Master's thesis

NTNU
Norwegian University of Science and Technology
Department of Computer Science

Fredrik Svendsen

Collaborative code editing Tool: Design and Evaluation

Master's thesis in Informatics
Supervisor: Kshitij Sharma
June 2022

NTNU
Norwegian University of
Science and Technology

Fredrik Svendsen

Collaborative code editing Tool: Design and Evaluation

Master's thesis in Informatics
Supervisor: Kshitij Sharma
June 2022

Norwegian University of Science and Technology
Department of Computer Science

NTNU

Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

DEPARTMENT OF COMPUTER SCIENCE

IT3920 - MASTER THESIS FOR MSIT

Collaborative code editing Tool: Design and Evaluation

Author:
Fredrik Svendsen

Supervisor:
Kshitij Sharma

June, 2022

Abstract

Fusing data from multiple modalities is shown to be valuable in gaining further understanding of the learning experience. The learning technology research field is mostly analysed through easy-to-collect data as click-streams or log-data, and the incorporation of multimodal data is in the early stages; especially for the collaborative experience which is still quite unexplored. This article describes the development of a collaborative web-based code editor for data collection, and the integration of data streams from multiple modalities: (1) eye-tracker, (2) wristband, (3) video, and (4) audio. Following the development, data was collected from 15 pair-programming experiments (30 participants) to investigate the predictive accuracy of multimodal data for programming performance in collaborative settings. The results show that a fusion of all extracted features from the different modalities predicted pairs performance with an error-rate of 8%, only decreasing to 7% when including a measure of prior knowledge. Moreover, some features extracted from sensor data, (e.g., cognitive load) is shown to be more important than prior knowledge in increasing prediction accuracy. The findings in this research provides new insight into the collaborative programming experience that can be used in designing learning technologies.

Acknowledgements

I want to thank Kshitij Sharma, my supervisor, for great guidance and support during this work, and for the help with the data analysis.

I would also like to thank everyone who participated in the data collection.

I would also like to thank Ola Kappelslåen Plassen and Andreas Rimolsrønning for the collaboration in recruiting participants which was to great help - and for all the talks this semester.

Finally, I want to thank Julie Viktoria for the invaluable emotional support.

Sammendrag

Kombinering av sensordata fra ulike datakilder er blitt vist til å gi ny og dypere forståelse av læringsprosessen. Læringsteknologi er som oftest forsket på og analysert gjennom begrensede datakilder oftest basert på direkte input som tasteklikk. Bruken av alternative datakilder som fysiologisk data i forskning av læringsprosessen, samt kombinasjonen av datakilder, er fortsatt i en tidlig fase; spesielt for læring ved samarbeid. Denne rapporten beskriver utvikling og implementasjon av en samarbeids-basert kode editor på web og datainnhenting fra flere kilder som (1) eye-tracker, (2) sensor-armbånd, (3) webcamera, (4) lydopptak, samt synkronisering av datakildene. Etter utviklingen av systemet ble det utført 15 par-programmering eksperimenter (30 deltakere) for samling av data. Innsamlet data er videre analysert for å utforske hvilke variabler som er prediktive for prestasjon i par-programmering. Resultatene viser at samlingen av alle variabler fra samtlige sensorer predikerte parenes prestasjon med en feilrate på 8%, og sank kun til 7% ved inkludering av mål på forkunnskaper. Enkelte variabler, som mål på kognitiv belastning, ble funnet til å være viktigere enn forkunnskaper for prediktering av prestasjon. Funnene i dette arbeid bidrar med nye innsikt i forståelse par-programmering prosessen som kan brukes til å forbedre design av fremtidige læringsteknologier.

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

With the continuous advance in technology, universities are utilizing online education to a greater extent. More and more universities are offering online classes and makes greater use of blended classrooms (i.e., partially online classes). Moreover, the popularity of online education has increased dramatically the past few years due to the COVID-19 pandemic. The interaction between the teacher and individual students in online classes is severely limited, especially in programming courses. In order to receive help, students are usually required to share their screen which may feel like an invasion of privacy.

Collaborative learning is suggested to be the superior learning strategy, (Johnson et al., 2007; Johnson and Johnson, 2009) and is further supported in terms of programming (McDowell et al., 2002; Williams and Kessler, 2000; Nosek, 1998). With the strong case for collaborative programming and the challenges of online programming courses, the need for supporting tools is undeniable.

Fusion of MMD have been shown to more accurately measure cognitive and affective states, and enables us to gain deeper insight in the learning process (Ochoa and Worsley, 2016; Giannakos et al., 2019; Sharma and Giannakos, 2020; Sharma et al., 2019; Lane and D'Mello, 2019; Zheng et al., 2019). Despite the strong evidence for collaborative learning and MMD, there is a gap in research of collaborative programming from a MMD.

This report sets out to describe the development (e.g., design and implementation) of a collaborative code editing tool integrated with data collection from logs and through sensor, and an educational dashboard for presenting the data in a meaningful way. The research question guiding this work is as following;

Which metrics derived from MMD are useful in predicting the pair programming performance?

In what way can a dashboard presenting students current progress along with MMD be beneficial for teachers?

2 Background and related work

2.1 Collaborative Coding

Pair Programming is the process of two programmers working together at one computer on the same task (Williams et al., 2000). Collaborative learning is suggested to be the su-

perior learning strategy, resulting in increased learning outcome and enjoyment (Johnson et al., 2007; Johnson and Johnson, 2009). The benefits of collaborative learning seems to be reflected in pair programming, as pair programming has been shown to be an effective strategy resulting in less code errors and greater enjoyment for entry-level students (McDowell et al., 2002), senior students (Williams and Kessler, 2000), and experienced programmers (Nosek, 1998). Students' engaging in pair programming may also have a lower dropout rate (McDowell et al., 2002), are less likely to deliver assignments late, and achieves more consistent results (Williams et al., 2000). However, pair programming is subject to practical limitations as geographic location and the need for a physical space, and thus may not always be feasible.

Current research comparing collocated and distributed pair programming did not find a significant difference in productivity, quality of code, or feedback from developers (Baheti, Williams, Gehringer, Stotts and Smith, 2002; Baheti, Gehringer and Stotts, 2002), which indicates the the benefits of collocated pair programming translates to distributed scenarios. With the continuous growth of online education (Kumar et al., 2017), further magnified by the COVID-19 pandemic, the need for tools supporting online education and work is greater than ever.

2.2 Tools Supporting Distributed Pair Programming

Several tools have been developed to enhance the opportunities of pair programming, but all with different goals in mind. Some tools are designed to support interactions between teachers and learners, while others are intended to assist collaboration between developers in professional settings. Jo and Arnold (2003) built a standalone program for collaborative coding with a professional work setting as the primary focus. The program supports textual communication between collaborators, and noted distributed learning as a possible use case. The study did not mention compatibility with different operating systems [OS], and it is unclear whether a user is limited to a single session.

Plugins exist that enables online concurrent programming in integrated development environments [IDE] (e.g., CoVSCode (Fan et al., 2019)), but the plugins tend to be designed only for the purpose of collaboration and does not support data collection. (Boyer et al., 2008) made a collaborative plugin (*RIPPLE*) for Eclipse [1], and performed a usability test with students relatively new to programming. RIPPLE adds to Eclipse in the way that multiple people can connect to a same session and includes a window for textual communication. The results were promising for the case of distributed pair programming; students reported significantly greater enjoyment from collaborative work than previously

[1] https://www.eclipse.org/

individual assignments Boyer et al. (2008). However, the findings are due to some limitations; the results were computed solely from questionnaire data and the test-sessions were only 50 minutes. Considering the data were gathered at a single point in time, the students may have been affected by unmeasured factors such as emotions (e.g., happy emotion after test completion).

In contrast to the trend of collaborative programming environments as standalone programs or IDE plugins, *CodeHelper* is a lightweight web application [web-app] (Liu and Woo, 2021). It was developed with the intent to support online mentoring in programming courses. As a simple web-app, CodeHelper can be easily accessed from most browsers and effectively circumvents the challenges related to OS compatibility and installation that standalone programs and plugins are facing. On the other hand, it is designed mainly for one-to-one teacher-student interactions which presents some issues. A session can only be started by an administrator (or e.g., teacher) and may only be used as long as the administrator is present, eliminating the possibilities for collaboration between students. Each sessions starts with a blank editor and code have to be manually inserted by the teacher or student, which limits the usability for complex programs.

The lack of multi-session support is recurring in existing collaborative tools, resulting infeasible for use in a classroom setting where a teacher would like to observe multiple groups simultaneously. Although the collaborative environment created by Jo and Arnold (2003) support the creation of sub-groups for communication, it does not support separate and private work-spaces for different groups. Integrating support for multiple sessions in IDE plugins (e.g., CoVSCode (Fan et al., 2019) and RIPPLE (Boyer et al., 2008)) is a complex task. An alternative solution to monitoring multiple groups simultaneously through plugins is to open a separate instance of the IDE for each group, though not very convenient. Shen and Sun (2000) describes the (previously developed) collaborative programming environment *RECIPE*, made to assist in the research of distributed collaborative programming. RECIPE differs from similar collaboration tools in that the admin of a session can control access rights of the participants (e.g., read only, read/write). RECIPE is in line with previous findings in terms of multi-session support, as a user cannot be connected to multiple sessions simultaneously.

The lack of data logging is recurrent in existing tools for collaborative code editing. Being able to reconstruct the collaborative process can provide valuable data for further studies (Edson and Phillips, 2021; Boyer et al., 2008). Logging of editor-contents and user-data (i.e., cursor position, selection) has previously been implemented in the Eclipse plugin RIPPLE Boyer et al. (2008). Although RIPPLE does not support multiple real-time sessions, every session can be fully reconstructed post-test from the logged data.

2.3 Educational Dashboard

An educational dashboard is defined by Yoo et al. (2015) as ”a display which visualizes the results of educational data mining in a useful way”. The amount of raw data produced by different sources when capturing the learning process may be plenty and can be difficult to interpret. Processing (if necessary) and making sense of such data can provide valuable real-time insights to both students and teachers (Yoo et al., 2015). Visualizations tend to be in the form of graphs and charts. The presented data differs for each dashboard due to different cases. The challenges related to dashboards include presenting actually meaningful data and finding the optimal way of presenting such data (Schwendimann et al., 2017).

In a literature review of existing studies related to dashboard, Yoo et al. (2015) reported a research gap in students reactions to their data presented in dashboards. Dashboards have in the recent years seen a steady growth in popularity. However, the information presented in dashboards from existing studies tend to originate from a single data source (Schwendimann et al., 2017). Schwendimann et al. (2017) reviewed 55 studies related to educational dashboard and found approximately two-thirds to only use a single data source, with log-data being the most used. Due to the complexity of the learning process, there is a need for further research of educational dashboards with multiple data sources to gain better insight in the learning process.

Edson and Phillips (2021) tested the usability of dashboards for teachers in monitoring student collaboration in a mathematics course. The test were conducted in a classroom environment. Initially, the teachers reported limited use of the dashboard as they preferred to be actively walking around the classroom and helping students. As the primary downsides were strictly related to the physical classroom setting, such dashboard may be of great use in an online setting. The test were carried out over multiple years and an increased use of the dashboard were observed. When asked about this, the teachers agreed that the dashboard showing students' progress were very helpful in determining when to transition from the *working* phase to the *summarize* phase of the class.

2.4 Multimodal Data for Learning Analytics

The process of learning is complex and is affected by the learners cognition and affective state. Technology provides great opportunities in capturing a multitude of data streams from the learning process. Some data types commonly used in analyzing the learning process are video, audio, eye-tracking, physiological data, and click-streams (Giannakos et al., 2019). A single data source may be adequate in capturing a specific aspect of a

students learning experience: e.g., Giannakos et al. (2019) found eye-tracking data to be a good predictor for skill acquisition of subjects playing the Pac-Man game. However, Giannakos et al. (2019) found a fusion of eye-tracking, electroencephalogram [EEG], and facial data to result in even greater accuracy in predicting skill acquisition. The overall learning process is a product of innumerable sub-processes and thus cannot be completely captured from a single data source. Existing research of the learning experience from a MMD (MultiModal Data) perspective have shown great potential by resulting in more accurate predictions of effort (i.e., measured engagement) (Sharma et al., 2019), performance (Sharma et al., 2019; Giannakos et al., 2019; Andrade, 2017), and emotions (Zheng et al., 2019), among other things. Zheng et al. (2019) found both EEG and eye-tracking to be individually good predictors for emotions; however, EEG were superior in recognizing happy emotion and eye-tracking in recognizing fear emotion, strengthening the case for multiple data sources.

The research of MMLA is still in an early stage Sharma and Giannakos (2020). Despite the promising results of MMLA, the number of data sources used in existing studies of the learning experience tend to be limited (Sharma and Giannakos, 2020). The collection of data and fusion of modalities poses several challenges related to privacy, cost, and technical difficulties (Ochoa and Worsley, 2016; Sharma and Giannakos, 2020; Giannakos et al., 2019). Sharma and Giannakos (2020) reviewed studies related to *MMD capabilities for learning*, and found a significant negative correlation between the number of modalities and sample size, for studies using more than two modalities. MMLA (MultiModal learning Analytics (Blikstein and Worsley, 2016)) has the potential to provide unique insight in the learning experience, but research is currently lacking due to the aforementioned challenges.

Previous studies have found physiological data (e.g., blood volumse pulse [BVP] and heart rate variability [HRV]) Sharma et al. (2019); Giannakos et al. (2019) to be beneficial in explaining cognitive aspects of the learning experience, as skill acquisition and problem solving. Understanding the cognitive abilities is an empathized matter in education research, and is the primary target of tests in the educational system Lane and D'Mello (2019). A major advantage of including physiological data is that its' collected from sensors that can produce a continuous stream of data. As opposed to user-generated data as questionnaire data which is only representative for a single point in time, continuous data from sensors can capture every little change and reflects reality more accurately. Different related states (e.g., cognitive: cognitive load, fatigue) can be observed simultaneously by capturing and fusing multiple streams of continuous data, providing a stronger fundamental for a more fine-grained analysis (Lane and D'Mello, 2019; Sharma et al., 2019).

Despite the advantages of utilizing multiple streams of data in capturing different aspects of the learning experience, the measurements that can be derived often overlap. Estimates for learners cognitive state, such as cognitive load, can be derived from multiple different data sources, including EEG data Sharma and Giannakos (2020), motion data, and eye-tracking (pupil diameter) (Cowley et al., 2016). Integrating multiple modalities may be considered superfluous in studies not having MMLA as a primary focus. On the flip side, analyses utilizing overlapping metrics from MMD is less likely to result in erroneous conclusions due to increased validity of findings.

As previously mentioned, the research of MMLA is in the early stages and its full capabilities in terms of understanding the learning process is still unknown (Ochoa and Worsley, 2016; Blikstein and Worsley, 2016; Sharma and Giannakos, 2020). The arguably slow progress in MMLA research is due to difficulties in collecting and analyzing data. Firstly, collecting data can be an expensive process. Recording devices (e.g., cameras, microphones, eye-tracker, wristbands) tend to be expensive itself, and may be tricky to set up. Collecting MMD from online courses [or MOOC - Massive Open Online Course] is difficult as it requires the subjects to be in possession of the required recording devices. Sensors may communicate through different protocols (e.g., BLE; Empatica E4 Wristband [2], which requires additional technical setup for real-time streaming of data. In cases were computer-integrated webcams and microphones are considered adequate recording devices, privacy becomes an issue (Sharma and Giannakos, 2020). The collection of MMD is usually done in physical environments due to the infeasibility of online data collection, and the class-room setting has been found to be the most used Sharma and Giannakos (2020). Moreover, fusing MMD and making sense of large amounts of raw data is technically difficult (Ochoa and Worsley, 2016).

2.5 Multimodal Data and Distributed Pair Programming

MMLA has been shown to provide great insight in the collaborative learning process. (Sharma and Giannakos, 2020). Spikol et al. (2016) developed a framework for analyzing MMD in collaborative programming. Currently there is a gap in research of collaborative programming with MMD.

[2] https://www.empatica.com/research/e4/

3 Implementation

As no system exists that can gather and synchronize the data required for this research, this had to be developed. This section describes the development, architectural decisions, and challenges faced during the process.

3.1 Requirements

Due to uncertainty regarding the difficulty of integrating the different data sources, and the devices only being available to use for a limited time, the requirements were indecisive. However, the goal was to develop a collaborative code editor and collect physiological data from Empatica E4 wristbands[3], facial landmarks from webcameras, gaze data from eye-trackers, and audio recordings.

The objectives of the development can be broken down into four main parts:

(1) : A code editor that supports concurrent collaboration, integrated with extensive logging

(2) : Gather data from multiple data streams (i.e., eye tracking, webcam, audio and wristband)

(3) : Synchronize all gathered data from editor and user sensors

(4) : Replay the the experiments in a teacher-dashboard through logged data

The plan was initially to present real-time data in the dashboard, but this changed during the development due to unforeseen problems further described later in this section.

3.2 Front-end

3.2.1 Code editor

Technical decisions regarding the code editor were primarily affected by the planned dashboard, amount of data to be collected, and need for both users to be able to compile the code. Although several IDE plugins for collaborative programming exists, few support code compiling for both users and none, to my knowledge, includes sufficient logging required for this project. The final design of the code editor webapp can be seen in Figure 1.

[3] https://www.empatica.com/research/e4/

Firepad and Firebase

One option that stood out when selecting a text editor was Firepad[4]. Firepad is a collaborative web-based text editor integrated with Firbase Realtime Database[5]. Firebase realtime is a cloud hosted NoSQL database that automatically synchronizes changes and resolves merge conflicts. This is particularly useful for this project in the event of a user losing connection during an experiment, as lack of data integrity can be detrimental to the replay of a code session. Moreover, real-time synchronization enables multiple live collaboration sessions to be viewed in the dashboard in real-time. The Firepad text editor also includes features such as undo & redo, presence detection, text highlighting, and cursor position. These features are key to recreating a pair-programming session. Another feature of Firepad is version checkpointing, which is useful for both detecting and solving issues related to data integrity. Additionally, Firepad comes with a userlist[6] that is simple to integrate. The list of users are especially useful in a classroom-setting where a lecturer may frequently change between groups.

CodeMirror

Although Firepad itself only supports text editing, it can render the text from web-based code editors as Ace[7] and CodeMirror[8]. CodeMirror and Ace are shortly said just text editors with desired features for code editing such as syntax highlighting and code folding. CodeMirror was chosen for this project, simply due to superior documentation.

React

The JavaScript[9] library React[10] was used to build the front-end. The decision to use React was made with the dashboard in mind, as React being component-based. Reusable components with differing state is highly desired for a dashboard that does not only show each pair as a component in the overview, but also the many components visualizing metrics in each group view. For just the collaborative editor, disregarding the dashboard, a simple web-page would suffice.

After deciding on a React project embedded with a Firebase and CodeMirror editor for the front end, there was still missing functionality. Firepad is built only with real-time collaboration in mind and all user-related data is continuously overwritten in the database. An example of this is that data regarding a users action, e.g., cursor placement, is limited

[4] https://firepad.io/
[5] https://firebase.google.com/docs/database
[6] https://github.com/FirebaseExtended/firepad/blob/master/examples/firepad-userlist.js
[7] https://ace.c9.io/
[8] https://codemirror.net/
[9] https://www.javascript.com/
[10] https://reactjs.org/

to the last recorded value only. All user data are also dependent on the presence of the respective user, and removed if no presence is detected. The only persistent data after a collaboration session are modifications of the code.

Logging

As Firepad uses but does not store user data that is relevant for recreating the collaborative experience, such as cursor position and selection, Firepad had to be modified. The library was downloaded as shown in Code listing 1.

```
wget https://firepad.io/releases/v1.5.10/firepad.min.js
```

Listing 1: Downloading Firepad

In order to make the data persistent, the library were modified to, whenever a user event were pushed to the database, to also push the same data to another document in the NoSQL database that would never be overwritten. This way the functionality of Firepad is never affected and the user data persists. An example of how this were implemented in the *sendCursor* function is found in Code listing 2, where the added code are in the lines 3-18, and *userLogRef* is a reference to the log-document the database.

```
1  i.prototype.sendCursor = function (t) {
2      this.userRef_.child("cursor").set(t);
3      // START new code
4      let dt_client = new Date().getTime();
5      let sStart = null;
6      let sEnd = null;
7      if (t !== null) {
8          sStart = t.position;
9          sEnd = t.selectionEnd;
10     }
11     // userLogRef_ is a reference to a document in the NoSQL db
12     this.userLogRef_.child("cursor").push({
13         selectionStart: sStart,
14         selectionEnd: sEnd,
15         client_timestamp: dt_client,
16         timestamp: firebase.database.ServerValue.TIMESTAMP })
17     }
18     // END new code
19     this.cursor_ = t;
20 }
```

Listing 2: Extending Firepad

Code compilation

After developing a functioning collaborative editor, the last step was to find a way for users to compile and run their code. Initially, this was done using the library Pyodide (The Pyodide development team, 2021) which is a Python distribution for the browser. A webapp (e.g., the editor) cannot execute Python code on client machines, but Pyodide works in the way that it executes the code in a virtual machine. The output and error data streams from executing Python code, *stdout* and *stderr*, respectively, were redirected to a function updating the compile-box of the editor. The compile-box can be seen in Figure 1 at number 5. This worked well in the sense that all users can compile and run the code independently, and the output a user can see is limited to the output from when the user ran the program.

The debug task (see appendix B) for the experiment was provided by the supervisor of this project who intends to run further analysis on the collected data after this project. The debug task involved debugging a Python game built with the Python library Pygame[11]. Unfortunately, Pyodide showed to be incompatible with Pygame. The first issue that was faced when trying to make Pyodide run Pygame was loading assets for the game. Pyodide runs in the browser and JavaScript is not allowed to access local data files, however, this could be worked-around by loading all assets from a server[12]. Before implementing a server for assets, all assets were temporarily removed from the game to find out if that was the only issue causing problems. Unfortunately, Pyodide was not able to load the Pygame package. This was initially thought to be due to incompatibility between Python versions, as each version of Pyodide runs on a specific Python version and each version of Pygame only support specific Python versions. However, after trying out (*almost*) all permutations of Pyodide and Pygame versions, this was found to not be the cause of the error. Pyodide only support external packages that are pure Python, and as Pygame is not pure python due to SDL-libraries dependencies for hardware access, the Pyodide-Pygame combination turned out to be impossible.

The work-around for running a Pygame-game from the browser ended up being to send the code to a server running locally on both client computers, where the server executed the code and responded with the output and error streams. The implementation and details of the server is described in Section 3.4.

[11] https://www.pygame.org/docs/

[12] https://pyodide.org/en/stable/usage/faq.html#how-can-i-load-external-files-in-pyodide

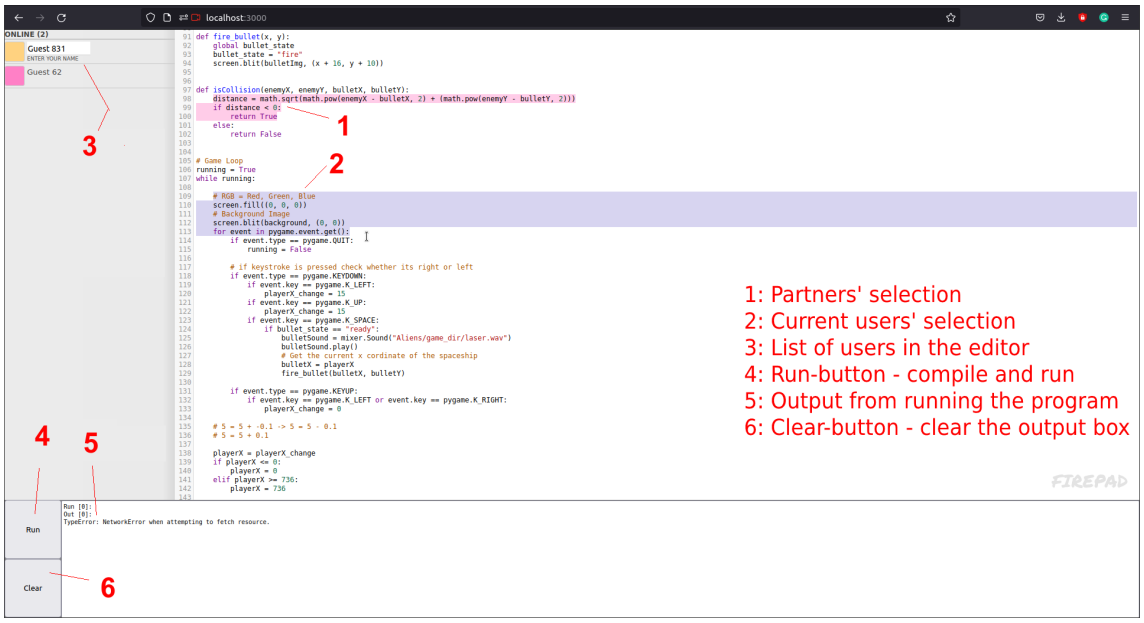


Figure 1: Final design of the code editor

3.2.2 Dashboard

Then plan was originally to develop a teacher-dashboard displaying metrics from the sensor data alongside the code editor (right side of the editor, see Figure 1), and evaluate said dashboard through qualitative interviews. The key metrics to display was going to be derived from the quantitative analysis of multimodal data and pair performance.

This was initially implemented in the same ReactJS-based webapp as the code editor by rendering additional components for users with privileged *admin*-roles. The data displayed were updated the same way as the text inside the code editor; through a websocket connection to Firebase where the data was stored. Unfortunately, this implementation was discarded due to restrictions related to storing personal data (described in Section 3.5).

The restriction of data storage mentioned above resulted in all data being stored locally. The second iteration of the dashboard used the Dash[13] framework for building the user-interface and the plotly[14] library for visualizing data. However, the time-cost of remodelling the system resulted in insufficient time for evaluating the dashboard, and the dashboard was therefore decided to be a part of future work.

[13] https://dash.plotly.com/
[14] https://plotly.com/python/

3.3 Capturing data streams

3.3.1 Empatica E4 wristband

The E4 wristbands stream collected data in real-time over BLE (Bluetooth Low Energy) to a BLED112 USB dongle[15]. Empatica's E4 streaming server for Windows[16] is used to forward the data received by the BLED112 dongle to socket connections over TCP, and includes an API for subscribing to devices and and their data streams. A Python client (named *pyE4Client* in figures) was developed for capturing and saving wristband-data from the E4 server in real-time. The E4 data collection process is illustrated in Figure 2 in the form of a sequence diagram.

If for any reason the BLE connection between E4 and the E4 streaming server (through BLED112) disconnects, the server is set to automatically reconnect. This was never observed to happen during testing nor in the experiments. The pyE4Client run independently of the webapp and must be started and stopped manually. When the client receives data from the E4 server, it first decodes the message and then appends each row of data to temporary lists with the corresponding data type (e.g., rows with data type E4_HR are appended to list HR_temp). Data in temporary lists are saved to disk when the list reaches a certain size or the process is stopped. The Python module atexit[17] was used to mitigate potential data loss by triggering a save of captured but unsaved data (i.e., data in temporary lists) in the event of an unexpected termination.

In the first implementation, all incoming data were appended to pandas.DataFrame's(Reback et al., 2022) and only written to file when the program was stopped. The time-cost of appending dataframes increases with the size of the dataframe, and the increasingly poor performance during a session eventually lead to data incoming at a higher rate than what the program was able to save. Although all data would eventually be saved if the program kept running after unsubscribing from the device (i.e., wristband) data streams, the ever-increasing time cost of appending new data makes this highly impractical, and the data would not be in real-time and thus unsuitable for a real-time dashboard. Depending on the duration of the data collection, this may also cause memory-related issues. To summarize this paragraph; performance is important when dealing with continuous data streams.

[15] https://www.silabs.com/documents/public/data-sheets/BLED112-DataSheet.pdf
[16] https://developer.empatica.com/windows-streaming-server.html
[17] https://docs.python.org/3/library/atexit.html

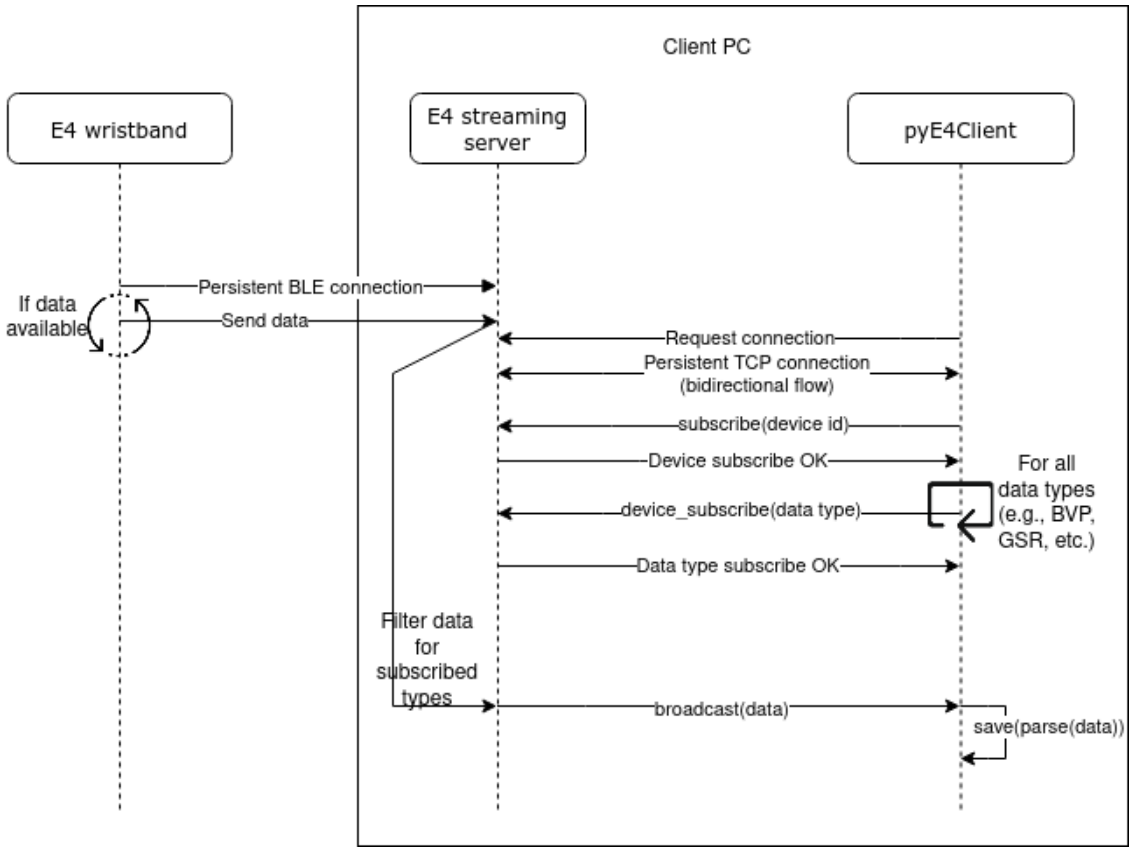


Figure 2: Sequence diagram of the E4 data collection process

3.3.2 Video

Video from the webcamera is streamed and processed in the front-end code of the code editor, and the process is set to automatically start when the user clicks a browser prompt starting the coding session in the editor. The Media Streams API[18] is used for accessing the data stream and the JavaScript API face-api.js[19] for processing. Face-api processes the video stream in real-time using pre-trained neural network models for image recognition to detects faces, facial landmarks, and facial expressions, among others. The video stream is not recorded since it is processed in real-time.

The decision to process the video in the browser was made at a time where the webcamera's were integrated and all data stored in firebase. The processed data were timestamped by the same clock as the editor log, so no need for additional synchronization. However, the data could no longer be saved in firebase due the personal data storage restrictions described in Section3.5, and ended up being sent to the server described in Section 3.4 instead.

[18]https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API
[19]https://justadudewhohacks.github.io/face-api.js/docs/globals.html

The limited time to redesign the system away from storing data in firebase resulted in some poor technical decisions, with they way face-api data was handled being one of them. Collected was only sent to the server over HTTP at the end of a test (25 minutes), instead of e.g., continuously over a TCP connection. A major overlooked issue with this was the risk of data loss that came with collecting the data in the browser. A reload of the webpage would result in loss of all data not already sent to the server. This risk was somewhat mitigated by sending data to the server at frequent intervals. In retrospect, a more optimal solution would be capturing and processing the video stream in a process separate from the webapp, whether it be running on the server or locally on the same machine.

3.3.3 Audio

The audio stream is started and captured the same way as the video stream (described in the previous paragraph), however, it is not processed for features in real-time but rather just saved as a recording. The audio is compressed in an OGG-container before being sent to the same server as the video data, and is saved once a minute for backup purposes. Similar to the video data, the sub-optimal technical design and the data-loss risks following the browser implementation also applies to this.

Mitigating the potential data loss is done slightly different for the audio data as it is not a collection of observations already timestamped (e.g., data from video processing). When the recording starts, a timestamp is logged to the database. Timestamps for the start of each recording (only multiple if a user refresh the webapp connection) can be used to detect exact intervals of missing audio. Valid audio segments can then be combined to a single file with each segment at their correct timestamp, and the empty intervals either filled (e.g., with silence) or excluded from further use.

3.3.4 Gaze data

The eye-trackers used were Tobii Pro X3-120[20] which connects over USB and broadcasts to localhost. Tobii's SDK for Python[21] is a programming interface for communicating with the eye-tracker, and was used in form of the Python package tobii_research[22]. Tobii_research simplified the process of collecting gaze data to a large extent, where it was down to a few function calls for connecting to the device and subscribing data streams.

[20] https://www.tobiipro.com/product-listing/tobii-pro-x3-120/
[21] https://developer.tobiipro.com/python/python-sdk-reference-guide.html
[22] https://pypi.org/project/tobii-research/

Gaze data is saved in similar fashion as wristband data from E4, and also uses the *atexit*-module to mitigate risk of data-loss.

Gaze data are timestamped by the eye-tracking devices internal clock, and some the clock in Tobii devices have been found to drift several seconds over sessions lasting approximately 20 minutes (Nüssli, 2011). Therefore, the client (named pyEyeClient in later figures) developed for capturing (through tobii_research) and saving the gaze data timestamps all data the the computers internal clock, eliminating clock drift between different data sources (e.g., gaze, wristband, etc).

3.4 Game and data server

A server for running the game and receiving data from the browser (i.e., facial features and audio data) was developed in Python using the Flask (Grinberg, 2018) framework. Two instances of the server ran separately on the two computers used in the experiments, and was only connected available on local network of each computer. The server is referred to as *GameServer* in architectural views shown later in this chapter.

The reason for running the games on a server was a workaround due to incompatibility between Pygame and browser-based python compilers, which was discovered at a late time, only days prior to starting the data collection (further described in the end of Section 3.2.1 and in Section 3.5).

The workaround for running a python script (i.e., Pygame) from the browser was to, whenever a user clicked the *Run*-button (see Figure 1), all code from the editor was sent to the server. The server has a directory with the game files (main script, assets) and runs in an environment where the Pygame-package is installed. Upon receiving code from the user, the server overwrites the main script (i.e., python file executed to run the game) with the received code. Then, the server proceeds to run the updated main script in a new process using Python's subprocess module[23]. Stdout and stderr (i.e., the output and error streams, respectively) in the sub-process are redirected to local variables so the server can return a proper response including any relevant information from running the program. The communication between the webapp and server is over HTTP, so the webapp only receives the output in a single response after the game is closed. Thus, in this workaround, the programs' output is only displayed to the user *after* closing the game, in contrast to the initial implementation (*not* Pygame) where the code was executed in the browser and output/error streams displayed in real-time,.

Assuming the code is free for breaking errors, the servers sub-process call opens the game

[23] https://docs.python.org/3/library/subprocess.html

in a new window. A downside of running the game in a sub-process is that the window opens in the *background* (i.e., behind the layers of currently active windows), and must be manually opened from the application bar. This was solved using Dexpot[24] - a program for Windows customization - by writing a custom rule forcing the game to be displayed in the foreground.

The server also has endpoints for receiving video and audio data from the browser, and writes said data to disk. This is simply for usability reasons; data saved directly from the browser must be downloaded and each download trigger a pop-up message, which is inconvenient in a pair-programming experiment and could be a source for annoyance.

3.5 Early iterations and challenges affecting the architecture

Initially, the plan was to send all collected data straight to the firebase database, so that any collaborate coding session could be observed from a dashboard in real-time. This involved each of the two computers collecting data from the respective users sensors. The E4 wristband was the first implemented data source external of the code editor, and Figure 3 illustrates the initial setup.

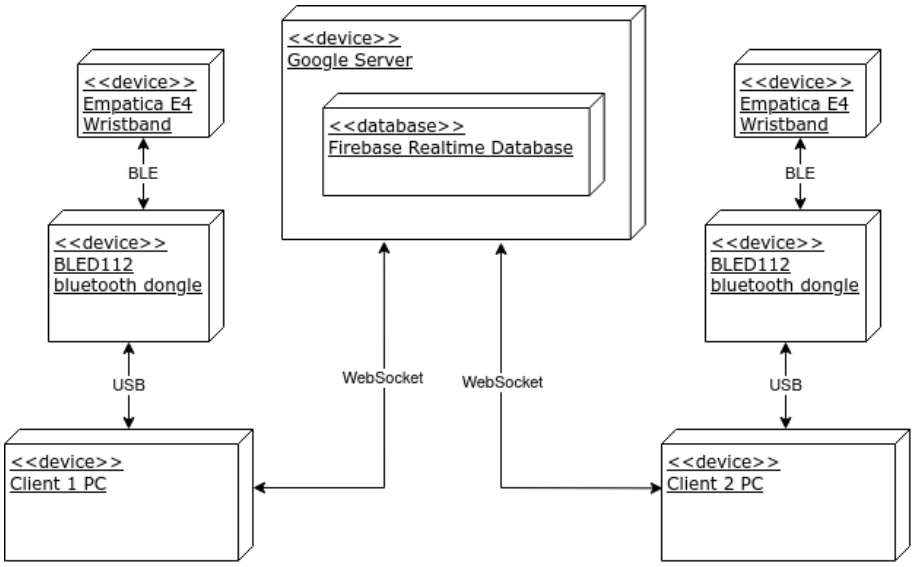


Figure 3: E4 wristband first iteration - physical view

During the development process, the value of connecting the sensor devices to a shared computer (with a shared server for data captured in the browser), was slowly realized. Running data collection scripts (for gaze and e4) and processing the data on the same device has multiple advantages: real-time clock synchronization of the data, less technical

[24] https://dexpot.de/?lang=en

setup before experiments (scripts only run on shared, not separate on both computers), no risk of users accidentally terminating data collection scripts, and the researcher can monitor the processing computer to quickly observe any potential errors. Moreover, starting the scripts simultaneously on the same device removes the need to start each script individually and *before* the experiment resulting in less data being saved (file size), and reduces the chance of errors in the setup process.

Figure 4 shows architecture from the first iteration of the system as a whole. In this iteration it was assumed that the computers to be used in the data collection would be laptops with integrated webcameras, therefore, the video and audio data were still collected in the browser before being sent to the processing server.

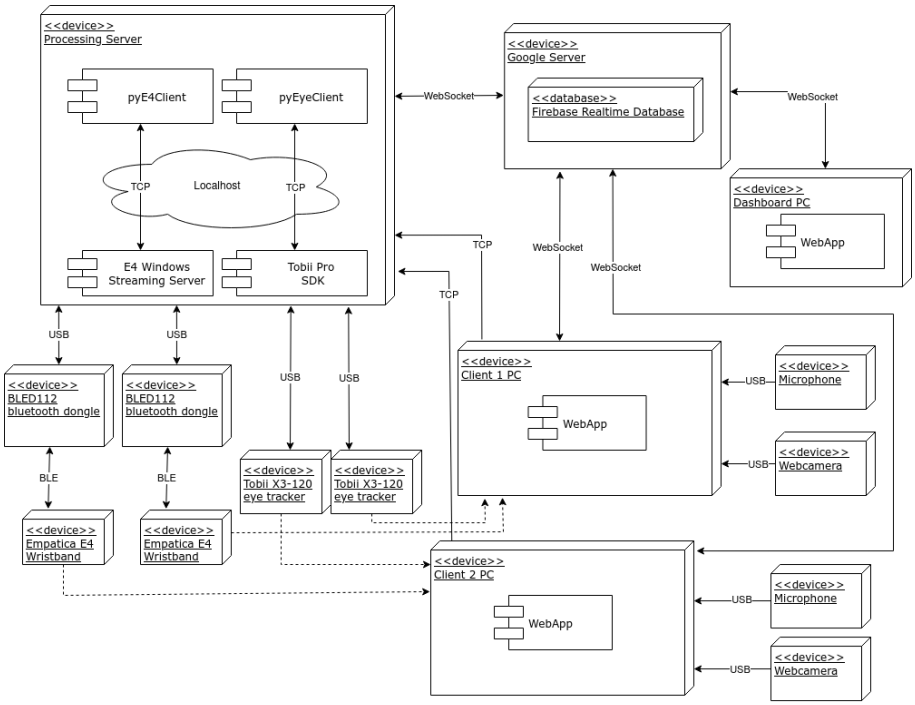


Figure 4: First version of the system - physical view with main processes

3.5.1 Personal data restrictions

Permission to collect and store the personal data was applied for to the Norwegian Centre for Research Data (NSD)[25]. The first application was sent in the middle of February, and was unfortunately rejected five weeks later, at the middle of March. The reason for rejection was because the personal data was planned to be stored in the firebase database. It is required that any external service processing personal data has an agreement with the university, and firebase, being hosted by Google, did not have such an agreement with the university.

[25] https://www.nsd.no/en/

Consequently, the personal data could no longer be stored in firebase. This did data storage restriction did not only affect the data collection, but was detrimental to the dashboard considering the dashboard got all data directly from firebase. The subsequent iteration of the system saved all data locally on the shared processing computer, illustrated in Figure 5, where removed lines of communication are marked with ”X”.

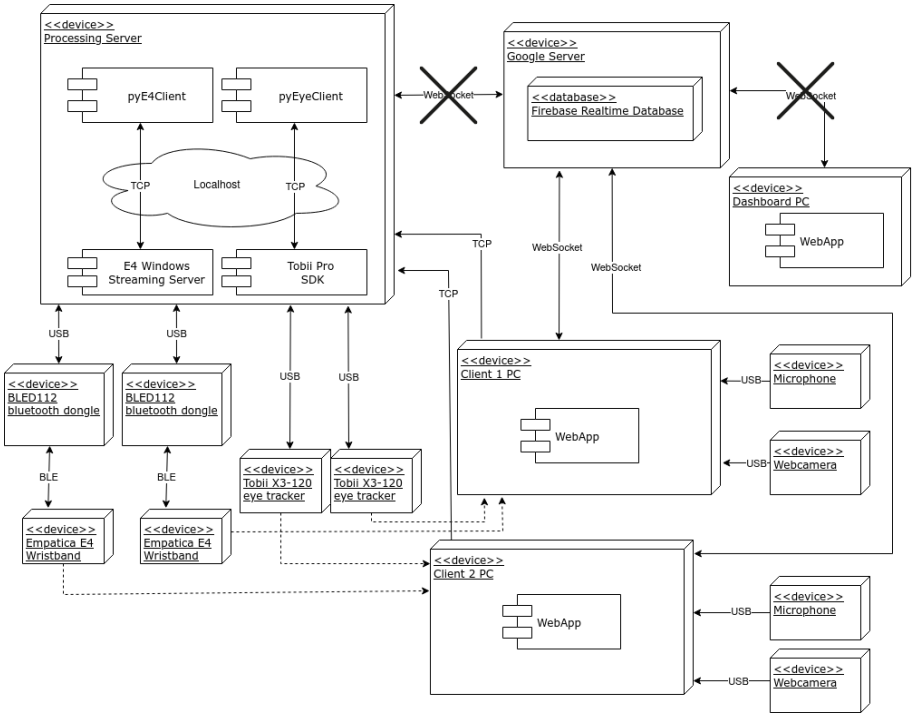


Figure 5: Second version of the system: data stored in firebase limited to logs

3.5.2 Multiple eye-trackers not supported

Data collection devices was only available for a limited time, and the second eye-tracker was received a two weeks prior to starting the data collection. As illustrated in Figure 5, the eye-trackers were planned to be connected to the same computer which used the developed pyEyeClient to collect data from both devices. However, it turned out that connecting multiple eye-tracking devices to the same computer was not supported.

This lack of support for multiple devices was first discovered during calibration in Tobii’s Eye Tracker Manager software where the program would only display a single eye-tracker at a time, regardless of the number of connected devices. This was thought to maybe be due to both devices broadcasting on the same port, however, it could not be concluded as no solution was found for configuring broadcast settings. On the other hand, the issue might not be port-related considering the tobii_research Python-package has a function for finding *all* devices running on a network. If the issue is due to limitations of the eye-tracking devices’ internal software that which cause a conflict when multiple devices

(i.e., eye-trackers) request processing from the same processing device (i.e., computer), a potential solution could be to process the two streams on separate intermediate devices such as Tobii's external processing unit[26]. These devices were not available and therefore this is not tested.

An alternative working solution to this problem is connecting the two eye-trackers to the user's two computers, disable the computers firewall, and then connect with the shared processing computer to capture the data. This was not done in the final implementation (see Figure 6) for security and privacy reasons.

3.5.3 Pygame in debug task

The debug task for data collection involved running a Python-based game through the Pygame-package. Pygame was incompatible with the compiler implemented in the code editor, further described in the end of Section 3.2.1 under *Code compilation*. As a consequence, a server had to be developed (see Section 3.4) so users could run the game. Although this did not affect existing lines of communication in the system, it increased complexity of the architecture.

3.6 Final architecture

Because of the challenges described in the previous subsection it was eventually decided to collect and save all sensor data separately on the two computers. Logs from the code editor are the only data stored in firebase. Data from all the streams are timestamped by the respective computers internal clock, and then synchronized after the data collection in respect to firebase's server time as described in section 4.6.1. Removing the processing computers also increases the need for additional pre-processing of the data before it can be analyzed. The pre-processing is described in Section 4.6.

[26] https://www.tobiipro.com/product-listing/external-processing-unit/

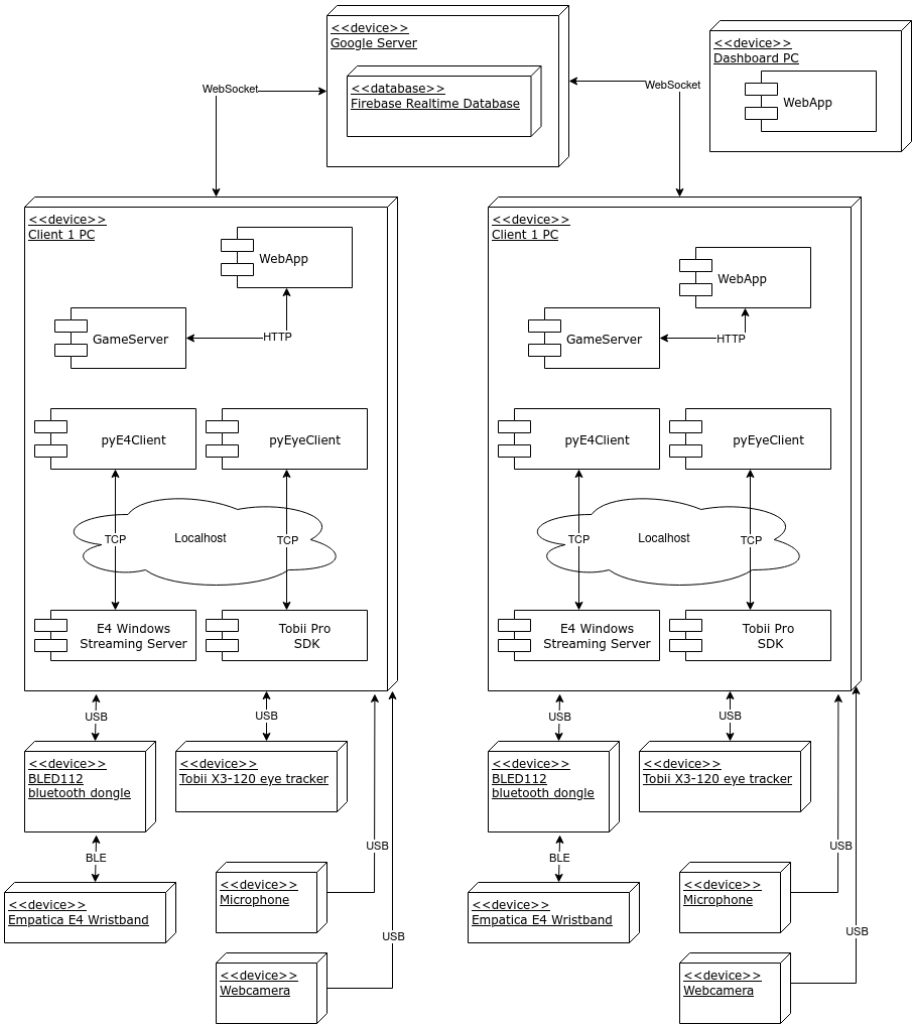


Figure 6: Final implementation - physical view with main processes

Regarding the dashboard; the restrictions to personal data storage broke the initial implementation. Development of a new dashboard not connected to firebase was in progress but due to the many setbacks it was eventually decided to be proposed for further work.

Figure 6 shows the physical view including some main processes in the final implementation.

4 Method

4.1 Context

An experiment was designed to collect quantitative data for a multimodal dataset representing the pair-programming experience. A total of 15 experiments each involving two participants were conducted over the course of two weeks. The experiments consisted

of an individual pre-test followed by a collaborative programming task, both provided by the supervisor of this project who will run further analysis on the collected data after this research project. The pre-test and the collaborative programming task, further referred to as the *debug task*, can be found in appendix A and B, respectively. The debug task in the appendix includes all of the six bugs which are marked with code-comments. The comments stating the bugs were not in the original source code used in the experiments, and the participants were unknown of the number of bugs. The debug task was time-limited and the objective was to detect and fix bugs in a *Space Invader* game written in Python, by collaborating in the web-based code editor described in Section 3. Space Invader, illustrated in Figure 7, has a basic concept with limited game controls (space bar and arrow keys) that most are familiar with.

Figure 7: Screenshot of Space Invader

4.2 Participants

A total of 15 pair-programming experiments were conducted. The 30 participants (26 male, 4 female) age range from 22 to 29 years old. The sampling strategy *convenience sampling* was used to recruit participants. The participants were recruited from my own personal network, open calls on social media platforms (Facebook[27], Slack[28]), e-mail, and through previous participants. I collaborated with a group of two fellow students

[27] https://www.facebook.com/
[28] https://slack.com/

conducting a similar experiment, and all the participants contacted via e-mail had given consent after participating in the other groups experiment. The inclusion criteria were programming skills at minimum equivalent to what is expected after the NTNU course Object-Oriented Programming[29], with no exclusion criteria. The majority of the participants consists of computer science students from NTNU, and five (5) of the participants were professional software developers. All participants were compensated with a gift card (Midtbykortet[30]) worth NOK200, and two NOK500 gift cards were given to a randomly selected pair after all the experiments. The experiments were conducted in May, which is a time were a lot of students are busy with exams or gone for the semester. Due to the difficulties of recruiting enough participants and the logistics of scheduling pairs, any previous participant recruiting a pair was awarded with a NOK200 gift card recruitment bonus. This bonus was only active at the end of the data collection phase.

4.3 Settings and Procedure

The experiments took place at a laboratory at NTNU (*UX-lab*). Each experiment started with the participants signing a contract, granting consent to collection, storage, and processing of their personal data. This was followed by an individual pre-test that was time-limited to 10 minutes. Afterwards, both participants were assigned a computer each. The monitors faced opposite ways and blocked the pairs view of each other, in order to simulate a remote collaboration process. The setup including all the devices can be seen in Figure 8.

[29] https://www.ntnu.no/studier/emner/TDT4100
[30] https://midtbyen.no/midtbykortet-gavekort-for-trondheim-sentrum

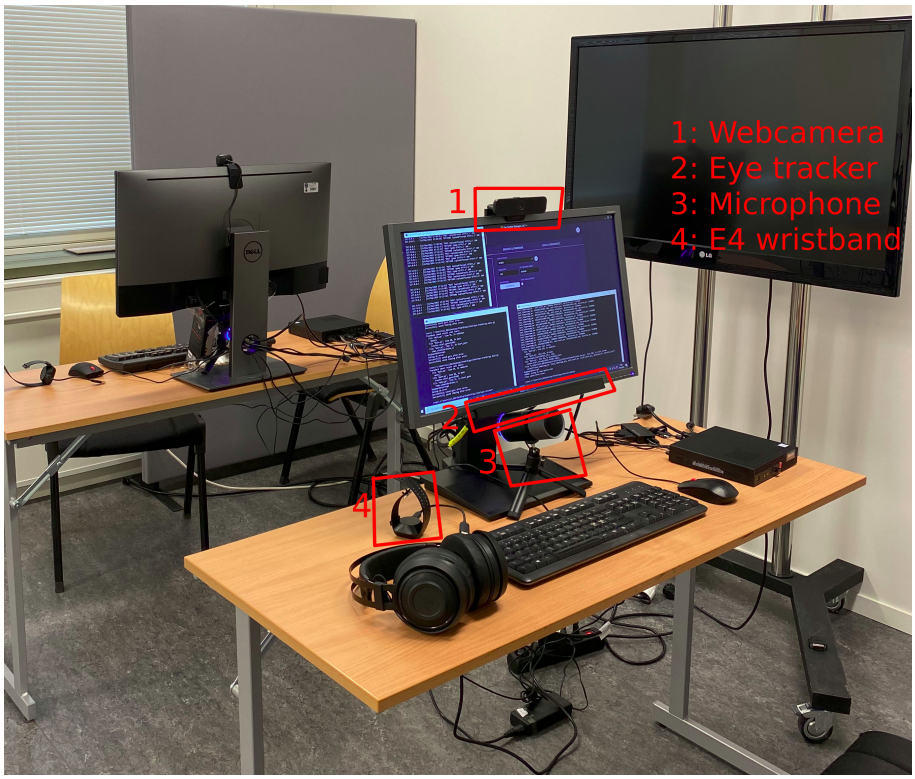


Figure 8: Environment of the experiment

The participants were then helped to put on the Empatica E4 wristbands[31] on their non-dominant hand as recommended in the user manual[32], although current research is conflicting on the importance of this (Schuurmans et al., 2020). The reason for assisting the participants in equipping the wristband is that in the pilot-tests, incorrect placement (e.g., *on* the wrist joint instead of above) were observed to have detrimental effects on the detection of interbeat intervals. Both wristbands were then connected to an E4 streaming server[33] running on the wearers computer. Then, the Tobii X3-120[34] eye-trackers were calibrated with a 7-point calibration using Tobii's Eye Tracker Manager[35]. Height and tilt of the monitors were adjusted before the calibration to assert proper positioning for each participant. The webcamera and microphone were tilted post-calibration for proper positioning.

The participants were subsequently presented with a brief introduction to the code editor where they learned how to run the game and where to look for output. They were then showed a short gameplay preview of how the game was intended to work. The pair were told that the game they receive is broken and their task is to collaborate in finding and

[31] https://www.empatica.com/research/e4/
[32] https://support.empatica.com/hc/en-us/articles/206374015-Wear-your-E4-wristband-#EDA-note
[33] https://developer.empatica.com/windows-streaming-server.html
[34] https://www.tobiipro.com/product-listing/tobii-pro-x3-120/
[35] https://developer.tobiipro.com/eyetrackermanager.html

fixing the bugs breaking the game. Both participants had access to a headset that were optional to use, with no value beyond listening to the game sounds. Two python scripts for collecting wristband and eye-tracking data (described in Section 3.3.1 and 3.3.4, respectively) were then started on both of the computers.

Finally, the pair was informed of the option to ask for help in case they got stuck (and the point reduction that comes along), and started the debug task by clicking a prompt overlaying the editor. The task was time-limited to 25 minutes and a timer was triggered upon starting the test, automatically ending the test if reaching the time limit. Pairs completing the task in less than the available time were ordered to click a button on the E4 wristband to timestamp the end of the session. The pairs spent on average 22 minutes and 29 seconds (SD = 3 minutes 49 seconds) on the task.

4.4 Data collection and measurements

The initial plan for this study included a follow-up qualitative study of a teacher-dashboard visualizing the metrics that were found to be the most significant in *this* study. As a result, the collected data is not limited to the data meant for analysing the collaborative experience, but also includes data necessary to recreate the experiment. The data were collected quantitatively and in the forms of questionnaire data (pre-test), sensor data, and log files. Data were collected from the following sources: pre-test, wristband, webcamera, microphone, eye-tracker, and the code editor. The measurements associated with the collected data are a mix of predefined and post-computed, and is further explained in the subsequent paragraphs. The physical setup of the data collection devices can be seen in Figure 8.

Pre-test

The pre-tests (see appendix A) were carried out on paper and the predefined measurement was the total score. The pre-test was time-limited to 10 minutes and consisted of 10 questions worth one point each.

Wristband data

The python client described in Section 3.3.1 was used to collect data from the E4 wristbands. The data collected from the wristbands were blood volume pulse (BVP) at 64 Hz, galvanic skin response (GSR) at 4 Hz 3-axis acceleration (ACC) at 32 Hz, body-temperature (TEMP) at 4 Hz, interbeat intervals (IBI), and hearth rate (HR). IBI and HR were captured at irregular intervals due to limitations of the wristband. Each and every measurement included a timestamp and a number identifying the user. All variables are

associated with pre-defined measures, and additional measures of change in the variables were post-computed.

Video

Video streams from the webcameras were processed in real-time during the experiments by using the JavaScript API *face-api*[36], so the video were never recorded. Integration and features of face-api are previously described in Seciton 3.3.2. Data were collected from the real-time processing at 1 Hz. The data included a timestamp, a 68-point list of facial landmarks with coordinates, and an estimate for each of seven different emotional states: (1) neutral, (2) happy, (3) sad, (4) angry, (5) fearful, (6) disgusted, and (7) surprised. The values representing emotional state were a predefined measurement. The facial landmarks were only intended for the dashboard and further research with the same dataset.

Audio

Audio was recorded from two Thornmax Pulse[37] microphones; one for each participant. The measurements for sound data were time proportions of silence, one speaking, overlap in speech, and were all post-computed. The audio was recorded by a script in the front-end and automatically began as the participants clicked to start the debug task. The technical details are described in Section 3.3.3.

Gaze data

Data from the eye-tracker was collected through a python client described in Section 3.3.4. The gaze data was collected at 120 Hz using Tobii X3-120 eye-trackers with a 7-point calibration. The gaze data consisted of pupil diameter and gaze point on screen individually for both eyes. In terms of the planned dashboard, all observed gaze points were important for recreating the collaborative process. For this analysis however, all measurements were post-computed. The measurements include cognitive load from pupil data and saccades from gaze points.

Code editor / webapp

Every single change in the code editor (write, remove, replace) were instantly logged to the firebase realtime database with a timestamp and user id. The editor was developed for this project and is based on the text-editor Firepad. Both users selection (i.e., single point, no selection, multiple characters/lines) within the code editor were logged with a timestamp upon change. Implementation and details of the code editor are described in

[36] https://github.com/justadudewhohacks/face-api.js/
[37] https://www.thronmax.com/product/thronmax-pulse/

Section 3. Data was also collected from interactions with elements in the editor that was not part of the *text area* specifically, and includes the timestamp at which a pair started the test, change of user colors with timestamps, timestamp of clicking *Run* and the output, and timestamp of clearing the output by clicking *Clear*. The output from running the program was independent for both users and logged accordingly. Moreover, both users front-end requested timestamps from the database server at 1 Hz to log request times and timestamps, which was later used for synchronization purposes described in Section 4.6.1.

All data collected from the webapp are essential in recreating the process in a dashboard. As the scope of this research changed, the data relevant for this research analysis is limited to the history of events in the code editor, timestamps of test starts, and the synchronization data. The code history was used to recreate the state of the code at the time the test was over, for computing pairs score on the debug task. The start-timestamps and synchronization data were used for trimming the dataset and synchronizing the data sources, and is further described in Section 4.6.2 and 4.6.1, respectively.

4.5 Research Design

The research design of this study is correlation research. All the pairs participating went through the exact same experiment with no manipulation of variables, resulting in no independent variable as all variables are strictly observed. This research design investigates relationships between performance (*i.e., score on the debug task*) and multimodal data, and if a fusion of modalities can be a more accurate predictor for performance than the pre-test. The relationships investigated are indeeed non-casual, as correlations does not necessarily imply causation.

4.6 Pre-processing

This section describes pre-processing of the data and post-computed measurements. The pre-processing was done using Python[38] with the pandas(Reback et al., 2022) library for data manipulation.

4.6.1 Time synchronization

Because all data relevant for the analysis were logged and collected on the two computers separately, each with their own clock, the first step of pre-processing was to synchronize

[38] https://www.python.org/

data from the two users. This involved replacing all timestamps with new *computed* timestamps of estimated server time.

As described under *Code editor* in section 4.4, both computers logged their timestamp, server timestamp, and request time in ms at 1 Hz. This process consisted of three steps:

(1) : Client requests server at client timestamp T_0

(2) : Server receives request, responds with servers timestamp S

(3) : Client receives response at client timestamp T_1

This was used to compute estimates for expected server timestamps (E_S) by using Cristian's algorithm Cristian (1989) for probabilistic clock synchronization.

$$E_S = T_0 + (T_1 - T_0)/2$$

Briefly explained, the expected server timestamp is the clients' timestamp plus half of the request time, i.e., an estimated time at which the server is expected to receive the request. The algorithm is *probabilistic* and the request-response times are not always equal and may vary, so the results are not error-free. The expected server timestamps were then used to find the time offset (D) between client and server, by subtracting the clients expected server time from the actual server time:

$$D = S - E_S$$

Before actually computing the estimated server time described above, the collection of client-server timestamps were split by minute-intervals where each new collection represented the data for the respective minute since start of the test. Then, for each minute-collection, only the row with the lowest *request-response* time were kept. The reason for this is that Cristian's algorithm is more accurate for lower request-response times as it is an estimation. *Then*, the calculations above for expected server time and offset were done for the one remaining row in each of the minute-collections.

The process described above resulted in *one* value for *each* minute representing the client-server offset using the lowest request-response time. The offset was observed to drift 0.2-1 ms per minute, totalling 5-25 ms over a full 25 minute test duration. This drift can be due to drift in the clients internal clock, but may also be due to performance issues of the webapp as the timestamps were sent from the front-end. However, this drift was considered negligible in this research. The mean of the offsets (one value for each minute)

were used as a single value representing the client-server offset. The original timestamps (T_x) of all sensor-data were then replaced with computed timestamps (T_y) by adding the offset mean (D_m).

$$T_y = T_x + D_m$$

Computing server offset and modifying timestamps were done *individually* for each of the participants. This process resulted in all of a users data being synchronized to the server time. Post-synchronization of both users data with the shared server removes the need for a designated processing server, effectively cutting a line of communication. Collecting data separately and directly on the computers can be beneficial in situations with limited network capacity or large amounts of data. Moreover, particularly for experiments of longer duration, the internal clock drift of each computer can be observed and taken into account.

4.6.2 Trimming data

Data collection from the different devices were manually started and stopped and therefore the dataset included a few observations from before and after the debug task. *Start* and *end* timestamp of each experiment were used to remove all data before and after, respectively. The start-timestamp used was the timestamp logged to the database when a pair clicked to start the debug task. The experiment ended automatically 25 minutes after the start, logging the end-timestamp to the database (although this was superfluous considering the known start-timestamp). Pairs finishing the debug task in less than the available time were asked to click a button on the wristband that marked the end of their test.

4.6.3 Debug score

The history of events in the code editor (i.e., changes in code) were used to reproduce the exact state of the code editor at end-time of the debug task. This was used asses the results of each group and give scores. The debug task had six (6) bugs, all worth the same amount of points. The task, including the bugs marked by comments, is found in appendix B. Each pairs' final score (S) were calculated by dividing the sum of points (P by time used (i.e., seconds available (S_0) - seconds remaining (S_1)) to credit pairs completing the task early.

$$S = P/(S_0 - S_1)$$

4.6.4 Audio measures

The Python library PyDub[39] was used for processing the audio files. The measurements, as stated in 4.4, were the amount of silence, one speaking, and speech overlap for each minute. The audio data were compressed during the data collection to lower the file size as it was sent to a server. Compressed audio has a reduced dynamic range of signals which evens out quiet and loud elements, and is highly undesirable for this analysis as each microphone is meant to only capture audio from one user. Naturally, the audio files were uncompressed before post-computing the measures.

First, silence intervals for each users were detected using Pydub.silence as shown in code listing 3. The minimum length of a silence interval was set to 1 second (1000 ms) to not incorrectly mark slow speech as silence. The threshold for detecting silence was set to volume at or below -30 dBFS (i.e., 30 less than relative maximum input). A threshold at -30 dBFS is rather high, but was necessary due to the microphones weak noise reduction causing the microphones to capture *too much* of the other participants voice. The measures (i.e., silence, one speaking, speech overlap) were the derived from comparing the two participants' silence intervals.

```
Pydub.silence.detect_silence(
        audio_segment, # input
        silence_thresh=-30,
        min_silence_len=1000,
    )
```

Listing 3: Detecting silence

4.6.5 Gaze measures

Post-computed measures for gaze included cognitive load, saccades, and fixations. The metric calculated as an indicator for cognitive load is the Index of Pupillary Activity (IPA) (Duchowski et al., 2018), measured from the frequency of pupil diameter oscillation. The process for this computation were as described by Duchowski et al. (2018). First, any gaze data 200 ms before and after a blink was removed (Jiang et al., 2014). The original code for calculating IPA is found in Duchowski et al. (2018), but was modified (see code listing 4) to work with Python3 and a different input type (pandas.DataFrame). One IPA value was computed for each minute interval of the test, representing the users cognitive load in the respective minute. The two measures used from cognitive load were (1) the pairs' average and (2) the difference in-between the pair. Both of these measures had one value for each minute.

[39] https://github.com/jiaaro/pydub/

```
import math
import pywt
import numpy as np


def modmax(d):
    # compute signal modulus
    m = [0.0] * len(d)
    for i in range(len(d)):
        m[i] = math.fabs(d[i])

    # if value is larger than both neighbours , and strictly
    # larger than either , then it is a local maximum
    t = [0.0] * len(d)
    for i in range(len(d)):
        ll = m[i - 1] if i >= 1 else m[i]
        oo = m[i]
        rr = m[i + 1] if i < len(d) - 2 else m[i]
        if (ll <= oo and oo >= rr) and (ll < oo or oo > rr):
            # compute magnitude
            t[i] = math.sqrt(d[i] ** 2)
        else:
            t[i] = 0.0
    return t


def ipa(d):
    # obtain 2- level DWT of pupil diameter signal d
    try:
        (cA2, cD2, cD1) = pywt.wavedec(d.pupildata.values, "sym16", "
    per", level=2)
    except ValueError:
        return

    # get signal duration (in seconds )
    tt = ((d.timestamp.values[-1] - d.timestamp.values[0]).item()) /
    1000

    # normalize by 1/2 j , j = 2 for 2- level DWT
    cA2[:] = [x / math.sqrt(4.0) for x in cA2]
    cD1[:] = [x / math.sqrt(2.0) for x in cD1]
    cD2[:] = [x / math.sqrt(4.0) for x in cD2]

    # detect modulus maxima , see Listing 2
    cD2m = modmax(cD2)

    # threshold using universal threshold  univ  =          (2 log n)
```

```
    # where      is the standard deviation of the noise
     univ  = np.std(cD2m) * math.sqrt(2.0 * np.log2(len(cD2m)))
    cD2t = pywt.threshold(cD2m,  univ , mode="hard")

    # compute IPA
    ctr = 0
    for i in range(len(cD2t)):
        if math.fabs(cD2t[i]) > 0:
            ctr += 1
    IPA = float(ctr) / tt
    return IPA
```

Listing 4: Modified implementation of IPA (Duchowski et al., 2018)

Saccades and fixations were identied using velocity-threshold fixation identification (I-VT) (Salvucci and Goldberg, 2000). This identification method involves measuring the duration of and distance between a users gaze points on the display, and uses the user-monitor distance to compute the velocity of eye movements in degrees/sec. Eye movements with speed (i.e., distance/time) higher than a set threshold are labeled saccades, while the rest (speed lower) are labeled fixations. The monitors used in the experiments were borrowed from the university and of unequal size in cm, and they were unfortunately not measured. However, both monitors had the same resolution of 1920x1080 pixels. Therefore, the velocity metric used for identifying saccadic eye movements was pixels/sec as opposed to the traditional degrees/sec.

The process of computing velocity is illustrated in code listing 5. Velocity was only computed for two consecutive valid values. The x and y gaze coordinates are relative numbers within the [0, 1] interval and were multiplied by monitors resolution x and y resolution.

```
# df = dataframe of gaze points

# If current and last gaze point are both valid (blinks or missing data
    = invalid)
if df.at[index, "gaze_validity"] and df.at[index-1, "gaze_validity"]:
    # Compute distance: gaze_point_x and _y are values in the [0,1]
   range
    last_x = df.at[index - 1, "gaze_point_x"] * 1920
    last_y = df.at[index - 1, "gaze_point_y"] * 1080
    now_x = df.at[index, "gaze_point_x"] * 1920
    now_y = df.at[index, "gaze_point_y"] * 1080
    distance = math.sqrt(
        math.pow(now_x - last_x, 2) + math.pow(now_y - last_y, 2)
    )
    # Velocity (pixels / sec)
```

```
14      velocity = distance / (df.at[index, "timestamp"] - df.at[index - 1,
        "timestamp"])
```

Listing 5: Computing velocity of eye-movements

The velocity threshold for labeling eye-movements saccadic depends on multiple factors as sampling frequency and distance to monitor, and is often inferred from the collected data (Salvucci and Goldberg, 2000; Sen and Megaw, 1984). The velocity threshold for this research were set to >= 10 px/ms and resulted in an average of 16% labeled as saccades and 84% as fixations.

For each user, the average saccade velocity, saccade-fixation ratio, number of saccades and number of fixations was computed for each minute. The final measures for each group (all with one value for each minute) were the groups average saccade velocity, difference in velocity between users, saccade-fixation ratio, difference in saccade-fixation ratio between users, total number of saccades, and total number of fixations.

4.7 Data Analysis

First, the dataset for each pair was quantified to a single value for each feature. Second, the non-casual relationships between performance (i.e., *debug_score*) and each of the selected features (see Table 1) were then investigated through correlation analysis. The predictive value of fusing all features from the modalities was then computed through a non-linear random forest analysis with cross validation. Last, the random forest analysis was run a second time with the *pretest_score* included to investigate if it could predict with higher accuracy than multimodal data alone.

4.7.1 Final features for analysis

The first step of the data analysis was to reduce the predefined (see Section 4.4) and post-computed (see Section 4.6) measures down to single variables representing the feature, which could be compared to a groups performance (i.e., debug_score). It should be noted that this step was exclusive to sensor data and did not include pretest- and debug_score, and minute-values for some features were computed in the pre-processing (e.g., features from gaze data - see Section 4.6.5). First, for each user, the data collected for each measure was split into minute intervals and then each interval replaced by *one* value; the mean of the entire interval. For the *normal* features (see Table 1), the minute-values for a pair was set to be the average of the two participants in each minute. For features with name including "_change" and "_change_abs", minute-values was the change in a pairs average

between each minute and the absolute of the value, respectively. For features with name ending in "_diff_users", the values were the difference between the participants. For audio features, the minute-values was numbers in the [0,1] range reflecting the relative amount of silence, both speaking and speech overlap in each minute. Finally, a single value for each of a pairs features were computed by taking the mean of the respective features minute-values.

Computing minute-values for each feature instead of simply using the mean of the entire data have several advantages with two of the most important being: (1) additional features as "_change.." can be extracted, and (2) features extracted from data stream with inconsistent logging rate (e.g., HR, eye-tracking) are less sensitive to the inconsistency.

Table 1: All 58 features - predefined and post-computed

Data stream	Feature	Supplemental features
Pre-test	pretest_score	
Logs	debug_score	
Video	neutral	neutral_change, neutral_change_abs
	happy	happy_change, happy_change_abs
	sad	sad_change, sad_change_abs
	angry	angry_change, angry_change_abs
	fearful	fearful_change, fearful_change_abs
	disgusted	disgusted_change, disgusted_change_abs
	surprised	surprised_change, surprised_change_abs
Wristband	hr	hr_change, hr_change_abs
	ibi	ibi_change, ibi_change_abs
	bvp	bvp_change, bvp_change_abs
	gsr	gsr_change, gsr_change_abs
	acc_x	acc_x_change, acc_x_change_abs
	acc_y	acc_y_change, acc_y_change_abs
	acc_z	acc_z_change, acc_z_change_abs
Audio	silence	
	both_speaking	
	one_speaking	
Eye-tracking	cognitive_load_mean	cognitive_load_diff_users
	saccades_velocity	saccades_velocity_diff_users
	saccade_ratio_mean	saccade_ratio_mean_diff_users
	saccade_count	
	fixation_count	

4.7.2 Correlation analysis

The correlation analysis was conducted in Python using the packages pandas(Reback et al., 2022) and SciPy(Virtanen et al., 2020) for data manipulation and statistical ana-

lysis, respectively.

Computing the correlations involved of the following steps:

(1) : Check each variable for normal distribution by running Shapiro-Wilk tests (SHAPIRO and WILK, 1965)

(2) : Compute the correlations between performance and each of the variables - using Pearson (Freedman et al., 2007) if both variables are normal distributed, otherwise Spearman (*Spearman Rank Correlation Coefficient*, 2008)

4.7.3 Regression analysis

The relationship between performance and the fusion of all features was investigated though a non-linear random forest analysis with cross validation, and was done using R (R Core Team, 2021).

Before training the model, the dataset was artificially increased due to the small sample size. Computing the results invovled the following steps:

(1) : Inflate the dataset - for every group of five values in each feature, an additional five were added (*SD = standard deviation*): (1) mean of the values, (2) mean - SD, (3) mean - (2*SD), (4) mean + SD, and (5) mean + (2*SD).

(2) : Then the random forest model using the inflated dataset was trained with a 5-fold cross validation

Subsequently, a second model was trained with pretest-scores included in the dataset.

5 Results

5.1 Distribution of variables

All variables were first checked for normality in Shapiro-Wilk test, with the chosen alpha level .05 (5%). This was done in order to select appropriate correlation tests for each variable. Results from checking normal distributions are grouped by modality and shown in the following tables.

Table 2: Results from checking normal distribution for pretest- and debug-score

Feature	p-value	Normal distributed
debug_score	.553	Yes
pretest_score	.488	Yes

Table 3: Results from checking normal distribution for wristband features

Feature	p-value	Normal distributed
hr	.322	Yes
hr_change	.726	Yes
hr_change_abs	.063	Yes
ibi	.842	Yes
ibi_change	.784	Yes
ibi_change_abs	.327	Yes
bvp	.002	No
bvp_change	.233	Yes
bvp_change_abs	.454	Yes
gsr	.004	No
gsr_change	.006	No
gsr_change_abs	.014	No
temp	.602	Yes
temp_change	.251	Yes
temp_change_abs	.093	Yes
acc_x	$< .001$	No
acc_x_change	.253	Yes
acc_x_change_abs	.193	Yes
acc_y	.841	Yes
acc_y_change	.761	Yes
acc_y_change_abs	.888	Yes
acc_z	.305	Yes
acc_z_change	.659	Yes
acc_z_change_abs	.855	Yes

Table 4: Results from checking normal distribution for eyetracking features

Feature	p-value	Normal distributed
cognitive_load_mean	$< .001$	No
cognitive_load_diff_users	$< .001$	No
saccades_velocity	.048	No
saccades_velocity_diff_users	.449	Yes
saccade_ratio_mean	.37	Yes
saccade_ratio_mean_diff_users	.028	No
saccade_count	.497	Yes
fixation_count	.261	Yes

Table 5: Results from checking normal distribution for video features

Feature	p-value	Normal distributed
neutral	.069	Yes
neutral_change	.333	Yes
neutral_change_abs	.067	Yes
happy	.002	No
happy_change	.531	Yes
happy_change_abs	.038	No
sad	$< .001$	No
sad_change	.021	No
sad_change_abs	$< .001$	No
angry	.001	No
angry_change	$< .001$	No
angry_change_abs	.001	No
fearful	.001	No
fearful_change	$< .001$	No
fearful_change_abs	$< .001$	No
disgusted	.001	No
disgusted_change	$< .001$	No
disgusted_change_abs	$< .001$	No
surprised	$< .001$	No
surprised_change	.02	No
surprised_change_abs	.008	No

Table 6: Results from checking normal distribution for audio features

Feature	p-value	Normal distributed
silence	.888	Yes
both_speaking	.17	Yes
one_speaking	.178	Yes

5.2 Correlation analysis

Pearson- and Spearman-correlation tests were used to investigate the non-causal relationship between debug_score (i.e., measure for performance) and each of the individual features. The distribution of debug scores is found to be normally distributed (see Table 2) so choice of correlation tests was decided by the distribution of the other variable in each comparison. Pearson correlation tests were used in in cases where the other variable was also normally distributed, and Spearman tests for the remaining. The alpha value was set to .005 (5%) in both tests.

pretest_score

Since both debug_score and pretest_score are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and pretest_score (see Figure 9) is not statistically significant,

$$r(13) = .43, p = .107$$

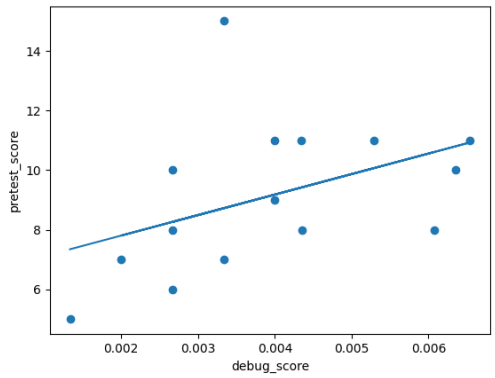


Figure 9: Scatter plot of **debug_score** and **pretest_score** with a linear regression line

neutral

Since both debug_score and neutral are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and neutral (see Figure 10) is not statistically significant,

$$r(13) = -.28, p = .320$$

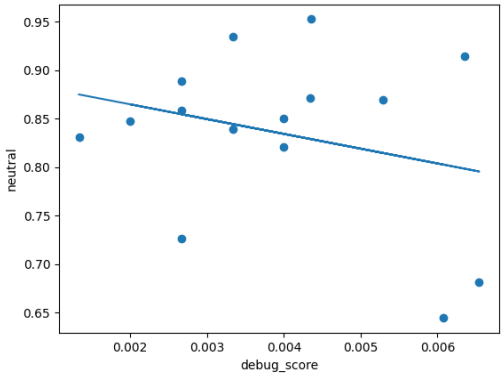


Figure 10: Scatter plot of **debug_score** and **neutral** with a linear regression line

neutral_change

Since both debug_score and neutral_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and neutral_change (see Figure 11) is not statistically significant,

$$r(13) = -.08, p = .786$$

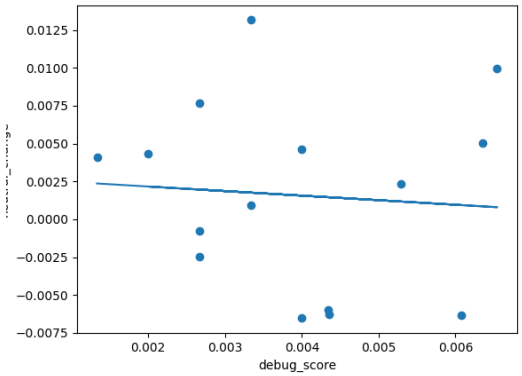


Figure 11: Scatter plot of **debug_score** and **neutral_change** with a linear regression line

neutral_change_abs

Since both debug_score and neutral_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and neutral_change_abs (see Figure 12) is not statistically significant,

$$r(13) = .14, p = .617$$

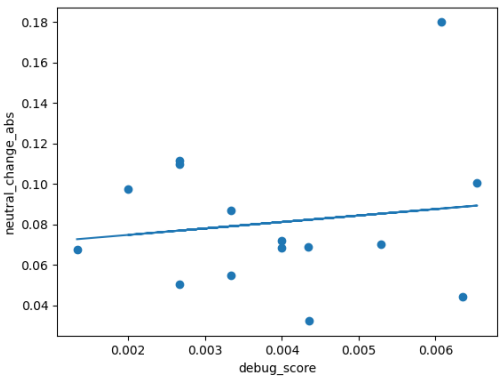


Figure 12: Scatter plot of **debug_score** and **neutral_change_abs** with a linear regression line

happy

Since happy is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and happy. The correlation between debug_score and happy (see Figure 13) is not statistically significant,

$$r(13) = -.2, p = .476$$

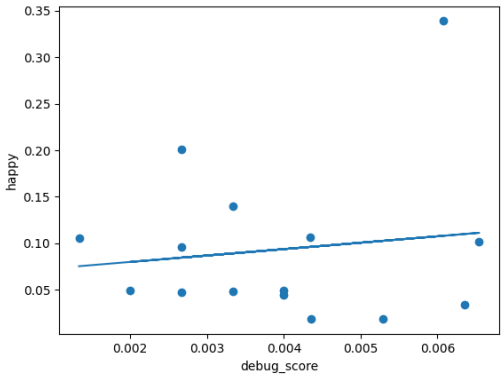


Figure 13: Scatter plot of **debug_score** and **happy** with a linear regression line

happy_change

Since both debug_score and happy_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and happy_change (see Figure 14) is not statistically significant,

$$r(13) = .37, p = .170$$

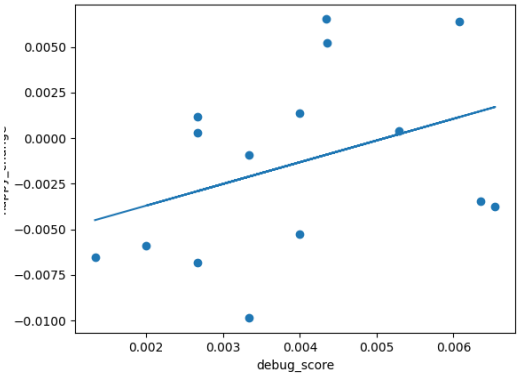


Figure 14: Scatter plot of **debug_score** and **happy_change** with a linear regression line

happy_change_abs

Since happy_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and happy_change_abs. The correlation between debug_score and happy_change_abs (see Figure 15) is not statistically significant,

$$r(13) = -.19, p = .497$$

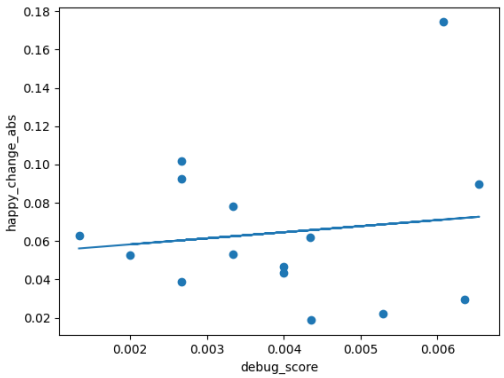


Figure 15: Scatter plot of **debug_score** and **happy_change_abs** with a linear regression line

sad

Since sad is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and sad. The correlation between debug_score and sad (see Figure 16) is not statistically significant,

$$r(13) = .02, p = .934$$

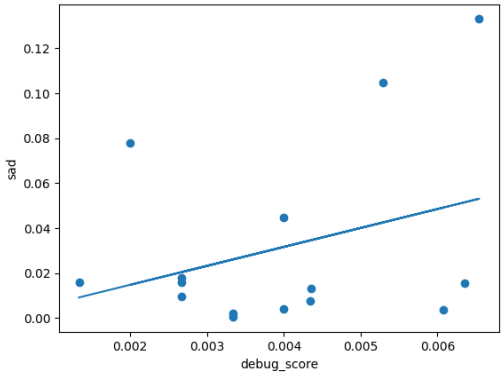


Figure 16: Scatter plot of **debug_score** and **sad** with a linear regression line

sad_change

Since sad_change is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and sad_change. There is is a moderate negative correlation between debug_score and sad_change (see Figure 17),

$$r(13) = -.66, p = .007$$

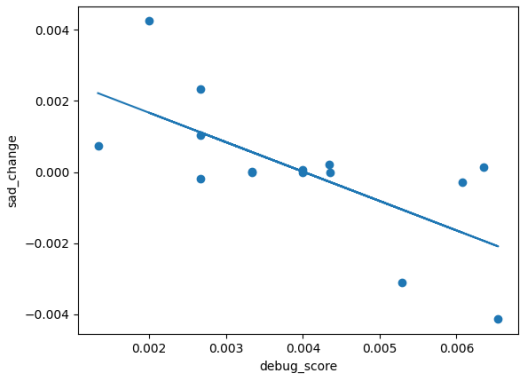


Figure 17: Scatter plot of **debug_score** and **sad_change** with a linear regression line

sad_change_abs

Since sad_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and sad_change_abs. The correlation between debug_score and sad_change_abs (see Figure 18) is not statistically significant,

$$r(13) = .17, p = .552$$

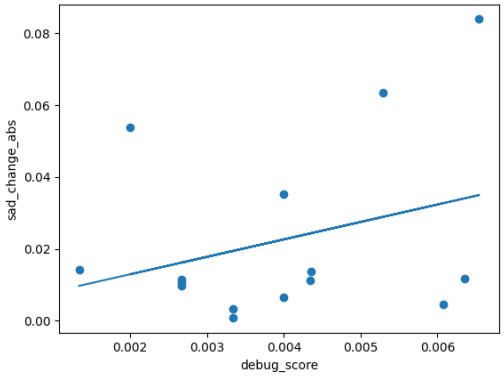


Figure 18: Scatter plot of **debug_score** and **sad_change_abs** with a linear regression line

angry

Since angry is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and angry. The correlation between debug_score and angry (see Figure 19) is not statistically significant,

$$r(13) = .34, p = .216$$

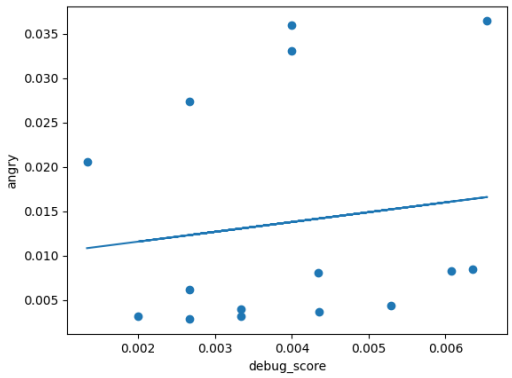


Figure 19: Scatter plot of **debug_score** and **angry** with a linear regression line

angry_change

Since angry_change is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and angry_change. The correlation between debug_score and angry_change (see Figure 20) is not statistically significant,

$$r(13) = -.11, p = .693$$

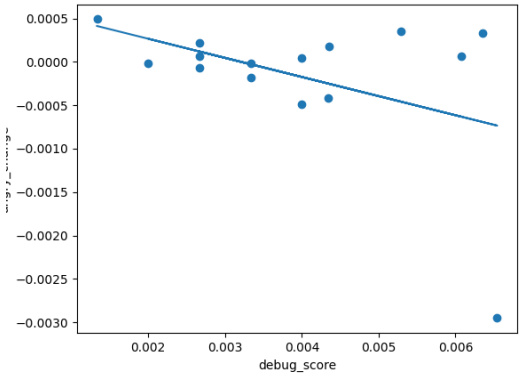


Figure 20: Scatter plot of **debug_score** and **angry_change** with a linear regression line

angry_change_abs

Since angry_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and angry_change_abs. The correlation between debug_score and angry_change_abs (see Figure 21) is not statistically significant,

$$r(13) = .16, p = .561$$

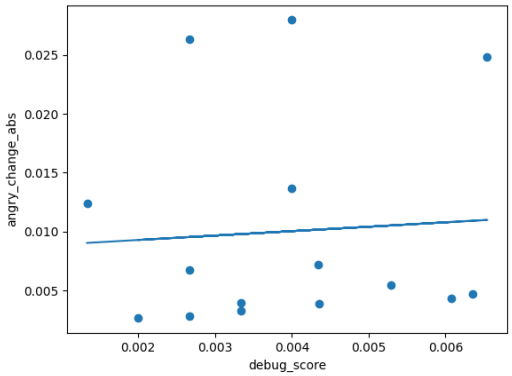


Figure 21: Scatter plot of **debug_score** and **angry_change_abs** with a linear regression line

fearful

Since fearful is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and fearful. The correlation between debug_score and fearful (see Figure 22) is not statistically significant,

$$r(13) = -.42, p = .122$$

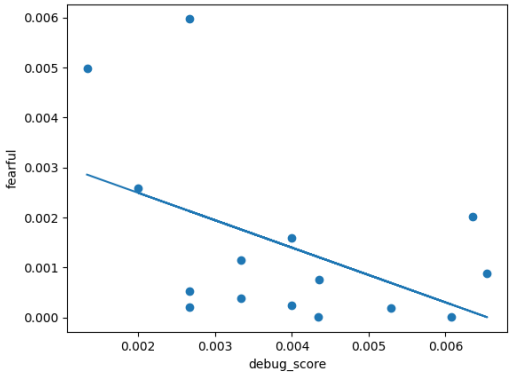


Figure 22: Scatter plot of **debug_score** and **fearful** with a linear regression line

fearful_change

Since fearful_change is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and fearful_change. The correlation between debug_score and fearful_change (see Figure 23) is not statistically significant,

$$r(13) = -.46, p = .086$$

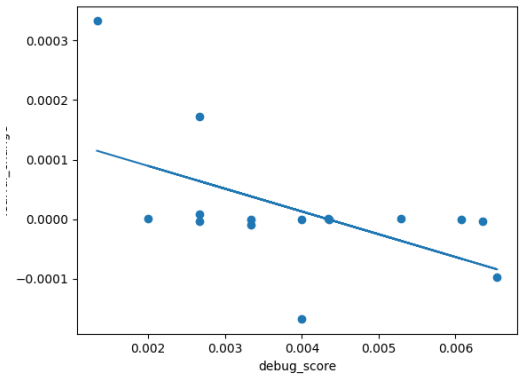


Figure 23: Scatter plot of **debug_score** and **fearful_change** with a linear regression line

fearful_change_abs

Since fearful_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and fearful_change_abs. The correlation between debug_score and fearful_change_abs (see Figure 24) is not statistically significant,

$$r(13) = -.46, p = .085$$

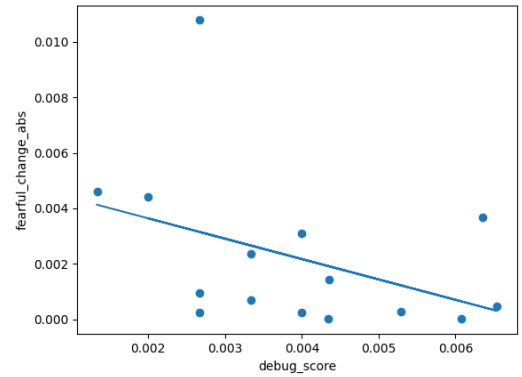


Figure 24: Scatter plot of **debug_score** and **fearful_change_abs** with a linear regression line

disgusted

Since disgusted is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and disgusted. The correlation between debug_score and disgusted (see Figure 25) is not statistically significant,

$$r(13) = -.19, p = .489$$

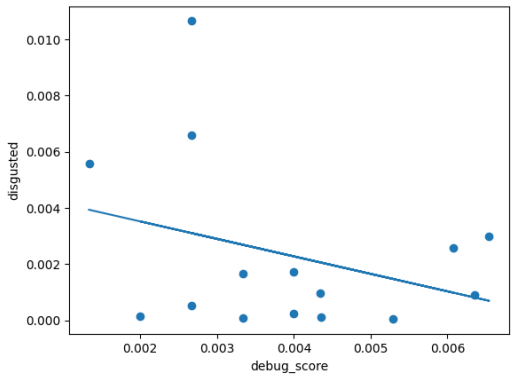


Figure 25: Scatter plot of **debug_score** and **disgusted** with a linear regression line

disgusted_change

Since disgusted_change is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and disgusted_change. The correlation between debug_score and disgusted_change (see Figure 26) is not statistically significant,

$$r(13) = -.22, p = .437$$

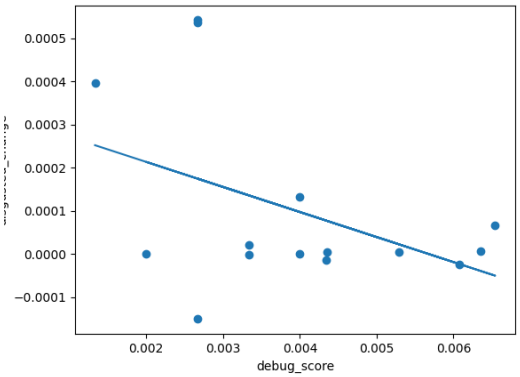


Figure 26: Scatter plot of **debug_score** and **disgusted_change** with a linear regression line

disgusted_change_abs

Since disgusted_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and disgusted_change_abs. The correlation between debug_score and disgusted_change_abs (see Figure 27) is not statistically significant,

$$r(13) = -.21, p = .452$$

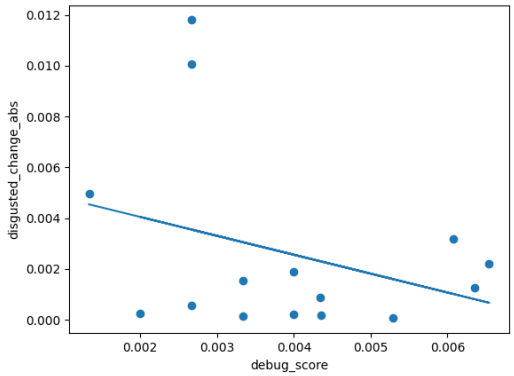


Figure 27: Scatter plot of **debug_score** and **disgusted_change_abs** with a linear regression line

surprised

Since surprised is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and surprised. The correlation between debug_score and surprised (see Figure 28) is not statistically significant,

$$r(13) = -.14, p = .610$$

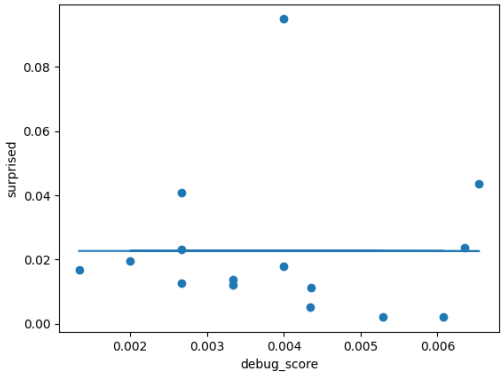


Figure 28: Scatter plot of **debug_score** and **surprised** with a linear regression line

surprised_change

Since surprised_change is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and surprised_change. The correlation between debug_score and surprised_change (see Figure 29) is not statistically significant,

$$r(13) = .34, p = .208$$

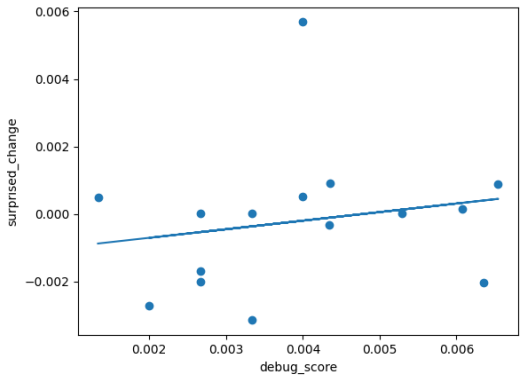


Figure 29: Scatter plot of **debug_score** and **surprised_change** with a linear regression line

surprised_change_abs

Since surprised_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and surprised_change_abs. The correlation between debug_score and surprised_change_abs (see Figure 30) is not statistically significant,

$$r(13) = -.13, p = .646$$

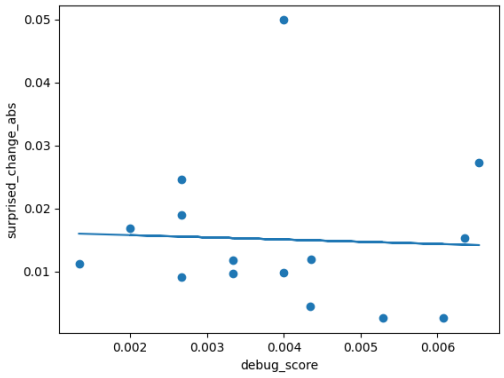


Figure 30: Scatter plot of **debug_score** and **surprised_change_abs** with a linear regression line

hr

Since both debug_score and hr are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and hr (see Figure 31) is not statistically significant,

$$r(13) = .46, p = .088$$

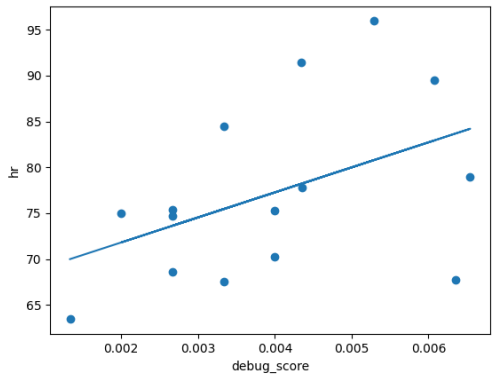


Figure 31: Scatter plot of **debug_score** and **hr** with a linear regression line

hr_change

Since both debug_score and hr_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and hr_change (see Figure 32) is not statistically significant,

$$r(13) = -.13, p = .638$$

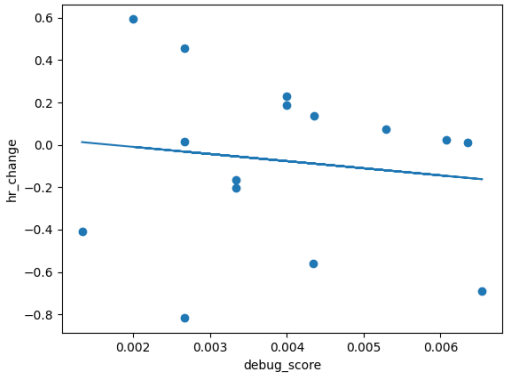


Figure 32: Scatter plot of **debug_score** and **hr_change** with a linear regression line

hr_change_abs

Since both debug_score and hr_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and hr_change_abs (see Figure 33) is not statistically significant,

$$r(13) = -.16, p = .563$$

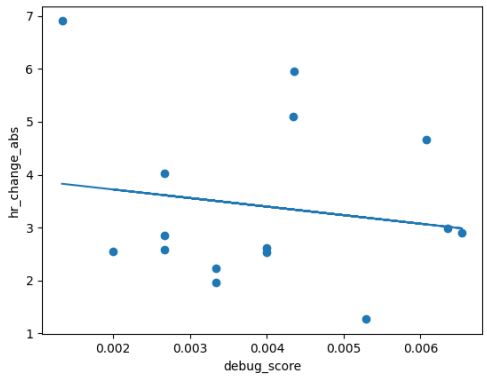


Figure 33: Scatter plot of **debug_score** and **hr_change_abs** with a linear regression line

ibi

Since both debug_score and ibi are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and ibi (see Figure 34) is not statistically significant,

$$r(13) = -.47, p = .074$$

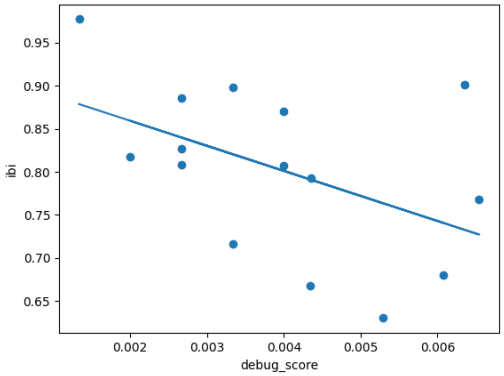


Figure 34: Scatter plot of **debug_score** and **ibi** with a linear regression line

ibi_change

Since both debug_score and ibi_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and ibi_change (see Figure 35) is not statistically significant,

$$r(13) = .02, p = .940$$

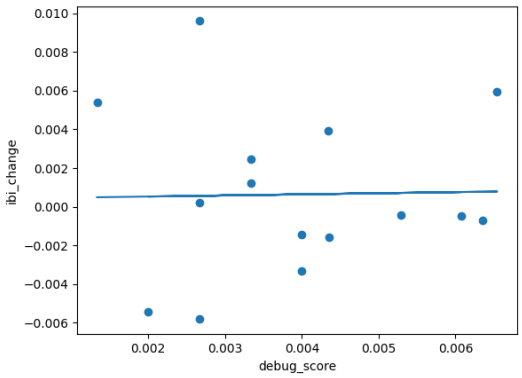


Figure 35: Scatter plot of **debug_score** and **ibi_change** with a linear regression line

ibi_change_abs

Since both debug_score and ibi_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and ibi_change_abs (see Figure 36) is not statistically significant,

$$r(13) = -.24, p = .384$$

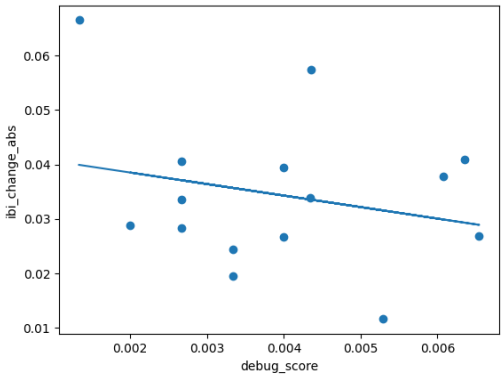


Figure 36: Scatter plot of **debug_score** and **ibi_change_abs** with a linear regression line

bvp

Since bvp is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and bvp. The correlation between debug_score and bvp (see Figure 37) is not statistically significant,

$$r(13) = -.03, p = .914$$

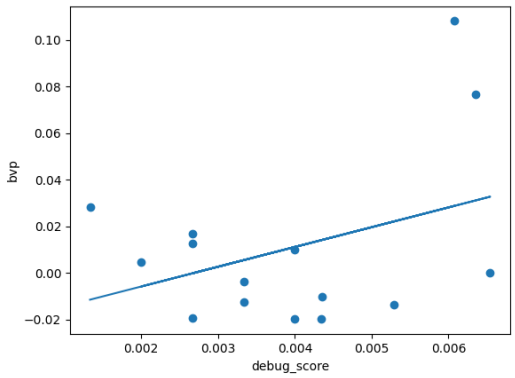


Figure 37: Scatter plot of **debug_score** and **bvp** with a linear regression line

bvp_change

Since both debug_score and bvp_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and bvp_change (see Figure 38) is not statistically significant,

$$r(13) = .43, p = .110$$

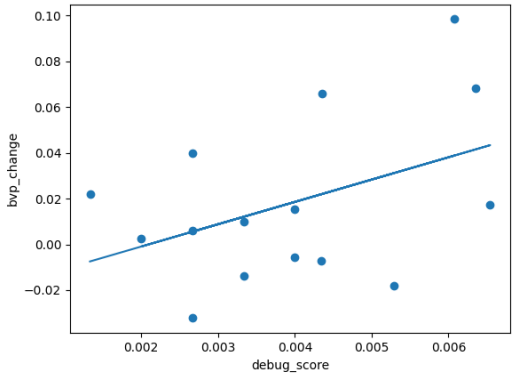


Figure 38: Scatter plot of **debug_score** and **bvp_change** with a linear regression line

bvp_change_abs

Since both debug_score and bvp_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and bvp_change_abs (see Figure 39) is not statistically significant,

$$r(13) = -.47, p = .079$$

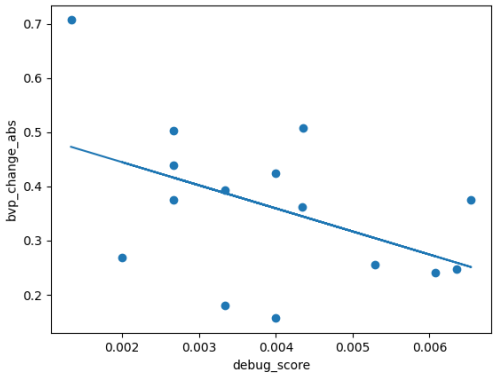


Figure 39: Scatter plot of **debug_score** and **bvp_change_abs** with a linear regression line

gsr

Since gsr is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and gsr. The correlation between debug_score and gsr (see Figure 40) is not statistically significant,

$$r(13) = .06, p = .824$$

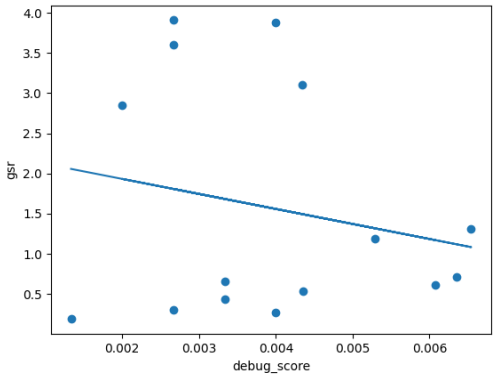


Figure 40: Scatter plot of **debug_score** and **gsr** with a linear regression line

gsr_change

Since gsr_change is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and gsr_change. The correlation between debug_score and gsr_change (see Figure 41) is not statistically significant,

$$r(13) = -.01, p = .959$$

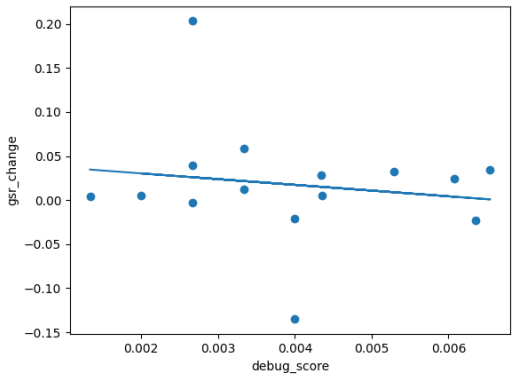


Figure 41: Scatter plot of **debug_score** and **gsr_change** with a linear regression line

gsr_change_abs

Since gsr_change_abs is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and gsr_change_abs. The correlation between debug_score and gsr_change_abs (see Figure 42) is not statistically significant,

$$r(13) = -.12, p = .669$$

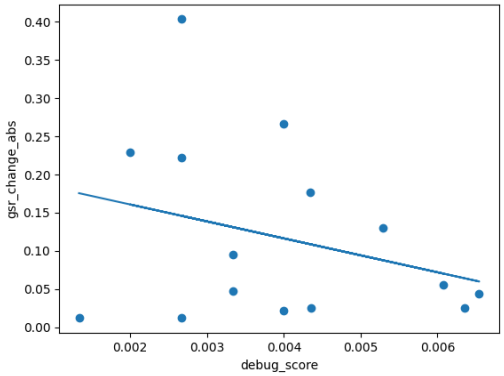


Figure 42: Scatter plot of **debug_score** and **gsr_change_abs** with a linear regression line

temp

Since both debug_score and temp are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and temp (see Figure 43) is not statistically significant,

$$r(13) = .03, p = .916$$

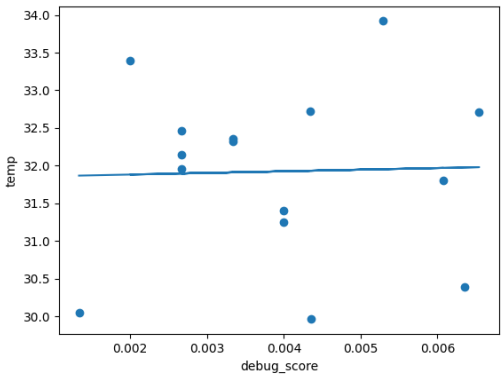


Figure 43: Scatter plot of **debug_score** and **temp** with a linear regression line

temp_change

Since both debug_score and temp_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and temp_change (see Figure 44) is not statistically significant,

$$r(13) = -.14, p = .623$$

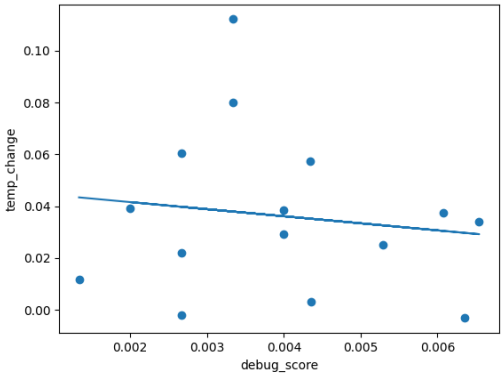


Figure 44: Scatter plot of **debug_score** and **temp_change** with a linear regression line

temp_change_abs

Since both debug_score and temp_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and temp_change_abs (see Figure 45) is not statistically significant,

$$r(13) = -.29, p = .288$$

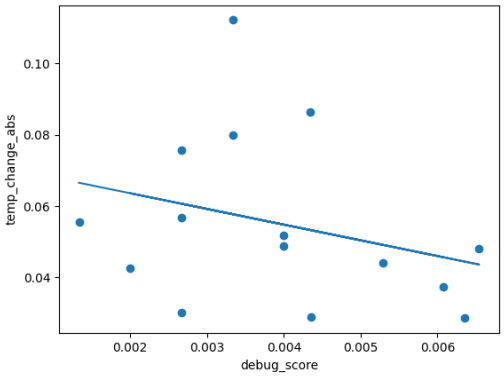


Figure 45: Scatter plot of **debug_score** and **temp_change_abs** with a linear regression line

acc_x

Since acc_x is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and acc_x. The correlation between debug_score and acc_x (see Figure 46) is not statistically significant,

$$r(13) = -.02, p = .949$$

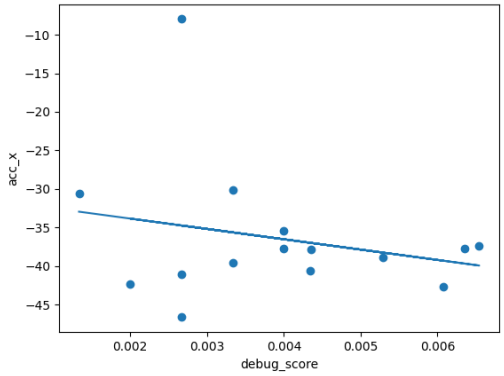


Figure 46: Scatter plot of **debug_score** and **acc_x** with a linear regression line

acc_x_change

Since both debug_score and acc_x_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_x_change (see Figure 47) is not statistically significant,

$$r(13) = -.17, p = .551$$

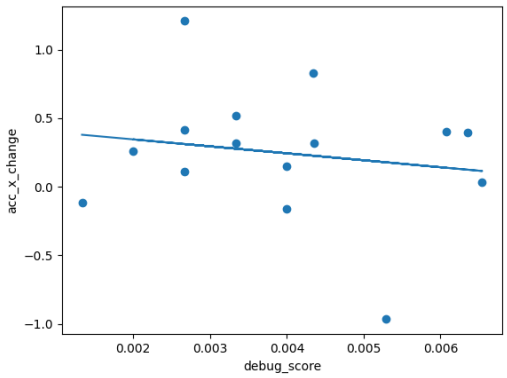


Figure 47: Scatter plot of **debug_score** and **acc_x_change** with a linear regression line

acc_x_change_abs

Since both debug_score and acc_x_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_x_change_abs (see Figure 48) is not statistically significant,

$$r(13) = -.35, p = .196$$

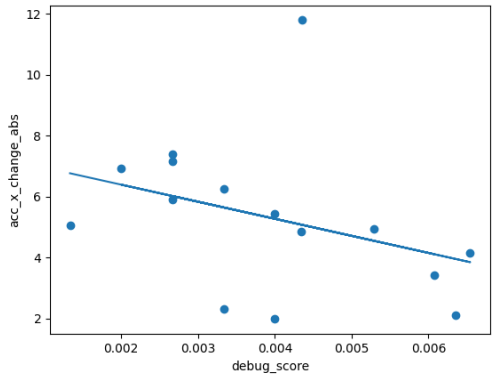


Figure 48: Scatter plot of **debug_score** and **acc_x_change_abs** with a linear regression line

acc_y

Since both debug_score and acc_y are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_y (see Figure 49) is not statistically significant,

$$r(13) = -.2, p = .472$$

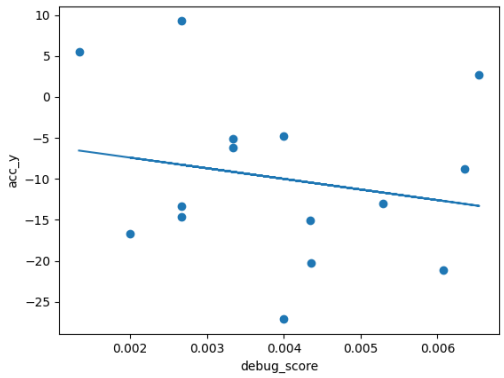


Figure 49: Scatter plot of **debug_score** and **acc_y** with a linear regression line

acc_y_change

Since both debug_score and acc_y_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_y_change (see Figure 50) is not statistically significant,

$$r(13) = .36, p = .183$$

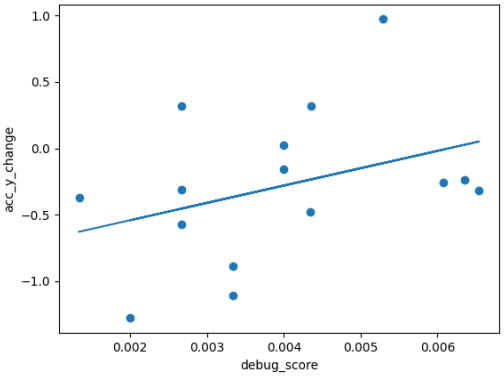


Figure 50: Scatter plot of **debug_score** and **acc_y_change** with a linear regression line

acc_y_change_abs

Since both debug_score and acc_y_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_y_change_abs (see Figure 51) is not statistically significant,

$$r(13) = -.44, p = .099$$

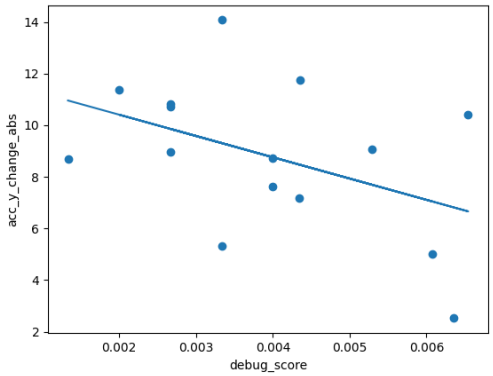


Figure 51: Scatter plot of **debug_score** and **acc_y_change_abs** with a linear regression line

acc_z

Since both debug_score and acc_z are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_z (see Figure 52) is not statistically significant,

$$r(13) = .39, p = .156$$

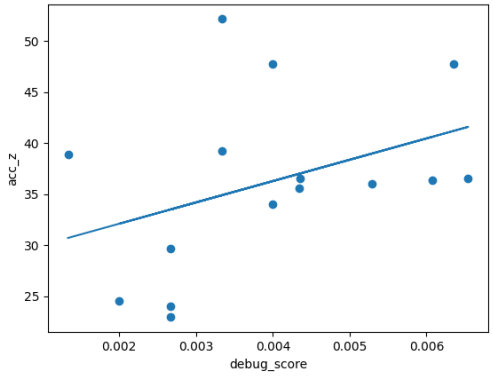


Figure 52: Scatter plot of **debug_score** and **acc_z** with a linear regression line

acc_z_change

Since both debug_score and acc_z_change are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_z_change (see Figure 53) is not statistically significant,

$$r(13) = .21, p = .444$$

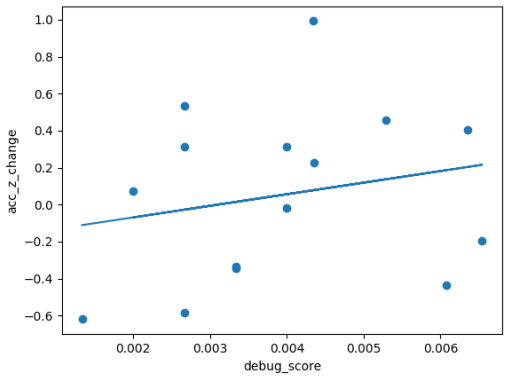


Figure 53: Scatter plot of **debug_score** and **acc_z_change** with a linear regression line

acc_z_change_abs

Since both debug_score and acc_z_change_abs are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and acc_z_change_abs (see Figure 54) is not statistically significant,

$$r(13) = -.22, p = .425$$

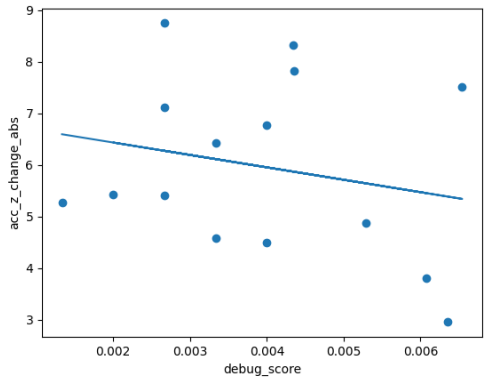


Figure 54: Scatter plot of **debug_score** and **acc_z_change_abs** with a linear regression line

silence

Since both debug_score and silence are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and silence (see Figure 55) is not statistically significant,

$$r(12) = .4, p = .154$$

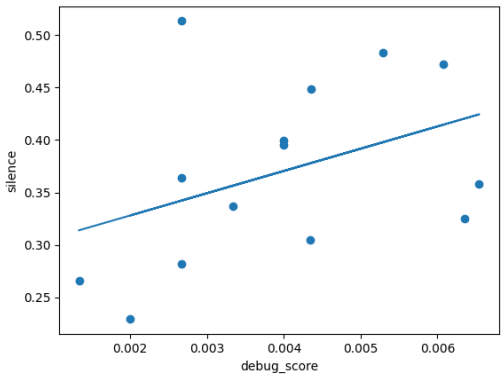


Figure 55: Scatter plot of **debug_score** and **silence** with a linear regression line

both_speaking

Since both debug_score and both_speaking are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. There is is a moderate negative correlation between debug_score and both_speaking (see Figure 56),

$$r(12) = -.55, p = .042$$

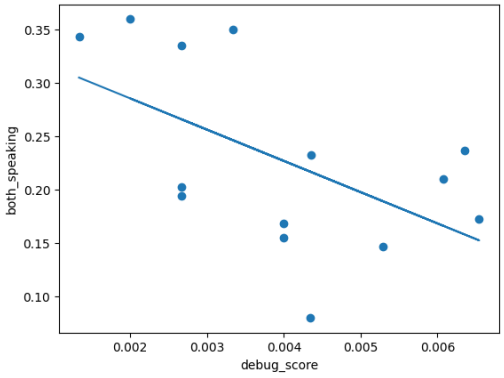


Figure 56: Scatter plot of **debug_score** and **both_speaking** with a linear regression line

one_speaking

Since both debug_score and one_speaking are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and one_speaking (see Figure 57) is not statistically significant,

$$r(12) = .16, p = .592$$

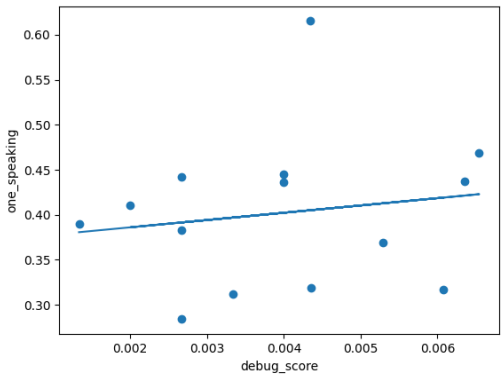


Figure 57: Scatter plot of **debug_score** and **one_speaking** with a linear regression line

cognitive_load_mean

Since cognitive_load_mean is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and cognitive_load_mean. The correlation between debug_score and cognitive_load_mean (see Figure 58) is not statistically significant,

$$r(13) = -.08, p = .770$$

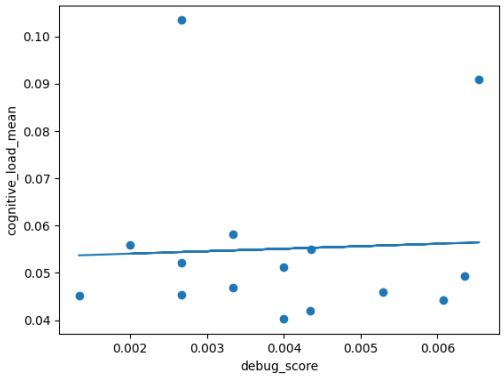


Figure 58: Scatter plot of **debug_score** and **cognitive_load_mean** with a linear regression line

cognitive_load_diff_users

Since cognitive_load_diff_users is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and cognitive_load_diff_users. The correlation between debug_score and cognitive_load_diff_users (see Figure 59) is not statistically significant,

$$r(13) = .25, p = .359$$

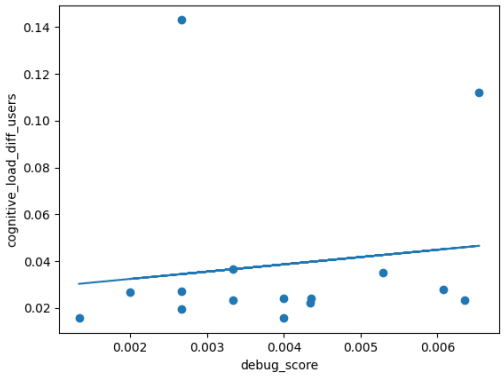


Figure 59: Scatter plot of **debug_score** and **cognitive_load_diff_users** with a linear regression line

saccades_velocity

Since saccades_velocity is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and saccades_velocity. The correlation between debug_score and saccades_velocity (see Figure 60) is not statistically significant,

$$r(13) = -.13, p = .632$$

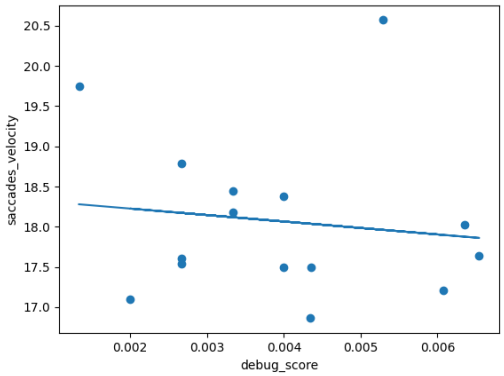


Figure 60: Scatter plot of **debug_score** and **saccades_velocity** with a linear regression line

saccades_velocity_diff_users

Since both debug_score and saccades_velocity_diff_users are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and saccades_velocity_diff_users (see Figure 61) is not statistically significant,

$$r(13) = -.07, p = .792$$

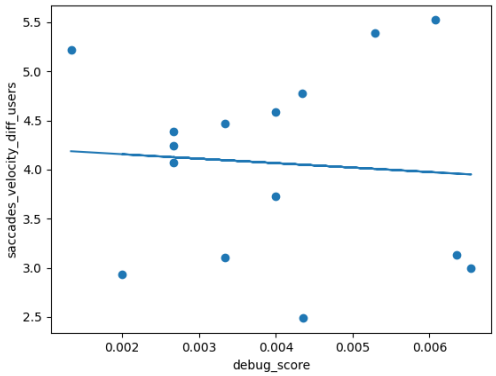


Figure 61: Scatter plot of **debug_score** and **saccades_velocity_diff_users** with a linear regression line

saccade_ratio_mean

Since both debug_score and saccade_ratio_mean are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and saccade_ratio_mean (see Figure 62) is not statistically significant,

$$r(13) = .21, p = .445$$

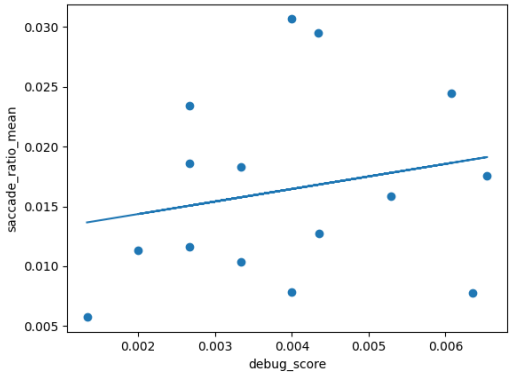


Figure 62: Scatter plot of **debug_score** and **saccade_ratio_mean** with a linear regression line

saccade_ratio_mean_diff_users

Since saccade_ratio_mean_diff_users is not normally distributed, Spearman's rank correlation was computed to assess the relationsihp between debug_score and saccade_ratio_mean_diff_users. The correlation between debug_score and saccade_ratio_mean_diff_users (see Figure 63) is not statistically significant,

$$r(13) = .28, p = .318$$

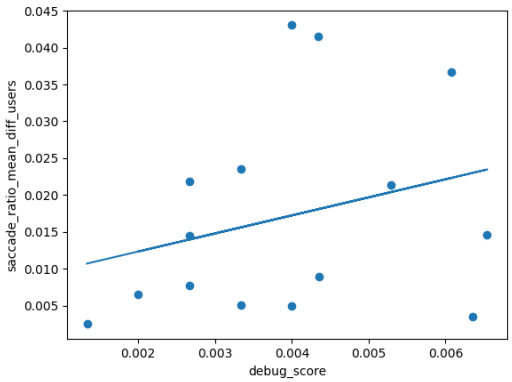


Figure 63: Scatter plot of **debug_score** and **saccade_ratio_mean_diff_users** with a linear regression line

saccade_count

Since both debug_score and saccade_count are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correla-

tion between debug_score and saccade_count (see Figure 64) is not statistically significant,

$$r(13) = -.04, p = .875$$

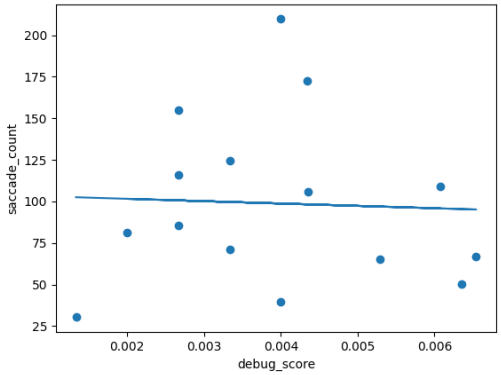


Figure 64: Scatter plot of **debug_score** and **saccade_count** with a linear regression line

fixation_count

Since both debug_score and fixation_count are normally distributed, the Pearson correlation coefficient was computed to assess the linear relationship between them. The correlation between debug_score and fixation_count (see Figure 65) is not statistically significant,

$$r(13) = -.41, p = .130$$

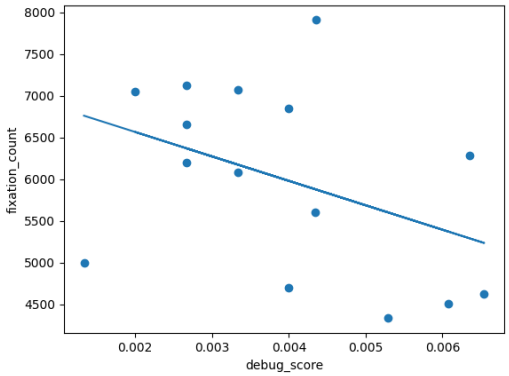


Figure 65: Scatter plot of **debug_score** and **fixation_count** with a linear regression line

5.3 Regression analysis

The random forest algorithm with 5-fold cross validation was used to investigate how the strongly the combination of features (see Section 4.7 can predict pairs' performance. The algorithm was first ran once without including the pre-test data, and a second time with the pre-test included.

First model:
Validation R-squared value = 0.81
Validation normalized root mean squared error (NRMSE) = 0.08
Top 10 most important variables is shown in Table 7 - values are in the [0,100] range and reflect the variable's relative importance (*most* important variable=100, *least* important=0).

Table 7: First run (without pretest) - Top 10 most important variables

	Overall
gsr_change	100.00
hr_change	92.46
cognitive_load_diff_users	89.78
hr	83.09
cognitive_load_mean	79.02
saccades_velocity_diff_users	76.16
saccades_velocity	71.12
one_speaking	65.25
both_speaking	63.60
gsr	60.82

Second model:
Validation R-squared value = 0.82
Validation normalized root mean squared error (NRMSE) = 0.07
See Table 8 for variable importance.

Table 8: Second run (with pretest) - Top 10 most important variables

	Overall
hr_change	100.00
cognitive_load_mean	98.23
pretest_score	79.72
hr	78.68
gsr	77.50
cognitive_load_diff_users	73.56
one_speaking	69.44
both_speaking	65.56
saccades_velocity_diff_users	63.97
saccades_velocity	56.57

6 Discussion

In the correlation analysis, only two variables were found to be significantly correlated with performance: (1) *sad_change* with coefficient -.66, and (2) *both_speaking* with coefficient -.55. The first model in the regression analysis predicts performance with an error-rate of 8% (NRMSE=0.08). The second model, including the pre-test, resulted did not achieve signficantly greater accuraty for predicting performance, and had an error-rate of 7% (NRMSE=0.07). The pre-test was expected to be significantly correlated with performance because it is a measure of prior knowledge (Dochy et al., 1999), but correlation analysis resulted insignificant. For the random forest model, it was expected to be significant due to the inflated dataset and amounts of model training - and was found to be the third most import variable. However, it did not significantly increase prediction performance compared to when it was not included. This means that the multimodal data alone can achieve a high accuracy - and there is very little room for improvement as the multimodal data already explains a large proportion of variance in performance.

Interestingly, gsr_change was the most important variable in the first model, and disappeared from the top 10 in the second – actually down to second *least* important – after including the pre-test. The mutual information between two pairs of variables can reduce the mutual information between other pairs of variables. Including the pre-test in the set of predictors might have similar effect on the mutual information between gsr_change and performance – which means the mutual information between pretest and performance can be partially explained by change in GSR.

One of the features found to be significant in the correlation analysis, *Sad_change* - the average change between each minute - where a higher value indicates the sad-expression

occurring at an increasing rate over the test duration. *Sad* facial expressions is a natural reaction when facing problems that seem overwhelming, and the negative correlation to performance is expected to be due to participants with lower performance feeling gradually more overwhelmed during the debug test.

Students with low performance have been found to be generally less engaged in collaborative activities (Chi and Wylie, 2014). Further, students performing well tend to engage in the collaborative process to a greater extent by sharing ideas and thought process (Hausmann et al., 2004). People with high performance tend to perform less individual problem solving in collaborative settings, and consequently, high performing pairs is expected to have more similar mental effort (Hausmann et al., 2004). The difference in cognitive load between participants in each pairs was the third most important variable in the first model and the 6th most important in the second – complementing current knowledge on the predictive value of mental effort difference in collaboration (Sharma et al., 2021). Moreover, the groups average cognitive load was found to be the third and second most important variable in the first and second model, respectively.

Another interesting find is the second significant correlation from the correlation analysis - *both_speaking*, the relative amount of time with speech overlap – which was found to have a negative correlation with performance. This contradicts current theories of how greater engagement in the collaborative process lead to higher performance (Chi and Wylie, 2014; Hausmann et al., 2004; Sharma et al., 2021). The significance of speech overlaps' affect on performance prediction is further strengthened by the fact it was valued in he top 10 variables from both regression models (9th and 8th most important, respectively). Further adding to the importance of engagement; GSR (*note: referred to as EDA in some studies*) is indicative of engagement and both models considered gsr to be one of the 10 most important variables, and gsr_change the most important by the first model. However, it is unknown how this indicator of engagement (GSR) relates to performance due to the insignificant results in the correlation analysis.

To further elaborate on the results from the audio features, the variables for time ratio of *one*-speaking and speech-overlap (i.e., both_speaking) were both top 10 most important features in both regression models. Audio is one of the least common modalities in learning analytics research, and features are often extract from pitch and sound levels. The significance of features derived from speech-detection in predicting collaborative performance is a novel contribution to the research field.

Heart rate is known to be indicative of physiological stress and high values is expected to negatively affect performance. Although the correlations resulted invalid due to insufficient alpha level, the regression models considered heart rate to be an important feature in

predicting performance: hr ranked fourth most important by both models, and hr_change the most important by the second model.

Saccade velocity and the difference in a pair is indicative of familiarity and was ranked in the top 10 by both models. Low difference for a pair reflects greater engagement in collaboration and is expected to result in higher performance (Sharma et al., 2021; Chi and Wylie, 2014; Hausmann et al., 2004). Saccade-features were found to be important in predicting performance, however, the relationship to performance cannot be further investigated due to insignificant alpha levels from the correlations.

Although the teacher-dashboard implementation was eventually discarded due to unforeseen data storage restrictions (see Section 3.5, these findings show metrics that can be valuable to include in a dashboard for monitoring the collaborative learning experience. Displaying a metric for cognitive load can be helpful so that teachers can intervene and help - when a pair has high mental effort, or motivate – when a pair lack mental effort. A metric for saccadic velocity can be used to detect unfamiliarity which can be reduced by providing the pair with additional tips. Further, metrics from different modalities can be used in the dashboard to simultaneously describe multiple aspects of the learning process; providing teachers a way to immediately detect the exact need of a pair and act accordingly.

6.1 Limitations

A major limitation of this research is the small sample size and was detrimental to the correlation analysis. Another limitation was due to the pairs different social setting in the experiment environment: approximately half of the sample was recruited from the researchers private network, and some participants joined as a pair, while others were matched with strangers. Different social settings where some pairs naturally feel more comfortable is not ideal when physiological data is being collected.

The pre-test measuring participants prior knowledge consisted of ten questions where six of them were multiple-choice type. The multiple-choice questions did not have options for "*I dont know*" which added a degree of randomization to the pre-test results. Although this limitation may be interpreted as speculative, and insignificant for similar studies with a sufficient sample size, the results in this research are most likely affected by the additional randomization due to the very small sample size.

Another limitation of this research is the extraction of eye-movement- and audio-features. The velocity of eye-movements for labeling saccades and fixations was computed in pixels/time, instead of the traditional degress/time (described in Section 4.6.5). In prac-

tice this serves the same purpose, however, the results are less comparable to other studies. Regarding the audio features, the short distance between a pair in combination with the microphones poor noise filtering caused overlap in the audio streams. The speech-detection used a set sound level and had varying accuracy for time segments where the communication were generally louder or quieter.

7 Conclusion and Future Work

First, this work provides insight to the development behind a web-based collaborative code editor, multimodal data collection from a collaborative process, and clock-synchronization of not only the modalities - but also between users. Second, this work provides detailed technical insight in computation of features.

This work shows that fusing features from multiple modalities - (1) eye-tracker, (2) video, (3) audio, (4) wristband - can provide high accuracy for predicting performance in collaborate programming. Last, the most predictive features are discussed in terms of how they can improve the design of learning technologies for the collaborative process.

The first proposal for further work is to train additional models with the random forest algorithm and grouping the models by modality to investigate which being the most predictive.

Second, conduct a similar study with an adequate sample size and investigate the relationship between performance and the features individually.

Last, finish the implementation of the teacher-dashboard visualizing metrics from this study, or from a follow study, and evaluate through a qualitative analysis.

Bibliography

Andrade, A. (2017), Understanding student learning trajectories using multimodal learning analytics within an embodied-interaction learning environment, *in* 'Proceedings of the Seventh International Learning Analytics Knowledge Conference', LAK '17, Association for Computing Machinery, New York, NY, USA, p. 70–79.
URL: *https://doi.org/10.1145/3027385.3027429*

Baheti, P., Gehringer, E. and Stotts, D. (2002), Exploring the efficacy of distributed pair programming, *in* D. Wells and L. Williams, eds, 'Extreme Programming and Agile Methods — XP/Agile Universe 2002', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 208–220.

Baheti, P., Williams, L., Gehringer, E., Stotts, D. and Smith, J. (2002), 'Distributed pair programming: empirical studies and supporting environments', *TR02-010. University of North Carolina at Chapel Hill Dept. of Computer Science* pp. 86–94.

Blikstein, P. and Worsley, M. (2016), 'Multimodal learning analytics and education data mining: using computational technologies to measure complex learning tasks', *Journal of Learning Analytics* **3**(2), 220–238.
URL: *https://www.learning-analytics.info/index.php/JLA/article/view/4383*

Boyer, K. E., Dwight, A. A., Fondren, R. T., Vouk, M. A. and Lester, J. C. (2008), 'A development environment for distributed synchronous collaborative programming', *SIGCSE Bull.* **40**(3), 158–162.
URL: *https://doi.org/10.1145/1597849.1384315*

Chi, M. T. and Wylie, R. (2014), 'The icap framework: Linking cognitive engagement to active learning outcomes', *Educational psychologist* **49**(4), 219–243.

Cowley, B. U., Filetti, M., Lukander, K., Torniainen, J., Helenius, A., Ahonen, L., Barral Mery de Bellegarde, O., Kosunen, I. J., Valtonen, T., Huotilainen, M. J. et al. (2016), 'The psychophysiology primer: a guide to methods and a broad review with a focus on human-computer interaction', *Foundations and Trends in Human-Computer Interaction* .

Cristian, F. (1989), 'Probabilistic clock synchronization', *Distributed Computing* **3**(3), 146–158.
URL: *https://doi.org/10.1007/BF01784024*

Dochy, F., Segers, M. and Buehl, M. M. (1999), 'The relation between assessment practices and outcomes of studies: The case of research on prior knowledge', *Review of educational research* **69**(2), 145–186.

Duchowski, A. T., Krejtz, K., Krejtz, I., Biele, C., Niedzielska, A., Kiefer, P., Raubal, M. and Giannopoulos, I. (2018), The index of pupillary activity: Measuring cognitive load vis-à-vis task difficulty with pupil oscillation, *in* 'Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems', CHI '18, Association for Computing Machinery, New York, NY, USA, p. 1–13.
URL: *https://doi.org/10.1145/3173574.3173856*

Edson, A. J. and Phillips, E. D. (2021), 'Connecting a teacher dashboard to a student digital collaborative environment: supporting teacher enactment of problem-based mathematics curriculum', *ZDM – Mathematics Education* **53**(6), 1285–1298.
URL: *https://doi.org/10.1007/s11858-021-01310-w*

Fan, H., Li, K., Li, X., Song, T., Zhang, W., Shi, Y. and Du, B. (2019), 'Covscode: A novel real-time collaborative programming environment for lightweight ide', *Applied Sciences* **9**(21).
URL: *https://www.mdpi.com/2076-3417/9/21/4642*

Freedman, D., Pisani, R. and Purves, R. (2007), 'Statistics (international student edition)', *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* .

Giannakos, M. N., Sharma, K., Pappas, I. O., Kostakos, V. and Velloso, E. (2019), 'Multimodal data as a means to understand the learning experience', *International Journal of Information Management* **48**, 108–119.
URL: *https://www.sciencedirect.com/science/article/pii/S0268401218312751*

Grinberg, M. (2018), *Flask web development: developing web applications with python*, " O'Reilly Media, Inc.".

Hausmann, R. G., Chi, M. T. and Roy, M. (2004), Learning from collaborative problem solving: An analysis of three hypothesized mechanisms, *in* 'Proceedings of the annual meeting of the cognitive science society', Vol. 26.

Jiang, X., Atkins, M. S., Tien, G., Bednarik, R. and Zheng, B. (2014), Pupil responses during discrete goal-directed movements, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '14, Association for Computing Machinery, New York, NY, USA, p. 2075–2084.
URL: *https://doi.org/10.1145/2556288.2557086*

Jo, C.-H. and Arnold, A. J. (2003), A portable and collaborative distributed programming environment., *in* 'Software Engineering Research and Practice', Citeseer, pp. 198–203.

Johnson, D. W. and Johnson, R. T. (2009), 'An educational psychology success story: Social interdependence theory and cooperative learning', *Educational Researcher*

38(5), 365–379.
URL: *https://doi.org/10.3102/0013189X09339057*

Johnson, D. W., Johnson, R. T. and Smith, K. (2007), 'The state of cooperative learning in postsecondary and professional settings', *Educational Psychology Review* **19**(1), 15–29.
URL: *https://doi.org/10.1007/s10648-006-9038-8*

Kumar, A., Kumar, P., Palvia, S. C. J. and Verma, S. (2017), 'Online education worldwide: Current status and emerging trends', *Journal of Information Technology Case and Application Research* **19**(1), 3–9.
URL: *https://doi.org/10.1080/15228053.2017.1294867*

Lane, H. C. and D'Mello, S. K. (2019), *Uses of Physiological Monitoring in Intelligent Learning Environments: A Review of Research, Evidence, and Technologies*, Springer International Publishing, Cham, pp. 67–86.
URL: *https://doi.org/10.1007/978-3-030-02631-8*$_5$

Liu, X. and Woo, G. (2021), Codehelper: A web-based lightweight ide for e-mentoring in online programming courses, *in* '2021 3rd International Conference on Computer Communication and the Internet (ICCCI)', pp. 220–224.

McDowell, C., Werner, L., Bullock, H. and Fernald, J. (2002), 'The effects of pair-programming on performance in an introductory programming course', *SIGCSE Bull.* **34**(1), 38–42.
URL: *https://doi.org/10.1145/563517.563353*

Nosek, J. T. (1998), 'The case for collaborative programming', *Communications of the ACM* **41**(3), 105–108.

Nüssli, M.-A. (2011), Dual eye-tracking methods for the study of remote collaborative problem solving, Technical report, EPFL.

Ochoa, X. and Worsley, M. (2016), 'Augmenting learning analytics with multimodal sensory data', *Journal of Learning Analytics* **3**(2), 213–219.

R Core Team (2021), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
URL: *https://www.R-project.org/*

Reback, J., jbrockmendel, McKinney, W., den Bossche, J. V., Augspurger, T., Roeschke, M., Hawkins, S., Cloud, P., gfyoung, Sinhrks, Hoefler, P., Klein, A., Petersen, T., Tratner, J., She, C., Ayd, W., Naveh, S., Darbyshire, J., Garcia, M., Shadrach, R., Schendel,

J., Hayden, A., Saxton, D., Gorelli, M. E., Li, F., Zeitlin, M., Jancauskas, V., McMaster, A., Wörtwein, T. and Battiston, P. (2022), 'pandas-dev/pandas: Pandas 1.4.2'.
URL: *https://doi.org/10.5281/zenodo.6408044*

Salvucci, D. and Goldberg, J. (2000), Identifying fixations and saccades in eye-tracking protocols, pp. 71–78.

Schuurmans, A. A. T., de Looff, P., Nijhof, K. S., Rosada, C., Scholte, R. H. J., Popma, A. and Otten, R. (2020), 'Validity of the empatica e4 wristband to measure heart rate variability (hrv) parameters: a comparison to electrocardiography (ecg)', *Journal of Medical Systems* **44**(11), 190.
URL: *https://doi.org/10.1007/s10916-020-01648-w*

Schwendimann, B. A., Rodríguez-Triana, M. J., Vozniuk, A., Prieto, L. P., Boroujeni, M. S., Holzer, A., Gillet, D. and Dillenbourg, P. (2017), 'Perceiving learning at a glance: A systematic literature review of learning dashboard research', *IEEE Transactions on Learning Technologies* **10**(1), 30–41.

Sen, T. and Megaw, T. (1984), The effects of task variables and prolonged performance on saccadic eye movement parameters, *in* A. G. Gale and F. Johnson, eds, 'Theoretical and Applied Aspects of Eye Movement Research', Vol. 22 of *Advances in Psychology*, North-Holland, pp. 103–111.
URL: *https://www.sciencedirect.com/science/article/pii/S0166411508618245*

SHAPIRO, S. S. and WILK, M. B. (1965), 'An analysis of variance test for normality (complete samples)', *Biometrika* **52**(3-4), 591–611.
URL: *https://doi.org/10.1093/biomet/52.3-4.591*

Sharma, K. and Giannakos, M. (2020), 'Multimodal data capabilities for learning: What can multimodal data tell us about learning?', *British Journal of Educational Technology* **51**(5), 1450–1484.
URL: *https://bera-journals.onlinelibrary.wiley.com/doi/abs/10.1111/bjet.12993*

Sharma, K., Olsen, J. K., Verma, H., Caballero, D. and Jermann, P. (2021), Challenging joint visual attention as a proxy for collaborative performance, *in* 'Proceedings of the 14th International Conference on Computer-Supported Collaborative Learning-CSCL 2021', International Society of the Learning Sciences.

Sharma, K., Papamitsiou, Z. and Giannakos, M. (2019), 'Building pipelines for educational data using ai and multimodal analytics: A "grey-box" approach', *British Journal of Educational Technology* **50**(6), 3004–3031.
URL: *https://bera-journals.onlinelibrary.wiley.com/doi/abs/10.1111/bjet.12854*

Shen, H. and Sun, C. (2000), Recipe: a prototype for internet-based real-time collaborative programming, *in* 'Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems. Philadelphia, Pennsylvania, USA', Citeseer.

Spearman Rank Correlation Coefficient (2008), Springer New York, New York, NY, pp. 502–505.
URL: *https://doi.org/10.1007/978-0-387-32833-1*$_3$79

Spikol, D., Avramides, K., Cukurova, M., Vogel, B., Luckin, R., Ruffaldi, E. and Mavrikis, M. (2016), Exploring the interplay between human and machine annotated multimodal learning analytics in hands-on stem activities, *in* 'Proceedings of the sixth international conference on learning analytics & knowledge', pp. 522–523.

The Pyodide development team (2021), 'pyodide/pyodide'.
URL: *https://doi.org/10.5281/zenodo.5156931*

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P. and SciPy 1.0 Contributors (2020), 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', *Nature Methods* **17**, 261–272.

Williams, L. and Kessler, R. R. (2000), The collaborative software process, *in* 'International Conference on Software Engineering 2000', Citeseer.

Williams, L., Kessler, R. R., Cunningham, W. and Jeffries, R. (2000), 'Strengthening the case for pair programming', *IEEE software* **17**(4), 19–25.

Yoo, Y., Lee, H., Jo, I.-H. and Park, Y. (2015), Educational dashboards for smart learning: Review of case studies, *in* G. Chen, V. Kumar, Kinshuk, R. Huang and S. C. Kong, eds, 'Emerging Issues in Smart Learning', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 145–155.

Zheng, W.-L., Liu, W., Lu, Y., Lu, B.-L. and Cichocki, A. (2019), 'Emotionmeter: A multimodal framework for recognizing human emotions', *IEEE Transactions on Cybernetics* **49**(3), 1110–1122.

Appendix

A Pre-test

Pre-test: Debugging

Name:

For each question you will be given a code snippet and you have to figure out what the ouput of the snippet is or what it does.

Question 1

What is the output of this program? Circle the correct answer.

a)

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

A. 0 1 2 3 4 Here
B. 0 1 2 3 4 5 Here
C. 0 1 2 3 4
D. 1 2 3 4 5

Answer: C

Question 2

What is the output of this program? Circle the correct answer.

```
for i in range(5):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

A. 0 1 2 3 4 Here
B. 0 1 2 3 4 5 Here
C. 0 1 2 3 4
D. 1 2 3 4 5

Answer: A

Question 3

What is the output of this program? Circle the correct answer.

```
a = [0, 1, 2, 3]
i = -2
for i not in a:
    print(i)
    i += 1
```

A. -2 -1
B. 0
C. error
D. none of the mentioned

Answer: C

Question 4

What is the output of this program? Circle the correct answer.

```
class Demo:
    def __new__(self):
        self.__init__(self)
        print("Demo's __new__() invoked")

    def __init__(self):
        print("Demo's __init__() invoked")

class Derived_Demo(Demo):
    def __new__(self):
        print("Derived_Demo's __new__() invoked")

    def __init__(self):
        print("Derived_Demo's __init__() invoked")

def main():
    obj1 = Derived_Demo()
    obj2 = Demo()
main()
```

A.

1. Derived_Demo's __init__() invoked
2. Derived_Demo's __new__() invoked
3. Demo's __init__() invoked
4. Demo's __new__() invoked

B.

1. Derived_Demo's __new__() invoked
2. Demo's __init__() invoked
3. Demo's __new__() invoked

Answer: B

C.

1. Derived_Demo's __new__() invoked
2. Demo's __new__() invoked

D.

1. Derived_Demo's __init__() invoked
2. Demo's __init__() invoked

Question 5

What is the output of this program? Circle the correct answer.

```
class Test:
    def __init__(self):
        self.x = 0

class Derived_Test(Test):
    def __init__(self):
        self.y = 1
def main():
    b = Derived_Test()
    print(b.x,b.y)

main()
```

A. 0 1
B. 0 0
C. Error because class B inherits A but variable x isn't inherited
D. none of the mentioned

Answer: C

Question 6

What is the output of this program? Circle the correct answer.

```
count={}
count[(1,2,4)] = 5
count[(4,2,1)] = 7
count[(1,2)] = 6
count[(4,2,1)] = 2
tot = 0
for i in count:
    tot=tot+count[i]
print(len(count)+tot)
```

A. 25
B. 17
C. 16
D. Tuples can't be made keys of a dictionary

Answer: C

Question 7

What is the output of this program? Write down the output.

```
def fn(**kwargs):
    for emp, age in kwargs.items():
        print ("%s's age is %s." %(emp, age))

fn(John=25, Kalley=22, Tom=32)
```

Answer:

John's age is 25.
Kalley's age is 22.
Tom's age is 32.

Question 8

What is the output of this program? Write down the output.

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def set_processor(self, new_processor):
        processor = new_processor

class Desktop(PC): # Derived class
    os = "Mac OS High Sierra" # Personalized attribute
    ram = "32 GB"

class Laptop(PC): # Derived class
    os = "Windows 10 Pro 64" # Personalized attribute
    ram = "16 GB"

desk = Desktop()
print(desk.processor, desk.os, desk.ram)

lap = Laptop()
print(lap.processor, lap.os, lap.ram)
```

Answer:

Xeon Mac OS High Sierra 32 GB
Xeon Windows 10 Pro 64 16 GB

Question 9

What is the output of this program? Write down the output.

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def __init__(self, processor, ram):
        self.processor = processor
        self.ram = ram
    def set_processor(self, new_processor):
        processor = new_processor
    def get_PC(self):
        return "%s cpu & %s ram" % (self.processor, self.ram)

class Tablet():
    make = "Intel"
    def __init__(self, processor, ram, make):
        self.PC = PC(processor, ram) # Composition
        self.make = make

    def get_Tablet(self):
        return "Tablet with %s CPU & %s ram by %s" % (self.PC.processor, self.PC.ram, self.make)

if __name__ == "__main__":
    tab = Tablet("i7", "16 GB", "Intel")
    print(tab.get_Tablet())
```

Answer:

Tablet with i7 CPU & 16 GB ram by Intel

Question 10

What is the output of this program? Write down the output.

```
def multiply_number(num):
    def product(number):
        'product() here is a closure'
        return num * number
    return product


num_2 = multiply_number(2)
print(num_2(11))
print(num_2(24))

num_6 = multiply_number(6)
print(num_6(1))
```

Answer:

22
48
6

B Debug task

```
import math
import random
from os import environ

environ["PYGAME_HIDE_SUPPORT_PROMPT"] = "1"
import pygame
from pygame import mixer

# Intialize the pygame
pygame.init()

# create the screen
screen = pygame.display.set_mode((800, 600))

# Background
background = pygame.image.load("background.png")

# Sound
mixer.music.load("background.wav")
mixer.music.play(-1)

# Caption and Icon
pygame.display.set_caption("Space Invader")
icon = pygame.image.load("ufo.png")
pygame.display.set_icon(icon)

# Player
playerImg = pygame.image.load("player.png")
playerX = 370
playerY = 480
playerX_change = 0

# Enemy
enemyImg = []
enemyX = []
enemyY = []
enemyX_change = []
enemyY_change = []
num_of_enemies = 6

for i in range(num_of_enemies):
    enemyImg.append(pygame.image.load("enemy.png"))
    enemyX.append(random.randint(0, 736))
    enemyY.append(random.randint(50, 150))
    enemyX_change.append(4)
```

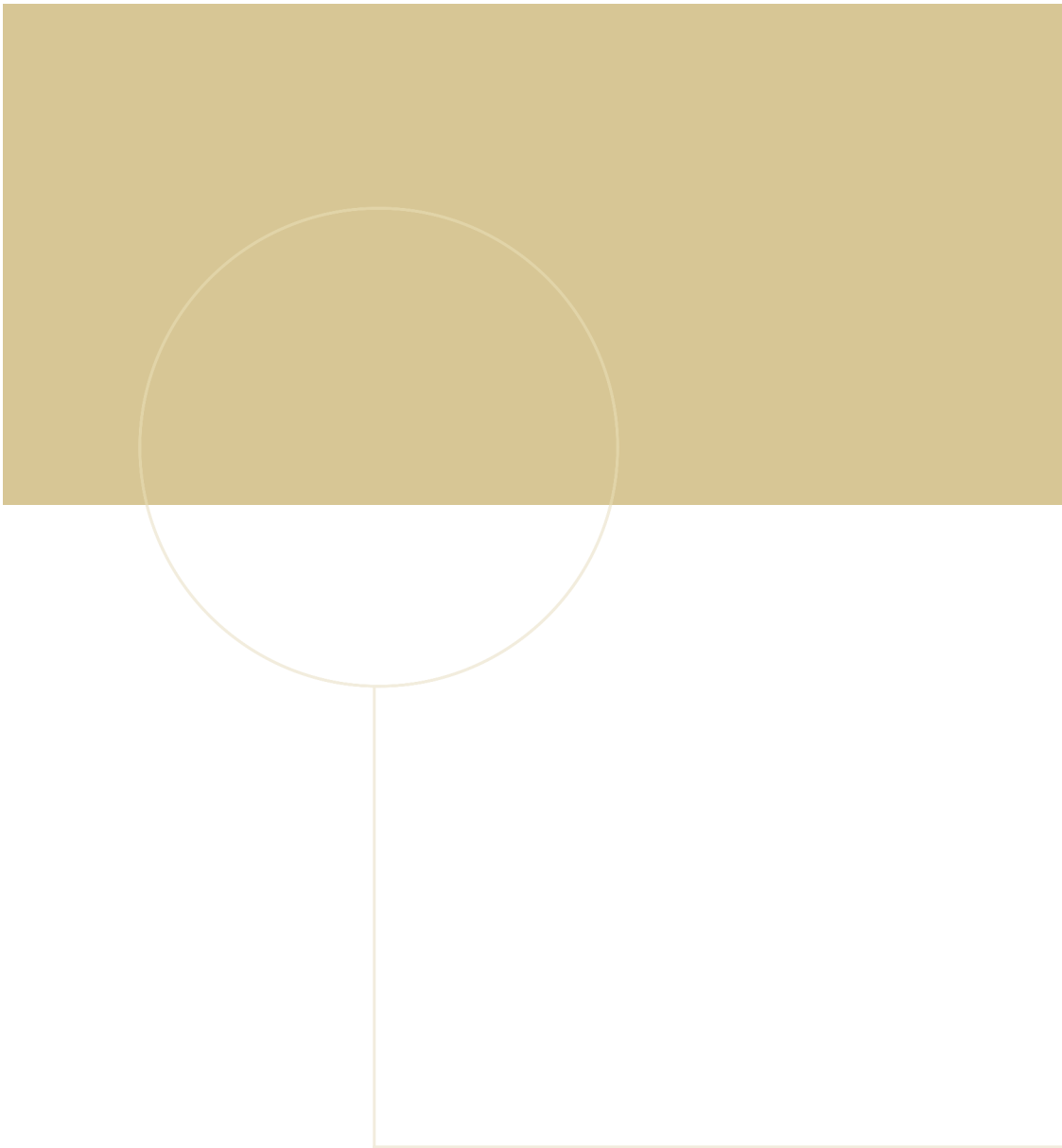
```
    enemyY_change.append(40)

# Bullet

# Ready - You can't see the bullet on the screen
# Fire - The bullet is currently moving

bulletImg = pygame.image.load("bullet.png")
bulletX = 0
bulletY = 480
bulletX_change = 0
bulletY_change = 10
bullet_state = "ready"

# Score

score_value = 0
font = pygame.font.Font("freesansbold.ttf", 32)

textX = 10
testY = 10

# Game Over
over_font = pygame.font.Font("freesansbold.ttf", 64)


def show_score(x, y):
    score = font.render("Score : " + str(score_value), True, (255, 255, 255))
    screen.blit(score, (x, y))


def game_over_text():
    over_text = over_font.render("GAME OVER", True, (255, 255, 255))
    screen.blit(over_text, (200, 250))


def player(x, y):
    screen.blit(playerImg, (x, y))


def enemy(x, y, i):
    screen.blit(enemyImg[i], (x, y))


def fire_bullet(x, y):
    global bullet_state
```

```
    bullet_state = "fire"
    screen.blit(bulletImg, (x + 16, y + 10))


def isCollision(enemyX, enemyY, bulletX, bulletY):
    distance = math.sqrt(
        math.pow(enemyX - bulletX, 2) + (math.pow(enemyY - bulletY, 2))
    )
    # Bug 1: Bullet-enemy collision
    #   Distance is only 0 if both objects are on the exact same (x,y) coordinate
    if distance < 0:
        return True
    else:
        return False


# Game Loop
running = True
while running:

    # RGB = Red, Green, Blue
    screen.fill((0, 0, 0))
    # Background Image
    screen.blit(background, (0, 0))
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # if keystroke is pressed check whether its right or left
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                # Bug 2: Spaceship movement logic
                # Left arrow should set playerX_change to be a negative value
                playerX_change = 15
            # Bug 3: Player input
            if event.key == pygame.K_UP:  # Should be right arrow (pygame.K_RIGHT)
                playerX_change = 15
            if event.key == pygame.K_SPACE:
                if bullet_state == "ready":
                    bulletSound = mixer.Sound("laser.wav")
                    bulletSound.play()
                    # Get the current x cordinate of the spaceship
                    bulletX = playerX
                    fire_bullet(bulletX, bulletY)
```

```
136
137         if event.type == pygame.KEYUP:
138             if event.key == pygame.K_LEFT or event.key == pygame.
    K_RIGHT:
139                 playerX_change = 0
140
141     # 5 = 5 + -0.1 -> 5 = 5 - 0.1
142     # 5 = 5 + 0.1
143
144     # Bug 4: Spaceship movement
145     playerX = playerX_change  # Add playerX_change to playerX
146     if playerX <= 0:
147         playerX = 0
148     elif playerX >= 736:
149         playerX = 736
150
151     # Enemy Movement
152     for i in range(num_of_enemies):
153
154         # Game Over
155         if enemyY[i] > 440:
156             for j in range(num_of_enemies):
157                 enemyY[j] = 2000
158             game_over_text()
159             break
160
161         enemyX[i] += enemyX_change[i]
162         if enemyX[i] <= 0:
163             enemyX_change[i] = 4
164             enemyY[i] += enemyY_change[i]
165         elif enemyX[i] >= 736:
166             enemyX_change[i] = -4
167             enemyY[i] += enemyY_change[i]
168
169         # Collision
170         collision = isCollision(enemyX[i], enemyY[i], bulletX, bulletY)
171         if collision:
172             explosionSound = mixer.Sound("explosion.wav")
173             explosionSound.play()
174             bulletY = 480
175             bullet_state = "ready"
176             # Bug 5: Increasing score
177             score_value = 1  # Increment score
178             enemyX[i] = random.randint(0, 736)
179             enemyY[i] = random.randint(50, 150)
180
181         enemy(enemyX[i], enemyY[i], i)
```

```
182
183     # Bullet Movement
184     if bulletY <= 0:
185         bulletY = 480
186         bullet_state = "ready"
187
188     # Bug 6: Bullet movement
189     if bullet_state == "fire":
190         fire_bullet(bulletX, bulletY)
191         bulletY = bulletY_change  # Subtract bulletY_change from
    bulletY (y starts from bottom)
192
193     player(playerX, playerY)
194     show_score(textX, testY)
195     pygame.display.update()
```

Listing 6: Debug task



NTNU

Norwegian University of
Science and Technology