Adaptive HCI: Air Writing Tracking Application

Table of Contents

Technical Outline

Overview

The application, tentatively called **Xamera**, targets Android devices (with VR compatibility) and enables multiple users to write in the air simultaneously. The system uses:

1. A single camera with adjustable shutter rate.
2. LED-equipped gloves—each glove features a green LED on the index finger, configured with a unique On-Off Keying (OOK) signal to differentiate users.

In addition, the application addresses the needs of users with motor control issues (e.g., Parkinson's disease) by implementing noise mitigation, path smoothing and text inference.

---

1. Mobile Application

- **Platform**: Android Studio (Kotlin).
- **Features**:
    1. **Start/Stop Tracking**: Easily toggles the system on or off.
    2. **Processed Pathway Visualization**: Displays real-time or recorded air-writing strokes, compatible with VR headsets.
    3. **Adjustable Settings**: Users can configure shutter speed, brightness thresholds, or smoothing parameters.

Selected Implementation / Workflow

1. **UI Setup**
    - Implement a main activity with two buttons: **Start** and **Stop**.
    - A settings menu allows users to adjust parameters such as shutter speed, brightness threshold, or motion smoothing level.
2. **Camera Preview & VR**
    - Integrate the **Camera2 API** to show live camera preview within the main activity.
    - For VR compatibility, render the camera feed and processed paths on a **Unity3D** scene or on a **Google VR** surface.

---

2. Video Input

- **Technology**: **Camera2 API** for Android.
- **Functionality**:
    - Captures video frames in real time at an adjustable shutter rate.
    - Streams frames to the subsequent processing pipeline (Pre-Processing → to Detection → to Tracking).

Selected Implementation / Workflow

1. **Permissions & Camera Setup**
    - Request camera permissions at runtime.
    - Open a CameraDevice session and configure a CaptureRequest to optimize shutter speed for LED tracking.
2. **Frame Access**
    - Implement an ImageReader to acquire **YUV_420_888** images for lower-latency image processing.
    - Convert to the desired color format (e.g., RGBA) if needed by the detection/processing library.

---

3. Pre-Processing

Pre-processing aims to improve image quality by reducing noise and enhancing relevant features—particularly the green LED.

Technique	Pros	Cons	Applicability
Grayscale Conversion	Simplifies shape-based tasks	Removes color information (critical for LED)	**Not** recommended for LED tracking
Brightness Thresholding	Isolates bright LEDs in noisy conditions	May remove other relevant features	Useful in extreme lighting cases
Histogram Equalization	Normalizes brightness across frames	Can introduce artifacts in well-lit scenes	Useful for inconsistent lighting
Gaussian Blur	Reduces noise for smoother edges	Can reduce sharpness of features	Good for high-frequency noise reduction

Selected Pre-Processing Workflow

1. **Brightness Normalization**
    - Apply **histogram equalization** specifically to the **green channel** to preserve color information vital for LED detection.
2. **Noise Reduction**
    - Use a **light Gaussian Blur** (e.g., kernel size 3x3 or 5x5) to reduce high-frequency noise without overly smearing key features.
3. **ROI Cropping**
    - If the LED's approximate region of interest (ROI) is known or can be quickly estimated, crop to that ROI to reduce computational overhead.

---

4. Object Detection & User Identification

- **Primary Approach**: **Custom-trained YOLO (You Only Look Once)** model.
    - Specialized to detect the green LED on the finger.
    - Specialized in detecting the shutter rate pattern resulting in per user identification
    - Trained across a variety of lighting, occlusion, and rotation scenarios.

Why YOLO?	Pros	Cons
Efficient object detection	High accuracy for bounding box detection	Doesn't track objects across frames
Lightweight for real-time	Easy integration with frameworks	Requires high-quality labeled training data

Workflow for YOLO Implementation

1. **Data Collection**
    - Capture videos of users wearing the LED gloves under various conditions (indoor, outdoor, different backgrounds).
    - Extract frames (via **OpenCV** or **FFmpeg**) and maintain balanced sets for training and validation.
2. **Annotation**
    - Use tools like **LabelImg** to draw bounding boxes around the LED finger.

- Export annotations in YOLO format.

3. **Dataset Preparation**
    - Split data into **training (80%)** and **validation (20%)** sets.
    - Optionally add a small test set (5–10%) for final performance evaluation.
4. **Model Training** (using PyTorch)
    - Adjust hyperparameters (batch size, learning rate, epochs) to find an optimal balance between speed and accuracy.
    - Utilize data augmentation (random brightness, rotations) to improve robustness.
5. **Deployment**
    - Export the trained model (e.g., model.pt).
    - Integrate into the Android app via **PyTorch Mobile**.

---

5. (Maybe?) Alternative User Identification

- **Approach**: **Signal Processing** for On-Off Keying (OOK) decoding.
    - Each LED emits a unique On/Off pattern.
    - The system correlates bounding boxes (from YOLO) with these patterns to assign user IDs.

Why Signal Processing?	Pros	Cons
Robust differentiation	Distinguishes identical-looking objects	Susceptible to noise in low-light conditions
Lightweight computation	Easy integration with YOLO output	Decoding might fail for overlapping signals

Alternative: Use **distinct LED colors** for each user. (Simpler to decode but limits scalability if many unique colors are required.)

Workflow for OOK Signal Processing

1. **Extract Signal Region**
    - Crop the bounding box around the LED finger from the YOLO output.
2. **Decode OOK Signal**

- Track intensity changes in the **green channel** over consecutive frames.
- Match the pattern to a database of known OOK signatures to identify each user (e.g., User 1, User 2).

---

6. Path Extraction

- **Approach**: Use bounding box centers (x, y) to track the path in 2D.
- **Depth (z)**: Estimated from bounding box size (area) or alternative depth approaches.

Technique	Pros	Cons
Bounding Box Scaling	Lightweight, straightforward	Less accurate if distance from camera varies
Monocular Depth Models	Potentially more accurate depth from a single view	Higher computational load, requires a trained model

Selected Implementation

1. **Calculate 2D Coordinates**
   - Find the bounding box center: (xcenter, ycenter).
2. **Estimate Depth**
   - Approximate **Depth** (**Z)** based on bounding box area or aspect ratio.
   - Optionally incorporate a simple calibration procedure: measure bounding box size at known distances to build a lookup table.
3. **Store Path Data**
   - Maintain a list or buffer of (x,y,z) for each user ID over time.

---

7. Path Smoothing

Air-writing can be noisy, especially for users with motor control challenges. Smoothing can enhance clarity before text inference.

Technique	Pros	Cons
Moving Average	Fast, easy to implement	Loses sharp details
Kalman Filter	Real-time, good for dynamic noise	Requires parameter tuning
LSTM/GRU (DL-based)	Learns complex spatio-temporal patterns	Higher computation cost
Savitzky-Golay Filter	Preserves features, polynomial-based	More complex than a moving average
Spline Interpolation	Finds best fit curve through points	Doesn't preserve features as well.

Selected Implementation / Workflow

1. **Kalman Filter**
    - Initialize the state: (x,y,z).
    - Each new bounding box update refines the prediction.
    - Tweak the process and measure noise matrices to account for typical user motions.
2. **Post-Processing**
    - **Spline Interpolation** or **Savitzky-Golay filter** on the final path to smooth edges while maintaining writing strokes' shape.

---

8. Text Inference

Convert smoothed 3D paths (or 2D with approximate depth) into text.

Model	Pros	Cons
LSTM/GRU	Lightweight, good for sequential data	Struggles with very long sequences
CRNN	Tailored for handwriting recognition	Less flexible for drastically varied input
Transformers	High accuracy, can handle global context	Higher computation and memory demands

Model	Pros	Cons
Hybrid Models	Combines spatial & temporal features	Increased complexity

Selected Implementation

1. **Data Representation**
    - Represent the path as a series of (x,y,z,t) points or a time-sequence of 2D images (if you choose a CNN-based approach).
2. **Model Choice**
    - A **CRNN** (Convolutional Recurrent Neural Network) is often effective for handwriting-style recognition.
    - Alternatively, a lightweight **LSTM** can be used if hardware constraints demand minimal overhead.
3. **Training**
    - Generate labeled data by capturing known air-written letters or words.
    - Pair each user's path data with the corresponding text label.
    - Train the model to predict the text sequence from the path sequence.

---

9. Visualization

- **Technology**: **Unity3D** for real-time rendering and VR compatibility.
- **Features**:
    1. Renders the user's 3D path in a scene.
    2. Displays recognized text in real-time.

Why Unity3D?	Pros	Cons
Easy VR integration	Streamlined development for VR (Oculus, etc.)	Additional learning curve
Cross-platform compatibility	Can deploy to Android, Windows, other platforms	May require performance optimization

Selected Implementation / Workflow

1. **Data Transfer**
   - From the Android app, send the smoothed path (and optional text) to Unity3D.
   - This can be done via local network sockets, or by integrating Unity as a library.
2. **Scene Rendering**
   - Create a minimal 3D environment in Unity.
   - Instantiate a line renderer or a mesh to visualize the real-time path.
3. **VR Integration**
   - Use **XR Interaction Toolkit** or vendor-specific SDKs (e.g., Oculus or SteamVR) to view the path in 3D.
   - Optionally attach controllers or gestures to allow user interaction with the rendered text or path.

Overall Workflow

1. **Video Input**: Real-time capture with the **Camera2 API**.
2. **Pre-Processing**: Normalize brightness (especially the green channel), reduce noise, and (optionally) crop ROI.
3. **Object Detection**: Use YOLO to detect the LED finger(s).
4. **User Identification**: Decode OOK signals or use alternative color-based methods to label each user.
5. **Object Continuity**: Track objects across frames with DeepSORT (or another multi-object tracker).
6. **Path Extraction**: Obtain (x,y,z)(x, y, z) from bounding box centers and scaled size.
7. **Path Smoothing**: Apply filters (e.g., Kalman) to reduce noise.
8. **Text Inference**: Translate the final smoothed paths into text using a suitable model (e.g., CRNN, LSTM).
9. **Visualization**: Display 3D strokes and recognized text in real-time through the Android app interface or a Unity3D VR scene.

Key Takeaways

- **Maintain Color Information**: Never discard color channels if you rely on LED color for tracking.
- **Accurate User Identification**: OOK signals or color-coding each glove is critical for multi-user scenarios.

- **Scalable Architecture**: Each module (detection, identification, tracking, smoothing, text inference) should be independently upgradable for future improvements.
- **Adaptive Smoothing**: Provide adjustable smoothing for users with different motor control levels. This customization can significantly improve user experience and accuracy.