Mouse and Gaze Tracking Application: Project Plan

ESTIA Gaze Project

2025-05-08

Mouse and Gaze Tracking Application: Project Plan

Summary

Aiming to develop an application that will track, analyze, and visualize how people use their mouse and where they look on their computer screen. This application will help us better understand and how people interact with digital interfaces by capturing both their physical interactions (mouse movements) and visual attention (where they're looking) to utilize in our main project.

App will be built using Python and its powerful libraries across different OS. This document outlines our plan for building this application, including the technical approach, features, development process, and timeline.

Project Vision

Being able to see exactly where a person is looking on their screen compared to where their mouse is moving. This powerful insight can reveal usability issues, identify points of confusion, and help create more intuitive digital experiences.

Our vision is to create a comprehensive tracking tool that:

- Records every movement, click, and scroll of the mouse
- Tracks where a person is looking on their screen
- Combines this information to reveal patterns and relationships
- Provides clear visualizations and reports
- Works on standard computers without requiring expensive specialized hardware

This tool will bridge the gap between what users see and what they do, providing unprecedented insight into human-computer interaction.

Overall Structure

The application will be organized into five main components:

1. **Mouse Tracking System**: This part will capture and record every mouse movement, click, and scroll action. It will work silently in the background, recording the exact coordinates of where the mouse moves along with timestamps for each action.

2. **Gaze Tracking System**: This component will track where the user is looking on the screen. We plan to support three different methods for tracking eye movements:

    - Using a standard webcam and computer vision technology
    - Connecting to specialized Tobii eye tracking hardware (for professional research)
    - Generating realistic simulated data (for testing and demonstration)

3. **Synchronization System**: This crucial component will connect the mouse and gaze data, matching them by timestamp to understand the relationship between where people look and where they click.

4. **Analysis Engine**: This part will process all the recorded data and generate meaningful insights. It will calculate statistics, create visualizations, and identify patterns in the data.

5. **User Interface**: This is the part users will interact with directly. It will provide simple controls to start and stop tracking, display real-time visualizations, and allow users to generate reports.

File Organization

We plan to organize the project files in a clear, logical structure:

```
MouseTracking/
├── mouse_tracker.py         # Core mouse tracking code
├── gaze_tracker.py          # Core gaze tracking code
├── sync_tracker.py          # Synchronization system
├── mouse_analytics.py       # Analysis and visualization
├── mouse_tracker_gui.py     # Basic interface for mouse tracking
├── sync_tracker_gui.py      # Advanced interface for combined tracking
├── run_sync_tracker.py      # Easy starter script
├── requirements.txt         # Software dependencies
├── Documentation files      # Various guides and explanations
├── models/                  # Directory for AI models
└── mouse_data/              # Directory for storing tracking data
```

Each file will have a specific purpose, making the code easier to understand, maintain, and extend in the future.

Detailed Feature Plan

Mouse Tracking Features

The mouse tracking system will capture comprehensive data about how users interact with their mouse:

Every movement of the mouse across the screen will be tracked with exact x,y coordinates. When users click any button (left, right, or middle), we'll record not only when and where they clicked, but also whether they pressed or released the button. Scroll wheel movements will also be tracked with direction and intensity.

From this raw data, our system will calculate useful metrics like the total distance the mouse has traveled, the speed of movement, acceleration patterns, and how long the cursor remains in certain areas (dwell time).

All of this information will be recorded with precise timestamps, allowing us to understand the sequence and timing of actions.

Gaze Tracking Features

The eye tracking system will work in three different modes to accommodate different needs and budgets:

Webcam Mode: Using the computer's built-in or external webcam, we'll employ computer vision techniques to: - Detect the user's face in the video feed - Locate their eyes using facial landmark detection - Track their pupils to estimate where they're looking - Map these estimates to screen coordinates

This approach won't be as precise as specialized hardware but will work with equipment most people already have.

Tobii Mode: For professional research requiring high accuracy, we'll integrate with Tobii eye tracking devices. These specialized tools use infrared technology to track eyes with high precision.

Dummy Mode: For testing or demonstrations without tracking hardware, we'll generate realistic simulated gaze data that mimics natural eye movement patterns.

In all modes, we'll identify key eye movements: - Fixations (when gaze stays in one place) - Saccades (rapid eye movements between fixations) - Blinks

We'll also record metrics like pupil size (when hardware permits) and assign confidence scores to our tracking data.

Synchronized Tracking Features

The true power of our application comes from combining mouse and gaze data. Our synchronization system will:

Match mouse and gaze events by their timestamps to create a unified timeline of interaction. For each moment, we'll calculate the distance between where the user is looking and where their mouse cursor is located. This will help identify whether users are looking at what they're clicking on.

We'll detect patterns of attention, such as whether eyes tend to lead the mouse (looking at a target before moving the cursor there) or follow it. The system will calculate an "attention match percentage" that shows how often a user's gaze and cursor are in the same area.

Analysis Features

Our analysis engine will turn raw tracking data into meaningful insights:

We'll generate heat maps that show where mouse movement and gaze attention concentrate on the screen. These visualizations make it easy to identify areas of focus and interest. Trajectory plots will show the path of both mouse movements and eye movements, helping to understand navigation patterns.

Distance plots will show how the gap between gaze and cursor changes over time, revealing coordination patterns. Statistical analysis will provide numerical insights into interaction behaviors. All of this will be compiled into comprehensive HTML reports with interactive visualizations that can be saved and shared.

User Interface Features

We'll create an intuitive interface that makes the system accessible to non-technical users:

Simple controls will allow users to start and stop tracking sessions with a single click. Real-time visualizations will show tracking data as it's being collected, providing immediate feedback. Users will be able to switch between different tracking modes through a simple dropdown menu.

A "Clean Session" feature will let users reset their data and start fresh without having to restart the application. When a session is complete, users can generate comprehensive reports with a single button click. Settings panels will allow for customization without overwhelming users with technical options.

Technical Implementation Plan

Programming Approach

We plan to build this application using Python, a versatile programming language well-suited for this type of project. Python offers several advantages: - It works consistently across different operating systems - It has excellent libraries for data processing and visualization - It can easily integrate with computer vision systems - It allows for rapid development and iteration

We'll use an object-oriented programming approach, organizing the code into logical classes and modules. This will make the code more maintainable and easier to extend with new features in the future.

Event-Driven Architecture

The tracking system will use what programmers call an "event-driven architecture." This means that rather than running in a fixed sequence, the program will respond to events as they happen:

1. When the user moves or clicks their mouse, the operating system generates an event
2. Our program will listen for these events, capture them, and process them
3. The processed events are stored in memory and periodically saved to files
4. The user interface updates based on these events

Data Storage Strategy

The application will save tracking data in two complementary formats:

CSV files (Comma-Separated Values) are simple text files that can be opened in spreadsheet programs like Excel. These make it easy for researchers to work with the data using familiar tools.

JSON files (JavaScript Object Notation) are more structured and preserve all the details of the data. These are ideal for further processing or analysis with programming tools.

Both formats will contain the same information, just organized differently for different purposes. The data will be saved to a designated folder on the user's computer, with filenames that include timestamps to keep sessions organized.

Computer Vision Techniques

For the webcam-based eye tracking, we'll implement several computer vision techniques:

First, we'll detect the user's face in the webcam image using a technique called "facial detection." Once we find the face, we'll locate specific points on it (particularly around the eyes) using a "facial landmark detector."

With the eye regions identified, we'll zoom in on them and use image processing techniques to find the dark centers of the pupils. By tracking how the pupils move relative to the eye corners, we can estimate where the person is looking.

This approach isn't as precise as specialized hardware, but it works surprisingly well for many applications and doesn't require any special equipment beyond a webcam.

Multithreaded Processing

To ensure the application runs smoothly without freezing or dropping data, we'll use a technique called "multithreading." This means the program will run several processes simultaneously:

- One thread will handle the user interface, keeping it responsive
- Another thread will track mouse events
- A third thread will process webcam frames or receive data from eye tracking hardware
- A background thread will periodically save data to files

These threads will communicate with each other in a coordinated way, sharing data when needed while preventing conflicts.

Data Visualization Techniques

We'll use a powerful data visualization library called Matplotlib to create informative visualizations:

For heat maps, we'll use a technique called "2D histogram binning" followed by Gaussian smoothing to create an intuitive representation of activity concentration.

Trajectory plots will use line and scatter plots to show the path of movements across the screen. Distance plots will use time-series visualization to show how mouse-gaze coordination changes over time.

All of these visualizations will be embedded directly in the user interface for real-time feedback, and also included in the generated reports.

Cross-Platform Compatibility Plan

We want our application to work well on all major computer operating systems, so we're planning specifically for compatibility with:

- Windows 10 and 11
- macOS (Catalina and newer, including Apple's new M1/M2/M3 computers)
- Linux (Ubuntu, Fedora, and other major distributions)

Each of these systems has some unique characteristics that we'll need to address:

Windows Considerations

On Windows, we'll use the Windows API through a library called pynput to capture mouse inputs. We'll use DirectShow for webcam access, which works well on Windows systems.

We'll also account for the way Windows handles display scaling on high-resolution screens, ensuring that our tracking coordinates match what the user actually sees. The

application will work well whether launched from Windows Terminal or the traditional Command Prompt.

macOS Considerations

Apple's macOS has specific security features that require special attention. We'll implement a system to guide users through granting the necessary permissions for input monitoring.

We'll account for the high resolution of Apple's Retina displays, ensuring tracking coordinates are accurate. For camera access, we'll use AVFoundation, which works well with macOS.

For Apple's newer computers with M1/M2/M3 chips, we'll ensure our code and dependencies work properly with their architecture.

Linux Considerations

Linux comes in many flavors and with different display servers (X11 or Wayland). Our application will detect which system is being used and adapt accordingly.

For webcam access, we'll use V4L2 (Video for Linux 2), which is widely supported. We'll also handle the permission systems of various desktop environments to ensure the application can access the inputs it needs.

Cross-Platform Implementation

To make sure our application works well across all these systems, we will:

- Use Python's platform-independent libraries whenever possible
- Include code that detects which operating system is being used and adapts accordingly
- Provide setup scripts tailored to each platform
- Handle file paths in a way that works across different systems
- Use relative paths within the application rather than hard-coded locations
- Include robust error handling for platform-specific issues

This careful approach will ensure that users have a consistent experience regardless of their operating system.

Tech Stack

We've carefully chosen the technologies and libraries we'll use to build this application:

Core Libraries

pynput will be used for mouse event monitoring. We selected this library because it works reliably across different operating systems and provides low-level access to input events.

OpenCV (Open Computer Vision) will handle our computer vision processing. This industry-standard library offers comprehensive tools for image processing and computer vision tasks.

dlib will provide face detection and facial landmark detection. This library includes high-accuracy pre-trained models that will help us locate and track eyes precisely.

NumPy will handle numerical operations and data manipulation. It's extremely efficient and forms the foundation of scientific computing in Python.

Matplotlib will create our data visualizations and plots. It offers comprehensive plotting capabilities that can be customized to our exact needs.

Pandas will help with data analysis and manipulation. Its powerful data structures are perfect for processing and analyzing our tracking data.

Tkinter will be used to create our graphical user interface. It comes standard with Python and works consistently across platforms.

Support Libraries

We'll also use several supporting libraries:

threading from the Python standard library will handle our multithreaded processing.

dataclasses will help us create clean, well-structured data representations.

logging will provide detailed application logs for debugging and auditing.

json and **csv** modules will handle data serialization and storage.

argparse will parse command-line arguments for users who prefer to launch the application from a terminal.

datetime will handle timestamp creation and processing.

We've selected these libraries based on several criteria:

- Reliability: All are stable, well-maintained packages with proven track records
- Performance: They're efficient enough for real-time processing
- Cross-platform support: They work consistently across Windows, macOS, and Linux
- Community support: They have active development communities and good documentation
- Licensing: All use compatible open-source licenses

Development Best Practices

We plan to follow several software development best practices throughout this project:

Code Organization

We'll organize our code to make it clear, maintainable, and extensible:

Each module will have a specific responsibility, making the code easier to understand and modify. Implementation details will be hidden within classes, exposing only what other parts of the system need to know.

Interfaces between components will be clean and focused, making it easier to modify one part without affecting others. Each class and function will be designed to do one thing well, rather than trying to handle multiple responsibilities.

Error Handling

Robust error handling will ensure the application works reliably even when things go wrong:

All operations that might fail (like opening camera feeds or saving files) will include error handling code that catches problems and responds appropriately. If preferred options aren't available (like a webcam), the system will fall back to alternatives (like dummy mode) rather than simply failing.

Users will receive clear error messages that explain what went wrong and suggest solutions. Detailed logs will be maintained for debugging and troubleshooting.

Resource Management

We'll carefully manage system resources to ensure the application runs efficiently:

Resources like file handles and camera connections will be properly closed when they're no longer needed. Large datasets will be processed in manageable chunks to avoid excessive memory usage.

All threads will be properly managed and joined when tracking stops to prevent resource leaks. Configuration settings will be stored in consistent locations and formats.

Testing and Validation

Our testing approach will ensure the application works reliably under various conditions:

The dummy mode will allow testing without specialized hardware dependencies. The application will handle unexpected conditions gracefully, degrading functionality rather than crashing.

User inputs will be validated to prevent errors from invalid data. The application will be tested across multiple operating systems to ensure consistent behavior.

Documentation

Comprehensive documentation will help both developers and users understand the system:

Code will include clear comments and explanations of purpose and usage. Python type annotations will improve clarity and provide better IDE support.

User guides will explain how to use the application effectively. This project plan and additional technical documentation will explain the system design and implementation.

Project Timeline

Proposed a phased development approach with the following timeline:

Phase 1: Foundation (Weeks 1)

- Set up development environment and project structure
- Implement basic mouse tracking functionality
- Create simple data storage and retrieval
- Develop minimal user interface for testing

Phase 2: Core Components (Weeks 2)

- Implement webcam-based gaze tracking
- Develop dummy mode for testing
- Create synchronization system
- Build basic data visualization

Phase 3: Enhanced Features (Weeks 3)

- Implement Tobii integration (for professional hardware)
- Enhance data analysis capabilities
- Create report generation system
- Improve visualizations

Phase 4: User Interface and Experience (Weeks 4)

- Develop comprehensive GUI
- Add real-time visualization
- Create settings and configuration panels
- Implement Clean Session feature

Phase 5: Testing and Refinement (Weeks 5)

- Cross-platform testing
- Performance optimization

- Bug fixing
- Documentation finalization

Phase 6: Deployment (Weeks 6)

- Package application for easy installation
- Create user guides and tutorials
- Final testing
- Release and distribution

Future Expansion Possibilities

While this project plan outlines a complete, functional system, there are several exciting ways the application could be expanded in the future:

Machine Learning Integration

We could incorporate machine learning to identify patterns in how people use their mouse and where they look. This could help predict attention and identify unusual interaction patterns that might indicate confusion or difficulty.

Advanced Visualization

Future versions could include 3D visualizations of interaction patterns, interactive data exploration tools, and side-by-side comparisons of multiple tracking sessions.

Mobile and Tablet Support

The technology could be adapted to work with touch interactions on mobile devices and tablets, potentially including stylus tracking correlated with gaze patterns.

Integration Capabilities

We could develop an API for other applications to access tracking data in real-time, a browser extension for web-specific tracking, or a software development kit for embedding these capabilities in other research tools.

Accessibility Features

The technology could be extended to support gaze-controlled cursor movement for people with motor impairments, attention-based interface adaptations, and other assistive technology research.

Conclusion

The proposed Mouse and Gaze Tracking Application represents a significant advancement in human-computer interaction research and usability testing. By capturing both physical

interactions through mouse tracking and visual attention through gaze tracking, the application will provide unprecedented insight into how people use digital interfaces.

The carefully designed architecture, thoughtful technology selections, and comprehensive feature set will ensure the application is powerful yet accessible to non-technical users. The cross-platform compatibility and multiple tracking modes will make it useful in a wide range of settings, from academic research to commercial usability testing.

We believe this project has the potential to significantly contribute to our understanding of human-computer interaction and help create more intuitive, user-friendly digital experiences.