**1. \_\_init\_\_**

This is the class constructor, responsible for initializing all necessary components and variables for the gaze-based mouse controller.

* **mp.solutions.face\_mesh.FaceMesh**: Initializes MediaPipe's face mesh for detecting facial landmarks.
* **pyttsx3.init()**: Initializes the text-to-speech engine for providing audio feedback.
* **pyautogui.size()**: Gets the screen width and height for proper mouse movement scaling.
* **Various Deque Buffers**: Used for smoothing gaze coordinates and storing gaze history.
* **Blink Detection Parameters**: Manages blink duration, long blinks, and triple blink actions.
* **Kalman Filter**: Used for smoothing iris movement predictions.
* **Heatmap and Visualizations**: Tracks gaze direction on a virtual heatmap for better feedback and visualization

**2. setup\_kalman\_filters**

This function sets up two Kalman filters for both left and right eye tracking, smoothing their gaze positions.

* **measurementMatrix**: Defines how measurements (iris positions) map to the state vector (position + velocity).
* **transitionMatrix**: Defines how the state evolves over time.
* **processNoiseCov**: Controls how much noise is expected in the system.
* **measurementNoiseCov**: Controls how much noise is expected from the sensor.

**3. apply\_kalman\_filter(x, y, kalman)**

Applies a Kalman filter to smooth the raw iris coordinates (x, y).

* **kalman.correct(measured)**: Corrects the current state estimate with a new measurement.
* **kalman.predict()**: Predicts the next state of the iris position based on the current state.

**4. get\_iris\_position(iris\_landmarks, image)**

Calculates the normalized iris position based on the detected landmarks.

* **mesh\_points**: Converts MediaPipe landmarks into pixel coordinates.
* **center**: Calculates the center of the iris.
* **normalized\_x, normalized\_y**: Returns the iris position as a fraction of the screen dimensions.

**5. smooth\_coordinates(x, y, smooth\_buffer, kalman)**

Smooths the raw iris coordinates using both a Kalman filter and a weighted moving average.

* **Kalman Filtering**: Reduces jitter by predicting smoother positions.
* **Weighted Moving Average**: Smooths the output further using historical gaze points stored in smooth\_buffer.

**6. calculate\_eye\_aspect\_ratio(eye\_upper, eye\_lower, landmarks, image)**

Calculates the **Eye Aspect Ratio (EAR)** for blink detection.

* **EAR Formula**: EAR=Vertical DistanceHorizontal Distance\text{EAR} = \frac{\text{Vertical Distance}}{\text{Horizontal Distance}}
* **upper\_mean & lower\_mean**: Calculate the average vertical positions of the upper and lower eyelids.
* **Used to detect if the eye is closed (low EAR) or open (high EAR).**

**7. detect\_blinks(landmarks, frame, current\_time)**

Detects blink patterns and determines if a specific action (e.g., click, pause) should be triggered.

* **both\_eyes\_closed**: Checks if both eyes are closed based on the EAR.
* **left\_eye\_only**: Checks if only the left eye is closed.
* **blink\_start\_time**: Tracks the start time of a blink.
* **Triple Blink**: Detects rapid triple blinks for a double-click action.
* **Long Blink**: Toggles between system pause and resume.

**8. handle\_mouse\_actions(action, current\_time)**

Executes mouse actions based on the detected blink patterns.

* **left\_click**: Single blink triggers a left-click.
* **right\_click**: Left-eye blink triggers a right-click.
* **double\_click**: Triple blink triggers a double-click.

**9. move\_mouse(gaze\_x, gaze\_y)**

Moves the mouse pointer based on the user's gaze direction.

* **gaze\_history**: Smooths the gaze direction over several frames to ensure stable movement.
* **distance\_from\_center**: Calculates how far the gaze is from the center of the screen.
* **Adaptive Speed**: Adjusts mouse speed based on the distance from the center of the screen.
* **Momentum & Friction**: Adds realistic movement physics for smoother mouse control.

**10. calculate\_adaptive\_speed(distance\_from\_center)**

Calculates the mouse speed based on the distance from the center of the screen.

* **speed\_factor**: Increases mouse speed proportionally to how far the gaze is from the screen center.
* **Returns a dynamic speed** that scales from a base speed to a maximum speed.

**11. speak\_feedback(text)**

Provides audio feedback using the text-to-speech engine.

* **speech\_cooldown**: Ensures that the system does not repeat the same feedback too frequently.

**12. run()**

The main loop that captures frames from the webcam, processes facial landmarks, detects blinks, tracks the gaze, and controls the mouse.

* **cap = cv2.VideoCapture(0)**: Opens the webcam for live video capture.
* **face\_mesh.process(rgb\_frame)**: Processes each frame to detect facial landmarks.
* **detect\_blinks()**: Checks for blink-based actions.
* **get\_iris\_position()**: Extracts gaze positions for both eyes.
* **smooth\_coordinates()**: Applies smoothing to the gaze positions.
* **move\_mouse()**: Moves the mouse pointer based on the smoothed gaze direction.
* **Visualization**: Draws various overlays on the video frame, such as iris positions, gaze direction, and blink indicators.

**Visualization Functions within run()**

1. **Draw Eye Contours**: Draws white outlines around both eyes.
2. **Draw Iris Centers**: Marks the centers of both irises with colored circles.
3. **Draw Gaze Direction**: Draws arrows to indicate the gaze direction.
4. **Blink Indicators**: Shows a bar that represents the current EAR for both eyes.
5. **Information Text**: Displays helpful instructions and real-time values on the screen.