Virtual Air Canvas with Hand Gesture Recognition

Air Canvas is an interactive computer vision application that allows users to draw in the air using hand gestures captured by a Raspberry Pi camera. The application tracks hand movements and specific finger gestures to create a virtual drawing experience without physical touch or keyboard interaction.

Core Components

1. Hand Detection Module (HandDetectorMP class)

This is a custom wrapper around Google's MediaPipe hand tracking solution:

- **Initialization**: Sets up parameters for detection sensitivity, tracking confidence thresholds, and number of hands to track
- **find_hands**: Processes each camera frame to detect hand positions
- **find_position**: Converts normalized coordinates to pixel coordinates and creates a list of landmark points
- **fingers_up**: Determines which fingers are extended using relative positions of landmarks:
    - Thumb is "up" if its tip is to the right of the previous joint
    - Other fingers are "up" if their tips are above their middle knuckles

2. Main Drawing Application

Setup and Configuration

- Defines canvas dimensions (Default: 1280×720 pixels with 125px header)
- Sets drawing parameters (color, brush thickness)
- Creates regions on screen for color selection and brush controls
- Loads overlay images for the header UI from specified folder

Camera Initialization

- Uses Picamera module 3 to interface with Raspberry Pi camera
- Configures for RGB capture (Default 1280×720)
- Uses Hailo for hardware-accelerated inference

Interaction System

The application implements a sophisticated gesture recognition system:

1. **Two-Finger Selection Mode** (index + middle finger up):
   - Selecting colors by touching the header area
   - Adjusting brush size with controls on right side
   - Visual feedback with rectangle between fingers
2. **One-Finger Drawing Mode** (only index finger up):
   - Drawing lines in selected color and thickness
   - Special eraser mode with wider brush and collision detection
   - Visual feedback with circle at fingertip
3. **Canvas Management**:
   - Stores strokes as tuples with position, color, thickness, timestamp
   - Auto-expires strokes after 30 seconds (Can be changed if necessary)
   - Implements intelligent eraser by detecting collisions between eraser movement and existing strokes
4. **Auto-Clear Feature**:
   - Tracks when hands are present/absent
   - Automatically clears canvas after 10 seconds of no hand detection
   - Shows countdown timer

Visual Pipeline

The program creates a layered display system:

1. Camera feed as background
2. Drawing canvas rendered on top
3. UI header with color options
4. Brush size controls on right side
5. Visual indicators for current mode

This is accomplished through:

- Separate canvas buffer for drawing
- Bitwise operations to combine layers
- Mask creation for proper blending

Keyboard Controls

- 'q': Quit application
- 'c': Cycle through color options
- 'x': Clear canvas
- '+'/'-': Increase/decrease brush thickness
- Number keys: Enter custom RGB color values

Technical Implementation Details

1. **Color Selection Regions**:
    - Precisely positioned rectangles in the header
    - Each associated with a specific color and overlay image
    - Visual feedback when selected
2. **Stroke Storage System**:
    - Each stroke stored as: (x1, y1, x2, y2, color, thickness, timestamp, is_eraser)
    - Two-pass rendering: first identify eraser strokes, then render visible strokes
    - Collision detection between eraser path and existing strokes
3. **Visual Rendering Pipeline**:
    - Creates canvas with current strokes
    - Converts to grayscale and threshold masks
    - Uses bitwise operations to composite with camera feed
    - Adds overlay header and UI elements
4. **Timing and Animation**:
    - Frame-by-frame processing in real-time
    - Stroke lifetime management with timestamps
    - Smooth transitions between modes