Deployment Manual for MotionInput 3.4 - EyeGaze

Welcome to the deployment manual for the EyeGaze Tracker, part of the MotionInput 3.4 project. This guide will walk you through the steps to deploy the EyeGaze Tracker project successfully after obtaining the source code. This hands-free mouse control solution uses eye movements captured via a standard webcam, employing computer vision techniques and the MediaPipe library to translate these movements into cursor navigation on your screen.

Table of contents

Welcome to the deployment manual for the EyeGaze Tracker, part of the MotionInput 3.4 project. This guide will walk you through the steps to deploy the EyeGaze Tracker project successfully after obtaining the source code. This hands-free mouse control solution uses eye movements captured via a standard webcam, employing computer vision techniques and the MediaPipe library to translate these movements into cursor navigation on your screen.

Prerequisites

- Features
- Requirements
- Installation
- Usage

Configuration
How it works
Classes and Methods
Configuration
Build from scratch

Prerequisites

Ensure you have Python version 3.10 or newer installed on your machine. The EyeGaze Tracker relies on several external libraries, including OpenCV, MediaPipe, and PyQt5. These dependencies can be installed through the package manager pip

Features

- Real-time eye tracking using the computer's camera
- Calibration process to map eye movements to screen coordinates
- Projective transformation for accurate cursor control
- Distance checking to ensure optimal user positioning
- Click control through winking or dwelling on a specific point
- Transparent overlay for visual feedback and user guidance

Requirements

- Python >= 3.10
- OpenCV (`opencv-python`)
- MediaPipe (`mediapipe`)
- PyQt5 (`PyQt5`)

Installation

1. Clone the repository:

```
git clone https://github.com/MotionInput/MI3.4_EyeGaze.git
cd MI3.4_EyeGaze
```

2. Create a virtual environment (optional but recommended):

```
python -m venv venv
source venv/bin/activate  # For Windows: venv\Scripts\activate
```

3. Install the required dependencies:

```
pip install -r requirements.txt
```

Usage

1. Run the main script:e for the launch

```
python main.py
```

2. Click "OK" in the start dialog to begin the Eye Gaze Tracker.
3. Follow the on-screen instructions for the distance check and calibration process.
4. Once calibration is complete, the Eye Gaze Tracker will start controlling the mouse cursor based on your eye movements.
5. Press the 'ESC' key to exit the program.

Configuration

The project's configuration settings can be adjusted in the `config.json` file. Some notable settings include:

- `debug`: Specifies whether to show debug information, default is false.

- `cursor_denoise_level`: Specifies the waiting time to take the average of mouse movements for smoother cursor control.
- `distance_varifier_time`: Specifies the time to allow users to see the distance checker.
- `calibration_point_duration`: Specifies the duration to show each calibration point.
- `calibration_point_color`: Specifies the color of the calibration point.
- `cursor_highlight_color`: Specifies the color of the cursor highlight.
- `cursor_highlight_radius`: Specifies the radius of the cursor highlight.

How it works

1. Eye tracking: The `EyeTracker` class uses the MediaPipe library to detect and track the user's eye landmarks from the camera feed. It calculates the average coordinates of the left and right eyes.
2. Calibration: The `Calibration` class handles the calibration process. It defines calibration points on the screen and guides the user to look at each point for a certain duration. It collects eye landmark data for each calibration point and calculates the average points to create calibration planes.
3. Projective transformation:
   The `ProjectiveTransformer` and `FourSurfaceProjectiveTransformer` classes are used to transform the eye coordinates from the camera space to the screen space. They use projective transformation to map the calibration planes to the corresponding screen regions.
4. Distance checking: The `DistanceChecker` class verifies the user's distance from the camera by analyzing the triangle formed by the left eye, right eye, and nose tip landmarks. It provides visual feedback to ensure the user is at an appropriate distance.
5. Click control (deprecated): The `ClickController` class handles click events based on user actions, such as winking or dwelling on a specific point for a certain duration.

6. Transparent overlay: The `TransparentWindow` class creates a transparent window overlay that displays visual elements like calibration points and the cursor highlight. It communicates with the main process using multiprocessing queues.
7. Window control: The `WindowControl` class provides utility functions to focus on specific windows, such as the Chrome browser or the control panel.
8. Main process: The `main` function orchestrates the overall flow of the program. It initializes the necessary components, runs the calibration process, and continuously tracks the user's eye movements to control the mouse cursor. It communicates with the control panel process using multiprocessing queues and events.

Classes and Methods

`Configuration`

- `__init__(self)`: Initializes the configuration object by loading settings from `config.json`.
- `__getattr__(self, name)`: Allows accessing configuration settings as attributes.

`DistanceChecker`

- `check_distance(self, frame, face_mesh, frame_w, frame_h)`: Checks the user's distance from the camera by analyzing facial landmarks.

`EyeTracker`

- `__init__(self, click_controller)`: Initializes the eye tracker with a click controller.
- `get_a_camera_image(self)`: Captures an image from the camera.
- `get_average_eye_coordinates(self, landmarks)`: Calculates the average coordinates of the left and right eyes.
- `get_eye_coordinates(self, frame)`: Detects and returns the eye coordinates from the camera frame.
- `track_eyes(self, transformer, frame)`: Tracks the user's eye movements and updates the cursor position.

`Calibration`

- `__init__(self)`: Initializes the calibration object with predefined calibration points.
- `calibrate(self, frame, landmark, screen_w, screen_h, queue)`: Performs the calibration process by collecting eye landmark data for each calibration point.
- `add_calibration_point(self, landmark)`: Adds a calibration point to the collected data.
- `complete_calibration(self, screen_w, screen_h)`: Calculates the projective transformation based on the collected calibration data.

`ProjectiveTransformer`

- `__init__(self, l: Quadrilateral, r: Quadrilateral)`: Initializes the projective transformer with source and destination quadrilaterals.
- `to_left(self, r_points)`: Transforms points from the right quadrilateral to the left quadrilateral.
- `to_right(self, l_points)`: Transforms points from the left quadrilateral to the right quadrilateral.

`FourSurfaceProjectiveTransformer`

- `__init__(self, l: [Quadrilateral], r: Quadrilateral)`: Initializes the four-surface projective transformer with source and destination quadrilaterals.
- `to_right(self, l_point)`: Transforms points from the left quadrilaterals to the right quadrilateral.
- `transform_ratio(self)`: Calculates the accuracy measure of the transformation.
- `screen_size_to_eye_pixel_ratio(self)`: Calculates the ratio between the screen size and the eye pixel size.

`WindowControl`

- `focus_chrome()`: Focuses on the Chrome browser window.
- `focus_control_panel()`: Focuses on the control panel window.
- `minimize_code_runner()`: Minimizes the code runner window.

`TransparentWindow`

- `__init__(self, window_name)`: Initializes the transparent window with the specified name.

- `initUI(self)`: Initializes the user interface of the transparent window.
- `display_marker(self)`: Displays the calibration marker on the window.
- `hide_marker(self)`: Hides the calibration marker.
- `paintEvent(self, event)`: Handles the painting of the window, including the calibration marker and cursor highlight.
- `animateMarker(self, x, y, size=100)`: Animates the calibration marker to the specified position and size.

`ApplicationOverlay`

- `__init__(self, paused_state, q)`: Initializes the application overlay with a paused state and a queue for communication.
- `on_press(self, key)`: Handles key press events.
- `on_move(self, x, y)`: Handles mouse movement events.
- `run(self)`: Runs the application overlay.

Build from scratch

```
python -m nuitka main.py  \
--standalone \
--output-dir=./dist \
--include-data-files=./config.json=./config.json \
--enable-plugin=pyside6 \
--user-plugin=plugins/mediapipe.py \
--windows-disable-console
```