

Air Canvas Documentation

System Requirements

- Raspberry Pi with camera module
- Python 3.x
- OpenCV
- MediaPipe
- NumPy
- Picamera2 library
- Hailo hardware acceleration (optional)

Installation

Ensure all dependencies are installed:

1. `pip install opencv-python mediapipe numpy`

Set up Picamera2 for Raspberry Pi:

2. `sudo apt-get install python3-picamera2`
3. Place overlay images in the configured folder path (my path):
</home/cotadmin/Downloads/Interfaces>

To pull from github:

1. <https://github.com/Jsopaseuth/AirCanvas.git>

Or

2. `gh repo clone Jsopaseuth/AirCanvas`

To start/end program:

1. In command line change directory to where python file is stored (my path):
<cd/picamera2/examples/hailo>
2. To start the program call the file in the command line: `python3 AirCanvas.py`
3. To end the program while the “Canvas” frame is selected: Press ‘q’ on the keyboard

Configuration (AirConfig.py)

The application uses a centralized configuration module ([AirConfig.py](#)) to manage settings:

(See Configuration File tab for a more detailed descriptions)

Canvas Settings

- [CANVAS_WIDTH](#): Width of the canvas (default: 1280 pixels)
- [CANVAS_HEIGHT](#): Height of the canvas (default: 720 pixels)
- [HEADER_HEIGHT](#): Height of the UI header (default: 125 pixels)
- [STROKE_LIFETIME](#): Duration before strokes fade (default: 30 seconds)

Drawing Parameters

- [default_color](#): Initial drawing color (default: red)
- [default_brush_thickness](#): Initial brush thickness (default: 5 pixels)
- [eraser_brush_multiplier](#): Eraser size multiplier (default: 2x brush thickness)

Detection Settings

- [detection_confidence](#): Hand detection confidence threshold (default: 0.85)
- [tracking_confidence](#): Hand tracking confidence threshold (default: 0.5)
- [hand_timeout](#): Time before canvas auto-clears when no hand is detected (default: 10 seconds)

Color Definitions

- [RED_COLOR](#): Red in BGR format (0, 0, 255)
- [BLUE_COLOR](#): Blue in BGR format (255, 50, 10)
- [GREEN_COLOR](#): Green in BGR format (0, 255, 0)
- [ERASER_COLOR](#): Eraser color in BGR format (255, 192, 203)

UI Elements

- [color_regions](#): Defines “clickable” regions for color selection
- [brush_control_regions](#): Defines regions for brush size adjustment
- [overlay_paths](#): Maps color names to overlay image filenames
- [show_countdown](#): Enables/disables auto-clear countdown display

Features

Drawing Capabilities

- Real-time drawing using index finger movement
- Multiple color options (red, blue, green)
- Dedicated eraser tool
- Adjustable brush thickness

- Strokes automatically fade after x amount of time
- Auto-clear functionality when no hand is detected for x amount of time

User Interface

- Interactive header with color selection options
- Visual indicators for current drawing mode
- Brush size adjustment controls
- Visual feedback for current brush position
- Auto-clear countdown display

Controls

Hand Gestures

- **Selection Mode** (index finger + middle finger raised):
 - Move over header area to select colors
 - Move over brush controls to adjust brush size
- **Drawing Mode** (only index finger raised):
 - Move finger to draw on canvas
 - Strokes only appear below the header area
- **No Gesture** (no fingers raised or no hand detected):
 - Resets drawing position
 - Auto-clear after timeout period

Keyboard Controls

- **q**: Quit application
- **c**: Cycle through color options
- **x**: Clear canvas
- **+**: Increase brush thickness
- **-**: Decrease brush thickness
- Number keys (0-9): Enter custom RGB color values (nine digits total: three each for blue, green, and red)

Technical Implementation

Hand Detection

The [HandDetectorMP](#) class encapsulates MediaPipe's hand tracking functionality:

- Detects hands in camera frames
- Identifies hand landmarks (21 points on each hand)

- Determines which fingers are raised
- Converts normalized coordinates to pixel positions

Key methods:

- [find_hands\(\)](#): Detects and visualizes hand landmarks
- [find_position\(\)](#): Extracts landmark coordinates
- [fingers_up\(\)](#): Returns an array indicating which fingers are extended [0,0,0,0,0] when:
 - Thumb is up: index 0 = 1
 - Index finger is up: index 1 = 1
 - Middle finger is up: index 2 = 1
 - Ring finger is up: index 3 = 1
 - Little finger is up: index 4 = 1

Drawing Pipeline

1. **Capture**: Obtain frame from Raspberry Pi camera
2. **Process**: Detect hand and finger positions
3. **Interact**: Interpret gestures as drawing or UI commands
4. **Render**: Update canvas with new strokes or UI changes
5. **Display**: Combine camera feed, drawings, and UI elements

Stroke Management

Strokes are stored as tuples with the following information:

- Start coordinates ([x_start](#), [y_start](#))
- End coordinates ([x_end](#), [y_end](#))
- Color (BGR tuple)
- Thickness (pixel width)
- Timestamp (creation time)
- Eraser flag (boolean)

The application uses a two-pass rendering system:

1. First pass: Identifies eraser strokes and builds a list of active erasers
2. Second pass: Processes each normal stroke to:
 - Check if it has expired (based on [STROKE_LIFETIME](#))
 - Determine if it collides with any eraser strokes
 - Render only non-erased, non-expired strokes
 - Update the strokes list for the next frame

Eraser Functionality

The eraser implements a sophisticated collision detection system:

1. When in eraser mode, finger movements don't create visible strokes
2. Instead, they create invisible "eraser strokes" with a wider area of effect
3. For each existing stroke, the system:
 - Creates a bounding box around the eraser path (with margins)
 - Creates a bounding box around the existing stroke (with margins)
 - Uses rectangle intersection test to detect collisions
 - Removes strokes that intersect with the eraser path
4. This approach allows for precise erasing of specific strokes without affecting nearby content

The eraser provides visual feedback with a larger white circle showing the exact area being erased.

Customization

Adding New Colors

1. Add color definition to [AirConfig.py](#)
2. Add color to [color_options](#) list
3. Create corresponding overlay image
4. Add entry to [overlay_paths](#) dictionary
5. Add region definition to [color_regions](#) list

Modifying Behavior

- Adjust [STROKE_LIFETIME](#) to change how long strokes remain visible
- Modify [hand_timeout](#) to change auto-clear behavior
- Adjust detection and tracking confidence thresholds for different lighting conditions
- Set [show_countdown](#) to False to hide the auto-clear countdown
- Enable [debug_mode](#) for troubleshooting and development

Troubleshooting

Common Issues

- **Overlay Images Not Loading:**
 - Verify the correct path in [folder_path](#)
 - Check that image files exist and have correct permissions
 - The application will create a basic gray header if images aren't found
- **Hand Detection Issues:**

- Adjust lighting for better hand visibility
 - Try lowering [detection_confidence](#) if hands aren't being detected
 - Ensure hands are within camera frame
- **Drawing Performance:**
 - If the application runs slowly, consider reducing canvas resolution
 - Verify Hailo hardware acceleration is properly configured
 - Decrease [STROKE_LIFETIME](#) value

Debugging

Enable [debug_mode](#) in [AirConfig.py](#) to view detailed information about:

- Overlay image loading
- Color selection events
- Brush size changes
- Auto-clear operations