Segmentation project

Zun Cao

4/21/2024

The goal of this project is to develop a solution for the eye fixation segmentation problem in eye-tracking data. Given a stream of x,y eye coordinates from a subject, the task is to identify fixation boundaries and map them to ground truth regions of interest (ROIs). Fixations are defined as periods where the subject's gaze is relatively stable, indicating visual attention on a particular point or region.

This project is very challenging for me, as I have never previously explored related topics or programming. I am also unsure about which methods to employ. After conducting some online research, I discovered a useful article: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9699548/. (Birawo, Birtukan, and Pawel Kasprowski. “Review and Evaluation of Eye Movement Event Detection Algorithms.” Sensors (Basel, Switzerland) vol. 22,22 8810. 15 Nov. 2022, doi:10.3390/s22228810)

In the article, I encountered several interesting methods that may be relevant to this project. These include: 4.2. Dispersion Threshold-Based Event Detection Methods, 4.3. Velocity Threshold-Based Methods, 4.4. Fixation and Saccade Detection with the Presence of Smooth Pursuit, and 4.5. Automated Velocity Threshold Data Driven Event Classification Method. Among these, the Velocity Threshold-Based Methods (Section 4.3) are relatively simple and straightforward to implement.

Velocity Threshold-Based Methods determine when the eye stops fixating on a point and begins moving to another by analyzing the velocity of eye movement coordinates. Specifically, when the eye movement velocity falls below a set threshold, it is considered that the eye is fixating, meaning the gaze is stable on a specific point. Conversely, when the eye movement velocity exceeds this threshold, it indicates a shift in gaze, signifying a saccade.

“To use this method, one must first calculate the velocity for each point in the eye-tracking data, usually by measuring the distance between consecutive coordinate points divided by the time difference. Then, an appropriate velocity threshold is set, which can be adjusted based on experimental data to best differentiate between fixations and saccades. Once the threshold is set, fixations and saccades in the eye movement data can be automatically detected, allowing for the analysis of the distribution and changes in visual attention.”

The advantage of this method is its simplicity and low computational demand, making it very suitable for beginners to quickly implement in eye-tracking analysis. Additionally, it offers an intuitive way to analyze visual behavior, especially in preliminary studies where rapid identification of eye movement patterns is needed.

Therefore, I chose the Velocity Threshold-Based Method as my primary approach to segment fixations. This method identifies fixation periods based on the velocity of eye movements, categorizing low-velocity intervals as fixations.

The first step is the data processing steps.

1. I began by loading eye-tracking data (child\_eye.txt) which contained timestamps, x, and y coordinates of pupil positions. **Input part**
2. Then converted complex timestamp formats into seconds for consistent processing.

**def parse\_timestamp(timestamp)**

1. Then computed the velocity between consecutive points using the Euclidean distance formula divided by the time difference. **def calculate\_velocity(data)**
2. Then applied a velocity threshold to identify when the eye movement was sufficiently slow to be considered a fixation.

I also encountered lots of difficulties:

1. Initially, converting timestamps posed a significant challenge due to their complex format. I had to refine the method to correctly parse these into a usable form.
2. Then I noticed that some segments had the same onset and offset times, indicating extremely short or non-existent fixations. To handle this, I implemented a filter to remove these entries, ensuring only meaningful fixations were considered.
3. Then I found that my initial onset and offset value is different from the ground truth value since its start from 1. Then I noticed the (“30 - N” seconds vs. “0 - N” seconds) in the instruction, so I add 29 seconds to each of them.(which is wrong after carefully reading the instruction again and I correct this later)
4. Finally, finding the optimal velocity threshold was tricky. I experimented with different values (initially set too high at 50 and too low at 10) and adjusted based on the distribution of calculated velocities and the proximity to ground truth data. In the end, I set the velocity threshold to 18.9(around 20 and I enter 20), for the sake of being close to the ground truth value.
5. There are also some small errors, such as OS error input file, but I solve that.

However, I also have questions that I have not solved already. There's a noticeable difference between the range of our processed data (30 to over 1300 seconds) and the ground truth data (30 to around 645 seconds). Why? (At this time, it’s close to the ground truth value but exceed the range, but I change the code later and it became not close to the truth value)

One consideration could be to truncate any data points occurring beyond 650 seconds to better align with the ground truth. This step could help in focusing the analysis on the most relevant segments of the eye-tracking data, potentially improving the accuracy and relevance of our fixation detection. I’m not sure about this.

Maybe that’s related to the extracted range? I modify my code according to the range of subject 1 [2525, 26317], but the data is more different than the ground truth value.

One day after. I found that I miss a column which is very important : recordFrameCount. So, I add this column and add two input() to let the user enter the extract range(because the range of subject1 and subject 2 are different).

Specifically, I create some new methods and parameters. However, even after adjusting the timestamp parsing and data range limitations, the generated timestamps did not meet expectations (originally starting at 30 seconds before adding the extract range, but actually ranging from about 80 to over 800 seconds now). (I’m not sure about what it means, and I regard it as starting from frame 2525+1=2526 to frame 26317.)

Although the current results might not have fully met expectations, I try my best and this project has taught me how to handle complex data, implement functional code, and attempt to solve real-world problems. Working with eye-tracking data, particularly dealing with complex timestamps and specific data ranges, has deepened my understanding of data processing and analysis in ways that textbooks and typical projects might not.