ChatGPT

Revised Requirements for Multi-Device GSR and Dual-Video Recording System

System Architecture and Multi-Device Control

- **Central PC Controller:** A desktop application will serve as the central control hub. The PC can connect to and manage multiple Android smartphones concurrently. This allows one PC to coordinate recording across several phones in sync.
- **Remote Recording Control:** Users can initiate or stop recordings and adjust settings on any connected Android device remotely from the PC interface. Recording and sensor settings (e.g. which streams to capture, frame rates) on the phones are configurable via the PC. Each phone should acknowledge and execute commands from the PC (e.g. start/stop recording) with minimal latency.
- **Standalone Phone Operation:** Each Android phone is also capable of independent operation. In absence of a PC connection, the phone app can start/stop recordings and manage sensors on its own (local UI controls). This ensures data can still be collected if the PC is unavailable. When a PC is later connected, it can sync up and retrieve status or data from the phone.
- **Multiple Device Support:** The system must support **multiple smartphones simultaneously** connecting to the PC. The PC interface should handle multiple video streams and sensor streams in parallel, distinguishing each device (e.g., by device name or ID). This enables synchronized multi-angle recordings (using multiple phones) under one session.
- **PC Camera Integration:** The PC itself can act as an additional sensor node. It will support recording from a USB webcam (e.g. a Logitech Brio or any UVC-compliant camera) connected to the PC. The PC's camera feed is treated as another video stream in the system, to be time-synchronized with the phone data. The PC application should be able to record this webcam video locally and/or stream its frames into the synchronization framework so it aligns with the phones' data timelines.

Camera and Sensor Data Acquisition on Phones

- **Dual Camera Streaming on Phone:** Each Android phone will capture **two video streams** simultaneously:
- **RGB Video** from the phone's built-in camera (using the full capabilities of the Camera2 API, including focus, exposure control, etc.).
- **Thermal Video** from a USB thermal camera (e.g., Topdon TC001/InfiRay P2 Pro) attached via USB-OTG [1] [2].
  Both streams run in parallel and are **precisely timestamp-synchronized** on the phone [3], ensuring that each thermal frame can be correlated to the corresponding RGB frame. The phone app must manage the dual-camera capture pipeline efficiently to maintain sync.
- **Shimmer GSR Sensor Integration:** The system **must integrate the Shimmer3 GSR+ sensor** (or a similar GSR measurement device) as a core component [4]. Each phone can connect to a Shimmer sensor via Bluetooth to collect Galvanic Skin Response data in real time. The GSR data should have high sampling rates (configurable 128 Hz up to 1024 Hz) with low latency (on the order of 20ms) [4] to capture rapid skin conductance changes. The app should support Shimmer's multi-range settings (covering 10 kΩ to 4.7 MΩ resistance ranges with auto-ranging [5]) to accommodate different skin conductance levels.

- **Flexible Sensor Attachment:** The user may choose where to pair the Shimmer sensor. By default, each phone will handle its own GSR sensor over Bluetooth. Optionally, the system should allow a Shimmer sensor to pair directly with the PC (if the PC has a Bluetooth interface) and integrate that data into the overall stream. This flexibility means the GSR data can either be streamed via the phone or via the PC, depending on the experimental setup. In all cases, the Shimmer is considered a **required sensor** in the system, and its data must be available in sync with video.
- **Raw Data Capture from Camera Sensor:** The phone's camera subsystem should support an **optional raw image capture mode**. Users can toggle whether to save raw sensor data (e.g., Bayer RAW images or DNG files) from the phone's camera. When enabled, the app will capture images **before ISP processing** (unmodified by image signal processing) to preserve maximum data fidelity. This is crucial for research scenarios that require analyzing the original sensor data (for example, advanced computer vision or psychophysiological analysis). If raw capture is disabled, the system will record the standard compressed video (e.g., YUV/JPEG or MP4) as usual. The requirement is to make raw capture available as a user-selectable feature on each phone.
- **Selective Recording Toggles:** The phone application must provide controls to toggle each data modality on/off. For instance, a user may choose to record RGB video + GSR but not thermal, or only thermal + GSR, etc., depending on their needs. Similarly, if raw image capture is an option, it can be turned on or off. These settings can be configured per device from the PC or on the phone. This ensures efficient use of storage and bandwidth by only recording necessary streams for a given session.

Data Streaming and Real-Time Preview

- **Live Data Streaming to PC:** During an active recording session, each phone will **stream data to the PC in real time** for monitoring. All data streams – including RGB video frames, thermal images, and GSR samples – are transmitted over a network connection (e.g., Wi-Fi). The streaming architecture will use the **Lab Streaming Layer (LSL)** framework for unified handling of multiple streams and time synchronization [6]. LSL uses standard network protocols to send/receive data and automatically aligns time stamps between devices [7], making it well-suited for multi-device setups.
- **Two-Tier Streaming (Preview vs Full Quality):** The system will adopt a two-tier approach for streaming:
- The **PC receives a live preview** of each video stream at a limited frame rate or resolution (configurable to manage bandwidth). For example, the phone might send a 5-10 FPS, lower-resolution preview stream to the PC just for monitoring purposes.
- **Full-quality data is not sent live** to avoid overloading the network. The phones handle full-resolution recording locally (see Data Storage section) while the PC shows a downsampled preview. This ensures real-time oversight without risking data loss or quality reduction in the saved recordings. For sensor channels like GSR, the full data (at the native sampling rate) can be streamed since it's low-bandwidth. The PC can plot or display these sensor values in real time.
- **Time Stamp Synchronization:** Each phone must time-stamp its outgoing data at the source to ensure cross-device alignment. We will use a reliable clock sync method – for example, embedding timestamps based on an NTP-synchronized clock or using RTCP sender reports for the video streams – so that all devices share a common time reference. In practice, LSL will provide an internal time synchronization across the network: all LSL streams have **initial synchronized clocks and known drift** between devices [7]. This means when a phone publishes a stream, it uses a timestamp aligned to the LSL global clock (which the PC and other devices also reference).
- **On-Phone Timestamping:** The phone app should timestamp each video frame and sensor sample **on the phone** as data is captured (to avoid any network-induced jitter). For example, if

using RTP/RTCP for video preview, the phone would use RTCP to periodically inform the PC of the mapping between the video frame timestamps and absolute time (wall clock). These timestamps ensure that even if network latency varies, the data can be realigned later by their capture time.

- **Continuous Clock Alignment:** The system should periodically check and correct clock drift between devices. LSL does this under the hood by sending time sync packets and computing offset/drift [8] [9], but if we use any custom streaming (like RTP), we must similarly ensure the phone's clock remains in sync with the PC's clock throughout the session. This may involve having all devices sync to a common NTP server or the PC acting as a time server. The requirement is that any drift (clock differences) over time should be minimized or compensated so that long recordings remain synchronized to at least within a few milliseconds accuracy.
- **Low Latency Monitoring:** The streaming protocol and network setup should be optimized for low latency. Using Wi-Fi (or a dedicated local network) will provide the needed bandwidth for multiple video streams. The preview streams should have as little delay as possible (target a few hundred milliseconds or less) so that the operator at the PC can see what's happening on each camera virtually in real time. If using LSL, it inherently supports real-time streaming; if needed, specialized streaming for video (e.g., RTSP or WebRTC) could be employed to achieve smooth previews. The overall system should guarantee that the time from capture on phone to display on PC is within acceptable real-time limits for monitoring.

Data Recording and Storage

- **On-Device Data Storage: All high-resolution data is recorded locally on each phone.** The phone app will save the RGB video, thermal video, and GSR readings to the phone's storage during recording. Storing locally avoids network bottlenecks and ensures no data loss if connectivity is interrupted. Each data type should be saved in an appropriate format: e.g., MP4 or a series of image files for video, CSV or binary log for GSR, and possibly a specialized format for raw images (such as DNG or a proprietary raw format).
- **Timestamped Data Files:** The recorded files on the phone must include timing information. For video, this could mean ensuring the file container has proper frame timestamps or an associated log of frame times. For sensor data, each sample in the log should have a timestamp. The timestamps should correspond to the synchronized clock used across devices (as noted above). This way, after the experiment, data from different phones can be aligned by time for analysis.
- **Offline Data Transfer:** After a recording session, the system will support **offline transfer** of the recorded data from each phone to the PC. This can be done via USB cable, Wi-Fi file transfer, or any secure method, depending on convenience. The requirement is that researchers can easily aggregate all the per-phone data onto the PC for post-hoc analysis. The transfer process should preserve the directory structure and filenames to keep data from different devices organized (e.g., each phone's data in separate folder, labeled by device or participant).
- **Data Format and Export:** To facilitate analysis, the system should offer data export in standard formats. For instance, it could integrate with the LSL **LabRecorder** utility on the PC to save all incoming streams into a single **XDF file** (Extensible Data Format) [10]. An XDF can contain multiple streams (video, GSR, etc.) with synchronized timestamps, which is ideal for replay and analysis in tools like MATLAB or Python. If using XDF, the PC would record the streams in real time. Alternatively, the phone can save data in common formats (MP4, CSV) which the user can later import into analytic software. Ensuring **consistency of format** (e.g., all videos encoded similarly, all sensor logs with a common timestamp format) is important so that merging data from multiple phones is straightforward.
- **Storage Management:** Given potentially large data (especially if raw video is recorded), the app should have options for storage management. For example, allowing the user to choose the save location (internal storage vs. SD card) [11], and perhaps warn if storage is low. It might also allow setting a maximum recording duration or size per file to prevent files from becoming too

large. These settings ensure the system can handle long recordings without running out of space or hitting file size limits.

Synchronization and Timing Details

(This section highlights the synchronization mechanisms which are critical in a research-grade multi-sensor system.)

- **Initial Clock Sync:** When the PC and phones connect (session start), the system will perform an initial clock synchronization. If using LSL, this is automatic – LSL finds clock offset between each phone and the PC [7]. Otherwise, the PC could send a sync command or ping to all phones to estimate round-trip time and offset. The requirement is to establish a baseline offset for each device's clock relative to the PC (or a designated master clock) before or at the moment recording begins.
- **Ongoing Drift Correction:** Clocks on different devices can drift apart over time [8], so the system must correct for this continuously or periodically. With LSL, each data sample's timestamp is converted to a common reference time, and drift is measured; our system should leverage that so all streams remain aligned. If we implement our own protocol, we'd send periodic sync messages (e.g., every few seconds) using RTCP or NTP to measure drift and adjust timestamps. The goal is to maintain synchronization accuracy throughout the recording without needing a wired trigger signal (we avoid hardware sync by using network sync methods as per best practices [12]).
- **Network and Protocol Considerations:** The solution will use a combination of wireless protocols: **Bluetooth** is used locally on each phone to gather sensor data (e.g., Shimmer via BLE/SPP), and **Wi-Fi (or Ethernet)** is used for device-to-device communication (phone-to-PC streaming). The LSL framework rides on top of TCP/UDP networking, so as long as the phones and PC are on the same network (or hotspot), they can discover each other's streams. We must ensure that the network setup allows multicast or the discovery protocol LSL uses. If firewall or network issues arise, the system should document how to set up a direct connection (for instance, having the PC run a Wi-Fi hotspot to which phones connect). The requirement is robust connectivity: data should stream reliably over the chosen channel with minimal packet loss. In case of temporary disconnections (e.g., phone goes out of Wi-Fi range briefly), the phone should continue recording locally (so no data is lost) and attempt to reconnect to the PC. The PC should alert the user of any lost connections.
- **Use of RTCP (if applicable):** The system design mentions RTCP for timestamping; if video preview streams use RTP/RTSP, each phone will send RTCP sender reports containing an NTP timestamp and the corresponding RTP timestamp of the video frames. This allows the PC to map video frame times to a global clock. The requirement here is that any real-time media streaming protocol used must include such time synchronization so that the preview frames can be lined up with sensor data. (If we rely solely on LSL, this may be handled inherently, but we note this to emphasize time-consistency across modalities).
- **Calibration and Alignment:** If multiple sensing devices need calibration (for example, aligning the thermal camera's field of view with the RGB camera, or calibrating timing), the system should provide a method to do so. For camera alignment, a one-time calibration (like recording a checkerboard or known reference with both cameras) might be performed and the transformation stored. While this is more of a setup procedure than a requirement, it affects data quality. The requirement is that the software should not assume perfect alignment without calibration; any necessary calibration procedures should be documented or integrable. For timing alignment, since we assume network sync, no manual user action is needed, but the system could offer a way to verify sync (e.g., flashing an LED in view of all cameras to check if frames coincide, if the researchers desire).

User Interface and Configuration

- **PC Application UI:** The PC controller software will provide a clear dashboard to manage devices and view data. Key UI elements include:

- A device list showing all connected Android phones (with indicators for their status: online/offline, recording or idle, battery level perhaps, etc.).
- Controls for each device (and a master control for all) to start/stop recording. The UI should reflect the recording state in real time (e.g., a red dot or timer for each phone when recording is active).
- Settings panels to adjust configuration on each phone remotely. For example, toggling the phone's raw capture, selecting which streams to record, setting the preview frame rate, or adjusting camera settings like resolution, focus mode, etc., if needed.
- Live preview windows for each video stream. The PC should be capable of displaying at least thumbnail video feeds from each phone's RGB and thermal cameras. The user can arrange or resize these previews, and possibly select one to view in larger size. The frame rate of these previews will correspond to the limited stream sent by the phone (as discussed, possibly a user-set value like 5 FPS to conserve bandwidth).
- Real-time data displays for sensor readings. For GSR, for example, the PC UI might show a live plot of the GSR signal from each connected Shimmer, updating as data comes in. It could also display numeric values (like current skin conductance in microsiemens) and basic stats.
- Status and notifications area: The UI should inform the user of important events (e.g., "Phone 3 lost connection, attempting to reconnect…", "Low storage on Phone 2", "Shimmer battery low", etc.). This helps the operator react to issues during a recording session.
- **Phone Application UI:** On each Android device, the app will also have a local user interface (for when it's used standalone or for initial setup):
- The phone UI should allow the user to connect to the thermal camera and Shimmer sensor, showing their connection status.
- It provides buttons to start/stop recording locally, and indicators that it's under remote control when connected to a PC (to avoid confusion, e.g., "Remote Controlled by PC" message when applicable).
- Configuration toggles for which streams to record (RGB, Thermal, GSR, Raw) should be accessible on the phone as well, with the ability to override or update from the PC. If the PC changes a setting, the phone UI should reflect that change in near real time (e.g., if PC turns on raw capture, the phone's raw toggle switches on).
- Basic preview on phone: The phone could display its own camera previews when recording (especially if used standalone). However, to save processing, if the PC is controlling and already showing previews, the phone might not need to render the preview on-screen (to reduce load). Possibly provide an option to black out the phone screen during remote control (to save battery), while still showing status info.
- **Automation and Scripting:** Building on the original system's remote control features [13] [14], the new PC software could expose an API or scripting interface (e.g. Python bindings) to allow automated control. For example, a researcher could write a script to start recordings on all devices at a scheduled time, or integrate this system into a larger experiment control framework. While not mandatory for initial release, designing with an API in mind (perhaps using the existing Bluetooth commands or a new network protocol) will add flexibility. Essentially, anything the user can do via the UI should eventually be doable via a programmatic interface for automation.
- **User Settings Persistence:** The system should remember configuration settings between sessions. For instance, if a particular phone had raw capture enabled and a certain save path selected, it should default to those settings next time. The PC app similarly should recall which devices were paired, and any user preferences in the UI layout. This saves setup time in recurring experiments.

Additional Requirements and Suggestions

- **Background Operation:** The Android app must support running as a background service for long recordings [15]. If the user switches apps or turns the screen off to save battery, the data recording should continue uninterrupted. The app should acquire the necessary wake locks or foreground service notifications to avoid being killed by the system. Similarly, the PC app should handle losing focus or running in background (it should continue to receive streams and record if needed even if the user tabs away, until explicitly stopped).
- **Data Quality and Reliability:** The system should ensure that data quality is maintained: for example, no frame drops beyond what is unavoidable, and any dropped data (frames or sensor packets) should be logged. For video, the phone could store a frame counter or identifier with each frame; if frames are dropped in preview, the PC can ignore that (since full data is on phone), but if any frames fail to record on the phone, the app should log an error. For GSR, if the Bluetooth connection momentarily drops, the system should attempt reconnection and mark any gaps in the data. Essentially, robust error handling for sensor disconnects, camera errors, or storage issues is required, with clear logging so researchers know if any data segment is unreliable.
- **Scalability and Performance:** The design should be scalable to at least 2-3 phones initially, but aim for even more if needed (limited by Wi-Fi bandwidth and PC processing). Performance testing should be done to ensure the PC can handle decoding multiple video previews and writing data without lag. If the PC has a GPU, using hardware acceleration for video decode (and perhaps encoding if recording PC webcam video) is recommended. On the phone side, optimize the dual recording pipeline (e.g., using efficient video codecs and perhaps dedicated threads for each camera) so that high-resolution dual recording plus streaming does not overwhelm the device. If necessary, allow adjusting parameters (such as using 720p preview while recording 1080p locally) to fit within performance limits of the hardware.
- **Calibration of Sensors:** In addition to camera alignment, consider calibration for the GSR sensor (e.g., zeroing or baseline measurement before start) and the thermal camera (flat-field calibration to account for sensor drift). The requirements could include a calibration routine that the user can run: for GSR, maybe shorting the leads to get a zero baseline or inputting known resistor; for thermal, using the provided calibration shutter (if any) or prompting the user to cover lens for a moment. These steps, while not part of every recording, ensure that the data collected is accurate and reproducible. The system should document and, if possible, automate these calibration steps.
- **Documentation and Research Notes:** Given this is a research-oriented project, the final system should come with thorough documentation. This includes a **user manual** detailing how to set up the hardware (phones, PC, cameras, sensors), how to operate the software (both phone app and PC app), and how to troubleshoot common issues. It should also explain the data format and how to post-process the data. For example, instructions on converting the recorded data into a form suitable for analysis (if not already in a standard format) would be valuable. Citing prior work (like Blum *et al.* 2021 who demonstrated synchronized multi-sensor smartphone recording [16]) can help users trust the methodology. Additionally, the documentation can encourage best practices such as synchronizing device clocks via internet or performing a test recording to verify sync before an actual experiment.
- **Future-Proofing:** While the immediate requirement is GSR, RGB, and thermal data, the system architecture should be extensible. It should be relatively easy to add new sensor types or data streams in the future. For instance, if one wanted to include an ECG sensor or another physiological signal, the framework (especially with LSL) can accommodate additional streams. Thus, a requirement is to design the software in a modular way – e.g., defining interfaces for "sensor modules" – so new modules (like "Shimmer ECG" or another camera) can be integrated with minimal changes. The configuration protocol between PC and phone should be flexible

enough to negotiate what sensors/streams a phone has and control them. This modular design will add longevity to the platform as research needs grow.

By addressing all the above requirements, the new application will provide a **comprehensive, synchronized multi-device recording system**. It combines the strengths of mobile sensors with a powerful PC control interface, ensuring high-quality data capture (thermal imagery, RGB video, and GSR signals) with precise time alignment across devices. Utilizing frameworks like LSL for synchronization and streaming ensures that even with multiple phones and sensors, the data can be integrated seamlessly for later analysis [7]. This design is well-suited for research in psychophysiology, mHealth, and multi-modal sensing, offering flexibility and reliability grounded in current best practices from recent studies [16]. The result will be a "pocket lab" system where data from wearable sensors and cameras can be collected in the field with laboratory-grade synchronization accuracy [17] [18], fulfilling the needs of the project and opening doors for future enhancements.

Sources: Recent research and documentation were consulted to inform these requirements, including the original project repository and relevant literature on synchronized mobile data acquisition [1] [4] [19] [16]. These sources underline the importance of precise time sync and robust data handling in multi-sensor setups, which the above requirements strive to achieve.

---

[1] [2] [3] [4] [5] [6] [11] [13] [14] [15] [19] README.md
https://github.com/buccancs/gsr-android-dual-video-stream/blob/91ed504b1bae3336b870e9452bc3ea2bae9b5c42/README.md

[7] [8] [9] [10] [12] [16] [17] [18] Pocketable Labs for Everyone: Synchronized Multi-Sensor Data Streaming and Recording on Smartphones with the Lab Streaming Layer
https://www.mdpi.com/1424-8220/21/23/8135