# SPLASH: U: PIXELDUST: Supporting Dynamic Area of Interest Tagging in Programming Studies with Eye Tracking

Jessica Cherayil
Wellesley College, Massachusetts, USA
jcherayi@wellesley.edu

## Abstract

Eye tracking studies are valuable for evaluating programming environments, but annotating what the programmer is looking at in a dynamic environment can be repetitive, time-consuming, and error prone. Through a participatory design exercise with two eye tracking researchers, I identified three significant challenges: search, extraction of code, and annotating transient objects. By applying computer vision algorithms to video traces, I developed a mixed-initiative system called PIXELDUST, which allows the researcher to train a system to recognize different objects on a screen. My preliminary results demonstrate the versatility of the approach; for example, the system can recognize return statements, method signatures, tool tips, and dialog boxes.

***Categories and Subject Descriptors*** D.2.2 [*Software Engineering*]: Design Tools and Techniques – User Interfaces

***Keywords*** AOI; eye tracking; tagging; static; dynamic; Eclipse

## 1. Motivation

Eye tracking studies are useful for investigating program comprehension, software tool usability, and programmer productivity, because they give researchers insight into participants' tasks passively, without need for explicit participant feedback. In eye tracking studies, the data recorded consists of pixel locations of participants' gazes, along with a capture of the screen video. Several analysis techniques, such as link analysis, require that these videos are divided into *areas of interest*, or AOIs. However, since tagging AOIs for analysis requires significant effort and time, some researchers design their studies such that the AOIs are static, meaning they stay in the same fixed coordinates on the screen throughout the entire eye tracking session.

Keeping AOIs static not only limits developers' ability to interact with the interface, but also simulates unrealistic conditions, since modern user interfaces are dynamic in nature. One example of a dynamic interface is the programming environment Eclipse, in which developers perform dynamic activities such as scrolling through lines of code and resizing dialog windows. Unfortunately for researchers, the support available to tag dynamic AOIs in programming environments like Eclipse is limited (Rodeghero et al. 2014).

To illustrate the difficulty in annotating AOIs in dynamic environments, consider the example of Abby, a Research Assistant. Given data from an eye tracking study conducted on software developers, Abby wants to know when each developer looked at the signature for the `toString` method. However, this signature can appear in several pixel locations on the screen, since the user can scroll the source code editor. As a result, Abby must traverse every frame in the video to search for these occurrences and note their positions and durations, which is incredibly labor intensive.

The *goal* of my research is to reduce the effort and time required for a person like Abby to tag eye tracking data. To achieve this goal, I developed PIXELDUST, which tags AOIs regardless of changes in position, size, or both. The prototype is currently able to recognize and log information about nine different AOIs in a matter of minutes. The *contribution* of my work demonstrates the effectiveness of computer vision techniques to support eye tracking research.

## 2. Background

**Instrumentation method.** One method of annotating videos is to instrument applications with a plug-in that records data as a task is completed (Shaffer et al. 2015). The instrumentation approach builds on previous methods in that it provides support for non-static applications, but it fails to capture gazes in non-instrumented areas. Therefore, all possible regions must be determined a priori. For example, prior studies require the participants use Eclipse exclusively. In order to improve this approach, the researcher would need

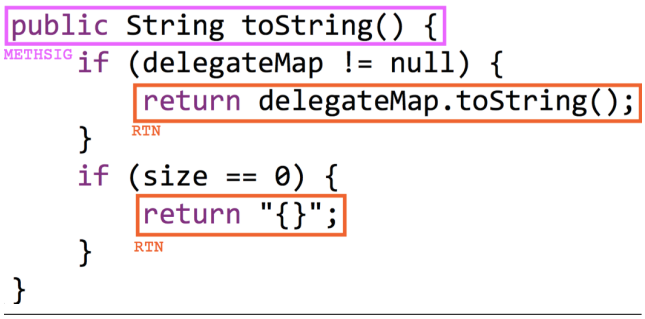


**Figure 1.** PIXELDUST is able to recognize AOIs at the line level, such as return statements and method signatures.

to similarly instrument other applications, such as browsers and text editors.

**Computer-aided method.** A second method is to defer video annotation until after data collection, but this can be limiting and ineffective. For instance, the Elan[1] video annotation tool supports annotations for a single frame, but has difficulty handling annotations that persist over multiple frames, called range annotations. A lack of support for range annotations is not practical for eye tracking studies, which frequently contain AOIs that last over multiple frames.

The annotation tool Vatic[2] recognizes the significant effort involved in annotation, and distributes this effort by crowdsourcing the task of video annotation through Amazon's Mechanical Turk.[3] Each video annotation is posted as a job that "Turkers" can accept and complete. However, this approach still requires each worker to step frame-by-frame through a video to annotate the AOIs (Vondrick et al. 2012). Furthermore, since each participant may label points differently, the researcher must reconcile implicit human variation in tagging. Lastly, by delegating the annotation task to others, researchers sacrifice valuable insight and familiarity with the data.

## 3. Approach

My approach involved four phases. First, I identified existing difficulties with eye tracking studies, through a participatory design exercise with two eye tracking experts. Second, I investigated various techniques to reduce these difficulties, including IDE instrumentation and computer vision. Third, I applied computer vision techniques, using the OpenCV[4] library, to reframe the generalized problem of computer vision and apply it to the domain of AOI tagging. Fourth, I adapted an image annotation tool, pylabelme[5], to operationalize this technique into a prototype tool, PIXELDUST.

PIXELDUST takes as input a video of participants' screens as they complete a task, as well as a set of *templates* – small screen captures of what a certain AOI might look like. The computer vision algorithms employed by PIXELDUST compare the template image across all pixels of each frame of the video. Using data mining to pick up on patterns in color or font, it reports the location where the template image corresponds the most to the video frame, making it possible to identify AOIs in different locations.

## 4. Results

I identified developer needs through two semi-structured interviews with researchers in eye tracking studies. Based on these participatory design exercises, I identified essential features needed for eye tracking and implemented the following four of these features:

**Search.** Locating and identifying an AOI in a video trace has proved to be a cumbersome task (Ponzanelli et al. 2016). One of our participants wanted to be able to distinguish between AOIs that appeared once within a video and those that appeared several times. PIXELDUST is able to do just that; as it traverses a video, it stores the *range* of frames on which an AOI appears (Figure 1). Just by parsing the output file, it is easy to tell which AOIs appeared only once and which appeared more than once. These ranges are useful in subsequent analysis.

**Line-Level recognition.** Our design exercises identified the need to recognize not only big objects, such as dialog boxes, but also fine-grained elements, such as a single line of code. Using image recognition made it possible to find AOIs of this size. After the user trains the tool just once, PIXELDUST is able to automatically recognize line-level elements and find AOIs such as return statements and method signatures (Figure 2). Recognizing lines of code is important, for example, in studies about developer navigation and control flow.

**Variable Object Tracking** Because PIXELDUST was created for dynamic interfaces, it is aware of not only a region on the screen, but what is in this region. Therefore, if an object moves from location A to location B, PIXELDUST is able to recognize that the object at A and the object at B are exactly the same object, just in different places on the screen. This is an improvement from tools created for static interfaces, which would be unable to pick up on different locations of the same object.

**Faster tagging.** Our data consisted of 50 developers each completing 10 tasks, with an average time of 1 minute per video. Factoring in 30 minutes of annotation time for each minute of data, it would take over 6 weeks to annotate this data. In comparison, PIXELDUST takes minutes to annotate this same data, since the researcher need only teach it the areas of interest once. Furthermore, unlike manual tagging, PIXELDUST is consistent in its labeling technique; if two objects have the same position and size, they will be labeled with the same coordinates. In comparison, manual annotations often differ by a few pixels due to human error.

[1] https://tla.mpi.nl/tools/tla-tools/elan/

[2] http://web.mit.edu/vondrick/vatic/

[3] https://www.mturk.com/

[4] http://opencv.org/

[5] https://github.com/mpitid/pylabelme

```
{
  "frames": [ "range(110,135)" ],
  "label": "Tooltip1",
  "fillColor": [ 255, 0, 0, 128 ],
  "lineColor": [ 0, 255, 0, 128 ],
  "times": [ "00:00:11-00:00:13" ],
  "points": [[836,546], [1437, 546],
             [1441,648], [836,650]],
  "videoName": "0007-scrn",
}
```

**Figure 2.** PIXELDUST writes information about each AOI to an output file.

## 5. Limitations

Despite the several advantages of computer vision, PIXELDUST is limited by its reliance on distinctive AOIs and its specificity. Because PIXELDUST identifies an AOI based on unique features in the image, such as color or text, AOIs without distinguishing characteristics are difficult to identify. For example, the close button icon (X) appears many times within an interface, which can confuse the identification algorithm.

In addition, using computer vision techniques for video annotation fails in the face of subtle differences in users' screens, such as varying color schemes in applications. For example, PIXELDUST recognizes a pre-defined set of AOIs that are all gray; therefore, if a participant's Eclipse interface was blue, PIXELDUST might not find any AOIs in the video trace at all. A third limitation is that we have only informally evaluated the tool.

## 6. Conclusion

The techniques used in this work, while specific to Eclipse, can be applied to any video data for user interfaces. Our results make it possible to conduct more realistic eye tracking studies, such as those that use dynamic interfaces. In addition, this work forms a foundation for future work in not only eye tracking annotation, but also video searching using computer vision. In future work, I intend to support a more general selection of AOIs via improved machine learning techniques, improve usability, and collect additional evaluations to measure the effectiveness and efficiency of PIXELDUST.

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1559593.

## References

L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza. Too long; didn't watch!: Extracting relevant fragments from software development video tutorials. In *ICSE*, pages 261–272, 2016.

P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello. Improving automated source code summarization via an eye-tracking study of programmers. In *ICSE*, pages 390–401, 2014.

T. R. Shaffer, J. L. Wise, B. M. Walters, S. C. Müller, M. Falcone, and B. Sharif. iTrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks. In *FSE*, pages 954–957, 2015.

C. Vondrick, D. Patterson, and D. Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, pages 1–21, 2012. ISSN 0920-5691. 10.1007/s11263-012-0564-1.