| ***Adaptive HCI:***  *Real-time Stabilization and 3D Reconstruction of Hand Gestures and Finger Movement Traces Using LED-Equipped Gloves*  ***Design Specification*** |  | | |  |
| --- | --- | --- | --- | --- |
|  | |  | |  |
| ***SOFTWARE DESIGN SPECIFICATION***  **1.0 Introduction**  This document describes all data, architectural, interface and component-level design of the Adaptive HCI system, detailing its implementation for real-time air-writing tracking, visualization, and text inference.  1.1 Goals and objectivesThe Adaptive HCI project aims to develop a mobile application that enables stable, accurate, and real-time air-writing tracking, allowing users to write in the air and visualize their strokes on a mobile application and in VR/AR environments. This system is particularly designed to support users with motor control challenges, such as patients with Parkinson’s disease, by mitigating tremors and enhancing writing stability. Additionally, it converts air-written strokes into text using inference models, improving accessibility in digital interactions.  **Key objectives:**   * To develop a real-time air writing system that displays virtual handwriting on mobile and VR/AR platforms. * To minimize hand instability and tremors using advanced tracking and smoothing techniques for clearer, more accurate strokes. * To implement text inference models to convert air-written strokes into readable text, enhancing usability and accessibility.   1.2 Statement of scopeThe Adaptive HCI software is a mobile application designed for real-time air-writing tracking, visualization, and text inference, particularly benefiting individuals with hand instability, such as patients with Parkinson’s disease. The system captures air-written strokes, smooths the motion to reduce tremors, and converts the writing into text while enabling real-time visualization in mobile and VR/AR environments.  **Major Inputs:**   * **LED-Equipped Glove Data:** Captures hand movements in real-time, tracking LED positions (x, y, z coordinates) for precise air-writing. * **Video Stream from Device Camera:** Provides a live feed to detect LED position and trace air-writing strokes using image processing techniques. * **User Commands and Settings:** Allows users to start/stop tracking, modify shutter speed, and configure VR/AR display options.   **Processing Functionality:**   * **Air-Writing Stroke Tracking:**  Uses the YOLOv11 object detection model to detect and map LED movements, generating a virtual writing path in real-time. * **Path Smoothing and Noise Reduction:** Applies Kalman filter and interpolation to minimize tremor effects and create clearer, more stable writing. * **Text Inference and Recognition:** Converts air-written strokes into text using CNN models for improved usability.   **Outputs:**   * **Real-Time Stroke Visualization:** Displays the user’s air-written strokes and text on the smartphone screen and in VR/AR environments. * **Converted Text Output:** Processes air-writing strokes into digital text for interaction in virtual spaces. * **Session Data Storage:** Saves processed stroke data for further analysis.   1.3 Software contextThe Adaptive HCI system will be accessible through a mobile application, primarily running on smartphones with rolling shutter cameras. As the project relies on real-time tracking, low-latency processing and multi-user scalability are essential. Additionally, it is designed for future expansion, enabling integration with various platforms (e.g., VR headsets, IoT devices) to support users with Parkinson’s disease and other mobility impairments.  1.4 Major constraintsThe Adaptive HCI system faces key constraints in time, hardware compatibility, privacy, and system performance, requiring careful optimization and efficient execution.   * **Time Constraints:** The project is designed for completion within an academic semester, requiring tight milestones and efficient time management. * **Hardware Limitations:** The application must support rolling shutter cameras on various smartphones while optimizing hardware-intensive image processing for lower-end devices. Additionally, VR/AR compatibility presents challenges in ensuring seamless performance across multiple VR/AR platforms with varying hardware capabilities. * **Privacy:** All data must be processed locally or securely stored, ensuring user privacy and filtering out non-relevant background information. * **Accuracy and Responsiveness:** The system must balance smoothing and responsiveness, maintaining real-time user control without excessive delay or unintended modifications, while ensuring precise recognition of numbers and characters for text inference.   **2.0 Data design**  This section outlines the data structures used within the Adaptive HCI system, including internal, global, and temporary data structures.  2.1 Internal software data structureInternal data structures are used within specific software components for real-time processing but are not shared globally.  **Data Structures Used:**   * **FrameBuffer (List[Frame]):**   + Temporarily stores video frames for real-time processing.   + Used by the Image Processing Module before extracting LED positions. * **GesturePath (List[Point3D]):**   + Stores the (x, y) coordinates of the detected LED position using YOLO object detection.   + Used for tracking, smoothing, and rendering air-writing strokes in real-time. * **FilteredPath (List[Point3D]):**   + A smoothed version of GesturePath, processed using Kalman filters and interpolation.   + Passed to the Text Inference Module for text inference. * **RecognizedText (String):**   + The final inferred text output from air-writing strokes.   + Generated by the Text Inference Module (CNN).   2.2 Global data structureGlobal data structures persist throughout the session and are shared across multiple modules.  **Global Data:**   * **UserSettings (Dict[String, Any]):**   + Stores user preferences such as:     - brightnessLevel: Float – Adjusts LED tracking sensitivity     - shutterSpeed: Int – Modifies camera Hz for optimized tracking     - VRMode: Boolean – Enables/disables VR/AR visualization * **SessionData (Object):**   + Stores session metadata to maintain state across interactions:     - sessionID: String – Unique identifier for the session     - startTime: DateTime – Timestamp when the session started     - deviceType: String – Smartphone/VR headset type     - rawGesturePath: List[Point3D] – Unprocessed air-writing strokes     - processedPath: List[Point3D] – Smoothed air-writing strokes     - recognizedText: String – Final inferred text output * **StoredFiles (Dict[String, String]):**   + Maps stored images and videos to session IDs for local retrieval:     - sessionID: String → filePath: String   2.3 Temporary data structureTemporary data structures are created and discarded within the session to facilitate processing.  **Temporary Data:**   * **TempFrameBuffer (Queue[Frame]):**   + Holds video frames temporarily before processing.   + Frames exceeding buffer limits are discarded to prevent lag. * **TempGestureBuffer (Queue[Point3D]):**   + Stores air-writing strokes before smoothing filters are applied. * **CalibrationData (Dict):**   + Used during initial device setup, containing:     - ambientLightLevel: Float – Helps adjust brightness thresholds     - defaultHandSize: Float – Used for depth estimation     - frameDelay: Int – Adjusts frame rate vs. processing speed   2.4 Database descriptionThe Adaptive HCI system does not use a formal database, as all processing occurs in real time without persistent storage. During development, videos and images may be temporarily stored for debugging purposes, but in practical use, no data is saved after processing. The system operates entirely in memory, ensuring minimal storage overhead and prioritizing real-time performance.  **3.0 Architectural and component-level design**  This section describes the architecture and components of the Adaptive HCI system. The system follows a modular architecture, where each component is responsible for a distinct function. The architecture ensures real-time performance, scalability, and low-latency processing by keeping components independent yet well-integrated.  3.1 Program StructureThe system consists of the following key components, interacting sequentially to ensure real-time object detection, contouring, processing, inference, and visualization.  3.1.1 Architecture diagramThe core architecture follows a pipeline-based design, where the mobile application captures LED-tracked motion which undergoes image processing and YOLO-based object detection before being processed for inference and rendered as text and 3D visualization.  3.1.2 Alternatives Several architectural styles were considered:   * **Layered Architecture:** Common in enterprise applications but unsuitable for real-time processing due to added latency. * **Client-Server Architecture:** Rejected since all processing is done locally on the device, eliminating the need for a server. * **Model-View-Controller (MVC):** Useful for UI-driven applications but not optimal for a sequential data pipeline like contour processing.   Therefore, a pipeline-based architecture was chosen for its sequential data flow, ensuring efficient LED finger tracking, processing, detection, inference, and visualization. It enhances modularity, real-time performance, and system clarity, making it ideal for LED-tracked motion recognition.  3.2 (a) Description for Mobile ApplicationThe Mobile Application, also known as “Xamera”, serves as the user interface and primary control center, capturing LED-tracked motion and sending video frames for further processing.  3.2.1 Processing Narrative (PSPEC)The mobile application initiates and manages gesture tracking. It captures video input using the Camera2 API and transmits frames to the Image Processing module in real time. The app allows users to configure settings such as shutter speed and brightness thresholds for better tracking performance. 3.2.2 Component Interface Description   * **Input:** Video feed from the camera, user input for configuration. * **Output:** Processed video frames sent to Image Processing.   3.2.3 Sub-Components  3.2.3.1 Interface description   * **User Interface (UI):** Allows users to start/stop tracking, configure settings, and view real-time feedback. * **Camera Module:** Captures live frames and passes them to Image Processing.   3.2.3.2 Algorithmic Model   | def start\_camera():  camera = open\_camera(device="rear", resolution="1080p")  set\_brightness(camera, mode="low")   set\_shutter\_speed(camera, values=[1/30, 1/60, 1/120])     while camera.is\_active():  frame = capture\_frame(camera)  send\_to\_processing\_module(frame)  display\_frame\_on\_ui(frame)    if ar\_mode\_enabled():  render\_ar\_visualization(frame) | | --- |   3.2.3.3 Restrictions/limitations   * The application is optimized for Motorola G Play and Google Pixel devices. Performance on other devices may vary. * Requires LED-equipped gloves for accurate motion tracking. * Designed for use in dark environments to enhance LED visibility and improve detection accuracy.   3.2.3.4 Local data structures   * FrameBuffer:Temporarily stores video frames before processing. * ConfigSettings: Stores user preferences for tracking parameters.   3.2.3.5 Performance issues   * Ensuring a high frame rate (>30 FPS) is critical for real-time processing. * Potential overheating issues in prolonged use.   3.2.3.6 Design constraints   * Must be optimized for mobile hardware with limited CPU/GPU resources.   3.2 (b) Description for Image Processing ModuleThe Image Processing Module enhances captured video frames to optimize LED tracking and gesture contour processing before passing the data to the Object Detection and Inference modules.  3.2.1 Processing narrative (PSPEC)  Once the mobile application captures video frames, they are preprocessed to improve detection accuracy and feature extraction. This includes:   * Normalizing brightness across frames to improve LED visibility. * Reducing noise for better feature extraction. * Highlighting smoothed LED contours for accurate tracking. * Converting frames to grayscale for uniform processing. * Saving the last written contour and converting it to 28x28 pixels before passing it to the Inference Model for character/digit recognition.   3.2.2 Component Interface Description   * **Inputs:** Raw video frames from the Mobile Application. * **Outputs:**    + Processed images sent to Object Detection for LED tracking.   + Resized grayscale images sent to Inference Model for text recognition.   3.2.3 Sub-Components  3.2.3.1 Interface Description   * **Preprocessing Module:** Normalizes brightness, reduces noise, and enhances feature detection. * **Feature Extraction Module:** Identifies LED contours and prepares data for object detection and text inference.   3.2.3.2 Algorithmic Model   | def process\_frame(frame):  grayscale\_frame = convert\_to\_grayscale(frame)  equalized\_frame = apply\_histogram\_equalization(grayscale\_frame)  filtered\_frame = apply\_gaussian\_filter(equalized\_frame, kernel\_size=(3,3))  edge\_detected\_frame = detect\_edges(filtered\_frame)    led\_contours = extract\_led\_contours(edge\_detected\_frame)  resized\_contour = resize\_image(led\_contours, size=(28,28))    save\_last\_contour(resized\_contour)  send\_to\_object\_detection(resized\_contour) | | --- |   3.2.3.3 Restrictions/limitations   * The application is optimized for Motorola G Play and Google Pixel devices. Performance on other devices may vary. * Requires LED-equipped gloves for accurate motion tracking. * Designed for use in dark environments to enhance LED visibility and improve detection accuracy. * Requires stable frame rates (≥30 FPS) for accurate contour extraction. * High computational load may impact performance on lower-end devices. * Extreme lighting conditions can interfere with LED detection.   3.2.3.4 Local data structures   * FrameBuffer: Temporarily stores raw video frames before preprocessing. * ProcessedFrame: Stores video frames after brightness adjustment and noise reduction. * ContourData: Stores extracted contour features, including:   + GrayscaleImage: Preprocessed frame converted to grayscale for contour detection.   + ResizedImage (28x28 pixels): Downsampled frame for compatibility with the inference model.   + LEDTrackingContours: Stores LED-tracked contours used for gesture recognition and inference processing. * TrackingParameters: Holds threshold values for LED detection, brightness levels, and adaptive filtering parameters. * FeatureData: Extracted shape and position data from LED contours, passed to object detection and inference.   3.2.3.5 Performance issues   * Maintaining real-time processing speed while applying brightness adjustment, noise reduction, and edge detection. * Increased computational load may lead to occasional frame drops, affecting LED tracking accuracy. * Processing efficiency depends on device hardware; lower-end devices may experience delays. * Ensuring smooth data transfer to object detection/inference without delays or interruptions.   3.2.3.6 Design constraints   * Must be optimized for efficient image processing on mobile devices with limited computational power. * Dependent on camera quality and lighting conditions for accurate LED tracking. * Must efficiently manage memory to handle continuous frame processing without slowdowns.   3.2 (c) Description for Object Detection (YOLO-Based LED Tracking)  The Object Detection Module is responsible for identifying and tracking the LED-equipped gloves using a YOLO-based model. This module detects the green LED on the finger, differentiates multiple users through On-Off Keying (OOK) signals, and extracts real-time positional data for contour tracking. 3.2.1 Processing Narrative (PSPEC)The object detection module processes video frames to identify and track LED motion for air-writing recognition. This involves:   * Receiving preprocessed frames from the Image Processing Module. * Detecting the LED-equipped finger using a YOLO-based model. * Differentiating users via unique OOK signal patterns. * Extracting bounding box coordinates and estimating depth from the LED size in the frame. * Passing extracted positional data to the Path Smoothing and Inference Modules for processing and text recognition.   3.2.2 Component Interface Description   * **Input:**    + Preprocessed video frames (grayscale, resized to 28x28) from Image Processing.   + Calibration settings (brightness threshold, motion sensitivity). * **Output:**    + LED bounding box coordinates (x, y).   + Depth estimation (z) derived from LED size.   + User identification via OOK decoding.   3.2.3 Sub-Components  3.2.3.1 Interface description   * **YOLO Detection Module:**    + Receives processed frames and detects LED location.   + Outputs bounding box coordinates and confidence scores. * **User Identification Module (OOK Processing):**   + Identifies users by decoding OOK LED signal patterns.   + Ensures multiple users can write simultaneously.   3.2.3.2 Algorithmic model   | def detect\_led(frame):  yolo\_output = yolo\_model.predict(frame)  if yolo\_output:  bbox = extract\_bounding\_box(yolo\_output)  led\_position = estimate\_depth(bbox)  user\_id = decode\_ook\_signal(frame, bbox)  return led\_position, user\_id  return None | | --- |   3.2.3.3 Restrictions/limitations   * Optimized for Motorola G Play and Google Pixel devices. Performance may vary on other hardware. * Requires high-contrast green LED visibility for reliable detection. * Performance may degrade in highly reflective environments or poor lighting conditions. * Limited to tracking two distinct users based on OOK signal differentiation; additional users may not be reliably identified.   3.2.3.4 Local data structures   * LEDPosition:Stores detected (x, y, z) coordinates of the LED. * OOKSignal: Buffers light intensity changes over frames for decoding. * BoundingBox: Saves YOLO output with LED position and confidence scores.   3.2.3.5 Performance issues   * Ensuring real-time detection without frame drops (<30ms per frame). * Maintaining low false positives for LED recognition. * Managing multiple LED sources without interference.   3.2.3.6 Design constraints   * Must be optimized for mobile hardware with limited CPU/GPU resources. * Must handle multiple user tracking via OOK decoding.   3.2 (d) Description for Inference ModuleThe Inference Module converts the processed motion data into text representations using a neural network-based approach. It takes smoothed gesture paths and predicts the corresponding characters or digits.  3.2.1 Processing Narrative (PSPEC)The Inference Module processes motion data extracted from the Object Detection Module and predicts air-written characters/digits using a trained deep learning model. This includes:   * **Data Standardization:** Adjusting gesture sequences to a uniform format. * **Feature Extraction:** Identifying spatial and temporal patterns in the smoothed path. * **Text Prediction:** Running the processed motion sequence through a trained model to infer the intended characters/digits.   3.2.2 Component Interface Description   * **Input:**    + Smoothed motion data from the Path Extraction Module (x, y, z, t).   + Resized grayscale image representations (28x28) of drawn contours for secondary text inference. * **Output:**    + Predicted character or digit.   + Confidence score of the prediction.   3.2.3 Sub-Components  3.2.3.1 Interface description   * **Preprocessing Module**: Formats contour data for model input. * **Neural Network Model:** Uses a CNN (Convolutional Neural Network) to predict text. * **Post-Processing Module:** Applies confidence filtering and output correction.   3.2.3.2 Algorithmic model   | def infer\_text(gesture\_path, contour\_image):  standardized\_data = standardize\_input(gesture\_path)  motion\_prediction = crnn\_model.predict(standardized\_data)  resized\_contour = resize\_image(contour\_image, (28, 28))  image\_prediction = cnn\_model.predict(resized\_contour)  return merge\_predictions(motion\_prediction, image\_prediction) | | --- |   3.2.3.3 Restrictions/limitations   * Optimized for pre-trained gesture datasets; requires retraining for new writing styles. * Limited multi-user functionality; models are optimized for individual handwriting calibration.   3.2.3.4 Local data structures   * StandardizedData: Stores preprocessed contour sequences, including 28x28 grayscale images, for model input. * InferenceResult: Holds final text output and confidence scores.   3.2.3.5 Performance issues   * Must generate predictions in <100ms to avoid lag. * Neural inference is compute-intensive on mobile hardware. * Prediction errors may occur due to fast motion artifacts.   3.2.3.6 Design constraints   * Requires TensorFlow Lite for optimized on-device inference. * Must be optimized for mobile hardware with limited CPU/GPU resources. * Designed for discrete character/digit recognition rather than continuous handwriting interpretation.   3.2 (e) Description for 3D Visualization ModuleThe 3D Visualization Module is responsible for rendering the final air-written text and motion path in a 3D space, ensuring real-time feedback and AR compatibility.  3.2.1 Processing Narrative (PSPEC)Once the inference module generates the predicted text, the 3D Visualization Module renders both the detected stroke path and corresponding text in a 3D space. Key steps include:   * **Path Rendering:** Displays the user’s air-written motion using a 3D stroke representation. * **Text Overlay:** Projects the recognized text alongside the drawn stroke. * **VR/AR Integration:** Supports real-time interaction via OpenGL for immersive visualization.   3.2.2 Component Interface Description   * **Input:**    + Processed gesture path (x, y, z) from Object Detection & Path Tracking.   + Predicted text output from the Inference Module. * **Output:**    + 3D-rendered path representation.   + Recognized text displayed in the virtual space.   3.2.3 Sub-Components  3.2.3.1 Interface description   * **Path Renderer:** Creates a continuous 3D stroke from detected motion data. * **Text Display:** Positions the recognized text within the 3D environment. * **AR Interface:** Enables real-time visualization in AR mode.   3.2.3.2 Algorithmic model   | def render\_visualization(gesture\_path, recognized\_text):  path\_3d = generate\_3d\_stroke(gesture\_path)  display\_text(recognized\_text, path\_3d.position)  if vr\_mode\_enabled:  enable\_vr\_rendering(path\_3d, recognized\_text) | | --- |   3.2.3.3 Restrictions/limitations   * Requires OpenGL or a compatible 3D rendering engine. * Designed for Android AR-compatible devices (e.g., Google Pixel, Motorola G Play). * Requires adequate lighting for accurate tracking and rendering.   3.2.3.4 Local data structures   * RenderedPath:Stores the 3D stroke representation of air-written text. * TextOverlay: Stores the positioned and formatted predicted text.   3.2.3.5 Performance issues   * Real-time rendering must maintain >30 FPS for smooth visualization. * Processing delays may occur with complex 3D scenes or large gesture paths. * Potential overheating issues in prolonged use.   3.2.3.6 Design constraints   * Must be optimized for mobile hardware with limited CPU/GPU resources. * Data must be lightweight for efficient rendering and low-latency feedback.   3.3 Software Interface DescriptionThis section describes the interfaces through which the software interacts with external machines, systems, and users.  3.3.1 External machine interfaces  The system interacts with external hardware components to enable contour tracking, processing, and visualization. These include:   * **Camera (Android Camera2 API):** Captures real-time video frames and streams them for image processing and LED tracking. * **LED-Equipped Gloves:** Provides a visible tracking point (LED) for motion capture. The system tracks LED movement and differentiates between users based on signal patterns. * **Mobile GPU (OpenGL Rendering):** Used for 3D visualization, rendering gesture paths in real time. * **VR/AR Headset:** Supports immersive gesture-based interaction by rendering air-writing in a virtual environment.   3.3.2 External system interfaces  The system operates fully on the device without cloud dependencies. Key components include:   * **TensorFlow Lite for Model Inference:** Loads the trained deep learning model for real-time text prediction, processing both motion-based and image-based contour data. * **Android Native APIs:** Utilized for hardware access, camera control, and rendering through OpenGL.   3.3.3 Human interfaceThe software provides an intuitive mobile interface with real-time visual feedback and adjustable settings:   * **Main Screen:**   + Displays Contour Tracking in real-time.   + Includes Start/Stop Tracking buttons to control recording.   + Provides an option to enter AR Mode for immersive air-writing visualization in a virtual environment. * **Settings Panel:**   + Allows users to adjust Brightness Levels for optimal LED tracking.   + Enables configuration of Shutter Speed to refine motion capture.   **4.0 User Interface Design**  This section describes the user interface design of Xamera, detailing its components, interactions, and design rules.  4.1 Description of the user interfaceThe user interface (UI) of Xamera is designed to be intuitive, providing users with seamless access to the application's core functionalities. The interface revolves around real-time camera interaction, tracking, and visualization modes.    4.1.1 Screen imagesBelow is an overview of the primary screens in Xamera:   1. **Permissions Request Screen (First Launch)**  * Users are prompted to grant permissions for camera, microphone, and file access.  1. **Home Screen (Live Feed View)**  * Displays the live camera feed as the primary focus. * **Bottom Controls:**   + **"Start Tracking" Button** – Initiates real-time LED glove tracking.   + **"Switch Camera" Button** – Toggles between front and rear cameras. * **Right-Side Controls:**   + **"Zoom In"** – Increases zoom level.   + **"Zoom Out"** – Decreases zoom level.   + **"2D View"** – Switches visualization to a 2D plane.   + **"3D View"** – Enables 3D representation of tracking data.   + **"AR View"** – Overlays tracked objects in an augmented reality space. * **Top Controls:**   + **"Settings"** Button – Opens the settings menu.   + **"About"** Button – Displays project and app information.   + **Top Banner:** Displays "Xamera Pre-Beta" indicating the app's name and version.  1. **Settings Screen**  * Allows users to configure camera behavior:   + "Rolling Shutter Speed" – Drop-down menu to select capture rate (5 Hz to 6000 Hz).   + "Lighting Mode" – Options include Low Light, High Light, or Normal.   + "Enable Flash" – Toggle switch to turn camera flash on/off.  1. **About Screen**  * Provides information about the project and app details.   4.1.2 Objects and actions   * **Camera Feed** – Displays real-time video input from the device camera. * **Buttons & Actions:**   + **"Start Tracking"** – Begins object detection and tracking.   + **"Switch Camera"** – Changes between front and rear cameras.   + **"Zoom In/Out"** – Adjusts the camera zoom level.   + **"2D/3D/AR View"** – Changes the display mode for visualization.   + **"Settings"** – Opens the configuration menu.   + **"About"** – Displays app and project information.   + **"Rolling Shutter Speed"** – Drop-down selection to control capture rate.   + **"Lighting Mode"** – Adjusts image processing parameters based on lighting conditions.   + **"Enable Flash"** – Turns the camera flash on or off.   4.2 Interface design rulesThe UI follows these design principles:   * **Minimalistic & Functional:** Focus on essential controls without unnecessary clutter. * **Consistency:** Unified design language across all screens. * **Accessibility:** Large buttons and clear icons for easy interaction. * **Real-Time Responsiveness:** Immediate feedback when users interact with the camera feed and tracking features. * **User-Centric Navigation:** Simple, direct paths to key functions with intuitive gestures and button placements.   4.3 Components availableXamera's UI is built using standard mobile UI components including:   * **Buttons:** For actions such as tracking, camera switching, and settings adjustments. * **Drop-Down Menus:** Used for selecting rolling shutter speed. * **Toggle Switches:** For enabling or disabling the flash. * **Camera Viewport:** Displays real-time feed and tracking visuals. * **Icon-based Controls:** For zoom and visualization mode selection.   4.4 UIDS descriptionXamera's UI is developed using:   * **Android UI Toolkit (Jetpack Compose / XML Layouts)** – For dynamic interface rendering. * **OpenCV Integration** – To draw the traced line in the Camera view. * **OpenGL** - To display the traced line in the 2D and 3D view, as well as assist ARcore in visualization. * **ARCore** – For implementing augmented reality-based tracking views. * **Touch and Gesture Controls** – To facilitate smooth interaction with zoom and tracking functionalities.   Overall, Xamera’s UI is designed for an optimized real-time tracking experience with intuitive user interaction and configurable settings tailored to different lighting and performance needs.  **5.0 Restrictions, Limitations, and Constraints**  The mobile application, Xamera, is designed for real-time video recording and processing, with several key constraints influencing its development:   * **Hardware Limitations:** The app runs on both the Moto G Play and Google Pixel 8A, each with distinct hardware capabilities. The Moto G Play has limited processing power and lacks a dedicated GPU, making CPU optimization crucial for real-time performance. The Google Pixel 8A, with its more powerful chipset and AI acceleration, allows for higher frame rates and improved processing efficiency. * **Frame Rate Constraints:** The Moto G Play currently achieves ~10 FPS using the YOLO Nano model, requiring further optimizations such as mixed-precision inference and multi-threading to improve real-time performance. The Google Pixel 8A can process at ~60 FPS+, benefiting from its more capable hardware. * **Battery Consumption:** Continuous real-time processing, especially involving OpenCV-based operations, can drain battery life rapidly. Power-efficient techniques such as frame skipping, adaptive processing, and hardware-accelerated inference must be employed to extend battery performance on both devices. * **Memory and Storage Constraints:** The devices have varying RAM and storage capacities, impacting buffer sizes, video processing efficiency, and temporary data storage. Optimized memory management is required, particularly on the Moto G Play. * **Software Dependencies:** The app relies on OpenCV, PyTorch (TorchScript), and Android’s Camera2 API, necessitating careful dependency management to ensure compatibility across both devices while maintaining stability and performance. * **Security and Privacy:** As the app processes live video data, secure handling of video streams is critical to protect user privacy and prevent unauthorized access. Encryption and local processing (when feasible) will help maintain security. * **Real-Time Processing Trade-offs:** Due to hardware limitations, some computationally expensive tasks (e.g., high-resolution image processing, advanced AI inference) may require optimizations or offloading to a server when necessary to balance performance and real-time responsiveness.   This dual-device approach ensures Xamera remains functional across different performance tiers while adapting optimally to each device’s strengths and constraints.  **6.0 Testing Issues**  The testing strategy for Adaptive HCI involves both black-box and white-box testing to ensure functionality, performance, and reliability.  6.1 Classes of tests   * **Unit Testing:** Testing individual functions such as frame extraction, grayscale conversion, contour detection, and object tracking. * **Integration Testing:** Verifying interactions between components, such as the video pipeline, object detection module, and tracking system. * **Performance Testing:** Evaluating FPS under different conditions, including varying lighting and movement speeds. * **Stress Testing:** Simulating prolonged usage to identify potential memory leaks and CPU overheating issues. * **User Testing:** Ensuring usability and smooth user interaction with the interface and live tracking results. * **Security Testing:** Checking for vulnerabilities in video handling and data transmission.   6.2 Expected software response   * The application should detect and track the LED light source reliably, even in varying lighting conditions. * The video stream should not freeze or lag excessively beyond acceptable real-time constraints. * Processed video frames should be displayed smoothly, with minimal frame drops. * The tracking module should correctly identify and follow the LED light source without significant false positives. * The app should not crash or produce memory-related errors during continuous usage. * Data storage (if applicable) should be efficient, without unnecessary duplication or excessive space consumption.   6.3 Performance bounds   * **Frame Processing Time:** Targeting <100ms per frame to approach real-time processing. * **Battery Consumption:** Must not exceed an unsustainable drain rate during prolonged usage. * **CPU Utilization:** Should remain below 80% to avoid excessive heating and system lag. * **Latency:** The delay between capturing and processing frames should remain minimal, ideally under 300ms. * **Model Accuracy:** The YOLO Nano model must maintain at least 85% accuracy in detecting the LED light source across different environmental conditions.   6.4 Identification of critical components   * **Camera Input Pipeline:** Ensuring smooth frame capture and delivery to the processing pipeline. * **YOLO Nano Object Detection:** Critical for real-time LED tracking; must be highly optimized. * **Contour Detection & Tracking:** Key for computing the LED’s position and motion path. * **Multi-Threading Implementation:** Essential for distributing processing loads efficiently. * **Memory Management:** Preventing memory leaks and excessive RAM usage during video processing. * **User Interface (UI) Responsiveness:** The application must remain interactive and not become unresponsive due to background processing loads   **7.0 Appendices**  7.1 Requirements traceability matrix   | **Requirement** | **Component** | **Data Structure** | **Description** | | --- | --- | --- | --- | | Real-time LED Tracking | Image Processing, Object Detection | FrameBuffer, GesturePath, FilteredPath | Tracks LED movement using YOLO detection and processes in real-time for visual output. | | Path Smoothing and Tremor Reduction | | Path Smoothing, Kalman Filter | | --- |  |  | | --- | | GesturePath, FilteredPath | Uses Kalman filtering to smooth gesture paths, reducing tremors for clarity. | | Text Inference | Text Inference Module | RecognizedText | Converts air-written gestures into readable text using an LSTM model. | | User Customization (Settings) | UI, Settings Screen | UserSettings | Allows users to adjust camera speed, lighting, and VR settings for optimal performance. | | AR/VR Integration | UI, ARCore Integration | SessionData, ProcessedPath | Visualizes real-time strokes in 2D, 3D, and AR environments. | | Session Data Storage | Session Management | SessionData, StoredFiles | Stores session metadata and user-specific data locally for future use. |   7.2 Packaging and installation issuesSpecial considerations for the packaging and installation of the Adaptive HCI system have been identified to ensure the software is easy to deploy, configure, and use. These considerations are focused on compatibility, performance, and user experience.  **Packaging Considerations:**   1. Platform Compatibility: The application is designed for smartphones (Android), with a focus on devices like Motorola G Play and Google Pixel. It will be packaged in APK format to be installed on Android devices. 2. Dependencies: The software requires integration with specific hardware components such as LED-equipped gloves and support for ARCore. Ensure ARCore and OpenGL libraries are correctly bundled and linked during packaging. 3. Image Processing Optimization: Due to the performance requirements of image processing, optimizations for GPU usage (e.g., OpenGL ES) should be included to minimize CPU load and enhance processing speed. 4. Camera and Hardware Permissions: The application requires permission for camera, microphone, and file access. These permissions need to be declared and managed in the installation process.   **Installation Considerations:**   1. Pre-Installation Checks: Ensure the device meets the minimum hardware requirements (e.g., stable frame rate greater than 30 FPS, ARCore support). 2. User Setup: The first-time installation should guide the user through initial setup, including calibration for the LED-equipped glove and customization of user settings (brightness, shutter speed, etc.).   7.3 Design metrics to be usedA description of all design metrics to be used during the design activity is noted here.  Design metrics are crucial to evaluate the performance, usability, and efficiency of the Adaptive HCI system during the design and development phases. The following metrics will be used:  **Real-Time Processing Performance (Latency):**   * **Metric:** Frame processing time (in milliseconds) * **Goal:** Maintain real-time processing (greater than 30 FPS) without significant delays between input capture and output visualization. * **Measure:** Time between capturing a video frame and displaying the visualized stroke on the screen.   **Accuracy of Gesture Tracking:**   * **Metric:** Detection accuracy rate (percentage of frames with correct LED detection) * **Goal:** Achieve greater than 90% accuracy in detecting LED positions across frames. * **Measure:** Number of correctly tracked frames divided by the total number of frames captured.   **Stability of Path Smoothing:**   * **Metric:** Path smoothness score (variance in filtered path coordinates) * **Goal:** Reduce path variance caused by tremors, aiming for a smooth user experience. * **Measure:** Variance in the smoothed gesture path (after Kalman filter application) compared to raw gesture path.   **Text Inference Accuracy:**   * **Metric:** Recognition accuracy (percentage of correctly inferred characters) * **Goal:** Achieve above 90% accuracy in text recognition for air-written letters and numbers. * **Measure:** The percentage of correctly identified characters out of all detected gestures.   **User Experience (UX) Feedback:**   * **Metric:** User satisfaction score (via post-session survey) * **Goal:** Achieve a satisfactory score for ease of use and real-time responsiveness. * **Measure:** Survey responses from test users after interaction with the application.   7.4 Supplementary Information   * **Device Compatibility:** Optimized for Motorola G Play and Google Pixel 8A; performance may vary on other devices. * **AR/VR Integration:** Uses OpenGL for rendering; future support for ARCore/ARKit can be explored. * **Security & Privacy:** All processing occurs on-device with no cloud dependency, ensuring user privacy. * **Performance Optimization:** Adjust brightness and shutter speed for tracking stability; frame skipping can reduce lag on lower-end devices. * **Future Enhancements:** Expanding multi-user support and improving handwriting recognition through adaptive learning. | | |  |  |