Eye-Tracker Developer Manual

Kaden Kornaus

CSCI460

2025

Table of Contents

How to set up a Tobii Project in Visual Studios 2022 … 3

Data Files … 4

Info Files … 4

Collecting Gaze Data … 4

General Flow of Data … 5

Heatmap Generation … 6

Rewatch Animation … 7

How set up a Tobii Project in Visual Studios 2022

This section of the manual describes how to get started with the Tobii eye-tracker in Microsoft Visual Studios 2022. This section was written by SNC alumni Taylor Wesolowski, and was the guide I used when setting up my eye-tracker projects.

Taylor Wesolowski
(SNC Computer Science / Mathematics Graduate - Class of 2024):
https://compsci04.snc.edu/cs460/2024/taylorwesolowski/readMe.pdf

Data Files

The data files generated by the test form are just strings of integers representing gaze coordinates. They look a little bit like this: x1, y1, x2, y2, x3, y3, ..., xn, yn. This means that, to pull out one gaze point, you actually need to pull out 2 integers. These integers are the coordinates of each gaze point gathered by the eye tracker in relation to the image on the screen. This means that (0,0) represents the very top left corner of the image, and (imgWidth, imgHeight) represents the very bottom right corner. These points are stored as integers so that they can be easily used on a windows form.

Info Files

There are two kinds of info.txt files that are used throughout the two forms. The purpose of these text files is to hold information that wouldn't make sense in a data file and/or should be easy to look at and edit the contents. The two kinds of files are:

Test info.txt

- Used to hold the answer key of the test. That is, it holds the strings of 1's and 0's that represent whether or not a given image matches the initImg or not. (For more information on this, please consult the User Manual's section on "Creating a test folder")

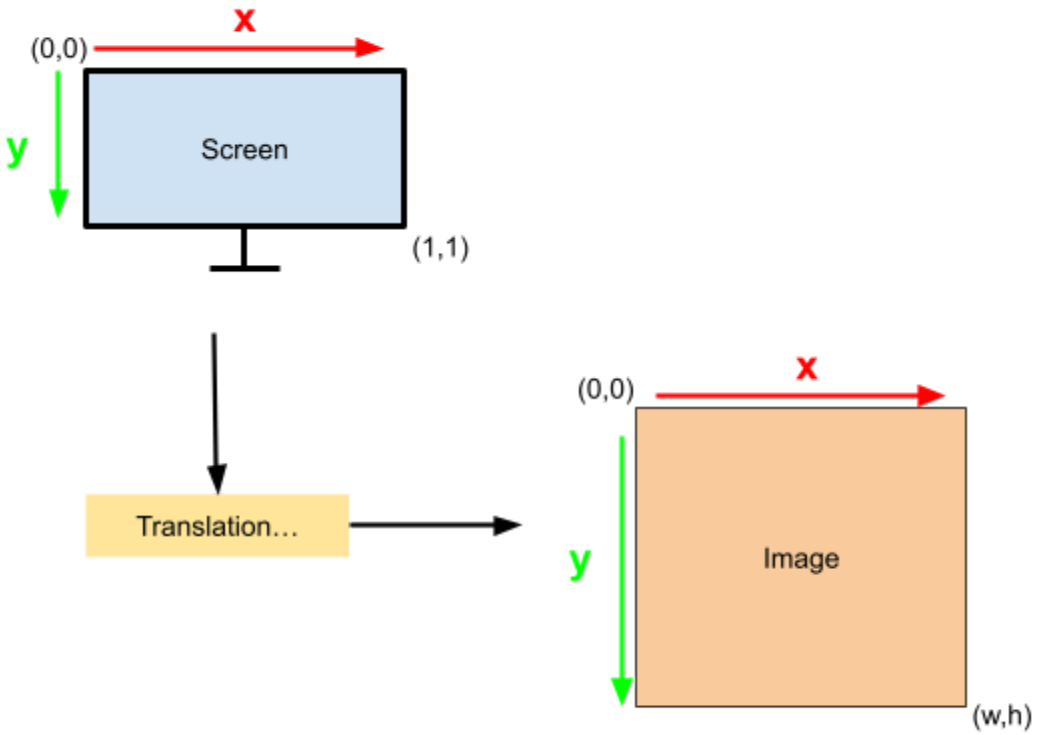
Results info.txt

- Used to hold the file path of the images that were used, the correct answer, the user's answer, and the user's overall score for the test.
- The correct answer and user's answer are stored as singular characters of either a 1 or a 0, which follow the same format as the answer key.
- The overall score is stored as a string in the format of: "num_Correct/total_Imgs"

Collecting Gaze Data

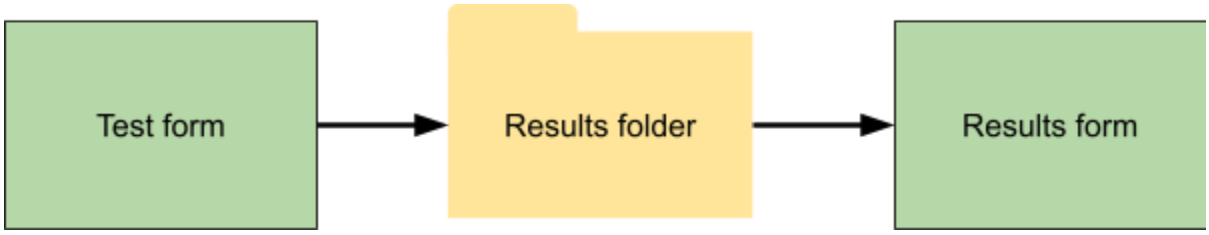
Eye data is collected from the EyeTrackerGazeDataRecieved event handler. This event is raised in the IEyeTracker object every time the eye-tracker has new data to send to the program. In this function, we first make sure that the data we received is data that we actually want to save (The points are on the screen, the users eyes aren't closed, etc.). If the data is not of use to us, we discard it by not saving it into the data file. If the data is valid, we then have to make sure that the user is looking at the image, and not somewhere else on the screen. If the data points coming in are valid and are located on the image, the coordinates are translated and saved into the data file. Since the coordinates that come in from the eye-tracker are relative to the screen ((0,0) is the top

left of the screen and (1,1) is the bottom right of the screen), we have to first translate them to coordinates on the window, then to coordinates on the image. So instead of being relative to the screen, (0,0) now represents the top left of the image, and (img_Width, img_Height) represents the bottom right of the image. This is done so that the data files are much easier to pick up by another program. It is also done so that the image can be placed anywhere on the screen and (in the future), could be resized and have the data files still be useful with minimal overhead computation.



General Flow of Data

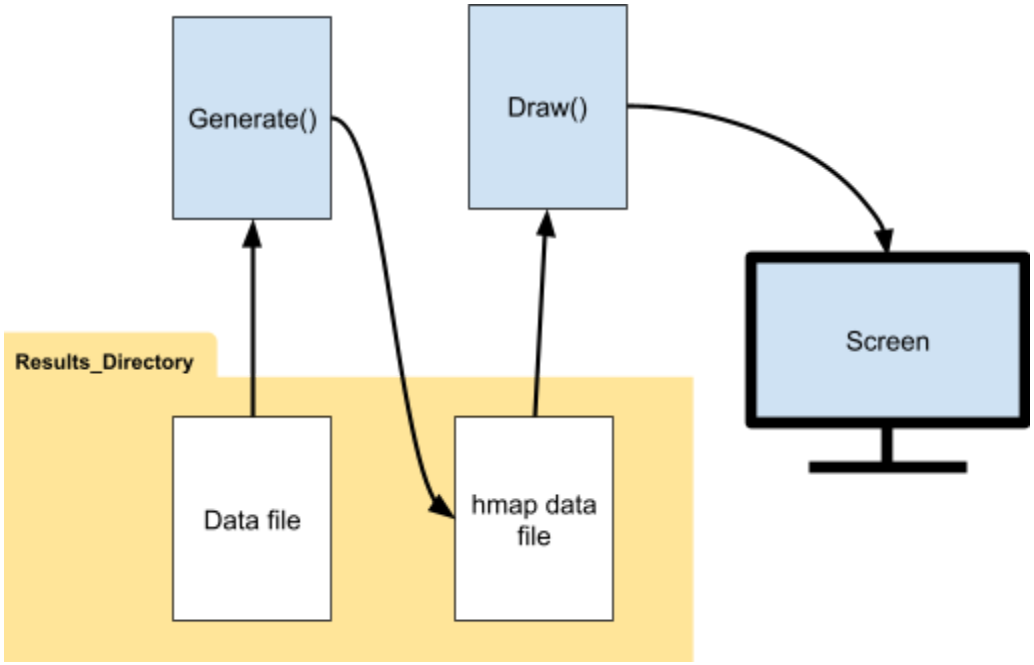
The way that data flows between the two forms is pretty simple. Since data really only flows in one direction (From test to results), storing data into a shared folder is a super simple way of passing data along. This folder is your “Result directory”, which is the folder where the test form will be storing all of the relevant test data. The flow chart is incredibly simple, and looks like this:



Heatmap Generation

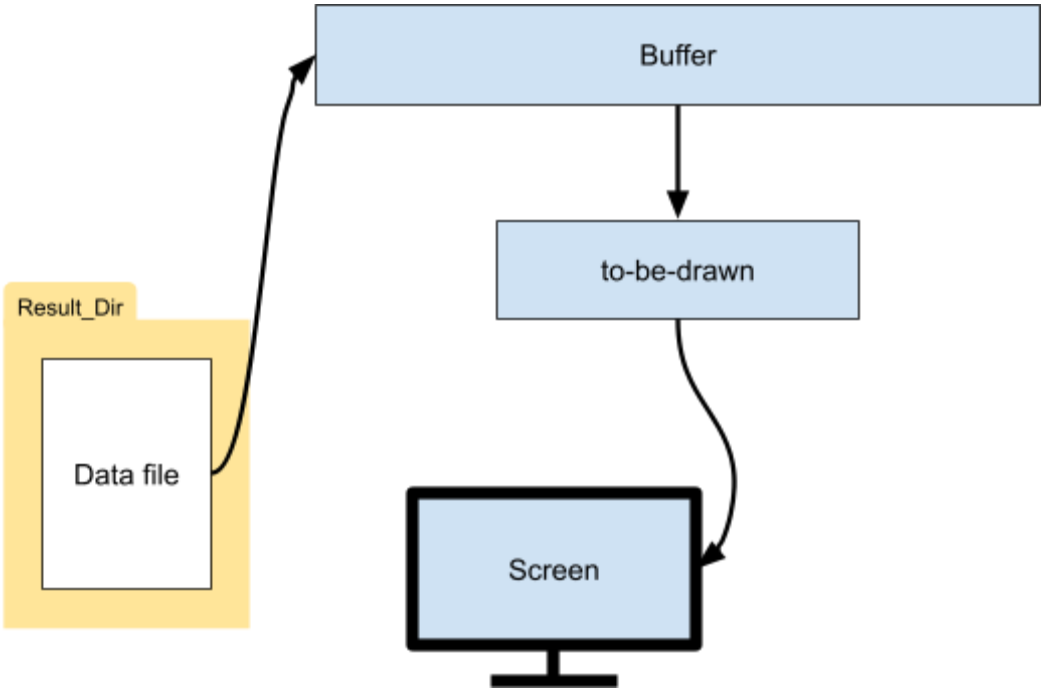
Heatmap generation is really done in two different steps. This is so that a developer is able to decide when each step is done.

- **Generation**
  This step is completed first. The main purpose of this step is to build the underlying data structure behind the heatmap. That is, it takes all of the points from the selected data file, and builds a frequency matrix (2D array) out of the data. It also saves the size of the largest “bucket”, which is used to help determine how each bucket should be colored. This structure is then saved into an hmap data file, which is picked up by the heatmap drawing function. The hmap data files are essentially an integer representing the size of the largest bucket, followed by a 2D array of integers representing the frequency matrix.
- **Drawing**
  This is the second step of the heatmap generation process. This is the function that handles the drawing process, and putting colors onto the screen. This function picks up the hmap data file created by the generation process, and uses the values stored in it to pick a color for each bucket, as well as paint it onto the screen. This function also handles the creation of the color array, which builds the gradient that will be used by the heatmap to color each bucket. This color array construction process takes a list of colors that you would like the gradient to shift through, and builds the gradient array using those colors.



Rewatch Animation

The rewatch animation runs in a thread that is handled by a background worker. In this thread, 200 coordinates (100 points) are pulled into a buffer from the file at a time. Next, these coordinates are turned into a “rewatchPoint” object, which makes them much easier and quicker to handle. These rewatchPoints are then pulled from the buffer into a “to-be-drawn” array of rewatch points that must be drawn on the screen. These points are pulled from the buffer until the buffer is empty, at which point more coordinates will be pulled from the file. This process repeats until the file is empty. Once this occurs, the algorithm continues until the buffer is empty, then finally will end after the “to-be-drawn” array is empty.



The Pause and Resume feature is pretty straightforward. Every time the algorithm ends (finishes, is paused, or is cancelled), the position of the file pointer is saved. This is only really useful if the algorithm is paused, since a completion or cancellation of the animation just resets the file pointer back to the beginning of the file (0). Every time the algorithm starts up and opens a data file, it sets the file pointer's position to wherever the last save point was, thus "Resuming" the animation.