB%20Camera%20%28Main%29%3A%20Capture%20high,for%20highest%20fidelity) [[135]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Network%20Sockets%20or%20HTTP%3A%20For,frames%20as%20they%20become%20available) [[136]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=high,to%20compute%20the%20intrinsic) [[137]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Video%20decoding%20on%20PC%3A%20OpenCV,integrating%20it%20gives%20more%20control) [[138]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=processing,5%5D.%29%20Use%20cases) [[139]](file://file-9JgS9hNU2GwaXbC4UsQQGa#:~:text=Once%20Setup%20is%20Done%3A%20After,It%E2%80%99s%20much) Updated\_Plan\_for\_Multi\_Sensor\_Recording\_System\_Android\_+\_PC.docx

<file://file-9JgS9hNU2GwaXbC4UsQQGa>

[[6]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L189-L197) [[22]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L199-L204) [[23]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L22-L28) [[25]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L156-L164) [[27]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L11-L19) [[28]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L14-L18) [[32]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L29-L37) [[33]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L65-L73) [[34]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L141-L149) [[35]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L283-L287) [[100]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L205-L213) [[101]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L215-L219) [[112]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L31-L39) [[123]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L80-L88) [[127]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L170-L173) [[133]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md#L221-L228) REPOSITORY\_IMPLEMENTATION\_SUMMARY.md

<https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/REPOSITORY_IMPLEMENTATION_SUMMARY.md>

[[7]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L90-L98) [[13]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L219-L223) [[14]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L220-L223) [[21]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L74-L81) [[26]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L94-L98) [[36]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L316-L325) [[37]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L257-L265) [[97]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L149-L157) [[98]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L150-L158) [[99]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L160-L168) [[113]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L134-L143) [[114]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L130-L138) [[125]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L282-L291) [[126]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L288-L297) [[140]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L48-L56) [[141]](https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md#L50-L58) MASTER\_PROJECT\_DOCUMENTATION.md

<https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/docs/MASTER_PROJECT_DOCUMENTATION.md>

[[38]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L14-L22) [[39]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L18-L26) [[40]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L22-L30) [[41]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L19-L27) [[42]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L78-L86) [[43]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L80-L88) [[48]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L94-L101) [[49]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L20-L28) [[50]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L34-L42) [[51]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L54-L61) [[52]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L58-L66) [[53]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L88-L96) [[54]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L98-L102) [[130]](https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex#L86-L94) pc\_developer\_readme.tex

<https://github.com/buccancs/fyp-gsr-android/blob/c34c2cbda374e614ad8c317ad82a991b851badb9/docs/pc_developer_readme.tex>

[[104]](https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/#:~:text=,Camera%20based%20motion%20capture%20systems) [[105]](https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/#:~:text=The%20application%20will%20allow%E2%80%A6) [[106]](https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/#:~:text=accelerometer%20range%20,settings%20into%20the%20application%20when) Shimmer launches Multi Shimmer Sync for Android - Shimmer Wearable Sensor Technology

<https://www.shimmersensing.com/shimmer-launches-multi-shimmer-sync-for-android/>