Using Eye Tracking and Attention Maps in Computer Science Education

Austin Edward Pernell

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
Engineering Science
Computer Science

University of Mississippi

2012

Abstract

Eye tracking has been used for many different areas of study where a user's eye movements are considered the most important information available. Before eye tracking was possible, the only way that information could be obtained was to ask the user where they looked and what caught their attention. Now, with the rise of eye tracking cameras, this data can be captured, stored, and processed in a meaningful way. These cameras aren't perfectly accurate however and there is still some interpretation that must be applied to more accurately represent the true gaze path of the user. This paper establishes some basic ideas to improve the accuracy when reading plain text as well as source code.

Dedication

This work is dedicated to my fianceé, Jessie Vincent.

Acknowledgments

I would like to thank Dr. Yixin Chen for his guidance in my research and for providing me with the idea for this project and research. His door was always open and he provided an excellent sounding board for all my ideas. I would also like to thank Ms. Cynthia Zickos and the students of CSCi 211 for their help in testing my project and providing valuable data for my research. My thanks to Dr. Jianxia Xue for the idea to implement a cursor to calculate accuracy. Also, my deepest appreciation goes to my fianceé, Jessie, who pushed me to finish strong. To my parents goes my gratitude for giving me every opportunity to succeed in life as well as the resources to do so. My brotherly love goes to my sister for challenging me to excel in my academics. And finally, I would like to thank all my professors here at Ole Miss for providing challenging and rewarding courses, as well as making me feel at home during my time here.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Eye Tracking Uses

Since the creation of eye tracking cameras, many new applications have arisen to take advantage of the ability to follow a user's gaze. The most basic of these applications are web usability studies. These studies involve tracking the gaze of the user while looking at a particular website and creating a heat map of that site which can be interpreted as the length of time the user gazed in a particular location. What can be inferred from the heat map is what the user actually read and what they ignored. The goal of web usability studies is to create web pages that minimize the amount of information that the user ignores. Other applications are more technical and the results must be interpreted by an outside source with knowledge on the subject. These types of studies are areas like differentiating between a novice and expert laproscopic surgeon based on eye gaze patterns Law *et al.* (2004). There are many applications in between that aid in ways that wouldn't be possible without an eye tracking camera, and this work looks to add a new application into that category.

1.2 Thesis Objective

The goal of this research is to incorporate basic eye tracking principles to aid Computer Science educators in giving more accurate assistance on an individual basis. This depends on the areas a student might not understand based on their gaze path while viewing the source code. This is done by creating a heat map in the form of a histogram to represent the length of time a user looked at each line in relation to the overall time looked at the code. Another map is also created which overlays the fitted gaze path onto the source code. In this map, the gazes are circles whose sizes are based on the length of time the user fixated on that point. The combination of these two maps allows the educator to determine if the gaze path and heat map don't match up to vital areas in the code in the instance where a student can not accurately describe what the particular block of code is trying to achieve.

The main problem with trying to come up with maps to represent the users gaze is that the data that is provided by the eye tracker is not related to the underlying information where the user is looking. Steps must be taken to determine what is actually being looked at for each piece of data that the camera provides. This presents a challenge because the cameras available today are subject to drift error and sub par accuracy for small fonts. In order to correct for these issues, steps were taken to overcome them and the results will be presented in this work.

1.3 Thesis Contributions

To solve the problem as mentioned above, using a larger screen as well as a larger font helped to correct the accuracy issue with smaller letter sizes. However, these steps are by no means a contribution to the realm of eye tracking. What is a contribution though is providing a model to follow for helping to more precisely determine the line that the user is reading. This is done by taking advantage of the fact that the way the English language is read, and therefore source code as well.

Another contribution this work makes is by describing a two pass median filter for gaze averaging. It is able to eliminate gazes that are considered too sparse to be accurate and improves performance over no filter by 17.95% on average.

Other measures are implemented to improve on accuracy and they will also be described later in this paper.

1.4 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 describes all the background information necessary to understand the key aspects in eye tracking. Chapter 3 lays out the other areas of eye tracking research that are related to this thesis. Chapter 4 documents the processes developed for this project and techniques used to refine those processes. Chapter 5 covers the testing process used and the results from those tests. Chapter 6 covers the conclusions and finally, Chapter 7 is the vita.

Chapter 2

Background

This chapter covers the background information needed to follow the eye tracking discussion. This will include the terminology used, as well as the basic concepts implemented to create the maps.

2.1 Equipment Used

For this research, a 24 inch ViewSonic VX2450wm-LED screen was used for display purposes. With a larger screen the font size of the source code was able to be increased so that when read, the accuracy could be increased without significantly reducing the number of lines available to display as would happen with a smaller screen. A 30 point font size eventually became the balance between size and accuracy, providing 25 rows of lines to be displayed at the same time, and it also displayed well over 90 columns of characters. With the default Linux terminal size being 80 x 24, albeit with the font size much smaller, the application for displaying code was at least proportional to the standard terminal size. Since programming convention usually imposes these size limits, the source code displayed on the screen didn't need to be modified between a normal size terminal and the output screen.

The camera used was a Mirametrix S2 Eye Tracker. This camera is a portable model that is able to be placed under the monitor being used. It works by using three infrared cameras, with one in the middle and one on each end which average the location of the eye. Once the software has been installed on the machine, a calibration sequence is able to be run; Mirametrix claims that accuracy is within the range of .5 to 1 degree. The camera also comes with a helpful API to develop custom applications and it is based on the open source eye gaze interface (Hennessey & Duchowski, 2010). This interface is based on a client-server architecture with the program created to take advantage of the camera being the client and the camera itself being the server. Prior to requesting data, the client sets variables based on the information needed from the camera. Each request and subsequent reply are in XML string format, and are sent over a TCP/IP connection. However, once data is requested to be sent, that changes to a UDP model where there is no acknowledgment of data received by the client, and it is continuously sent from the server regardless of packet loss.

There has been much work done on the lab setup by Pernice (2009) and the lab setting used for this research tried to follow as much of this as practical. The room used had overhead lighting and when testing, the blinds were closed to eliminate excessive light and glare which can lead to drift error. The user was given a chair that was able to be height adjusted to put the user in the optimal viewing window for the camera. It also had a reclining back, but was locked into place during use so that motion by the user wouldn't lead to any unnecessary head motion which could reduce the effectiveness of the camera.

2.2 Terminology

We will define a gaze simply as the computed position on the computer screen that the user is focusing. A saccade is defined as a small, jerky movement of an eye from one fixation to the next. An eye carriage return will be defined as a user's gaze that is moving from left to right and then returns to the left to start a new line.

2.3 Concepts

In eye tracking, the main goal is to translate raw eye movement data points into fixation locations (Salvucci & Goldberg, 2000). However, even while fixating, an eye still moves in small rapid motions. This presents a problem in distinguishing between the movements in a fixation and the movement of a saccade. Many cameras will distinguish between these two types of movement before they are sent, but in real time it is hard to interpret this data. So for this research, processing was done after the capture of all the data. Essentially the eye is doing one of two things, focusing or moving and usually the data where the eye is moving is disregarded as insignificant, but in this application, that information can be quite useful.

Chapter 3

Related Work

3.1 Comparison of other camera setups

There are many other setups that an educator could use which range from a similarly made portable bar camera like the one used in this research, to monitors with built in cameras, as well as headsets for the user to wear. Each camera works well in a particular setup, and each have their own degree of accuracy and precision. Gaze accuracy is defined as the average distance from the point being looked at on screen to the point that the camera returns. As mentioned before as with our Mirametrix camera, accuracy is measured in degrees. Gaze precision is the amount of variation between multiple trials of looking at the same point. This is the compactness of the calibration, and essentially, the smallest amount of pupil movement able to be detected. These concepts are further described in Figure 3.1.

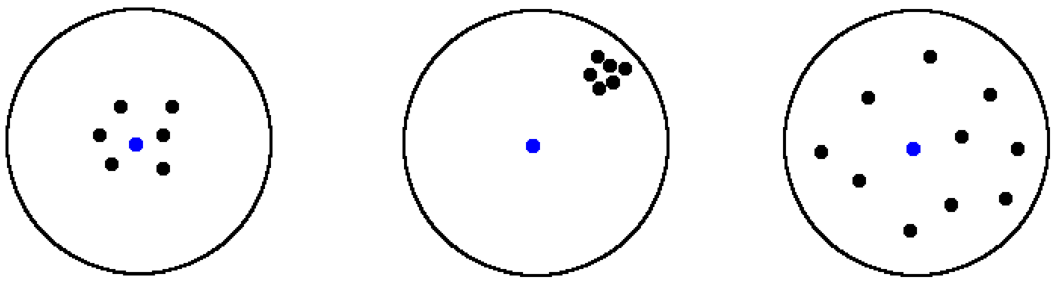


Figure 3.1. The circle on the left represents a gaze with high accuracy, but low precision. The circle in the middle represents a gaze with high precision, but low accuracy. The circle on the right is an example of an average gaze returned by the camera, with average accuracy and precision.

3.1.1 Tracking method

There are two types of pupil tracking, bright pupil and dark pupil, each having their own positives and negatives and ideal conditions. With bright pupil tracking, the eye is tracked based on the sharpness of the distinction between the pupil and the iris. The more contrast between the two colors, the easier it is to track the pupil. This works better for users who have lighter color irises and in darker environments. Dark pupil tracking works under the same premise, but it uses the infrared camera to change the color of the iris to a lighter shade, making the dark pupil stand out easier. This method works better in well lit environments as well as with natural lighting. It is important to make this distinction now, because it will play a key role later in Chapter 5 in the Limitations section.

3.1.2 Camera models

The SceneCamera from Arrington Research (Arrington, 2011) is a headset model built for freedom of body movement. The way this device works is by recording the scene viewed by the user though a small camera on top of the frames and relating eye location to what is being seen. This type of system is the only way to determine a user's gaze when a computer monitor is not the focal point. The SceneCamera tracks using both bright and dark pupil methods and can measure using the pupil only, corneal reflection, or both for enhanced accuracy and precision. The accuracy on the camera ranges from .25 to 1 degree, with .15 degrees of resolution, or precision. Arrington Research notes that better accuracy and precision numbers can be reported in ideal situations, but their numbers come from normal testing environments and are more realistic than under ideal conditions. They also state that small head movements are acceptable, as long as the pupil can be mapped to the image being viewed by the camera. This type of camera works well because it is only calibrated once per user, and its portability allows it to be used in many environments, regardless of lighting conditions. Its accuracy and portability are very comparable to non-headset models, and could compete in the lab setting as well.

A more widely known company in the eye tracking world is Tobii. Tobii produces many different types of cameras and their comparable camera to the one used in this research is the X1 Light Eye Tracker (Tobii, 2010). It is a portable bar model, able to be used in conjunction with any desktop or laptop monitor. It works by viewing the user's pupils and mapping that information in a calibration process shown on the screen being used. The X1 can be used with either bright or dark pupil methods, and a wide range of lighting conditions. Tobii tests under ideal conditions and are able to achieve a range of .2 to .8 degrees with a median of .4 degrees of accuracy and .15 to .35 degrees of precision with .21 degrees being the median precision. This compares well to the SceneCamera and comes with more restricted testing conditions and a lower cost, but higher overhead. Like with the rest of the models described, these cameras must be recalibrated for each new session with a user.

Tobii also makes a more precise model, the T60XL Tobii (2011). This is an integrated camera and 24 inch, 1080p widescreen monitor, using both bright and dark pupil tracking. It allows a range of head movement of 41w x 21h cm, with no depth movement without degradation of accuracy and precision. Again, under ideal conditions, this camera can achieve an accuracy range of .2 to .7 with the median being .4 degrees. The precision is where this model stands out as it reports precision of .09 degrees using only the raw data, but a precision of .03 degrees using the Stampe filter 2 (Stampe, 1993). The results achieved by this model are very tightly compacted gazes, with the accuracy being similar to those of the other models.

The Mirametrix S2 model that was used in this research is similarly comparable (Mirametrix, 2011). It is a portable bar model and works under the same calibration process as both the Tobii models. The main drawback is that it only uses the bright pupil tracking method and this will be discussed later in this work. Allowing a 25w x 11h x 30d cm range of head movement, Mirametrix claims a range of accuracy of .5 to 1 degrees and a precision of less than .3 degrees.

Each of these cameras have similar accuracy and precision numbers, with the T60XL standing above the rest in precision. Having that type of precision may have eliminated some of the processes described in this work, but no matter which camera was used, the accuracy metrics would still have needed to be put in place.

3.2 Similar Studies

3.2.1 Usability

The paper by Jacob (2003), describes the correlation between eye tracking and web usability, but it also lays the groundwork for defining eye tracking terminology and ideas. While usability is different than merely providing assistance based on a user's scan path, Jacob notes, "Difficulties relating eye tracking data to cognitive activity is probably the single most significant barrier to the greater inclusion of eye tracking in usability studies." While this statement holds true for many usability studies, this research takes ideas from those studies and applies them without regard to usability. The paper also states that it is still necessary to improve tracker accuracy and precision in order to get a firmer grasp on how to interpret usability results, and that is the goal of this work.

3.2.2 Code review and debugging

The focus of eye tracking in the realms of code review and debugging seek to reveal the technique differences between novices and experts. The studies focus less on the usability of the software used in concert with the eye tracking and more on the length of gazes on the areas of interest and scan path of the user between the source code and debugging tools. In the research done by Uwano *et al.* (2006), it is noted that if a user doing code review for debugging purposes doesn't spend the proper amount of time reviewing the source code, it will ultimately take them longer to find the errors. The more novice users tended to start with the debugger before reviewing the source code, and were not able to pinpoint the error without going back and scanning the source code. This shows that having an educator review a scan path of a student will be a viable way to improve the knowledge of novice programmers if they can't accurately debug a program or explain a concept used in the code being reviewed.

The work by Bednarik & Tukiainen (2008), deals with the same area and concludes that,"the retrospective relation of the eye tracking measures to the underlying processing is hard." He even admits that in his previous work, that deals with program comprehension (Bednarik & Tukiainen, 2006), that a new approach must be taken. The goal of this research isn't to define the relationship between gaze and cognitive processes, but to simply use the gaze information as an alternative way to determine deficincies in knowledge. Currently, the main way of uncovering these deficiencies is to have a student step through a section of code and for a given input, determine the output. The educator only sees a correct or incorrect answer with no knowledge of the reason for an incorrect answer. With the additional information of scan path and gaze length, if the scan path doesn't accurately follow the code, or a key area in the code is completely overlooked, this can benefit the educator in tailoring individual assistance based on areas deficient in the student's knowledge.

Chapter 4

Processes

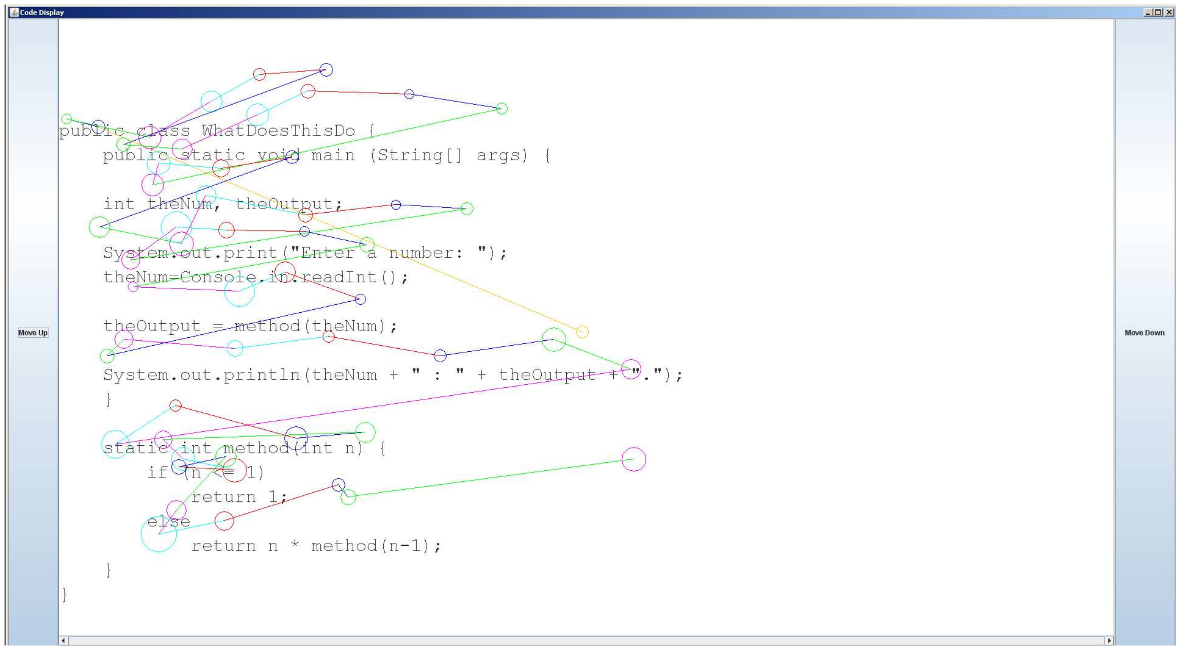
In this chapter the methods used and the ways those methods were refined will be discussed. This chapter is also the timeline of those events and how each discovery grew from the previous one.

4.1 Gaze path playback

The first step in being able to understand the data was to represent the data visually. In each line of data sent by the camera was a gaze ID which the camera increments each time it decides the user has focused on another point. Each unique gaze ID would be represented by a circle based on the new best gaze for that ID determined by the camera. Each gaze ID consisted of 1 or more lines which in essence, is the amount of time, or heat, the user focused on a particular point. Since each gaze ID had an independent number of lines, that information was applied visually by changing the size of each circle to correspond with the heat for that gaze ID. Each new gaze ID and representing circle was linked to the previous with a line denoting the saccade and a series of alternating colors was used to give the playback a path.

4.2 Picking a font size and style

Initially, the goal was to pick a font size that allowed the eye tracking data to be accurate on its own without any additional computation. For a calibration considered excellent by Mirametrix's standards, this size turned out to be 60 point font. This only allowed 13 lines



```
public class WhatDoesThisDo {
    public static void main (String[] args) {

    int theNum, theOutput;

    System.out.print("Enter a number: ");
    theNum=Console.in.readInt();

    theOutput = method(theNum);

    System.out.println(theNum + " : " + theOutput + ".");
    }

    static int method(int n) {
        if (n <= 1)
            return 1;
        else
            return n * method(n-1);
    }
}
```

Figure 4.1. Gaze path where the colored lines and circles are the raw camera data. The lines are the saccades and the circles are the gazes.

to be displayed at the same time, and much less than 80 columns, so many lines needed to be either line wrapped, or horizontally scrolled on the screen, effectively compounding the issue of solely using the font size to correct accuracy errors. With the lines needing to be wrapped, the number of unique lines was at a maximum of 13, and was usually less which was due to another programming convention of tabbing blocks of code which increases line length. When the scroll approach was used, there was so much eye movement toward the scroll bar that the noise it generated made the data almost unusable. At this point it was decided that wrapping would be used until the font size could either be decreased or another solution could be implemented to overcome this problem.

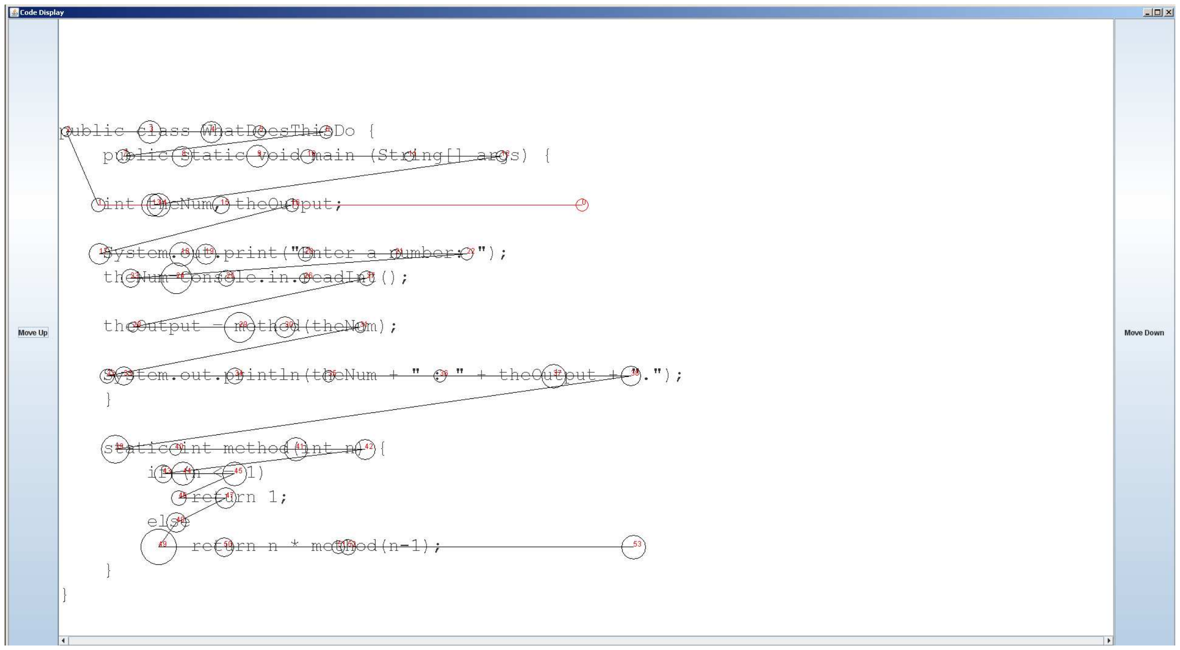
The style of the font that was used was Courier. It is a monospaced, or fixed-width font which is a traditional source code font believed to increase readability. It was used specifically in this instance though because of calculation purposes. In order to match the gaze of the user with a variable-width font, calculations would have to be done on each character on a line in order to determine width, and this would introduce unnecessary overhead. With a monospaced font, the length of a line can be calculated by finding the width at the start of the program and then by simply multiplying the number of characters by that width.

4.3 Assigning a line

Next came the task of trying to determine the line being looked at for a particular gaze. Even with the large font size of 60, assigning a line to each gaze wasn't inherently any easier due to imperfect calibration and drift error. At this point, only the line on the screen, which corresponded to the coordinates given in the data, was calculated. This consisted of taking each gaze ID's best gaze location and incrementing the count for the number of times that line was looked at by 1. This was not intended to represent the heat for a line, just the number of gazes calculated to be looking at a line. The assignment errors were based on the fact that the user would read every line in the source code and skip any lines that were blank. This could also be double checked by following the gaze path display and seeing where an assignment was missed. For a piece of source code with 13 lines, none of them wrapped, and 5 whitespace lines in the text the simple assignment got the line of each gaze correct 80% of the time. This was a decent preliminary number, and a good place to start from, but the accuracy needed to improve and the font size needed to come down.

4.3.1 Disregarding whitespace

One easy way to improve accuracy was to prevent the program from assigning heat to a line that was entirely whitespace or only had a single bracket on a line. The algorithm then had to decide where the next two closest lines were on the vertical axis. Once those were found, the shorter of the two distances were selected and that line got the credit for that particular point. This eliminated the errors where a blank, or almost blank line was receiving credit for a gaze. When in reality, it was the event where an accidental fixation was placed in the middle of a saccade between two actual fixations. Without having a definitive way of determining if the fixation was accidental, assigning it to the next closest line was



```
public class WhatDoesThisDo {
   public static void main (String[] args) {

      int theNum, theOutput;

      System.out.print("Enter a number: ");
      theNum = Console.in.readInt();

      theOutput = method(theNum);

      System.out.println(theNum + " : " + theOutput + ".");
   }

   static int method(int n){
      if (n <= 1)
         return 1;
      else
         return n * method(n-1);
   }
}
```

Figure 4.2. Gaze path where the black lines and circles are the fitted data with eye carriage return exploit. The raw data used for this fit is the same data used in figure 4.1. Each line is read only once and in order from top to bottom.

the most viable way of preventing an incorrect assignment.

4.4 Incorporating an eye carriage return

After visually double checking many trial runs for errors, a pattern soon became apparent. An eye carriage return, as previously defined, was noticeable on almost every line. This is when it was realized that an exploit could be made in the way the English language is read, from top to bottom and from left to right. Now, even where in source code, we may not always read from top to bottom, we do always read from left to right, so using that fact gave the algorithm a way to determine line separations. The focus now changed from assigning each gaze to a line, to assigning a string of average gazes to a particular line. Of course, this all had to be done after all the data had been taken, but with averaging not being a heavy computation, the results were still instant.

4.5 Readjusting the font size

Once the goal became line assignment over a series of averaged gazes, it was then possible to decrease font size while maintaining accuracy. In the beginning, it was necessary for the font size to be much larger because individual points, with no other reference to ones before or after it, were harder to assign. Then, once the eye carriage return exploit took advantage of that information, a more accurate guess could be made. While larger font sizes would provide a bigger range for accuracy, the problem was still trying to keep any lines from being wrapped, and getting more information on the screen without having to scroll vertically. As mentioned in Chapter 2, a font size of 30 rose to the top for its balance of size and accuracy. It was chosen because it was the largest font that could be used without showing less lines than a standard terminal.

4.6 Double returns

Now a new problem arose, where just like before, an accidental fixation after an eye carriage return would cause a single gaze to be represented as a line. However, where before there was no way of determining if it was accidental, it could now be determined that the gaze was a mistake in this instance. This conclusion came because after an eye carriage return, the eyes should next focus on the start of a line, which is the farthest left point of that line. If that gaze was followed by another gaze even farther to the left, the user was not reading anything at that point that could accurately be detected, and they were more likely just focusing for the purpose of finding the start of the next line. These type of accidental fixations rarely occurred, so omitting them only slightly positively affected the results.

4.7 Taking line length into account

Since it was determined that the algorithm would be basing a string of gazes on the line being looked at, another pattern arose. The last gaze before a carriage return was always close to the end of the line being looked at on a horizontal axis, not always the vertical.

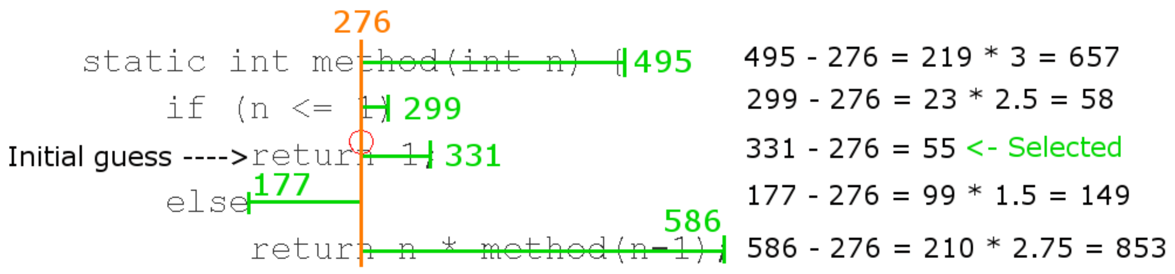


Figure 4.3. The red circle represents the last gaze before an eye carriage return. The orange number is its position on the horizontal axis. The green numbers are the ending positions on the horizontal axis. The formulas are shown on the right; line length - gaze position = difference * multiplier = weight.

This information could now be incorporated into the decision making process for a positive effect. For the line that was initially guessed, the two lines, whether valid or not, both above and below were taken into consideration for a total of five lines. For each, the distance was taken between the last point on that line, and the last actual gaze by the user. If a line was entirely whitespace and essentially invalid, then it was thrown out in the calculations, and not considered. Then each absolute value was taken and those became the starting points for comparison.

For the line initially guessed by the line average alone, no further calculations were done to that value. Its distance from the gaze was its weight. For the line two positions above the currently guessed line, its distance value was tripled. The idea behind this was that the error of the average should not be off by two lines worth of distance, but it should still be considered at a discounted value. For the line two positions below the currently guessed line, its distance value was multiplied by 2.75 times. The reason for the difference between two lines above and two lines below was again the reading principle that was exploited earlier. We tend to read from top to bottom, so the line two spots below should be discounted a little less because it is slightly more likely that the user is reading it, however it is still penalized because there should not be two lines worth of error in the system with proper calibration. Now, for the lines directly above or below the guessed line, the multipliers are 2.5 and 1.5 respectively. The reason for the whole point jump lower instead of a .25 decrease is again because of the exploit. The two most likely lines are the line initially guessed and the line below it, so they should be the most similar to their original distance values in order to have them selected more accurately.

All this means is that the algorithm is the most confident in the initially guessed line as represented by the average, but other options are also being considered, even if they are at a discounted rate. Once all the calculations are complete, the lowest weight represents the line that is the most probable to being the true line the user was looking at. It seems as if no other lines would ever be picked over the initial guess, however, with tabbed blocks of code, line lengths tend to vary from one line to the next, and that discrepancy became the key point in this heuristic.

If the initial line guessed is actually correct, then the distance between its length and the location of the last gaze should be very small. It would only be overturned if there was another line that matched the distance to the location even closer after discounting the distances. This situation almost occurs in Figure 4.3, but the initially guessed line has just a slightly lower weight after discounting. This would have been a false negative with respect to the initially guessed line had it not been selected. This rarely happens due to a number of different factors including line averaging, character size, line length, and the discounting process. What happens more frequently though is when the line guess is actually incorrect and the true line filters up to the top and is chosen. The same principle applies in the previous situation, it is just in reverse. The line incorrectly guessed will more than likely have a distance that is not similar to the gaze position, but the true line will have a distance that is very similar. With it being so similar, even by multiplying it by a weight doesn't increase the value to be larger than the incorrectly guessed line.

Taking the line length and respective distance to the gaze into consideration wouldn't make much of a difference if the lines were all relatively the same length like in a book, but source code is different. Each expression is normally placed on an separate line and with indentation conventions, lengths tend to vary enough to be exploited. This technique is a key rule in the algorithm, and works as long as the source code being viewed holds to those conventions. However, this technique doesn't always favorably help in the reverse case by taking the starting position into account. This is not because the information would not be useful, it is because of a possible camera limitation. If the user slowly reads to the end of one line, performs an eye carriage return, and begins reading immediately, the values sent by the camera tend to be much farther to the right than the true starting point of the user. This could be because of camera lag, or the way we read words and not individual letters. Regardless, comparing these starting positions in relation to the starting positions of the lines negatively affect the calculations because blocks of code are nested, and therefore, all start at the same tabbed position. The differences in indentation levels come from two lines where they are not tabbed to the same starting position. If the data sent is farther right than the line truly being viewed, that line would be discounted. And if a line in the same neighborhood started at that position, it would have a better chance of being selected on that fact. This would increase the number of false negatives, and therefore, it is not used in the current algorithm.

4.8 Adding weight to the next assignment

In addition to using a heuristic to adjust for line length, a separate heuristic can be used to further help the true line be selected. By keeping track of the last line that was selected, when the next selection comes around, the lines below that last line can have some distance taken off of their true values. This increases the line's chance to be selected. For instance, if the last line was nine, then more than likely, line ten will be the next one read, as long as it is valid. In order to account for the fact that line ten may very well be whitespace, by taking the last line and subtracting the initial, unmodified guess, the difference between the two lines is the result. If that number matches up to any of the lines being put through the length heuristic, a set number is subtracted given that is no larger than its current distance value. However, if it is, then it is automatically selected as then guessed line.

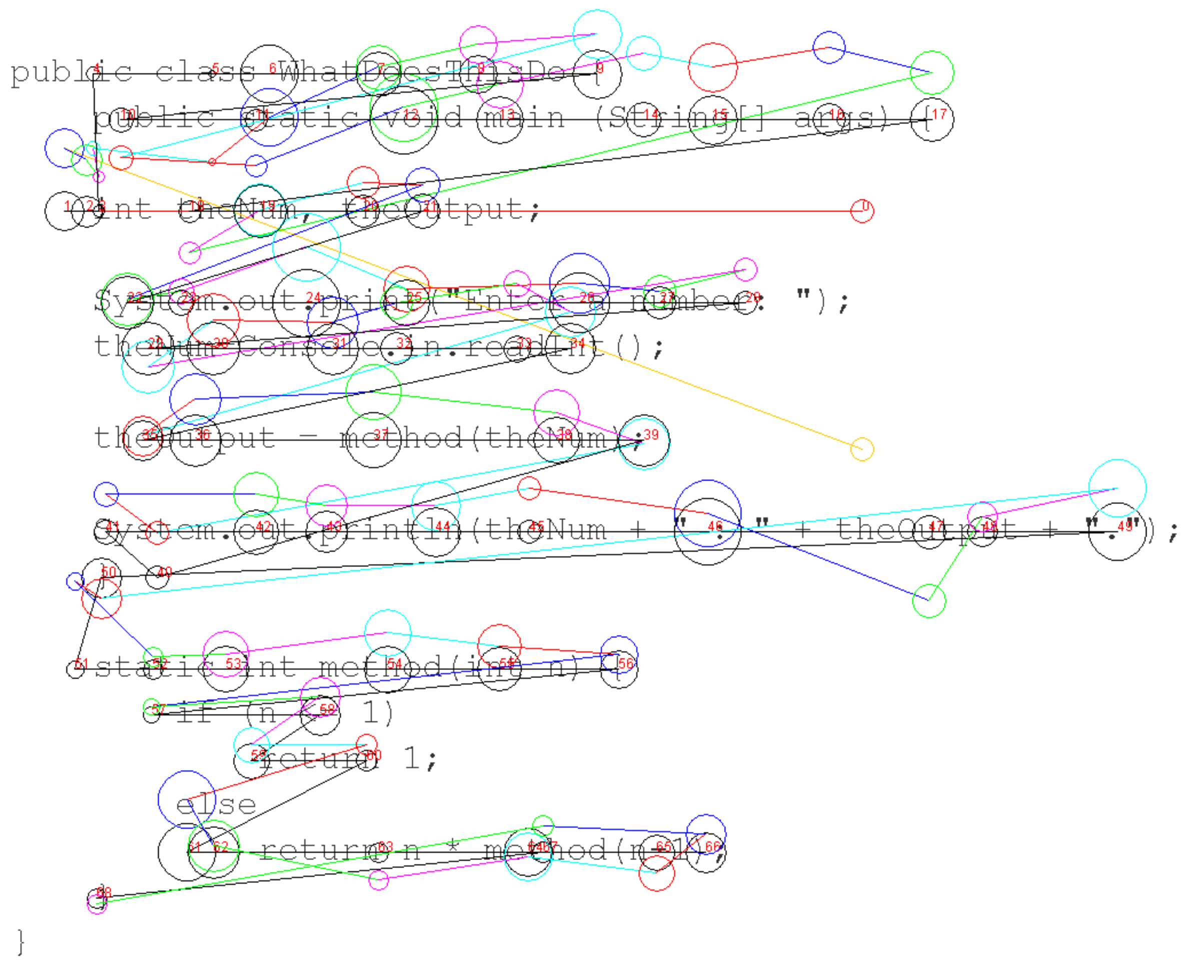
This heuristic can reinforce that the guessed line is the correct line, or it can bring light to the fact that it is more likely a line below that is the correct line even though it may have been penalized in the first line length heuristic. This isn't to say that it was incorrectly penalized, it is just due to the fact that this camera cannot be perfectly calibrated and calibration varies between each use and each user. The other lines in the region are considered but discounted, yet not at a rate that would automatically disqualify them. This weight subtracted based on the previous line selected only further allows the correct line to be chosen and doesn't increase the false negative rate by a substantial amount. These two heuristics only produce false negatives in extreme situations which can't be taken into account as normal occurrences.

4.9 Creating the heat histogram

Once these measures were put into place to ensure the correct line would be selected for a series of gazes, the next step was to take all of the assignment information and turn it into useful data. This data can be thought of as a histogram heat map. Each line in the source code is considered and for every time the user looks at that line, a counter is incremented by the length of the gaze upon that line. This gives a total count of where every single gaze was assigned. Then percentages are taken based on a line's heat and the overall time spent looking at the entire source code. When the percentages are displayed in a horizontal fashion, it is very obvious where the user spent the most time looking and this can be seen in Figure 4.5. This can help an educator point out areas to the student that may have been more vital to that code block if they weren't looking at the right group of lines or couldn't explain some of the concepts in the code.

4.10 Following a cursor

In the original test of assigning each individual gaze to a line, the program had no explicit way of knowing if its assignments were correct or incorrect. It took the user who was reading



```
public class WhatDoesThisDo {
   public static void main (String[] args) {

  int theNum, theOutput;

   System.out.print("Enter a number: ");
   theNum=Console.in.readInt();

   theOutput = method(theNum);

   System.out.println(theNum + " : " + theOutput + ".");


   static int method(int n) {
      if (n < 1)
         return 1;
      else
      return n * method(n-1);

}
```

Figure 4.4. Gaze path and fitting where the colored lines and circles are the raw camera data and the black lines and circles are the fitted data with various exploits.

```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 0.0  is: 0.0%
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 198.0  is: 8.4%       ---------
Line 6 heat: 285.0  is: 12.09%     -------------
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 240.0  is: 10.18%     -----------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 274.0  is: 11.62%    ------------
Line 11 heat: 227.0  is: 9.63%     ----------
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 203.0  is: 8.61%     ---------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 345.0  is: 14.63%    ---------------
Line 16 heat: 38.0  is: 1.61%      --
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 189.0  is: 8.02%     ---------
Line 19 heat: 68.0  is: 2.88%      ---
Line 20 heat: 70.0  is: 2.97%      ---
Line 21 heat: 0.0  is: 0.0%
Line 22 heat: 212.0  is: 8.99%     ---------
Line 23 heat: 9.0  is: 0.38%       -
```

Figure 4.5. Sample heat map for the gaze path sample in figure 4.4.

the source code to state where they were looking at each point given by the camera. There needed to be a way to leave all the ground truth with the program so the error rate could be automatically calculated and displayed. It was decided the best way to do this was a cursor. The source code would be displayed and a circle would float directly on top of each row for the user to follow. This way, the program would know exactly where the user should be looking as well as where they were actually looking. On the first implementation, the cursor size was chosen as an arbitrary number of 50 pixels in diameter. It was selected by drawing different sized circles around the characters and selecting a number that encompassed those characters without too much overlap on top of adjacent characters.

4.10.1 Averaging the points within a gaze

Initially, accuracy was poor for a fixed cursor size and it became necessary to average all the points within that gaze. The way the camera works is that it determines the best point of gaze for each unique gaze as the first point separated from the last gaze by a saccade. This doesn't always represent the true center of a gaze, so by averaging, a more stable center can be used in comparisons and calculations. This improved the accuracy, on average 10.12%, which meant that the camera was returning a best point of gaze that was very close to the center of that gaze, but it wasn't always correct. The change in the center, or best point of gaze, can be seen in Figure 4.7.

4.10.2 Variable cursor size technique

At this point, testing on a fixed cursor size, changing anything in the algorithm only resulted in finding out if those changes worked for that specific cursor size. It became necessary to test on multiple sizes, and that was the next step in the process. Instead of rerunning the algorithm for each cursor size, the data was reused and the diameter of the cursor was changed by the algorithm in each iteration. This generated a curve that determined the accuracy based on cursor size. This allowed the changes made to be seen in an overall view, instead of a localized one.

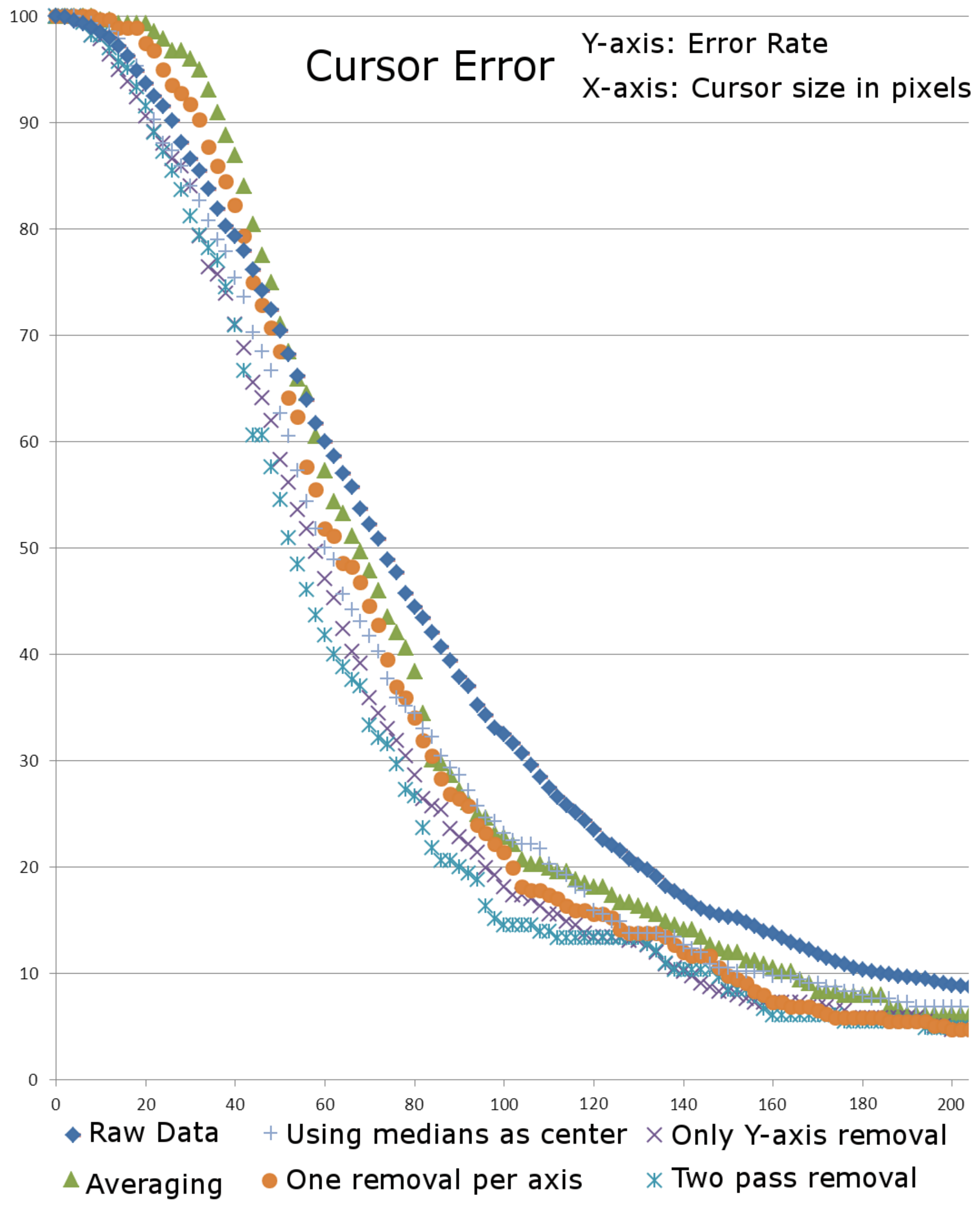


Figure 4.6. Error Rate in Respect to Cursor Diameter Size

4.10.3 Removing outliers

Another way to improve accuracy was to remove the data within a gaze that the camera sent as valid, because some of those points were actually outliers. In the XML string sent by the camera, a flag is included that determines whether or not the camera thinks the point is valid. After inspecting a group of lines that it set as either valid or invalid, it became clear that the camera wasn't accurately determining validity. Some lines were duplicated yet had different validity flag settings. The data sent was sometimes off by a line or two and it could not be determined if that was a problem that resulted in poor timing in the sending of the data, or an issue with the camera itself. Therefore, the algorithm had to accept everything as valid, but make the determination whether or not a point was valid instead of relying on the camera for that information. Multiple techniques were implemented to pick outliers, and each had their own strengths and weaknesses, but one eventually rose to the top.

4.10.4 Only vertical outlier removal

The first step was to test a one pass outlier removal on the vertical axis. The camera results as displayed on the screen showed much more vertical noise than horizontal, so that is why the vertical axis was chosen. First, for each gaze ID, all the data was ordered by the vertical axis and the median was chosen. Now, with the font size still being 30, this gave a row height of 40 pixels, so the bar set for being an outlier was a radius of 20. This is because a point in the middle of the character is selected, anything greater or less than 20 pixels from that point on the vertical axis became another character. Essentially, this made all the data points that were kept, within a character distance of each other on the vertical axis. This tightened up the average a good amount, by 14.16% on average from the raw data alone and by 4.04% over averaging the points of the gaze together, but if the median wasn't near the cursor center on the vertical axis, this wasn't going to correct the error rate significantly. This would turn out to be the second best performer, even though it was the first test that was tried.

4.10.5 One removal from each axis

The next step was to test how well a two pass removal would work, removing one data point from each axis, starting with the vertical pass and then moving to the horizontal pass. Instead of removing everything over a set bar, exactly one outlier was taken in each pass regardless. The value that was taken was the furthest from the median on each respective axis. The problem was, that if the true median on the horizontal axis was considered an outlier on the vertical axis and it was removed, it shifted the true center of the gaze. This test only slightly improved the error rate, by about 1.18% over the averaging heuristic, and it was in instances where there were true outliers that skewed the average enough to generate an error.

4.10.6 Using the median values as the center

Trying to approach the issue of accidental shifting of the center from a different angle, the next test was to keep only the median values on each axis as the center values, with no averaging needed. This technique didn't work as well in this research as it might have in others because the points for each gaze ID were sometimes very spread out. If the point selected as the true center by the median values was actually an outlier itself, there was no way to remove it, and this is why averaging helped to find the center in a much more stable way. The curve produced by this technique very closely followed the curve for averaging alone, and there were some instances where it was slightly better and slightly worse. It improved averaging by about 0.13% and improved over the raw calculation by 9.31%, on average. The reason for it being either slightly better, or even slightly worse was that in the cases it correctly picked the center, outliers were removed, and the results improved. In the case where it incorrectly picked the center, and was actually an outlier, it added noise and therefore errors to the results, making them worse than averaging would have. Even so, it was a useful test to run to figure out that the points in some gaze IDs were very sparse and this could now be used to improve the results even further.

4.10.7 Set distance from median removal

Upon seeing that some gazes tended to be sparse, and not good candidates for being valid, a method to remove these points was needed. Taking from the results of vertical removal with a bar, and coupling it into a two pass removal, the result was better than only a one pass removal. At this point, the bar was still a size of 20, but having previously automated the size of the cursor, fitting the size of the optimal bar in the same way was simple. It turned out that the highest accuracy curve came when the bar was of size 3 pixels and this was tested against multiple samples of data from multiple users. It improved over the raw data by 17.95% and over the averaged heuristic by 8.3% on average.

The way the two pass system worked was by ordering all of the points based on the vertical coordinate and marking every distance that was farther than 3 pixels from the median. Then the points were re-ordered by the horizontal axis and the same test was performed. After both tests had been run, each point that had been marked was removed and then the average was taken with the remaining points. In many cases, the pass on the vertical coordinate would mark the median on the horizontal axis and the pass on the horizontal course would mark the median of the vertical axis, and in both passes each would mark every other point for that gaze leaving no points to be averaged. This is how the sparsely grouped gazes were considered invalid. If not enough of the points were in a tightly compacted group, then the algorithm would remove them from being tested. If there were still points in a gaze after the outliers were removed, they would be very compact and give a good representation of the center of that gaze and be more likely to be closer to the center of the cursor than in any other technique used.

4.10.8 Variable cursor size results

After each technique had been plotted with the variable cursor it was shown that in order to get a letter by letter accuracy of 85%, a circle with a diameter of 2.5 times the height of the 40 pixel tall, 30 point font character had to be used. This reiterated the point that

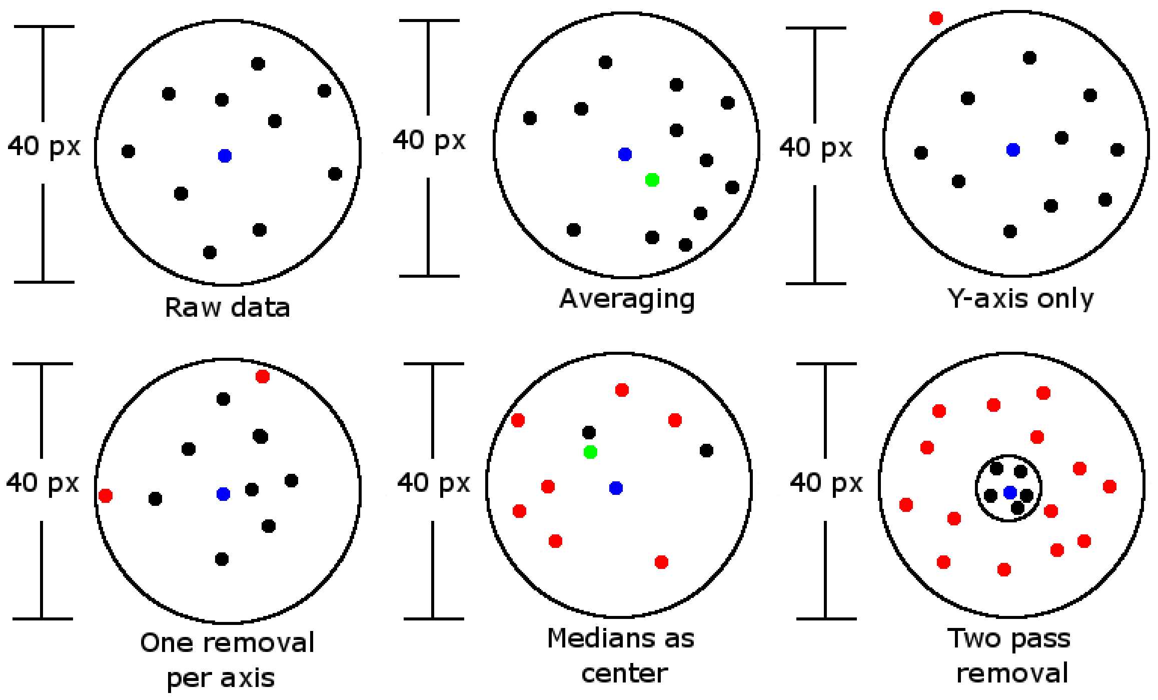


Figure 4.7. Examples of outlier removal. Black points are data that were kept. Red are data that were removed. Blue is the best point of gaze as reported by the camera. Green is the new best point of gaze as computed by each removal heuristic, if applicable.

either word, or in the case of this particular research, line accuracy was a more achievable goal.

4.11 Scrolling

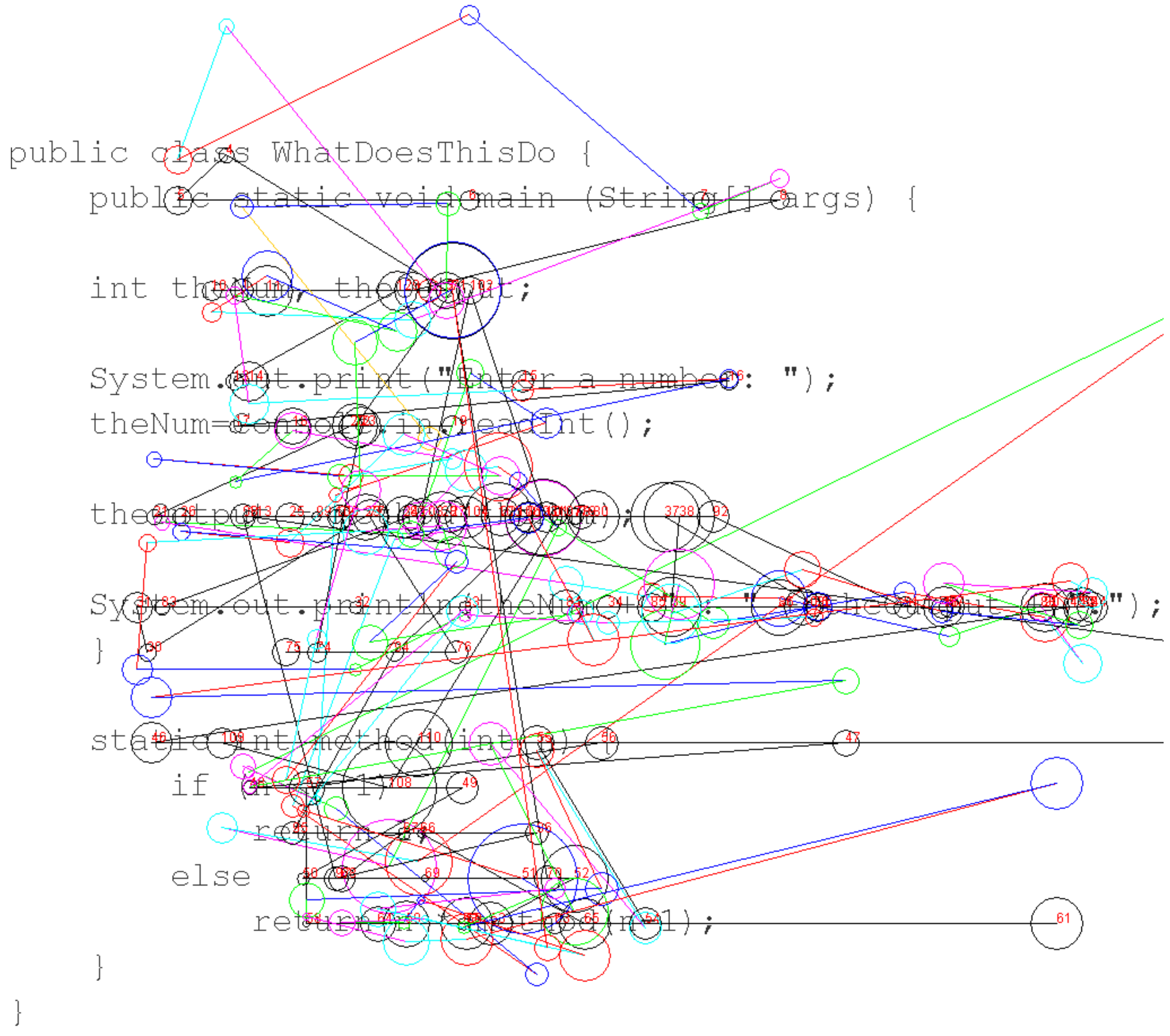
In the instances of scrolling, either vertically or horizontally, the algorithm was set to act as if an eye carriage return had been done by the user. This took all of the data up until that point and averaged it as if the user had begun looking at a new line. In order to scroll vertically, the user had to click a large button on either side of the code that ran the entire length of the screen. So to do that, they would usually look once to the button they wished to use, and then refocus on the code. Each time the button was clicked, the code would move by exactly one line, and the algorithm would assign the current gaze to a line being shown on the screen. Since the user was focusing on the code as it scrolled up in order to find what they were searching for, these gazes circles were very small and were accurately placed onto the corresponding line. Scrolling horizontally worked in the same fashion, but it occurred much less frequently due to the code not normally extending past 80 characters.

Chapter 5

Testing and Results

5.1 Testing

Since the goal of this project was to create a program to aid Computer Science educators in tailoring assistance on a case by case basis, the obvious testing group was students currently enrolled in courses on programming. Students in upper-level Java (who were also enrolled in the laboratory) were emailed and asked to participate in the research. They were given extra credit on a laboratory assignment if they agreed to help. Of the 15 enrolled in the course at the time, 6 agreed and scheduled times to test the system. On arrival, it was explained to the students how the testing would proceed. They would calibrate the camera as many times as necessary to get a good initial setup and then they would be shown the source code. The first few runs with the source code would be done with the cursor, and we would proceed after they felt comfortable with the speed of the cursor. Then, the next run would be reading every line from top to bottom with no cursor, skipping any whitespace. The last run the students were instructed to read the code, again with no cursor, with the intent of describing what the code was intended to do, after they had completed reading it. They were told to read just as they would in a debugging situation, trying to find an error in the logic. This test was only to see how students actually read code, not to determine any accuracy numbers. The following figures are examples of the students reading the code, with no cursor, to be able to describe what the code does.

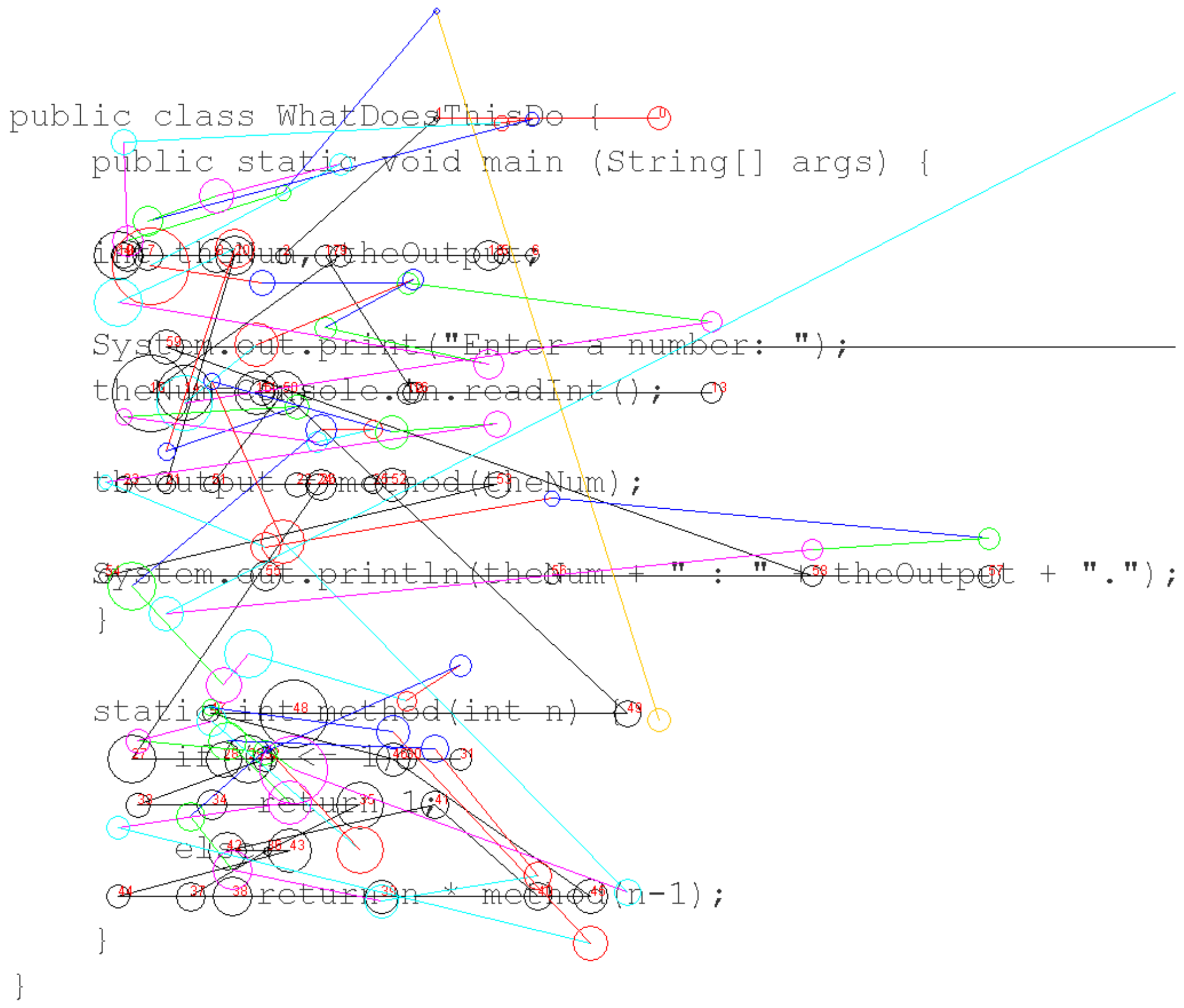


```
public class WhatDoesThisDo {
    public static void main (String[] args) {

    int theNum, theOutput;

    System.out.print("Enter a number:  ");
    theNum=Console.in.readInt();

    theOutput=method(theNum);

    System.out.println(theNum+": "+theOutput+"");
    }

    static int method(int n) {
        if (n <= 1)
            return 1;
        else
            return n * method(n-1);
    }
}
```

Figure 5.1. Gaze path for Student 1.

```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 0.0  is: 0.0%
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 26.0  is: 0.79%       -
Line 6 heat: 83.0  is: 2.52%       ---
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 280.0  is: 8.48%      ---------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 85.0  is: 2.58%      ---
Line 11 heat: 142.0  is: 4.3%      -----
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 912.0  is: 27.64%    ----------------------------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 642.0  is: 19.45%    --------------------
Line 16 heat: 127.0  is: 3.85%     ----
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 198.0  is: 6.0%      ------
Line 19 heat: 76.0  is: 2.3%       ---
Line 20 heat: 79.0  is: 2.39%      ---
Line 21 heat: 337.0  is: 10.21%    -----------
Line 22 heat: 313.0  is: 9.48%     ----------
Line 23 heat: 0.0  is: 0.0%
Line 24 heat: 0.0  is: 0.0%
```

Figure 5.2. Heat map for the gaze path for Student 1 in figure 5.1.

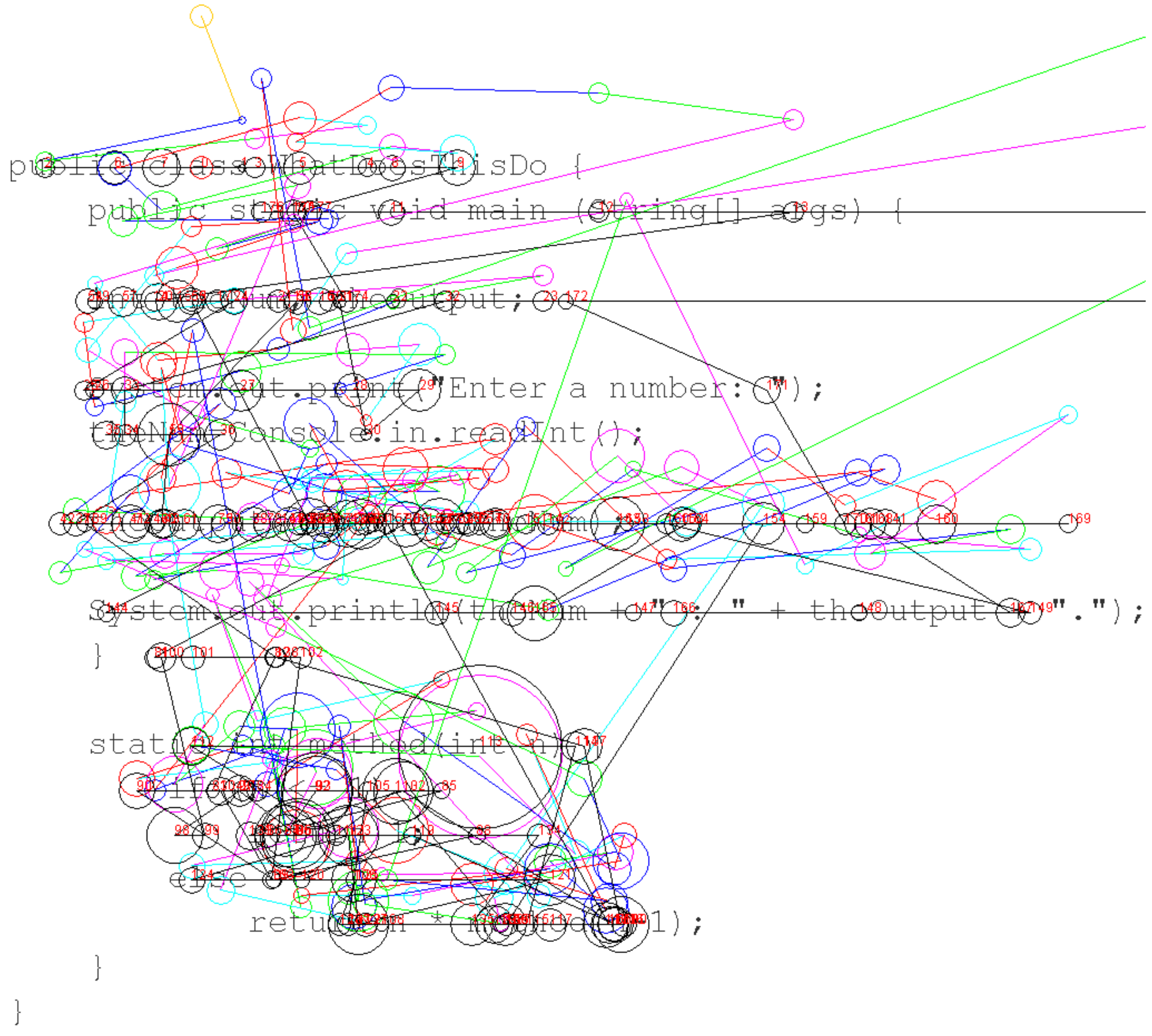


```
public class WhatDoesThisDo {
    public static void main (String[] args) {

        int theNum, theOutput;

        System.out.print("Enter a number: ");
        theNum = Console.in.readInt();

        theOutput = method(theNum);

        System.out.println(theNum + " : " + theOutput + ".");
    }

    static int method(int n) {
      if (n <= 1)
        return 1;
      else
        return n * method(n-1);
    }
}
```

Figure 5.3. Gaze path for Student 2.

```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 0.0  is: 0.0%
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 19.0  is: 1.19%       --
Line 6 heat: 0.0  is: 0.0%
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 345.0  is: 21.67%     ----------------------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 34.0  is: 2.14%      ---
Line 11 heat: 204.0  is: 12.81%    -------------
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 211.0  is: 13.25%    --------------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 111.0  is: 6.97%     -------
Line 16 heat: 0.0  is: 0.0%
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 123.0  is: 7.73%     --------
Line 19 heat: 172.0  is: 10.8%     -----------
Line 20 heat: 111.0  is: 6.97%     -------
Line 21 heat: 86.0  is: 5.4%       ------
Line 22 heat: 176.0  is: 11.06%    ------------
Line 23 heat: 0.0  is: 0.0%
Line 24 heat: 0.0  is: 0.0%
```

Figure 5.4. Heat map for the gaze path for Student 2 in figure 5.3.

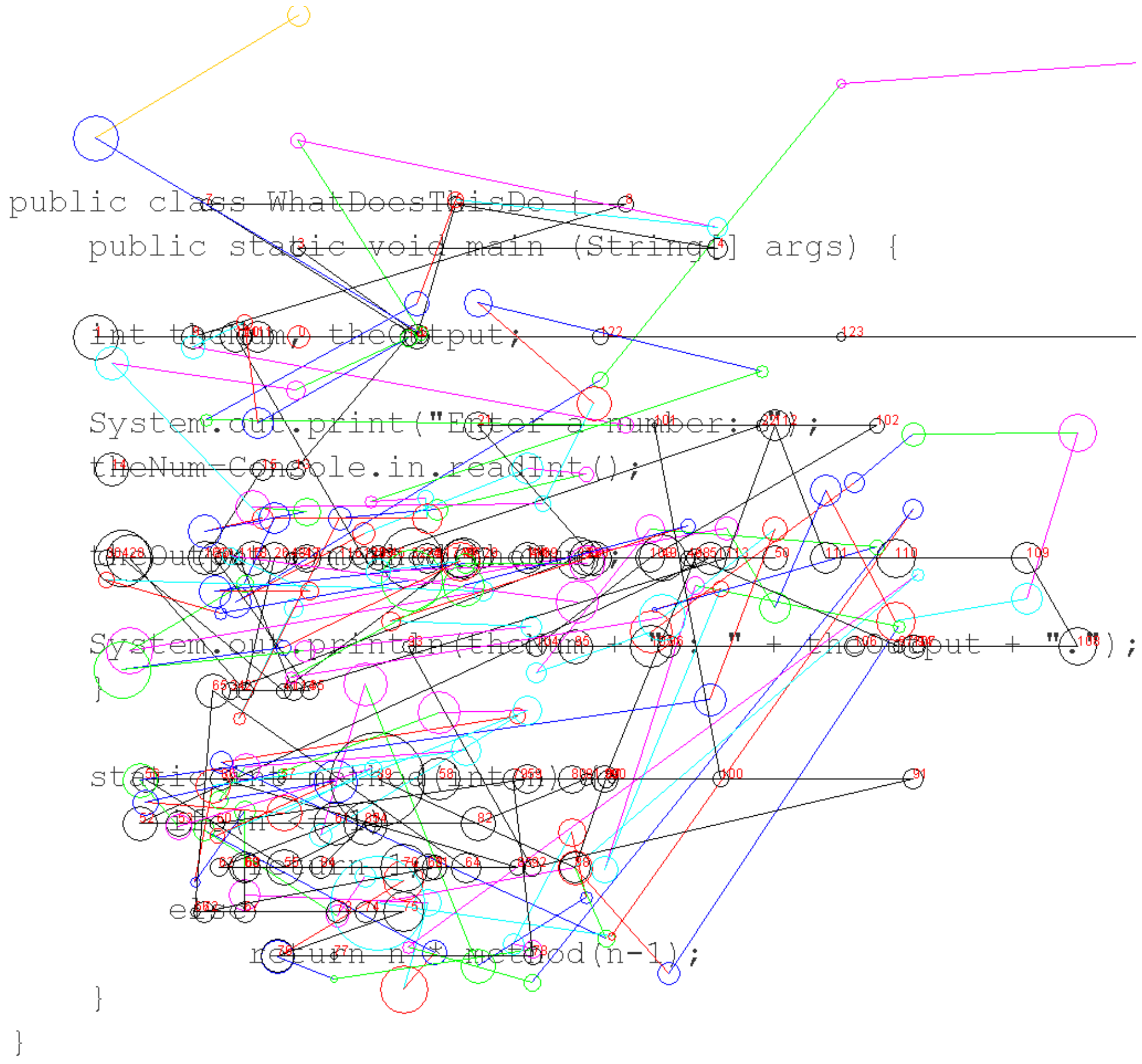


```
public class WhatDoesThisDo {
    public static void main (String[] args) {
        int theNum, theOutput;
        System.out.print("Enter a number: ");
        theNum=Console.in.readInt();
        theOutput = mymethod(theNum);
        System.out.println(theNum + " : " + theOutput + ".");
    }
    static int mymethod(int n) {
        if (n <= 1)
            return 1;
        else
            return n * mymethod(n-1);
    }
}
```

Figure 5.5. Gaze path for Student 3.

```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 0.0  is: 0.0%
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 231.0  is: 4.77%      -----
Line 6 heat: 129.0  is: 2.66%      ---
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 516.0  is: 10.66%     -----------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 170.0  is: 3.51%     ----
Line 11 heat: 117.0  is: 2.42%     ---
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 1501.0  is: 31.01%   --------------------------------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 196.0  is: 4.05%     -----
Line 16 heat: 106.0  is: 2.19%     ---
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 214.0  is: 4.42%     -----
Line 19 heat: 420.0  is: 8.68%     ---------
Line 20 heat: 414.0  is: 8.55%     ---------
Line 21 heat: 222.0  is: 4.59%     -----
Line 22 heat: 605.0  is: 12.5%     -------------
Line 23 heat: 0.0  is: 0.0%
Line 24 heat: 0.0  is: 0.0%
```

Figure 5.6. Heat map for the gaze path for Student 3 in figure 5.5.

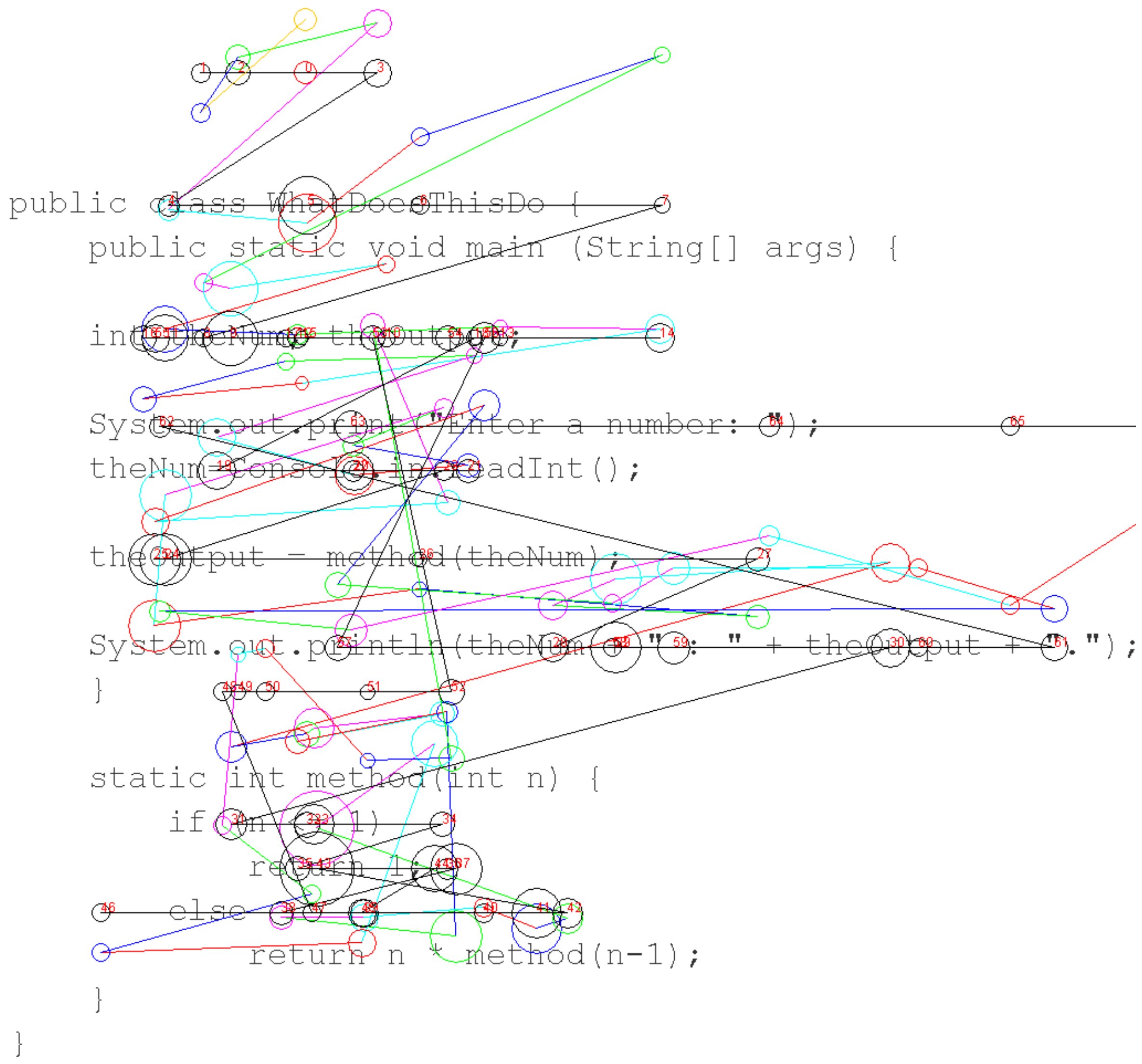


```
public class WhatDoesThisDo {
    public static void main (String[] args) {

    int theNum, theOutput;

    System.out.print("Enter a number: ");
    theNum=Console.in.readInt();

    theOutput=method(theNum);

    System.out.println(theNum + "! : " + theOutput + ".");
    }

    static int method(int n){
        if(n <= 1)
            return 1;
        else
            return n * method(n-1);
    }
}
```

Figure 5.7. Gaze path for Student 4.

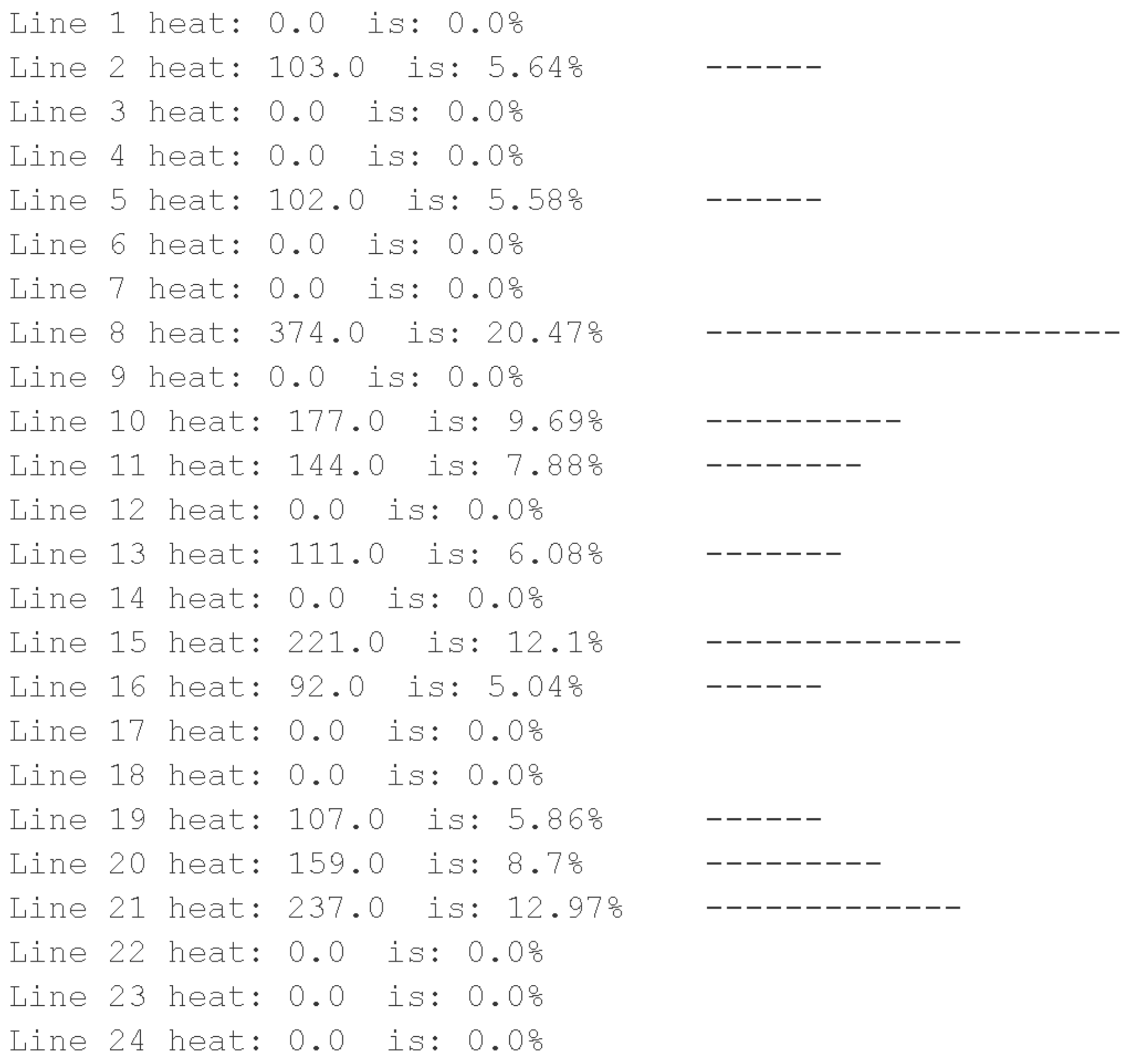
```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 0.0  is: 0.0%
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 60.0  is: 2.05%       ---
Line 6 heat: 34.0  is: 1.16%       --
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 208.0  is: 7.1%       --------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 87.0  is: 2.97%      ---
Line 11 heat: 82.0  is: 2.8%       ---
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 1042.0  is: 35.56%   ------------------------------------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 238.0  is: 8.12%     ---------
Line 16 heat: 164.0  is: 5.6%      ------
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 266.0  is: 9.08%     ----------
Line 19 heat: 191.0  is: 6.52%     -------
Line 20 heat: 356.0  is: 12.15%    -------------
Line 21 heat: 158.0  is: 5.39%     ------
Line 22 heat: 44.0  is: 1.5%       --
Line 23 heat: 0.0  is: 0.0%
Line 24 heat: 0.0  is: 0.0%
```

Figure 5.8. Heat map for the gaze path for Student 4 in figure 5.7.



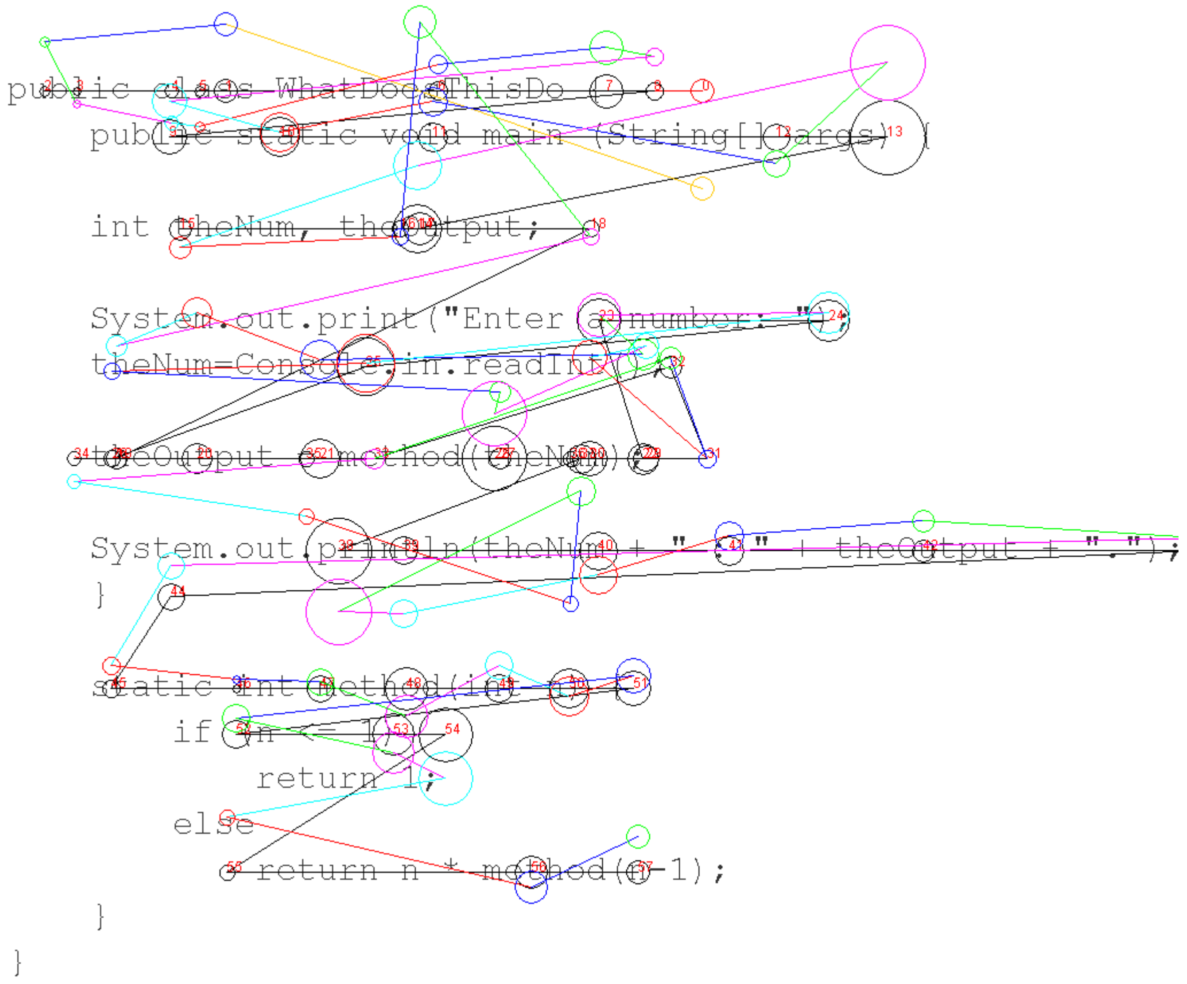
```
public class WhatDoesThisDo {
    public static void main (String[] args) {

    int theNum, theOutput;

    System.out.print("Enter a number: ");
    theNum=Console.in.readInt();

    theOutput = method(theNum);

    System.out.println(theNum + "! :  " + theOutput + ".");
    }

    static int method(int n) {
        if (n < 1)
            return 1;
    else
            return n * method(n-1);
    }
}
```

Figure 5.9. Gaze path for Student 5.



```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 103.0  is: 5.64%      ------
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 102.0  is: 5.58%      ------
Line 6 heat: 0.0  is: 0.0%
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 374.0  is: 20.47%     ---------------------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 177.0  is: 9.69%     ----------
Line 11 heat: 144.0  is: 7.88%     --------
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 111.0  is: 6.08%     -------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 221.0  is: 12.1%     -------------
Line 16 heat: 92.0  is: 5.04%      ------
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 0.0  is: 0.0%
Line 19 heat: 107.0  is: 5.86%     ------
Line 20 heat: 159.0  is: 8.7%      ---------
Line 21 heat: 237.0  is: 12.97%    -------------
Line 22 heat: 0.0  is: 0.0%
Line 23 heat: 0.0  is: 0.0%
Line 24 heat: 0.0  is: 0.0%
```

Figure 5.10. Heat map for the gaze path for Student 5 in figure 5.9.



```
public class WhatDoesThisDo {
    public static void main (String[] args) {

    int theNum, theOutput;

    System.out.print("Enter a number: ");
    theNum=Console.in.readInt();

    theOutput = method(theNum);

    System.out.println(theNum + " : " + theOutput + ".");
    }

    static int method(int n) {
        if (n <= 1)
            return 1;
        else
            return n * method(n-1);
    }
}
```

Figure 5.11. Gaze path for Student 6.

```
Line 1 heat: 0.0  is: 0.0%
Line 2 heat: 0.0  is: 0.0%
Line 3 heat: 0.0  is: 0.0%
Line 4 heat: 0.0  is: 0.0%
Line 5 heat: 155.0  is: 10.16%     -----------
Line 6 heat: 194.0  is: 12.72%     -------------
Line 7 heat: 0.0  is: 0.0%
Line 8 heat: 98.0  is: 6.43%       -------
Line 9 heat: 0.0  is: 0.0%
Line 10 heat: 88.0  is: 5.77%      ------
Line 11 heat: 33.0  is: 2.16%      ---
Line 12 heat: 0.0  is: 0.0%
Line 13 heat: 420.0  is: 27.54%    ----------------------------
Line 14 heat: 0.0  is: 0.0%
Line 15 heat: 159.0  is: 10.43%    -----------
Line 16 heat: 16.0  is: 1.05%      --
Line 17 heat: 0.0  is: 0.0%
Line 18 heat: 186.0  is: 12.2%     -------------
Line 19 heat: 99.0  is: 6.49%      -------
Line 20 heat: 0.0  is: 0.0%
Line 21 heat: 0.0  is: 0.0%
Line 22 heat: 77.0  is: 5.05%      ------
Line 23 heat: 0.0  is: 0.0%
Line 24 heat: 0.0  is: 0.0%
```

Figure 5.12. Heat map for the gaze path for Student 6 in figure 5.11.

Student	Eye Color	Glasses	Ability Level	Time Spent Considering Code
1	Hazel	No	Excellent	53.6 seconds
2	Blue	Yes	Average	27.8 seconds
3	Blue	No	Excellent	79.4 seconds
4	Brown	No	Poor	51.3 seconds
5	Blue	No	Average	29.5 seconds
6	Brown	No	Average	24.4 seconds

Table 5.1. Summary of the students who were tested.

5.2 Limitations

The limitations that follow are system dependent, and not necessarily able to be generally applied to all eye tracking systems.

5.2.1 Iris color

One of the biggest limitations in the testing process was on eye color and this was due to the fact that the S2 camera uses only the bright pupil tracking method. Of the students tested, only 2 had unusual difficulty in calibrating to the camera and both had very dark irises. Prior to the calibration process, the software provided with the camera displays a window so that the user can get situated and see if their eyes are within the viewing range. In this window, the software places a green and red square around each eye and then in two smaller windows, the left and right eyes are tracked separately. Each of these windows are displayed in grayscale, just as the camera sees. During this time, if the camera loses track of either eye, that window turns completely red. In both of situations, the camera lost track significantly more than it had with other users. Upon closer examination of users with lighter iris color, there was a very distinct color that represented the iris and the white color that represented the pupil of the eye. In the cases where calibration was difficult, the color difference between the iris and the pupil was almost impossible to differentiate. This is how the camera detects a pupil, it finds the whitest areas in the viewing window and hopefully finds the sclera, which is the white area surrounding the iris. Then it tries to detect a darker area within that white area, which is the iris. If it can not find the now brightened pupil inside the dark iris, it assumes that it is not focusing on the eye, and continues to search. This throws off the calibration process by a great deal and even if the calibration process can be completed with a satisfactory accuracy, it is very likely that it will continue to search during the testing phase because it doesn't feel confident of its lock onto the pupil.

5.2.2 Users wearing glasses

In the instances where users wore glasses, some of the same similarities arose, however they were not nearly as frequent nor as degrading to accuracy as dark irises. The main problem that arose from glasses was that when there was a glare on either lens, the camera may or may not lose the lock it had on that eye. It was usually only in the case where the glare obscured the pupil, and even then, with just a slight head movement, that problem was eliminated. So, glasses may be a slight limitation, but in any setup where glare on the lens is an issue, this will cause temporary problems, not just in the setup for this work with the Mirametrix S2 camera.

5.3 Results

The easily and accurately measurable results achieved from this work were from the testing done with the cursor. Every metric that was implemented, measured against the raw data from the camera alone, showed improvements. Using the averaging technique was a baseline for performance improvement and continued to be used in conjunction with every outlier metric implemented. Each of those also showed improvements from that new baseline, though some were more successful than others. Across all users, the metric that showed the highest rate of accuracy was the two pass median filter with a distance of 3 pixels. The reason this metric performed so well was that it was able to entirely eliminate gazes that were not compact, and therefore, not as reliable in their validity.

Cursor Size	Raw	Averaging	Median as Center	One Removal	Only Y	Two Pass
0	99.96	100	100	100	100	100
20	93.71	99.28	93.48	97.46	90.58	91.52
50	70.39	71.01	62.68	68.48	58.33	54.55
80	44.43	38.41	34.42	34.06	28.62	26.67
100	32.47	22.83	23.19	21.38	18.12	**14.55**
150	15.34	11.96	10.51	9.78	8.33	8.48
200	8.89	6.16	6.88	4.71	4.71	4.85

Table 5.2. Outlier removal error rates averaged over all the students who were tested.

Chapter 6

Conclusions

6.1 Conclusions

With the goal of this work being to aid Computer Science educators with an eye tracking program, that goal was met and contributions to knowledge in the field of eye tracking were made. It was shown that when the focus of the eye tracking was source code, there were several exploits that lent themselves to improving the accuracy of the camera. One of the more substantial advantages gained was by creating rules for finding eye carriage returns in a scan path. This was able to be further refined by adding weight to guesses based on line length and gaze location as well as adding additional weight based on reading tendencies. When testing accuracy while following a cursor, two improvements further improved reliability and those were averaging and outlier removal. All of these techniques improved overall line assignment accuracy as well as taking a step toward improving word and character accuracy.

6.2 Future Work

This work has yet to be implemented in a lab setting on a regular basis, but it is feasible to be used in that way in its current state. An instructor or an assistant would need to be briefed on calibration techniques and limitations with the current camera and laboratory setup. Even though the results are fairly straightforward, a quick explanation of the meaning of the results would be beneficial as well. Both of these short topics could be put into a user's manual along with other information required to perform the different tests described in this work.

Additionally, other projects could take advantage of this work and the improved accuracy techniques. Popular avenues for hands free programs are accessibility and video games. In the realm of accessibility, users who are physically unable to type could benefit from an on screen keyboard and mouse that rely on eye tracking. And through use in video games, the accuracy techniques in this research can be further refined for use in situations where reading is not the main goal. In the ever growing field of eye tracking, there are many other projects that could use this research, but hopefully this work will continue to be improved upon in its own right.

Bibliography

Bibliography

Arrington (2011) "Arrington SceneCamera System Specifications." Arrington Research Inc. 8

Bednarik, R. & Tukiainen, M. (2006) "An eye-tracking methodology for characterizing program comprehension processes." In *Proceedings of the 2006 symposium on Eye tracking research & applications*, ETRA '06, pages 125–132, New York, NY, USA. ACM. 11

Bednarik, R. & Tukiainen, M. (2008) "Temporal eye-tracking data: evolution of debugging strategies with multiple representations." In *Proceedings of the 2008 symposium on Eye tracking research &#38; applications*, ETRA '08, pages 99–102, New York, NY, USA. ACM. 11

Hennessey, C. & Duchowski, A. T. (2010) "An open source eye-gaze interface: expanding the adoption of eye-gaze in everyday applications." In *Proceedings of the 2010 Symposium on Eye-Tracking Research &#38; Applications*, ETRA '10, pages 81–84, New York, NY, USA. ACM. 5

Jacob, R. (2003) "Eye tracking in human-computer interaction and usability research: Ready to deliver the promises." Mind. 10

Law, B.; Atkins, M. S.; Kirkpatrick, A. E.; & Lomax, A. J. (2004) "Eye gaze patterns differentiate novice and experts in a virtual laparoscopic surgery training environment." In *Proceedings of the 2004 symposium on Eye tracking research & applications*, ETRA '04, pages 41–48, New York, NY, USA. ACM. 1

Mirametrix (2011) "Mirametrix S2 Product Sheet." Mirametrix Inc. 9

Pernice, N. (2009) "Eyetracking Methodology: How to Conduct and Evaluate Usability Studies Using Eyetracking.". 5

Salvucci, D. D. & Goldberg, J. H. (2000) "Identifying fixations and saccades in eye-tracking protocols." In *Proceedings of the 2000 symposium on Eye tracking research & applications*, ETRA '00, pages 71–78, New York, NY, USA. ACM. 5

Stampe, D. (1993) "Heuristic filtering and reliable calibration methods for video-based pupil-tracking systems." *Behavior Research Methods*, Vol. 25, pp. 137–142 10.3758/BF03204486. 9

TOBII (2010) “Tobii X1 Product Description.” Tobii Technology AB. 9

TOBII (2011) “Tobii T60XL Product Description.” Tobii Technology AB. 9

UWANO, H.; NAKAMURA, M.; MONDEN, A.; & MATSUMOTO, K.-I. (2006) “Analyzing individual performance of source code review using reviewers’ eye movement.” In *Proceedings of the 2006 symposium on Eye tracking research & applications*, ETRA ’06, pages 133–140, New York, NY, USA. ACM. 10

Vita

Austin Edward Pernell was born just outside of Boston, Massachusetts, on August 4th, 1988 to Jerry and Tammi Pernell. He was raised in Crossville, Tennessee and was lucky enough to have a family computer. After becoming fascinated with all the computer could do, he took two courses in programming once he entered high school. Upon enrolling in Computer Science at Tennessee Technological University in 2006, he excelled in the program and graduated at the top of his class. He decided to pursue his Master's Degree from Ole Miss in the fall of 2010.