Towards Webcam-based Eye Tracking in the Eclipse IDE

Sebastian Lohmeier
Freie Universität Berlin and eCube GmbH
sl@monochromata.de

Abstract

Eye tracking is used in program comprehension research and is potentially useful for enhancing integrated development environments (IDEs). While eye tracking plug-ins exist for the Eclipse IDE, eye trackers do not typically work on operating systems other than Windows. To make empirical studies accessible to programmers working on Linux or MacOS, a recently presented webcam-based eye tracker is being adapted to the Java platform and provided as a plug-in for the Eclipse IDE. The eye tracker shall demonstrate eye tracking in the IDE without external devices to stimulate ideas of how to improve webcam-based eye tracking and how eye tracking could be used to improve program comprehension research and IDEs.

Introduction

Progress has been made in recent years in integrating eye trackers into the Eclipse IDE (Shaffer et al., 2015; Lohmeier, 2015) and iTrace of Shaffer et al. (2015) is now open source. Eye tracking requires an eye tracker, though. Looking at commercially-available remote eye trackers, they are either prohibitively expensive, come with restrictive licensing terms, or are only available for Windows. The situation might be due to a chicken and egg problem: Few eye trackers are sold (at high prices) while the eye tracking killer app is yet to be found and eye tracking is not used due to high prices, making it hard to imagine an eye tracking killer app. It is therefore desireable to strive for low-cost eye trackers – both to ease eye tracking studies in the field and to be able to support programmers' work via eye tracking.

Webcam-based eye tracking

Webcam-based eye tracking has recently drawn attention with eye trackers based on constrained local models (CLM) that use mouse and keyboard interaction for implicit calibration. This form of calibration seems well-suited for the use of eye tracking in programming where participants sit in front of the computer for a long time and mouse and keyboard are used frequently. WebGazer and PACE are examples of such eye trackers.

WebGazer is a JavaScript-based in-browser eye tracker that is implicitly calibrated instantaneously via relatively few mouse clicks and mouse movements (Papoutsaki et al., 2016). Its authors report an average error of at least 100px between gaze and corresponding click locations from a study of 82 participants with normal vision, contact lenses, and glasses who were free to move their heads and took less than 10 minutes per participant on average.

PACE is an eye tracker for desktop applications that uses a webcam and is calibrated using hundreds of mouse and keyboard events (Huang, Kwok, Ngai, Chan, & Leong, 2016). The authors report an average accuracy of 2.56° of visual angle from a study with 10 participants that created at least 1500 interaction events each during 4 hours on average. PACE is interesting because it is based on a careful study of user interaction that might be suitable for programming tasks as well.

Eclipse plug-in

Like Papoutsaki et al. (2016), the Eclipse plug-in uses `https://www.auduno.com/clmtrackr/` for face tracking. Because Clmtrackr is implemented in JavaScript, J2V8 is used: it embeds the V8 JavaScript engine of Google Chrome in Java. An SVM is trained using eye images and mouse coordinates to predict gaze after training. The plug-in works with glasses and contact lenses and is tested on

Linux, Mac OS, and Windows. It is a proof-of-concept so far and requires further development, before its tracking accuracy can be evaluated. It will also be necessary to consider Huang et al. (2016) to identify forms of mouse and keyboard interaction during programming that are suitable for training the eye tracker.

Psychology of programming

The Eclipse plug-in is motivated by work like Lohmeier (2016) that aims at a cognitive model of program comprehension. Records of eye movements are input into that model to construct one (out of many) plausible representations of a programmer's memory of source code. These representations can be used to predict knowledge-related comprehension difficulties of individual programmers – parts of the code that integrate less known information might be hard to comprehend. Both working memory and long-term memory might be covered. Collecting such records might be feasible with low frequency and low accuracy webcam-based eye trackers.

One of the key challenges for such applications will be to shape problems in such a way that (a) permits the error that the low accuracy of webcam-based eye trackers introduces and that (b) capitalizes on the easy availablility of webcam-based eye trackers. One of such problems might be the identification of potential referents of a referring expression: Instead of presenting an exhaustive list of all identifiers in a program that are equal to the referring expression, potential referents from the programmer's field of view might be presented.

It would also be interesting to interpret eye tracking data to identify potential comprehension difficulties. It is yet unclear whether webcam-based eye trackers will be able to deliver data fast and accurate enough for such tasks and in sufficient amounts, though.

Webcam-based eye tracking lowers the barrier for industry programmers to participate in studies that use eye tracking to study program comprehension. Another idea behind the eye tracker plug-in is to have software developers consider eye tracking and become creative with this new input device in order to see what kinds of scenarios eye tracking could be useful for and to reflect on what happens when they read and write source code.

Conclusion

The current work shows that porting the concept of webcam-based eye tracking to a Java IDE is feasible. The described eye tracker plug-in shall make eye tracking in the Eclipse IDE accessible on a variety of operating systems. It is yet to be shown whether the interactions between programmers and IDEs are suitable for implicit calibration of the eye tracker. If that is the case, webcam-based eye tracking potentially provides easy access to low accuracy eye tracking data to improve IDEs and facilitate program comprehension research.

References

Huang, M. X., Kwok, T. C., Ngai, G., Chan, S. C., & Leong, H. V. (2016). Building a personalized, auto-calibrating eye tracker from user interactions. In *CHI 2016* (pp. 5169–5179).

Lohmeier, S. (2015). *Experimental evaluation and modelling of the comprehension of indirect anaphors in a programming language. Version 1.3.* Retrieved 2017/05/15, from `http://monochromata.de/master_thesis/`

Lohmeier, S. (2016). A formal and a cognitive model of anaphors in Java. In *PPIG 2016* (pp. 32–35).

Papoutsaki, A., Sangkloy, P., Laskey, J., Daskalova, N., Huang, J., & Hays, J. (2016). Webgazer: Scalable webcam eye tracking using user interactions. In *IJCAI 2016* (pp. 3839–3845).

Shaffer, T. R., Wise, J. L., Walters, B. M., Müller, S. C., Falcone, M., & Sharif, B. (2015). iTrace: Enabling eye tracking on software artifacts within the IDE to support software engineering tasks. In *ESEC/FSE 2015* (pp. 954–957).