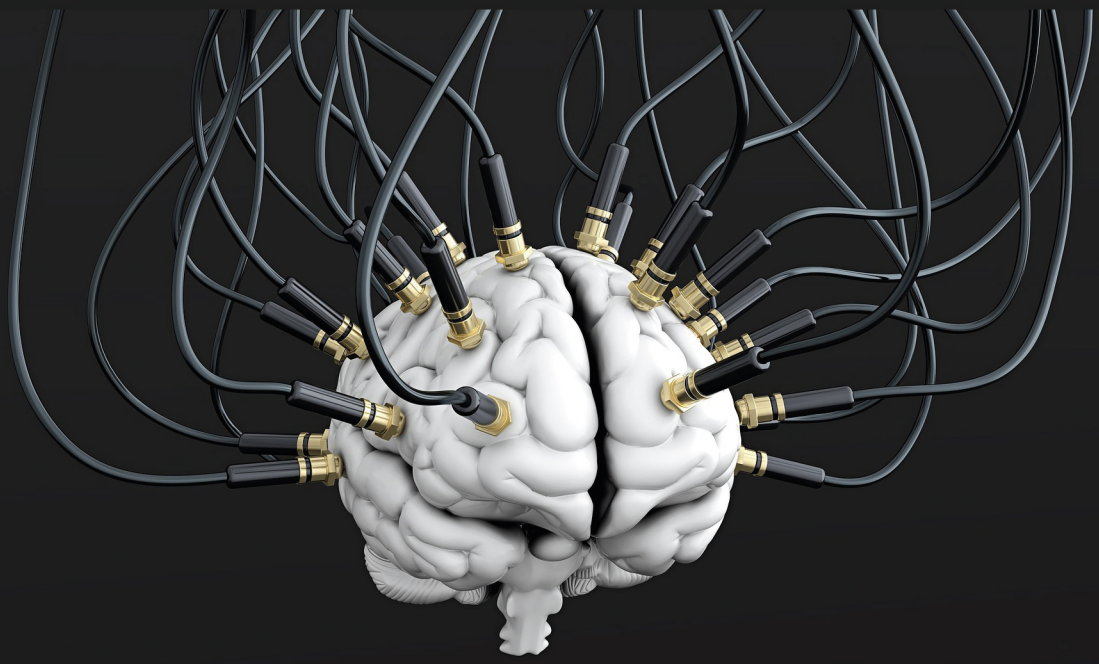
Influential stimuli characteristics on SNR in SSVEP-based interfaces

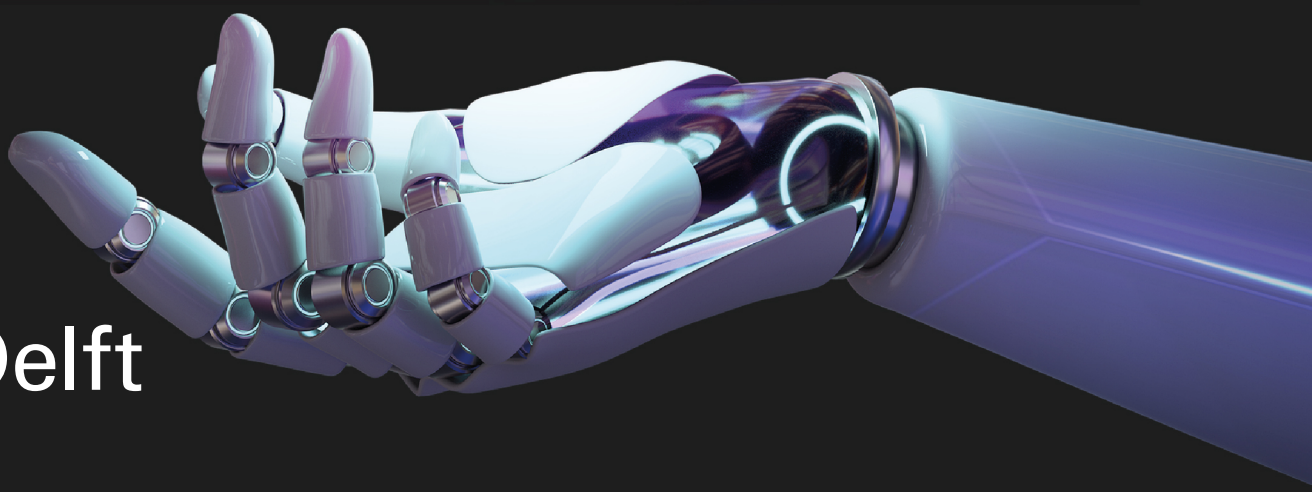
Thesis report: researching different aspects that influence the SNR in SSVEP-based interfaces

RO57035: MSc. Thesis

S.T. van Vliet

Delft University of Technology





TU Delft

Influential stimuli characteristics on SNR in SSVEP-based interfaces

Thesis report: researching different aspects that influence the SNR in SSVEP-based interfaces

by

S.T. van Vliet

Student Name	Student Number
Sjoerd Timo van Vliet	5180821

Thesis Supervisor:	Dr.ir. Y. B. Eisma
Institution:	Delft University of Technology
Place:	Faculty of Mechanical, Maritime and Materials Engineering, Delft
Project Duration:	December, 2021 - January, 2023

Cover Image: Brain wired to robotic arm, https://pixabay.com/

TU Delft

Preface

This report is my master's thesis of the study MSc. Robotics.
It is meant for scientists that want to investigate the influential aspects of SSVEP-based interface characteristics on the quality of the signal which is used for decoding commandos in brain-computer interfaces.
I am thankful for the excellent guidance and advice from Dr.ir. Y. B. Eisma. I would also like to thank L. Verweg for her help with the data collection during the experiments and J. de Winter for his advice during the thesis.

S.T. van Vliet
Delft, January 2023

Abstract

Controlling machines with brainwaves can be beneficial to society as they could potentially be used to reclaim some level of independence for patients that have lost a limb or are paralyzed. Different paradigms can be used to evoke brainwaves on command. These paradigms can be used to interact between the brain and a machine. These paradigms are called brain-machine interfaces. One of these methods uses external stimuli of which each flickers at a fixed frequency. This paradigm is called steady-state visually evoked potentials. To evoke these brainwaves the person has to look at the stimuli. In reaction to this, a response is elicited around the stimulus frequency in the brain. These external stimuli can have different characteristics, such as various colors, sizes, frequencies, and shapes. These properties can affect the quality of the evoked steady-state visually evoked potentials. The quality of the signal is measured by the ratio between the ”signal” and ”noise”, which is often referred to as the baseline. This ratio is called the signal-to-noise ratio. The quality of a signal is important as the real-time operation of applications is heavily dependent on the signal quality. Real-time operation of applications with stead-state visually evoked potentials is still a problem as this currently is only achievable using very expensive hardware. This limits its beneficial use to society as this is still too expensive for patients.

In this research, the relationship is investigated between the characteristics of the external stimuli and interface with respect to the signal quality. Hopefully, this will allow future research to create better interfaces that use steady-state visually evoked potentials and potentially on less expensive hardware. Two experiments were shown to dissect the relationship between signal-to-noise ratio and colors (red, white, green), sizes (10.000, 20.000, 30.000 pixels), frequencies (8Hz, 13Hz, 29Hz, 25Hz), and shapes (squares, triangles, circles).

The first experiment shows only one stimulus at a time across a combination of all these characteristics. The second experiment shows all of these 4 characteristics across multiple settings. The other experiment shows 4 stimuli in a steady-state visually evoked potentials speller consisting of multiple frequencies that differ by only 0.3Hz. The goal is to look at 1 of the 4 stimuli, which is the target. The frequency of these stimuli matches one of the frequencies of the earlier experiment. Additionally, all the stimuli here have the shape of a square. However, the color and size vary across the same settings as in the earlier experiment.
By comparing the results of these 2 experiments from 6 participants, a comparison can be made between a single stimulus versus a steady-state visually evoked potentials speller containing multiple stimuli. The speller contains multiple stimuli as it often has to support multiple commands simultaneously to let the brain-machine interface control an application. This comparison of a single stimulus versus multiple stimuli is to our knowledge novel. To investigate if there are any differences between the experiments and the different stimuli characteristics, analyses of variances are executed.

When comparing each category of the overlapping stimuli characteristics between the 2 experiments, no significant difference can be found between the means. The results suggest that the effects of the target stimulus surrounded by multiple stimuli are probably negligible in most cases. Furthermore, both experiments seemed to show significant differences between the frequencies and favored 25Hz. However, other research stated different results. Additionally, the preference for color and shape remains indecisive. However, the shape preference remains indecisive between the square and the circle as the best option. Moreover, the property pixel surface seems to have a positive correlation with the signal-to-noise ratio. Lastly, in an attempt to give the EEG recordings more context, eye-tracking was used during the experiments. This is considered novel. However, in around half of the participants eye-tracking still seemed to fail a lot. This lowers its supportive role significantly. The cause is still unknown.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ANOVA	Analysis of Variance
AR	Augmented reality
Bpm	Bits per minute
BCI	Brain-computer interface
c-VEP	Code visually evoked potential
df	Degrees of freedom
EEG	Electroencephalography
f-VEP	Frequency visually evoked potential
ITR	Information transfer rate
IDW	Inverse distance weighted
LCD	Liquid crystal display
ME	Motor execution
MI	Motor imagery
NaN	Not a Number
SNR	Signal-to-noise ratio
SSVEP	Steady-state visually evoked potential
SSVER	Steady-state visually evoked response
t-VEP	Time visually evoked potential
VEP	Visually evoked potential
VR	Virtual Reality

List of Figures

List of Tables

1 Introduction

Brain-computer interfaces **(BCIs)** are devices and frameworks that allow the brain to communicate with a computer. These devices can be especially beneficial to people that have lost a limb or are paralyzed, which allows them to reclaim a certain level of independence. These devices can be used to control different applications, such as artificial or external limbs. BCIs can be intracortical or not, which means that they are implanted or externally attached to the head. There are multiple methods that realize BCIs. The focus of this research is electroencephalography **(EEG)**. EEG uses external electrodes that measure the electrical activity from the scalp's surface. It measures mV over time. This electrical activity is evoked by the brain. There are multiple methodologies possible to evoke signals in the brain. These signals can be used as control signals for BCI-based systems. Motor imagery **(MI)** is an example that is evoked by imagining a movement [34]. This often allows for continuous and more fluent control. However, the problem is that MI is not very accurate and needs months of training. Furthermore, the number of different commands possible is limited. Another method is motor execution **(ME)**. This uses actual movements of body parts to generate the desired wave to control the BCI-based application [34]. However, besides the limitation of the number of different commands and accuracy, it also limits the useability for people that are paralyzed. Another option to elicit brain waves is the use of external stimuli. Methodologies that use external stimulation to evoke Visually Evoked Potentials **(VEP's)** can be categorized based on the aspect being modulated. These are code(c)-VEP's, frequency(f)-VEP's, and time(t)-VEP's. These VEP's can be observed in the occipital and parietal regions [7][34].

An example of a VEP that is used often is P300-waves. They are very accurate, people need less training, and can support a high amount of different commands [34]. However, the problem is that it takes a long time to issue just one command. This limits its useability in real-time applications. This methodology uses a matrix of characters of which a row or a column is elicited at random (see figure 1.1). When the person looks at the desired character and that row or column lights up, a brain wave is evoked. To do this it needs to go through all the columns and rows. Sometimes, even multiple times. A methodology that overcomes this issue is steady-state visually evoked potential **(SSVEP)**. It reaches high accuracies, is fast to issue, and needs no training time for the user [21][41]. However, it needs an external stimulus pulsating at a fixed frequency to be evoked. This means that additional hardware is needed, and is a disadvantage to other control methodologies, such as MI.

Figure 1.1: P300-speller matrix [30]

1.1. Influential factors in SSVEP-based BCI's

Research has reported that SSVEP can be observed best in the occipital area and parietal lobe region of the scalp [7][41]. This is important, as good signal quality is critical for the reliable and robust decoding of brain signals. Here an introduction is given to different contributing factors to signal quality in SSVEP-based interfaces and their corresponding terminology to create a better understanding of the multivariate nature of SSVEP-based interfaces before stating the purpose of this research.

First, the impedance of the electrodes is one of the influential factors that determine the signal-to-noise ratio **(SNR)**(dB), which is a measure of the signal quality. To lower the impedance of the electrodes, conductive gel is often used. Another important aspect of the hardware is that the sample rate can significantly differ between used hardware, which limits the sample resolution. Furthermore, some methods used to classify SSVEP cannot be used when this is too low [16].

Besides the used recording hardware for the brainwaves, the stimulation hardware also has influential aspects. For instance, a monitor induces the problem of portability of SSVEP-based interfaces. It also often requires a gaze-shifting time between the application that is controlled and the SSVEP-based interface [39][41]. Some SSVEP-based frameworks have now used augmented reality **(AR)** to overcome this problem [5][27][43]. However, using a monitor with respect to AR or virtual reality **(VR)** can still pose different results [41]. That the difference in stimulation hardware poses different results is also stated by Volosyak, Cecotti, and Graeser [39]. Volosyak, Cecotti, and Graeser [39] stated that using an array of LED's in comparison to using a liquid crystal display **(LCD)** screen results in significantly larger amplitudes.

According to Zhu et al. [45], besides LED's, most SSVEP-based interfaces use a screen to generate the stimuli. These stimuli can have different types of nature. They can show an object that appears and disappears (see figure 1.3a). Stimuli with a checkerboard or stripe-based pattern (grating stimulus) are often used, and their patterns are continuously reversed to evoke SSVEP's. Both reverse the displayed stimulus and alternate between these two states (see figure 1.3b). In this research, the focus is on single graphic stimuli that appear and disappear, as seen in figure 1.3a, and multiple graphic stimuli that appear and disappear at different rates. An example of this are SSVEP-speller matrices, which are used to control different applications. The example shown in figure 1.2 is used as a keyboard [19].

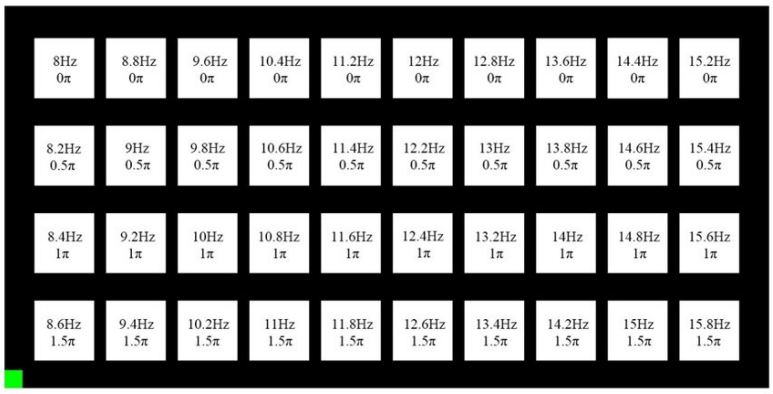
One of the most important values that measure if an SSVEP-based framework is suited for online control of an application is the information transfer rate **(ITR)**(bits/min) (see equation 1.1).

$$ITR = (\log_2 N + p * log_2 p + (1-p) * log_2[\frac{1-p}{N-1}]) * \frac{60}{T} \tag{1.1}$$

Where p is the recognition accuracy; T is the time to select and issue a target (includes the time to identify the target, the time needed to switch gaze, and the visual latency); and N is the number of targets to choose from [41].

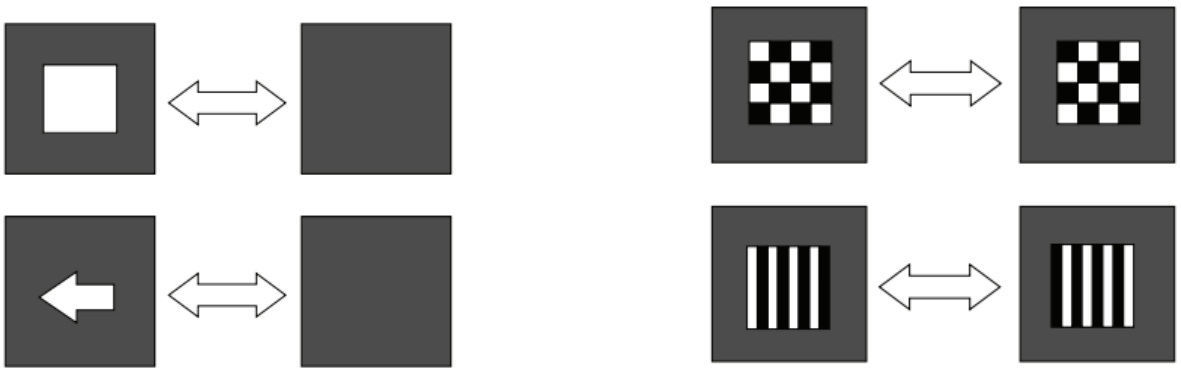
ITR depends on the accuracy and is a measure that indicates how fast and accurate a BCI-based framework is. The accuracy and ITR do heavily depend on the SNR. Besides the used hardware (e.g., the EEG headset), numerous aspects influence the SNR. The interface used to show the stimuli that evoke the SSVEP's in the brainwaves is greatly investigated by the scientific field. This is important as it paves the way for more reliable and robust decoding of SSVEP-based interfaces. Furthermore, a cleaner signal might also result in faster decoding of commands, allowing for real-time control of applications. Researchers claim to have succeeded in reaching high ITR's [7][16][29][41]. However, the technology has yet to mature as state-of-the-art performance often relies on expensive hardware, which limits its public use significantly. Contributing factors of poor signal quality in less expensive hardware are things such as lower sampling frequencies and higher impedance of the electrodes, which all influence the measured signal quality. However, how the stimuli are presented also influences the measured signal quality. By investigating and dissecting the contributing factor of the interface characteristics that display the stimuli in an SSVEP-based paradigm, the signal quality can be improved. This allows future research to push the boundaries of the scientific field. To formally state the main research question in this research: "What is the relationship between the stimuli characteristics and

(a) The actual SSVEP-based interface shown which is used as keyboard



(b) The frequencies and phase differences of each potential target shown on the interface

Figure 1.2: (a) The SSVEP-speller matrix.(b) The technical information of each target of the SSVEP-speller. [19]



(a) single graphic **(b)** pattern reversal

Figure 1.3: (a) During single graphic stimuli the object appears and disappears.(b) During pattern reversal, the object is mirrored each time. [45]

the measured SNR?"

Let's first introduce previous research to better grasp the previously investigated stimuli characteristics before stating the exact details of this research further.
Researchers are investigating how different variations of the interfaces showing the stimuli influence the SNR. Influential factors might be the inter-stimuli distance, stimulus color, the shape of the stimulus, the amount of stimuli, the size of the stimuli, and the used frequencies for the stimuli [9][33][35][41][45].

1.1.1. Color of the stimulus

To determine the best color for flickering is a hard-to-answer question, as previous research shows contrary results. Some research stated that using color enhances the SNR, while others do not [8]. While some research stated white flickering shows the best accuracy and others stated red [1][4][8]. However, Duart et al. [8] stated that this is dependent on the stimulus frequency and that around 5Hz, red showed the highest SNR. At 12Hz, Duart et al. [8] states that they did not perceive any differences between the red and white color used when showing flickering. Green is also often investigated as a stimulus color [4][8][45].

Cao et al. [4] also compared different colors of stimuli in combination with their frequency. It tested green, blue, red, gray, and white at frequencies between 8Hz and 17Hz. From their results, white was favored, and than gray. The other colors followed in the sequence are red, green, and blue.

Duszyk et al. [9] also investigated stimuli frequency and color. It researched the frequencies 14Hz, 17Hz, 25Hz, and 30Hz. The colors investigated are white, red, and yellow. Tests concluded that the differences between the means were not significant between the colors except with respect to blue. Duszyk et al. [9] did even more extensive research by also investigating the size of the stimuli, the shape of the stimuli, the influence of a fixation point, and the distance between stimuli. These aspects will be discussed later.

Cysewska-Sobusiak and Jukiewicz [6] used rectangular evoked waves at 10Hz, 20Hz, 30Hz, and 40Hz. The colors tested were green, blue, red, yellow, and white. However, the results varied significantly across each frequency which made it hard to choose a clear winner. An interesting measurement was that at 10Hz, the green color showed the highest SNR relative to the other colors. At the other frequencies, this difference between colors was less significant.

Besides stimulus color, the background color also has an effect on the decoded SNR, as studied by Zhang and Chen [42]. This study showed that when comparing backgrounds, black luminance has a positive effect on the SNR compared to grey luminance. This means that a combination of a red stimulus with a black background should result in a high SNR, which is in line with Albawardi et al. [1].

1.1.2. Size of the stimulus

Duszyk et al. [9] tested different sizes of squares with the sides being the length of 41, 102, 170, and 255 pixels. Overall, the increased pixel size resulted in increased SSVEP magnitude, except for the 30Hz, which peaked at 170 pixels.

1.1.3. Shape of the stimulus

Duszyk et al. [9] also compared different shapes. They compared circles to squares. They showed a slight, but not significant, increase in the SSVEP magnitude in the case of circles with respect to squares. This was the case for all tested frequencies (14Hz, 17Hz, 25Hz, 30Hz), except the 17Hz. The most often-used shape in SSVEP's stimuli when showing a single graphic square-based [45]. However, the widely used checkerboard pattern also exists out of numerous squares (see figure 1.3b).

1.1.4. Inter-stimuli distance

Duszyk et al. [9] tested various inter-stimulus distances across various frequencies and showed mixed results. They concluded that varying this aspect resulted in no significant effect on the SSVEP magnitude. Zhang et al.[41], however, showed a positive relation between inter-stimuli distance and the measured accuracy in an AR-SSVEP setup.

1.1.5. Stimuli frequency

Previous works state that frequencies lesser than half the refresh rate are suitable for SSVEP [37][40]. The refresh rate determines which frequencies are easier to display. For example, for a refresh rate of 60Hz, the 60 divided by a selected integer gives the frequencies that are the easiest to display, such as 30Hz, 20Hz, 15Hz, 10Hz, 8.57Hz, 7.5Hz, and 6Hz [23][39]. However, this is more the case when the flicker stimulation sequence can only switch between on and off states.
Other research has switched the on and off flickering for a sampled sinusoidal stimulation method (see equation 1.2) [25]. In this equation, i is the index of the frame, which is between 0 and the refresh rate. Next, ϕ is the phase shift, f is the stimulation frequency, and RefreshRate is the refresh rate of the monitor. This method allows for more flexibility in selecting stimulus frequencies because it can show values between the on and off states. Some research incorporates the phase shift to increase the separability between adjacent frequencies, as shown in equation 1.2 [25].

$$sin(f, \phi, i) = 0.5(1 + sin(2 * \pi * f * (i/RefreshRate) + \phi) \tag{1.2}$$

In this research, the sampled sinusoidal stimulation method is used without the phase shift, to allow more freedom in selecting the stimuli frequencies in contrast to the earlier mentioned method of dividing the refresh rate by an integer. This is similar to Kanoga et al. [20], which used sinus-type stimulus. The reason that it allows more freedom is due to the important fact that harmonics should be avoided [37]. This means that at multiples of stimulus frequencies, an increase in amplitude occurs, which should be avoided when selecting the different stimulus frequencies as a decrease in SNR occurs [37]. This also means that these multiples of different frequencies cannot be the same. Volosyak, Cecotti, and Graeser [39] made sure that at least the first two harmonics between different frequencies cannot overlap, as these can be used for SSVEP detection.

Besides harmonics, the type of stimulus also seems to greatly affect the SNR. High-frequency stimulus frequencies are more prone to timing issues as a delay of a few ms greatly influences the actual stimulus frequency compared to lower frequencies. This means that lower frequencies are more robust to stimuli timing issues. Furthermore, research has shown that high-frequency stimulation is more prone to evoke fatigue within subjects [22][37]. Some research also states that the smallest response in amplitudes is seen in the high-frequency spectrum [6][37][40].

Pastor et al. [33] investigated 14 frequencies between 5Hz and 60Hz across 16 subjects. According to their research, they claim that the amplitude of SSVEP peaked at 15Hz in the occipital region. They named SSVEP here steady-state visual-evoked response **(SSVER)**.

Duart et al. [8] used an analysis of variance **(ANOVA)** to determine the best color and frequency combination that results in the highest SNR. From the 5Hz, 12Hz, and 30Hz, their research preferred 12Hz. It showed similar results at 12Hz for red and white. However, at 5Hz, green was the preferred color. That frequencies show different results with different colors is in line with Regan [35].

Cysewska-Sobusiak and Jukiewicz [6] measured the SNR at different colors at 10Hz, 20Hz, 30Hz and 40Hz. However, the SNR values were significantly higher at 10Hz and 20Hz. At 20Hz, the highest mean SNR was reached. Herrmann [17] did mention that they achieved strong resonance peaks at 10Hz, and weaker peaks at 20Hz, 30Hz, 35Hz, and 45Hz. They also stated that the SSVEP spectrum peaked at the frequencies 10Hz, 20Hz, 30Hz, 40Hz, and 50Hz. This could explain the peaks at 10Hz and 20Hz measured by Cysewska-Sobusiak and Jukiewicz [6].

Volosyak, Cecotti, and Graeser [39] tested two sets of frequencies on LCD screens. Frequency set 1, containing 5 frequencies between 6.67Hz and 12.00Hz, showed overall significantly better accuracy and ITR in comparison to the other frequency set, containing frequencies between 13.00Hz and 17.00Hz. SNR thresholding was used as a classification method for over 10 persons. These results do seem to suggest that frequencies around the 10Hz are better to use as stimulation frequencies.

Looking at all of this research, it would suggest that most research prefers frequencies around 10Hz and 20Hz.

1.1.6. Fixation point

An interesting aspect often implemented in SSVEP-based interfaces is the use of fixation points in the middle of the displayed stimuli [36]. It is often believed that this is beneficial to the SNR, as participants

have reported that it helps them focus. Duszyk et al. [9] also compared the influence of the fixation point on SSVEP magnitude. Tests revealed no significant differences in this parameter across all tested frequencies. The results sometimes showed a higher SNR for a fixation point and sometimes without. However, participants reported here also that it helped them focus.

1.1.7. Method to evoke wave

Cysewska-Sobusiak and Jukiewicz [6] also investigated the method used to generate the stimuli. They favored a sinus-based method significantly over a saw-tooth method or a rectangular/square wave. The rectangular method realized the lowest amount of SNR. Oralhan and Tokmakçi [32] also investigated the influence that the duty cycle has on the accuracy and ITR when using a rectangular/square wave. Using one-way ANOVA with respect to the classification accuracy, it was concluded that the duty cycle had a significant effect. Besides square waves, Oralhan and Tokmakçi [32] also tested a sort of saw-tooth-based method where the brightness was step-wise increased and decreased.

1.1.8. Number of stimuli

The optimal number of stimuli in SSVEP-speller matrices depends on various aspects and does seem to influence the accuracy and ITR achieved with various decoding algorithms, and the optimal amount of stimuli may vary between these algorithms [13][41]. Gembler, Stawicki, and Volosyak [13] and Zhang et al. [41] showed that the highest ITR does not necessarily mean the highest accuracy. An interesting finding of Zhang et al. [41] was that the location of the stimuli in the matrix might also result in decoding accuracy differences.

1.2. Our Contribution

In the present study, the influential characteristics of the external stimuli on the SNR are investigated. It is important to show novel insights into the correlation between different interface characteristics as this is still unclear within the scientific field. To dissect the influence of the interface characteristics on the SNR the following questions are investigated:

- What is the relationship between color and the measured SNR?
- What is the relationship between stimulus size and the measured SNR?
- What is the relationship between shape and the measured SNR?
- What is the relationship between frequency and the measured SNR?
- Do surrounding stimuli in an SSVEP-speller affect the measured SNR of the target stimulus?

To answer these questions different stimulus characteristics are tested over 2 experiments. Experiment 1 shows only a single stimulus at various sizes, shapes, colors, and frequencies. All of these possible combinations are shown to participants in an attempt to capture how the combination of these various characteristics contributes to the measured SNR. The second experiment attempts to give insights into how these different combinations of characteristics function differently in an SSVEP-speller matrix environment. To our knowledge this is novel. Therefore, in this experiment each of these variables except shape, which is kept static at squares, are tested in a 2X2 SSVEP-speller matrix. The other 3 stimuli are the same size, shape, and color as the target stimulus that is to be recorded. Additionally, the frequencies of these stimuli are in the neighborhood of the frequency of the stimulus. This is recorded by using a fixed difference (step size) in Hz between neighboring frequencies.

To give the measurements more context eye-tracking is used which helps explain the possible deviations in the measurements. This approach using such an experimental paradigm is to our knowledge novel in nature with respect to SSVEP-based interfaces. However, eye-tracking issues did lower the supportive role that it could play significantly within half of the participants.

In the coming chapters, first, the developed SSVEP-based interfaces are discussed. Then, the experiments of this research are discussed in detail, followed by the chapter that mentions the results. Lastly, the discussion, limitations, and conclusion of the results are stated at the end of this research.

2

Developed SSVEP-based interfaces

The developed SSVEP-based interfaces are created by using a combination of python to create '.mp4' videos using AVC1 decoding. The videos are created frame by frame at 60Hz, with a resolution of 1920x1080 pixels. Then the videos are converted using the software named Handbrake. The videos are converted to videos of the same resolution but an adapted framerate of 59.94 frames per second. This is needed to display the videos in the Experiment Builder (v2.3.38) software used to record and display the experiments. The software is developed by the company SR Research Ltd. The software has built-in eye-tracking and EEG recording functionality. The EEG is recorded in combination with the software BrainVision Recorder (Brain Products GmbH). A problem within Experiment Builder is that a significant part of the frames will be dropped if the desired video framerate is equal to or higher than the framerate of the screen. This means that the stimuli will not be correctly displayed, which makes the measurements untrustworthy.

2.1. Hardware

All the hardware is bought from SR Research Ltd. The developed SSVEP-based interface is shown on an interface of 60Hz (to be precise 59.999Hz). The resolution of the screen is 1920x1080 pixels. The EyeLink Portable DUO is used for eye-tracking with a sample rate of 2000Hz. The EEG recordings are realized using the actiCHamp Plus amplifier of BrainVision in combination with the standard actiCAP snap, which is a gel-based EEG headset. The sample rate is 2500Hz. The conductive gel used was the ECI Electro-Gel. The system uses active electrodes. A total of 64 electrodes are used for the recordings. The electrode locations recorded are Fp1, Fz, F3, F7, FT9, FC5, FC1, C3, T7, TP9, CP5, CP1, Pz, P3, P7, O1, Oz, O2, P4, P8, TP10, CP6, CP2, Cz, C4, T8, FT10, FC6, FC2, F4, F8, AF7, AF3, AFz, F1, F5, FT7, FC3, C1, C5, TP7, CP3, P1, P5, PO7, PO3, POz, PO4, PO8, P6, P2, CPz, CP4, TP8, C6, C2, FC4, FT8, F6, AF8, AF4, F2, and Iz. The locations of these electrodes can be seen in figure 2.2.

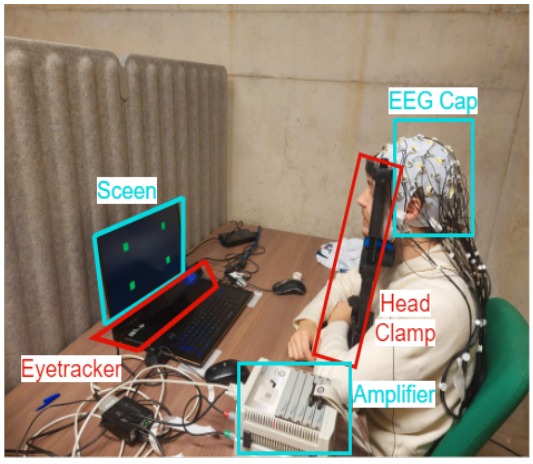


Figure 2.1: Experimental setup with hardware

To realize stable measurements a head restraint is used for eye-tracking. Additionally, this helps to keep the distance between each participant and the monitor used to display the stimuli at a constant level. This will improve the overall data quality. The display used is of a laptop, the ASUS GX701LV-D576. This laptop uses the Intel Core i7-10750H CPU @ 2.60GHz processor and the NVIDIA RTX 2060 that has 16GB video RAM to display the experiments and with that the videos. Additionally, the

laptop uses vsync to realize the exact timing of the frames and avoids screen tearing. An overview of the experimental setup can be seen in figure 2.1.

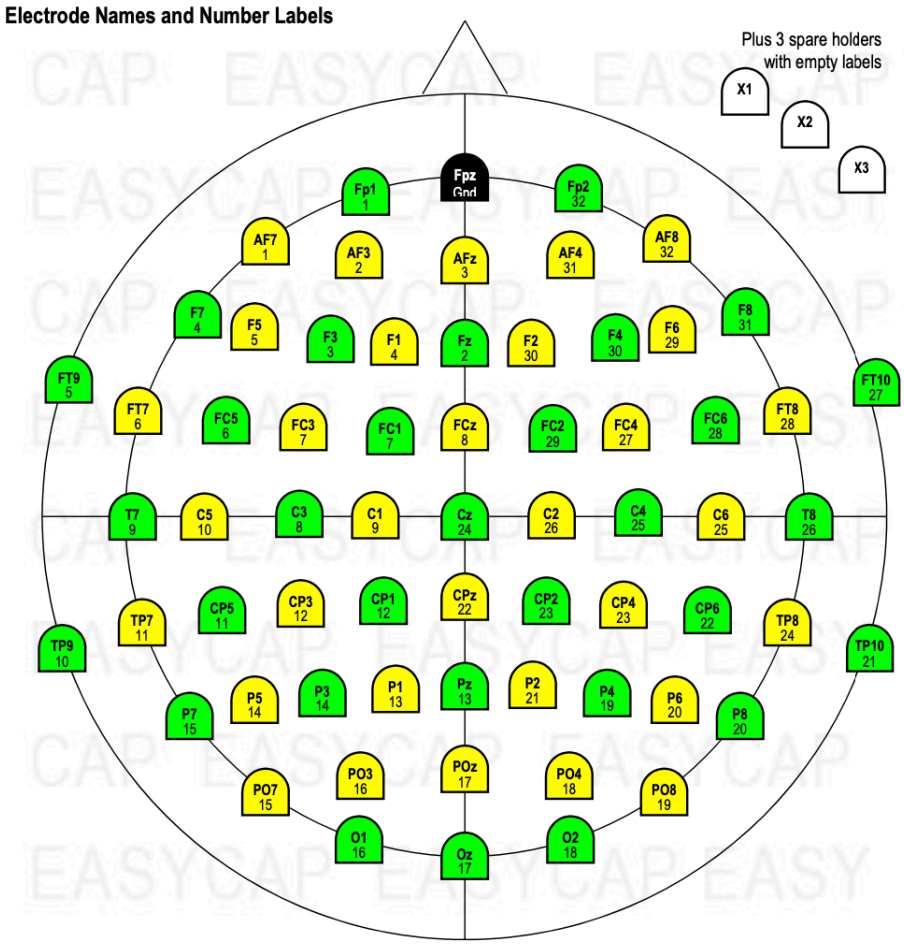


Figure 2.2: Electrode placement of the EEG headset [10].

2.2. Stimuli

Stimuli are tested in two experimental paradigms. Experiment 1, also named experiment 1x1, displays a single stimulus at a time. The stimulus here will vary by shape, size, color, and frequency. In experiment 2, also named experiment 2x2, 4 stimuli are presented simultaneously. One stimulus of these 4 stimuli is the target stimulus. This stimulus is displayed at the same frequencies used in experiment 1x1. The other stimuli are shown at frequencies close to the target frequency. The stimuli here will vary in size and color uniformly. The results of experiment 2 with respect to the results of experiment 1 can be used to give insight into how the effect of multiple stimuli in an SSVEP-Speller matrix affects the recorded SNR of the desired target.

To draw the stimuli OpenCV is used which is an image processing library. It contains draw functions to draw standard figures such as circles, but also methods to draw (non-)convex shapes.

2.2.1. Positioning

As mentioned in section 2.1, the display has a size of 1920x1080 pixels. In experiment 1, only one stimulus is shown with its center at the middle of the screen. Thus, x=960 and y=540 pixels. In experi-

ment 2, 4 stimuli are shown simultaneously. In each quadrant of the screen, a stimulus is shown. This means that following the (x, y) notation, the shapes are centered at (480, 270), (480, 810), (1440, 270), and (1440, 810). A visualization of this can be found in figure 3.1.

Important to note is that the origin of images is located in the upper-left corner of the image. The positive y-axis is oriented downwards.

2.2.2. Shapes and pixel surfaces

In this research, 3 different types of shapes are tested: triangles, squares, and circles. Using the desired pixel surface that makes up the shape, the corresponding shape is calculated. The corresponding shape may not exactly be the exact amount of pixel surface that is desired. However, it will be close to it. The reason is that it was chosen to let the shapes be symmetrical with respect to its y-axis and its center. The reason being that the shapes then have a true center. There are 3 levels of pixel surfaces tested in this research: 10.000 pixels, 20.000 pixels, and 30.000 pixels. The corresponding actual pixels surfaces per shape are shown in table 2.1.

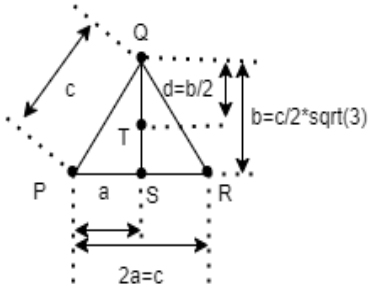


Figure 2.3: Iscoles triangle

Table 2.1: Pixel surface per shape

Shape	10.000 pixels	20.000 pixels	30.000 pixels
Circle	9.845	19.577	29.525
Square	10.201	19.881	29.929
Triangle	10019	20.289	30.077

Triangles

The triangles are designed to be an isosceles triangle where
all sides have the same length (see figure 2.3). Looking at figure 2.3, side a can be expressed according to 2.1.

$$a = \frac{c}{2} \tag{2.1}$$

Equation 2.2 shows the equation of Pythagoras for calculating the sides of a rectangular triangle consisting of sides a, b, and c, see figure 2.3.

$$a^2 + b^2 = c^2 \tag{2.2}$$

When substituting equation 2.1 in equation 2.2, equation 2.3 is achieved.

$$(\frac{c}{2})^2 + b^2 = c^2 \tag{2.3}$$

$$\frac{c^2}{4} + b^2 = c^2 \tag{2.4}$$

$$b^2 = c^2 - \frac{c^2}{4} \tag{2.5}$$

$$b^2 = \frac{3c^2}{4} \tag{2.6}$$

$$b = \sqrt{\frac{3c^2}{4}} \tag{2.7}$$

Eventually, b can be expressed in c according to equation 2.8.

$$b = \frac{c\sqrt{3}}{2} \tag{2.8}$$

(a) Green circle at ≈ 10.000 pixels

(b) Red triangle at ≈ 30.000 pixels

Figure 2.4: Examples of stimuli shown in experiment 1

The surface can then be expressed according to equation 2.9.

$$surface = \frac{1}{2} * 2a * b = a * b = \frac{c}{2} * \frac{c}{2} * \sqrt{3} \tag{2.9}$$

Equation 2.9 can then be rewritten to equation 2.12, to express the length of side c as a function of the desired surface.

$$surface = \frac{c^2}{4} * \sqrt{3} \tag{2.10}$$

$$c^2 = \frac{4 * surface}{\sqrt{3}} \tag{2.11}$$

$$c = \sqrt{\frac{4 * surface}{\sqrt{3}}} \tag{2.12}$$

The location of the desired center coordinate is at point T (see figure 2.3), which is used to define the points Q, P, and R. However, the actual center of the triangle is the average of the 3 points (Q, P, R) that make up the complete triangle. This center is calculated first. The x-coordinate of the desired center coordinate does not need a correction. Only the y-coordinate needs to be corrected. Thus, then the difference between point T and the actual center point of the triangle is calculated on the y-axis. Then all the points that make up the triangle are corrected by the calculated needed correction. The code function is named '_draw_triangles' and is located in appendix D.1.2.

Squares
The sides of the squares are calculated by taking the square of the pixel surfaces and converting the not-rounded answer to int. Then the floor division is taken by dividing the answer by 2 and using that answer to calculate the upper right corner and upper left corner of the square with respect to the center coordinate. The code function is named '_draw_squares' and is located in appendix D.1.2.

Circles
To approximate the desired pixel surface the radius is calculated according to equation 2.13. The code function is named '_draw_circles' and is located in appendix D.1.2.

$$radius = int(\sqrt{\frac{pixelsurface}{\pi}}) \tag{2.13}$$

2.2.3. Colors

To investigate how colors affect the SNR the following colors are investigated in combination with a black background: red, white, and green. The colors of each frame are calculated according to equation 1.2. However, as stated earlier the phase shift is kept at 0. To visualize how all the attributes, such as color, shape, and size come together, see figure 2.4 for examples.

2.2.4. Frequencies

To investigate what the influence of the stimuli frequencies is on the SNR the following frequencies will be the target frequencies: 8Hz, 13Hz, 19Hz, and 25Hz. In experiment 1, only one stimulus is shown at a time. Each stimulus is shown at one of these fixed frequencies. In experiment 2, 4 stimuli are shown simultaneously. The target stimulus will also be one of these target frequencies. The frequencies of the other 3 surrounding stimuli will be close to the desired target frequency. The stepsize of these neighboring frequencies is 0.3Hz from the adjacent frequencies. The distribution is one frequency lower and two frequencies higher than the target frequency. Thus, for example, 24.7Hz, 25Hz, 25.3Hz, and 25.6Hz. A 0.3Hz stepsize means that for an even distribution of intervals around each stimulus frequency, each interval is defined ±0.15Hz with respect to the stimulus frequency. The same interval is used for the analysis of experiment 1 to create a fair comparison. The reason that an interval is searched is that the response frequency measured in the brain does not always exactly match the stimulus frequency [26]. For example, Zheng et al. [44] searches an interval of ±0.1Hz with respect to the stimulus frequency.

3

Experiments

The experiments were recorded in the nuclear bunker of the UMC Amsterdam. An advantage of this location is that the external noise of electromagnetic fields is minimized with respect to the EEG recordings. A total of 6 participants of which 3 males and 3 females participated in the experiments. The age distribution is 24.8±3.4 years. All participants are new to SSVEP, except participant 1.

The experiments consist of two parts, experiment 1 and experiment 2. Half of the participants start with experiment 1 and the other half with experiment 2. After that, they execute the other experiment on the same day. This is done to counterbalance the data.

Before the start of the experiments, each participant is informed and has to give consent for the use of their recorded information of this research (see appendix E). They are also informed about any potential risks related to the experiment.
After attaching the EEG headset to each person and infusing it with conductive gel, each person is seated in a head clamp at a fixed distance of 68 cm with respect to the screen (see figure 2.1).

The experiments consist of two parts, experiment 1 and experiment 2. Each experiment starts with validating the impedance to be below 10kΩ for each electrode. After that, the eye-tracking is calibrated and an instruction screen is shown. By pressing the SPACE bar on the keyboard the practice trials are started. Each experiment shows 4 randomly sampled unique trials from all the unique combinations of interface characteristics covered in the trials. The trials are shown automatically in sequence. Each trial consists of an instruction screen, which is shown for 2s. This instruction screen consists of a static image that shows the target to look at using a blue dot (see figure 3.1). After these 2s, the target stimulus starts flickering. Important to note is that the other stimuli will also start flickering in experiment 2. After that, the video is frozen on the last frame and Experimental Builder loads the next trial, which takes on average 1s. This means the total trial duration for each combination is around 7s. After each block of trials, a 1 min break is induced for each person to relax. After the 1 min break, a drift correction is performed for the eye-tracking. After which, the next block of trials is started. Each experiment consists of 6 blocks for the experimental trials of which the size is experiment dependent and 1 block of practice trials which consists of 4 practice trials.

It is important to note that at the start of each video, a large part of the frames is dropped in Experimental Builder. This is only in the first 0.3s of the video. However, in the analysis of the EEG data, only the last 3.5s of the video is taken to calculate the SNR values. The frequency resolution can be measured according to equation 3.1 with a sampling frequency, F_s of 2500Hz [18]. This results in a frequency resolution of 2500/(3.5*2500)=0.2857Hz.

$$\delta f = \frac{F_s}{n_{fft}} \tag{3.1}$$

3.1. Experiment 1

In experiment 1, a single stimulus is shown across 3 different colors, 4 different frequencies, 3 different shapes, and 3 different sizes. This makes a total of 3*3*3*4=108 unique combinations of experiments. Each combination is shown 3 times in total in this experiment. All 108 combinations are shown in a random order. Then the combinations are randomized and shown again, and then randomized and shown again. As mentioned earlier, the block of practice trials consists of 4 trials. The experimental trials, which cover 3 times the 108 combinations, is divided up into 6 blocks, each of 54 trials. This means that the duration of each block is 54*7s=378s, which is 6.3 min. The total recording duration is thereby 7s*4 (practice trials)+60s*6+378s*6=2656s, which is $\approx$44.3 min.

3.2. Experiment 2

In experiment 2, 4 stimuli are shown at the same time. It stimulates the target in its 'natural environment'. It simulates an actual SSVEP-speller as neighboring frequencies only differ by 0.3Hz from one another. There is always one frequency lower and two frequencies higher than the target frequency. This means that for 25Hz, 24.7Hz, 25.3Hz, and 25.6Hz are shown across the other 3 stimuli. Similar frequency differences between neighboring frequencies are often used in SSVEP-speller matrices containing a significant amount of potential targets [3][19][25][26]. The locations of these neighboring frequencies besides the desired target to be recorded are randomized.
To indicate where to look during the trial the static photo that is shown before the start of the video contains a blue dot at the location of the target. This indicates where the subjects have to stare (see figure 3.1).

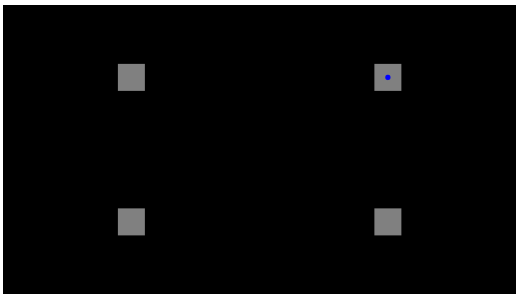


Figure 3.1: Example of a static instruction image for experiment 2

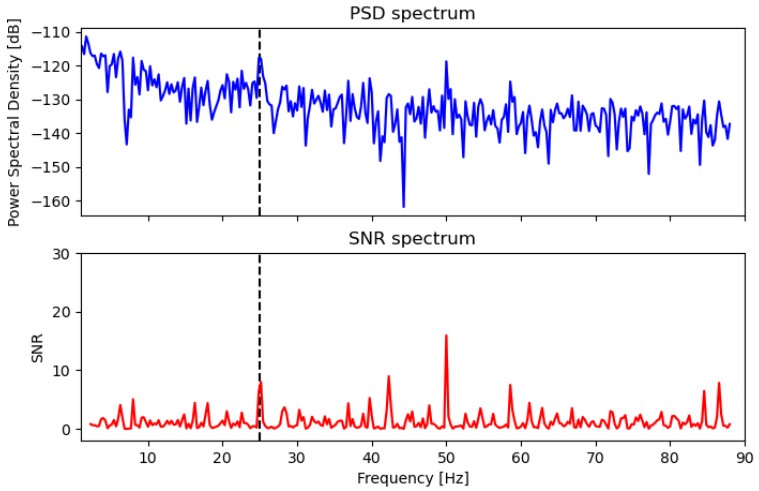
The trials cover a total of 144 combinations, following a similar procedure as experiment 1. All the combinations here are also randomized in order, and each combination is recorded 3 times. The only difference is that the size of each block covers here 72 trials. The 144 combinations cover each of the 4 positions and the same 4 frequencies (8Hz, 13Hz, 19Hz, and 25Hz), 3 sizes, and 3 colors as used in experiment 1. The shapes do not differentiate in this case as otherwise the number of combinations would be 432, which is too large to record. All combinations show a square, as this is most often used in SSVEP-spellers [19][24][25][26][45]. The total amount of experimental trials is 144*3=432 and the number of practice trials is 4. This results in 432*7s+6*60s=3384s time to record the entire trial, which is $\approx$56.4 min.

3.2.1. Questionnaire

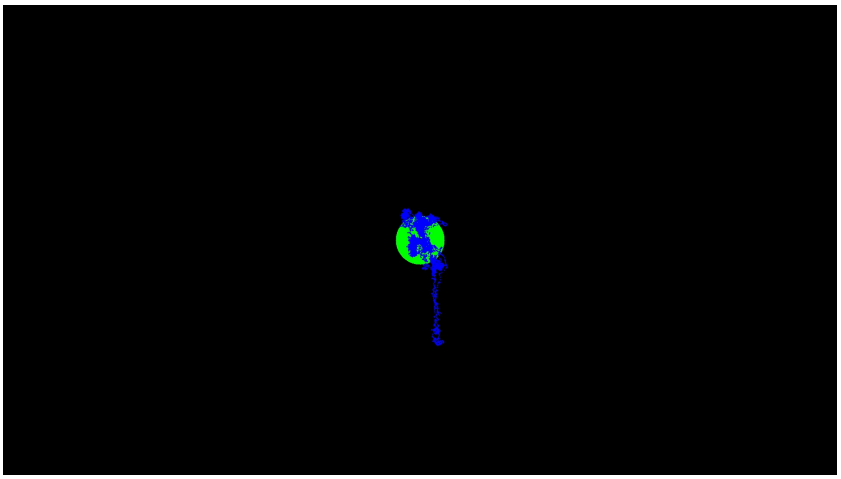
Before the start of the experiment, the questionnaire is shown to give an idea of which aspects will be graded by each participant. For each interface characteristic that was changed across the experiments, the participant could answer: very low(1), low(2), neutral(3), high(4), and very high(5). This means for every color, every size, every shape type, and for the low frequencies, high frequencies, or middle frequencies. Additionally, experiment 1 and experiment 2 are separately graded to see if there is any difference in the subjective perspective of the participants. Each of the characteristics was graded on the following attributes: comfort level, the level of focus of the participants, eye irritation, how easy it was to focus on the center of the stimulus, and if they had to blink a lot (see appendix C).

3.3. Analysis side notes

For the analysis of trials, the first 0.5 seconds of the trial will be removed from the EEG recordings. Reason one is that SSVEP can be stably measured 250ms after the stimulus onset [38]. Furthermore, Experimental Builder dropped the most frames at the start of the video causing artifacts. Thus, to receive a clean and stable signal the last 3.5 seconds are taken of the 4s trial. A visualization of the



(a) The power-spectral density and SNR plot of the raw data segment. The striped line indicates that the stimulation frequency is at 25Hz.



(b) The raw eye-tracking data visualized during the trial. The green circle is the stimulus and the blue dots are the gaze of the participant.

Figure 3.2: Visualizations of the raw data measured in a trial of participant 5 during experiment 1

raw EEG data measured can be seen in figure 3.2a. It shows the unprocessed data of the 3.5s segment.

For the analysis of the eye-tracking data, the whole 4s is taken as this indicates how well the person was focused during the whole trial. This is realized by segmenting the video from the moment the first frame was displayed until the 4s timer gave a timeout. The timer was started at the beginning of the video. An example of the visualized raw eye-tracking data in a trial can be seen in figure 3.2b.

4

Analysis & Results

One problem during the trials is that the eye-tracking in combination with a video showing flickering stimuli seemed to interfere with one another. This seemed especially the case when there was no light source in the background behind the display. The speculation is that the reflection of the display in the eyes causes this. However, still, in a large portion of the trials the eye-tracking failed when trying to mitigate this problem (see figure 4.1). The analysis of gaze data will not be discussed in this paper, as it is used to give the EEG measurements more context. Additionally, it is used to validate if the person was overall accurate and consistent with looking at the desired stimuli.

In the coming sections, the used methodology for analysis is explained, the influence of the interface characteristics when combining stimuli characteristics, the influence of stimuli characteristics per experiment, and the differences between the two experiments. After that, the results of the questionnaire are discussed.

Table 4.1: Eye-tracking in the overall number of trials

	Experiment 1				**Experiment 2**			
	No gaze data		**Gaze data**		**No gaze data**		**Gaze data**	
	Number of trials [N]	Percentage [%]	Number of trials [N]	Percentage [%]	Number of trials [N]	Percentage [%]	Number of trials [N]	Percentage [%]
p1	64/324	19.8%	260/324	84.2%	42/432	9.7%	390/432	90.3%
p2	38/32 4	11.7%	286/324	88.3%	32/432	7.4%	400/432	92.6%
p3	174/324	53.7%	150/324	46.3%	153/432	35.4%	279/432	64.6%
p4	25/324	7.7%	299/324	92.3%	65/432	15.0%	367/432	85.0%
p5	94/324	29.0%	230/324	71.0%	156/432	36.1%	276/432	63.9%
p6	228/324	70.4%	96/324	29.6%	246/432	56.9%	186/432	43.1%

Table 4.2: Bad electrode locations. See figure 4.1 for actual values of colors in kOhm.

	Experiment 1 Electrode locations			**Experiment 2 Electrode locations**		
	Red	Orange	Yellow	Red	Orange	Yellow
p1	-	-	C2, CP4, Iz	-	-	POz
p2	FC5, F4, TP10	C6, PO8	F5, F6, FT8, TP7, TP9	FT9	AF8, T8	F7, FT7, FT10
p3			PO4			FC2
p4	F1, FC3, C3, C2, CP2	-	Fc1, C1, POz, FCz	C4, C6, Cp6, TP7	P8, T7	Fz, F2, F4, F8, TP10, TP8, TP9
p5	FT10	F6, F8, FT8, C6, AF7	CZ, T7,TP9,TP7, CP4, CP6, TP8, TP10, T8, P8, PO8, PO4, O2, OZ, IL	CP3	-	FC3, C3, C1, T7, CP1, PZ ,PO3, POZ, PO4
p6	-	-	-	-	-	-

Table 4.3: Experiment remarks

	Electrodes used for analysis		Remarks
	Experiment 1	Experiment 2	
p1	O1, O2	O1, O2	Right eye was better tracked than left eye.
p2	O1, O2	O1, O2	Person is color blind and does not see the colors yellow and green.
p3	O1, O2	O1, O2	The right eye had irritation already before start experiments. Additionally, the participant deviated partially from the questionnaire
p4	O1, O2	O1, O2	In experiment 2, the eye-tracking had trouble with tracking the right eye.
p5	O1	O1, O2	The impedance of electrode O2 had increased after experiment 1 to yellow.
p6	O1, O2	O1, O2	Participant had to blink a lot.

4.1. Methodology

The methodology used can be separated in the workflow of the SNR, the method of grouping used to aggregate data before the statistical analyses are executed, and the workflow of the statistical analyses.

4.1.1. Signal-to-noise ratio workflow

The SNR is calculated according to Gramfort et al. [15]. Important to note is that the SNR can never go below zero. The first step to calculate this is calculating the power-spectral density spectrum. However, the resolution of this spectrum is dependent on the sample length. Using 3.5 seconds for analysis results in a frequency resolution/step-size of 0.2857Hz between the frequency bins. When calculating the SNR from this spectrum it inherits the same frequency resolution but not the same amount of frequency bins. Padding is used to solve this issue.

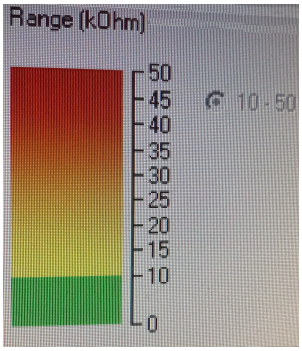


Figure 4.1: The kOhm meter of the electrode cap

SNR is the measure of the relative power between the ”signal” and ”noise”. According to our definition, SNR is a metric of relative power that compares the power in a specific frequency bin, or the ”signal,” to a ”noise” baseline, or the average power in the nearby frequency bins [28]. Similar to Gramfort et al. [15], this research uses the average power of the neighboring 3 bins on each side and skips the first bins that are located directly beside the stimulus frequency bin [15]. Thus, to put this in a kernel format: [1, 1, 1, 0, 0, 0, 1, 1, 1]. Then, the kernel is normalized to calculate the weight of each frequency bin. This results in the normalized kernel: $[\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, 0, 0, 0, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}]$. This kernel is used to average over the bins of the power-spectral density spectrum. After performing this convolutional operation, the calculated SNR spectrum is padded with Not a Number (**NaN**) values to match the number of bins again of the power-spectral density spectrum.

Because the stimulus frequencies were often in between frequency bins the approach of Gramfort et al. [15] is modified, as it only took the closest frequency bin as the output SNR value. The reason for this was that the immediate neighboring frequency bins sometimes had higher SNR values than the one located closest to our stimulus frequency bin. As mentioned in Liu et al. [26] and Zheng et al. [44], the peak of the SNR is located close to the actual stimulus frequency. That is why interpolation is used to calculate the SNR directly at the stimulation frequency and at ±0.15Hz from the stimulation frequency. Interpolation is realized according to the weighted average of the neighboring frequency bins with respect to the target frequency. Of the two neighboring bins, the one closest to the number has the largest weight. The interpolation method is called inverse distance weighted **(IDW)** [14]. In this approach, it is linear as the power taken is 1. Important to note is that if the target frequency matches one of the calculated frequencies in the frequency bins, then that value is taken and no interpolation is used. After creating the interval consisting of the 3 SNR measurements at the 3 frequencies: [SNR at target frequency-0.15Hz, SNR at target frequency, SNR at target frequency+0.15Hz], the average is taken at each frequency of the frequency interval across the channels. The channels/electrodes aver-

aged over are O1, and O2, located in the occipital region, as this is one of the best areas to measure them [7][41]. However, due to a faulty electrode, only O1 is used for experiment 1 in participant 5 (see tables 4.2 and 4.3). From the 3 SNR values that make up the interval, the maximal value is taken. A summary of all these steps can be found in figure 4.2, which describes the overall SNR workflow for SNR extraction from trials.

After the SNR is extracted for all the trials for each participant, each combination of stimuli characteristics has 3 SNR values that correspond to the 3 trials performed per trial setting combination per participant. This means all 108 unique trials have 3 values as each trial is performed 3 times per participant. After that, the data of all the participants is aggregated. The aggregation method is further explained in the next section (section 4.1.2).

4.1.2. Method of grouping

To explain the method of grouping used for the statistical analyses, let's take SNR as an example. When trying to investigate the relationships between SNR and stimuli characteristics the data has to be grouped. For each trial, the maximal SNR is calculated and the stimuli characteristics used for each trial are registered. Let's take experiment 1 as an example. There are 6 participants that each perform 3 trials per combination of interface characteristics. This means that in total 18 trials are performed per combination of interface characteristics over all the participants. In the experiment, there are 3 colors, 3 shapes, 4 frequencies, and 3 sizes of stimuli. Let's take the 3 colors as an example. Each color covers the 3 shapes, 4 frequencies, and 3 sizes across each color. This means that each color covers 3*4*3=36 unique combinations of stimuli characteristics. Multiply this by 6 participants across 3 trials each means that each color group contains the maximal SNR, measured at the stimulus frequencies, of 36*6*3=648 trials. This also means that for each participant there is a total of 36*3=108 trials where the maximal SNR is measured. Thus, to summarize it, all trials are grouped that meet the required characteristics of the group that we want to analyze.
Each of these groups of trials are used to represent that group for analysis which lead to the results of this research. The table 4.4 shows the resulting group sizes when this is performed for both experiments 1 and 2.
Important to recognize is that when statistical analyses are performed within the same experiment and category all the group sizes compared are the same. However, when this is performed between experiments 1 and 2 for the color red, it can be seen that the sample sizes are unequal. To explain the calculation for experiment 2, 1 (color)*3 (pixel surfaces)*4 (frequencies)*4 (for each location)*6 (participants)*3 (trials per participant)=864 trials for the color red.

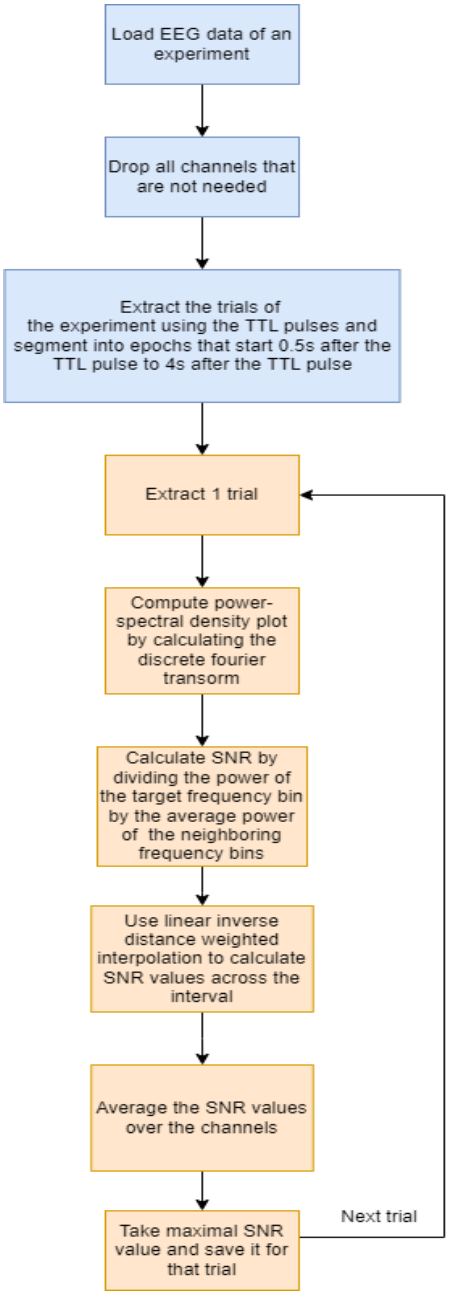


Figure 4.2: The SNR workflow for SNR extraction from trials

The grouping performed for the questionnaire is done per metric to compare the differences between each cate-

gory. Let's take the color red again. Each participant gives one number to grade red with respect to a metric. Let's take comfort as an example. This means that when comparing red to green each group contains 6 numbers, one for each participant (see table 4.21). This group size is consistent when performing statistical analyses to compare all the metrics between the groups of the aspects of a specific category of stimuli characteristics.
When calculating the correlation between the metrics (the reason is explained in section 4.6) all the data is aggregated of each metric. Each metric grades 14 different aspects and each participant grades each aspect once. This means that the group size for each metric is in this case 14*6=84 values.

4.1.3. Statistical workflow

To investigate the difference between means various statistical methods are applied in sequence. Assumed is that the data has a normal distribution in the analyses. In all of the applied tests, a 95% confidence interval ($\alpha = 0.05$) is used. First, Bartlett's method is applied to identify if the variances are homogeneous or not [2]. If the variances are equal, one-way ANOVA is applied to test if the means are different [11][31]. If this is the case, Tukey's post hoc test is applied to determine if these differences are significant [31]. If the variances are unequal, Welch's ANOVA is applied to determine if the means are different [11][31]. If this is the case, the Games-Howell post hoc test is used to determine if the differences are significant [11][31].
Furthermore, it is important to note that officially the notation for p values lower than 0.001 should have a notation of p<0.001. However, in the tables of this research, the notation of 0.000 is used. Another important abbreviation to explain is degrees of freedom (**df**) which is often used to explain ANOVA tests. These are also mentioned in the tables 4.5, 4.12, and 4.19, which describe the statistical workflow.

4.2. The influence of the combination of interface characteristics

Visualizations of the influence of the interface characteristics on the SNR can be found in the plots of appendices A and B. The visualizations show the mean and standard deviation of each group. The actual data that is visualized can be found in tables 4.4 and A.1 (see appendix A). To better compare the actual results and show that the SNR does not go below zero as the standard deviations suggest, figures 4.3 until 4.18 show the boxplots and means across the different combinations of settings. The statistical analysis for each specific combination of 3 or more categories of interface characteristics is not performed to see if the means are different. The reason is that this would make the analyses of the report overly complex and the dataset size might be too small to draw any solid conclusions.
Statistical analyses are performed within each category of stimulus characteristics of the experiments itself. After that, a one-on-one comparison between the experiments is made between the characteristics of each category.

Table 4.4: SNR table showing for each experiment per characteristic per category what the mean SNR and std are.

experiment	category	group	SNR group A average	SNR group A std	group A number of samples	SNR group B average	SNR group B std	group B number of samples	SNR group C average	SNR group C std	group C number of samples	SNR group D average	SNR group D std	group D number of samples
Experiment 1	pixel surface	[10000 20000 30000]	3.28	3.69	648	3.82	4.71	648	4.12	4.32	648			
Experiment 1	color	['green' 'red' 'white']	3.51	4.26	648	3.68	3.93	648	4.04	4.58	648			
Experiment 1	shape	['circles' 'squares' 'triangles']	3.80	4.01	648	4.03	4.92	648	3.40	3.79	648			
Experiment 1	frequency	[8 13 19 25]	3.26	4.40	486	3.20	3.16	486	3.89	4.16	486	4.62	5.02	486
Experiment 2	pixel surface	[10000 20000 30000]	3.11	3.45	864	3.59	3.92	864	3.99	4.57	864			
Experiment 2	color	['green' 'red' 'white']	3.31	4.02	864	3.59	3.89	864	3.78	4.15	864			
Experiment 2	frequency	[8 13 19 25]	3.41	4.26	648	2.95	2.81	648	3.82	3.97	648	4.07	4.71	648

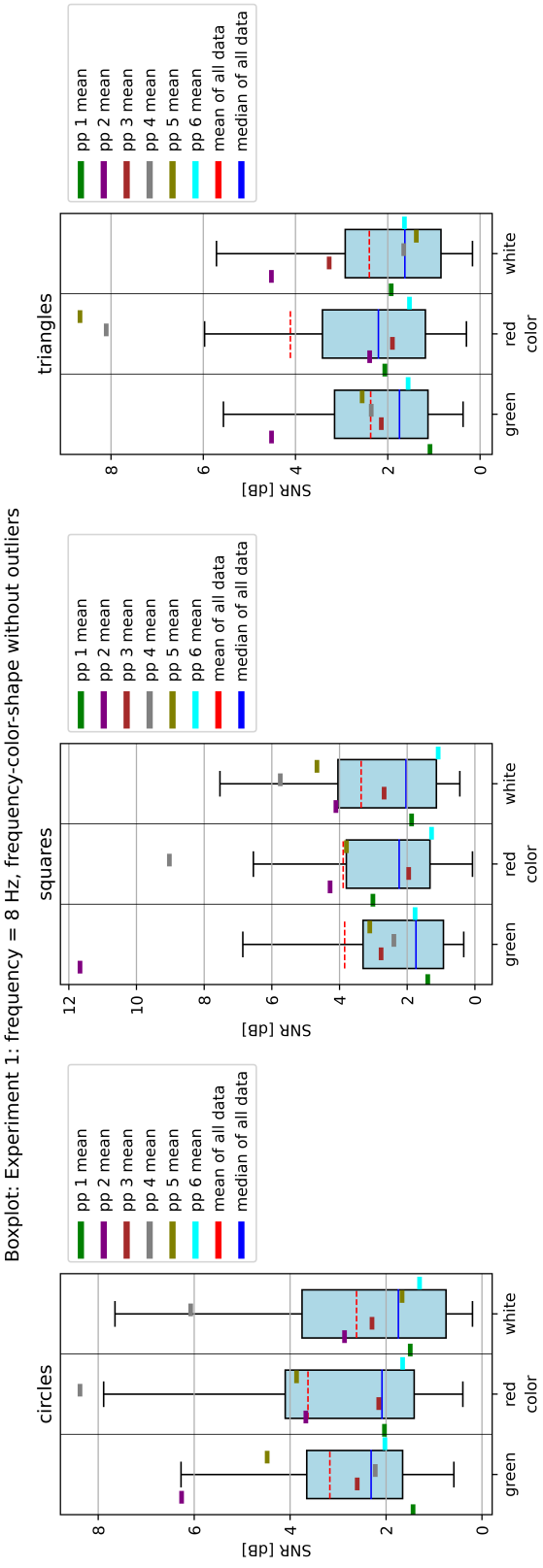


Figure 4.3: SNR boxplot of color vs shape of experiment 1 at 8Hz

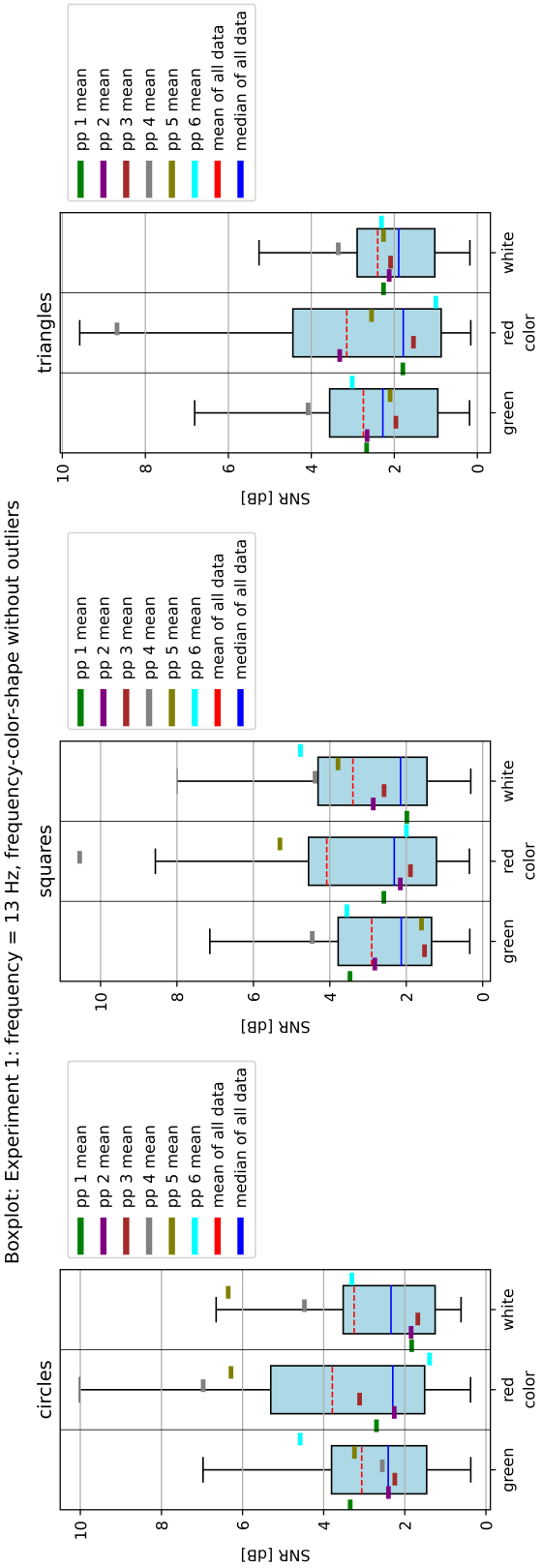


Figure 4.4: SNR boxplot of color vs shape of experiment 1 at 19Hz

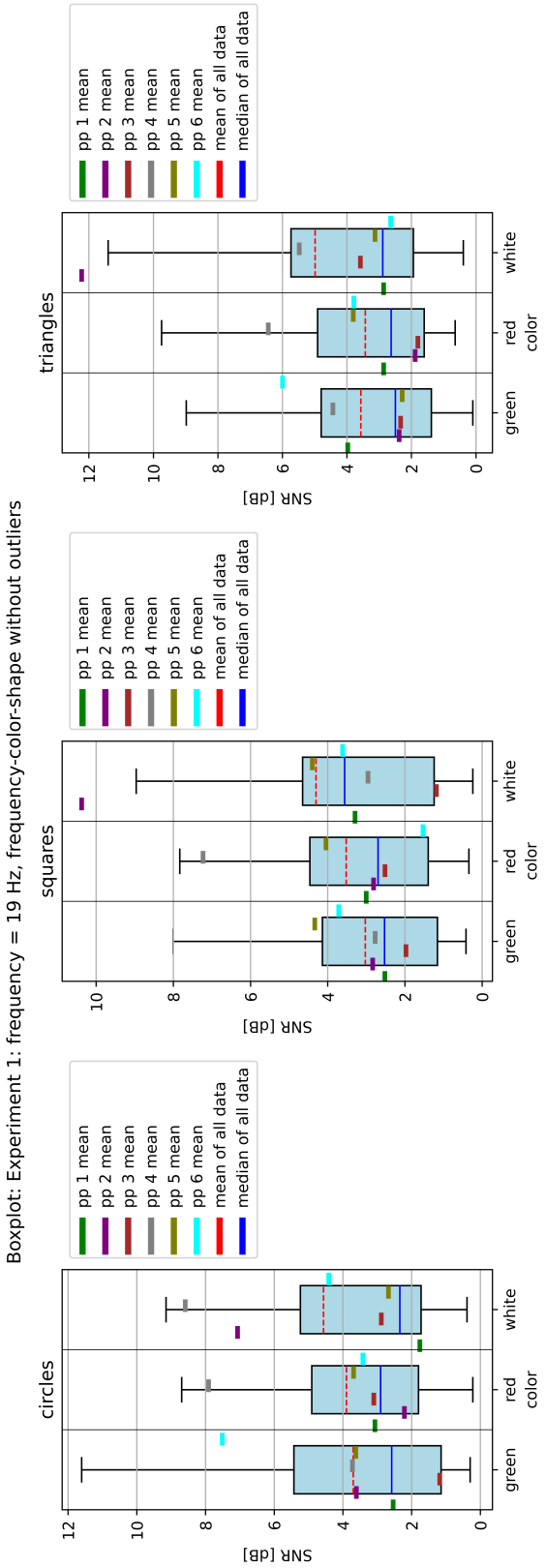


Figure 4.5: SNR boxplot of color vs shape of experiment 1 at 19Hz

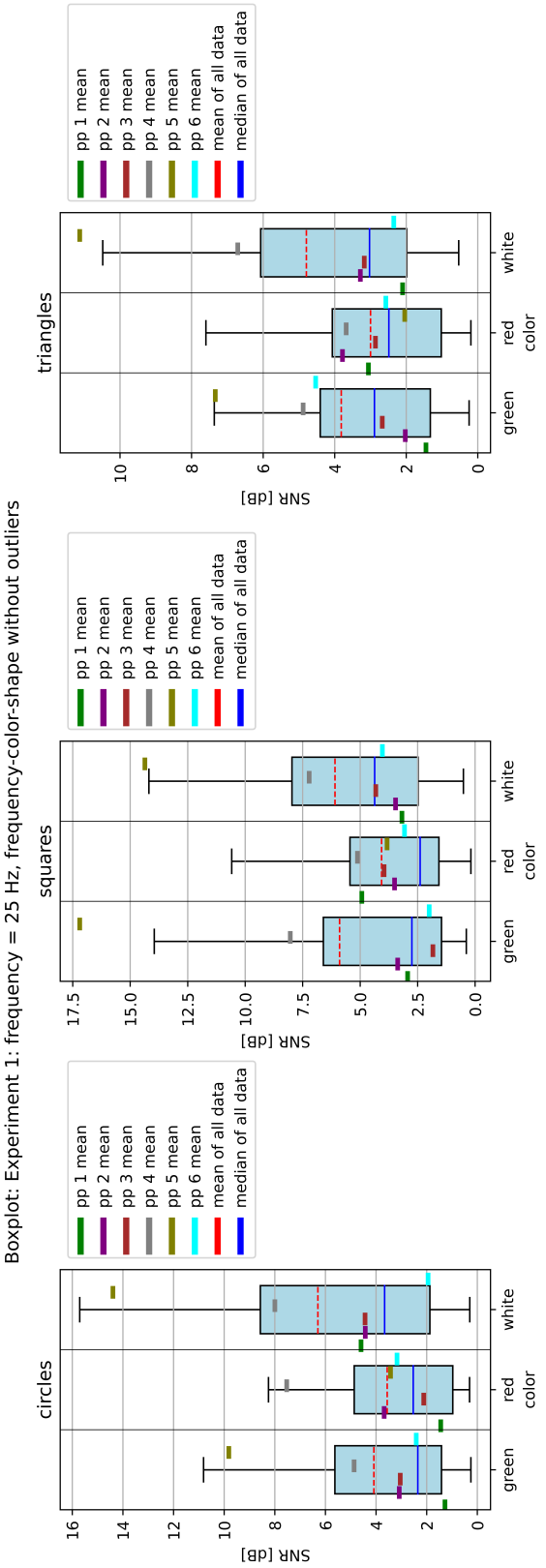


Figure 4.6: SNR boxplot of color vs shape of experiment 1 at 25Hz

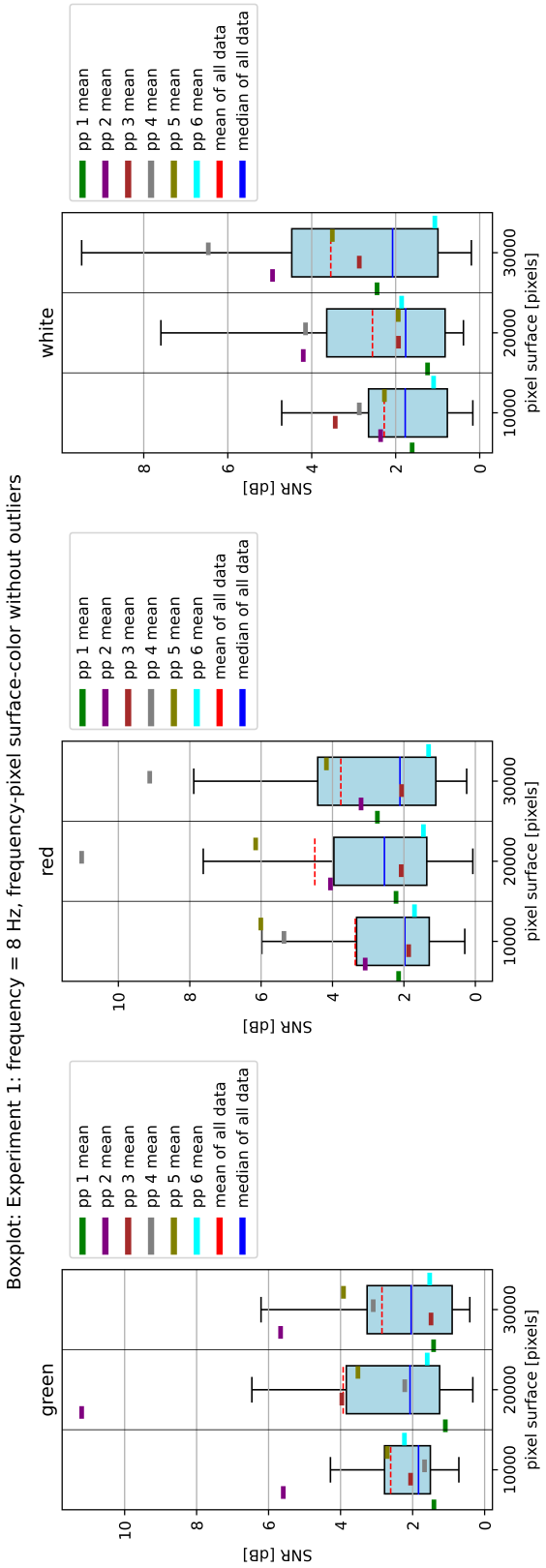


Figure 4.7: SNR boxplot of pixel surface vs color of experiment 1 at 8Hz

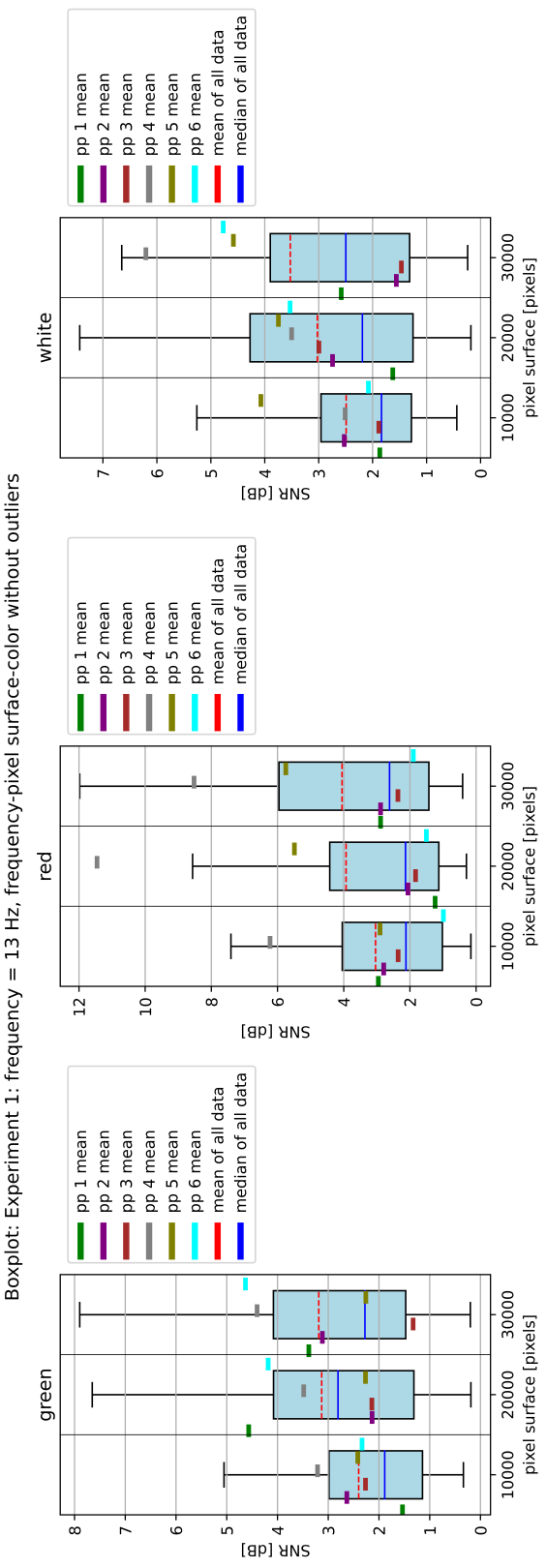


Figure 4.8: SNR boxplot of pixel surface vs color of experiment 1 at 13Hz

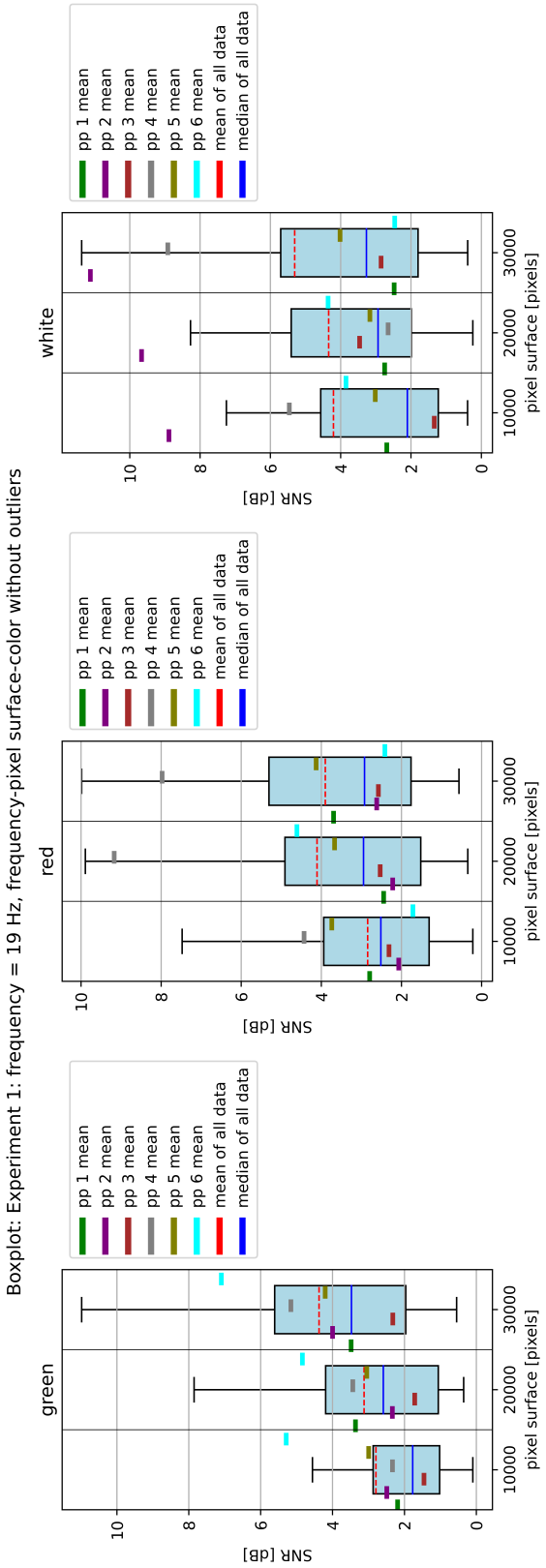


Figure 4.9: SNR boxplot of pixel surface vs color of experiment 1 at 19Hz

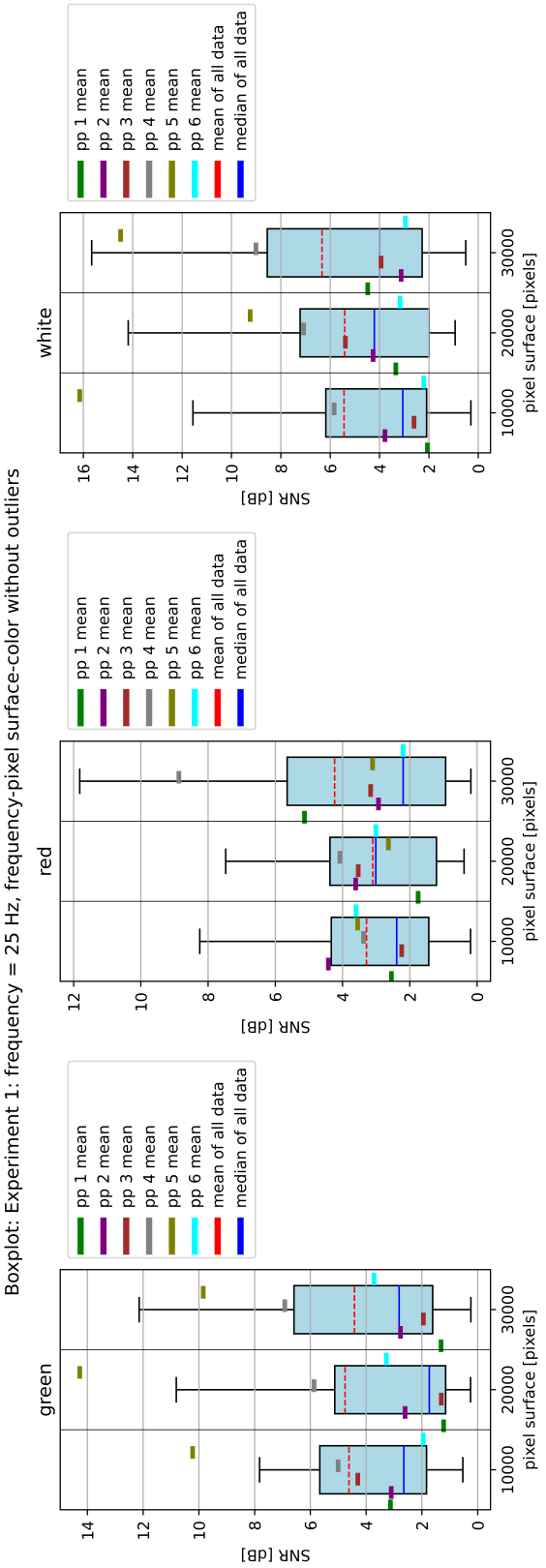


Figure 4.10: SNR boxplot of pixel surface vs color of experiment 1 at 25Hz

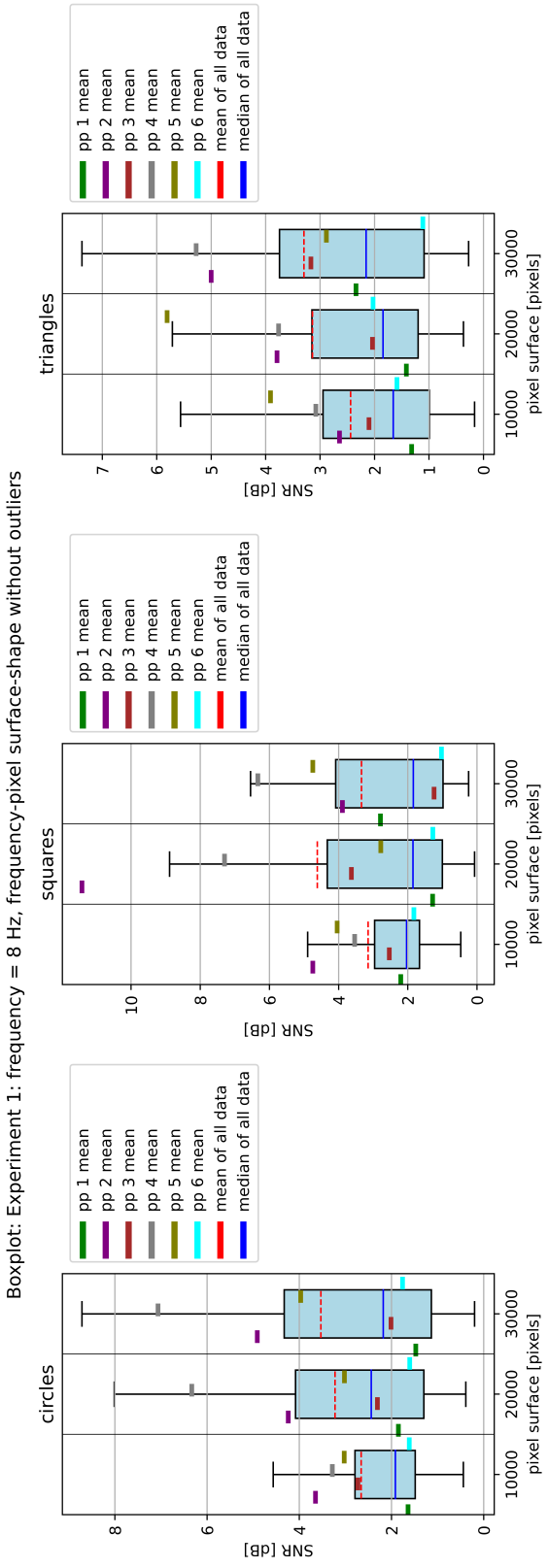


Figure 4.11: SNR boxplot of pixel surface vs shape of experiment 1 at 8Hz

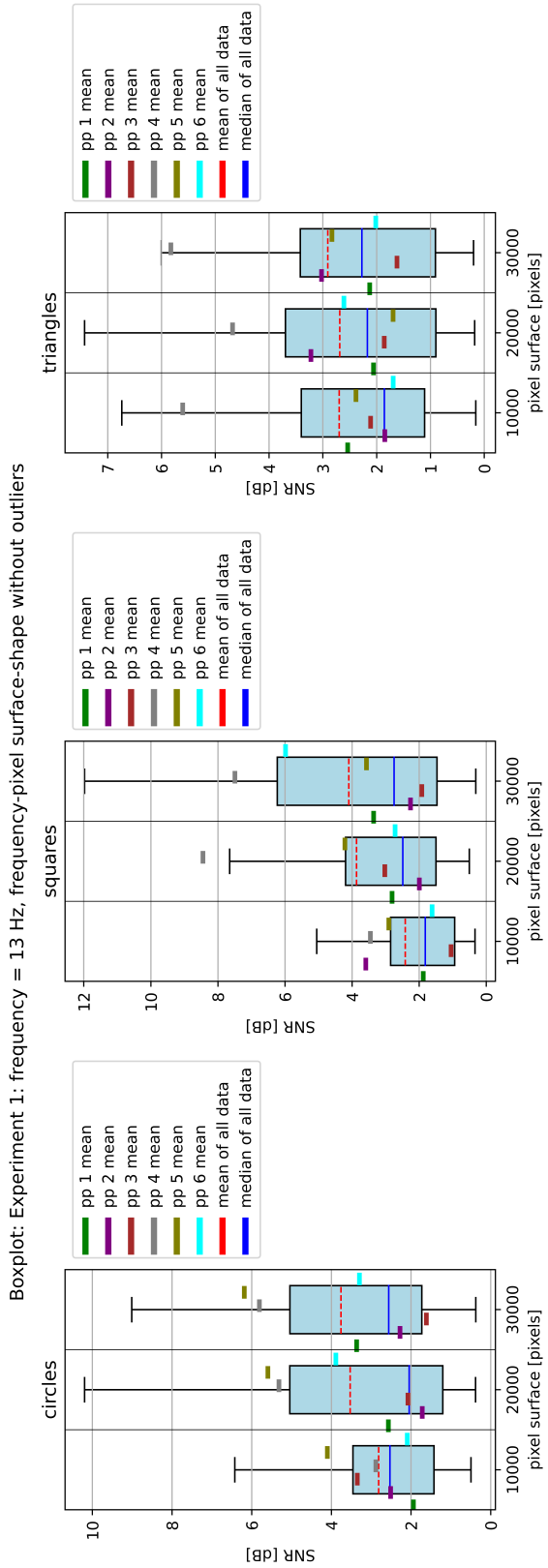


Figure 4.12: SNR boxplot of pixel surface vs shape of experiment 1 at 13Hz

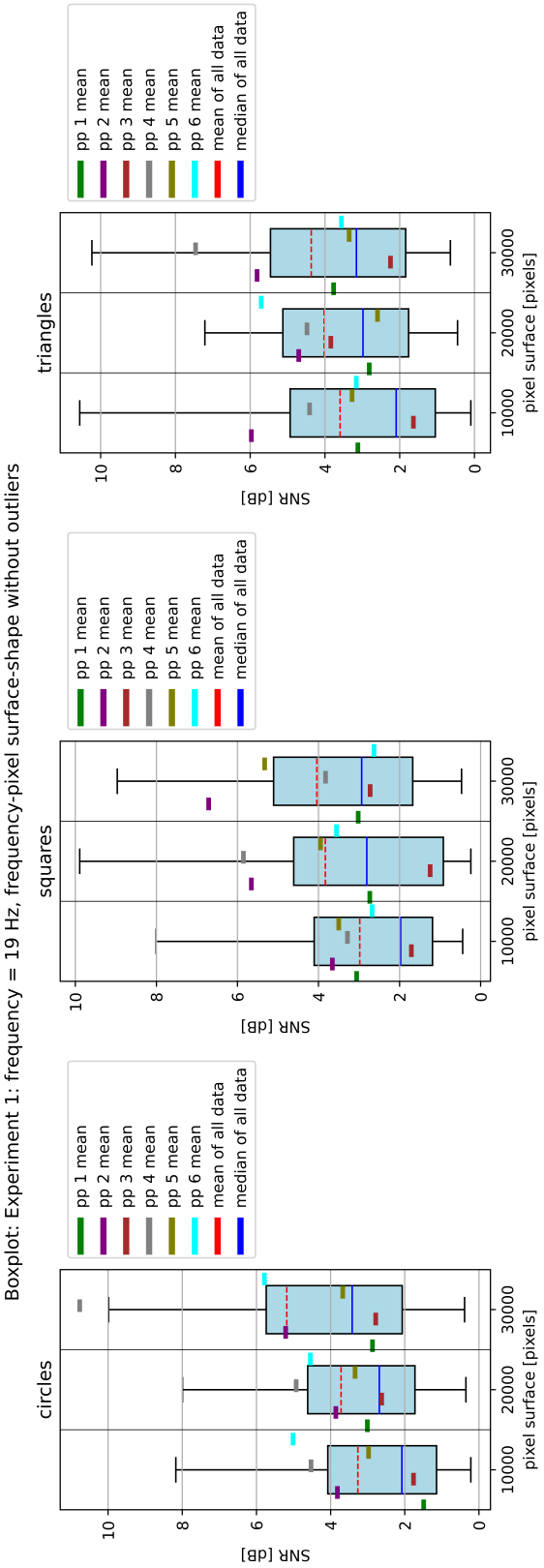


Figure 4.13: SNR boxplot of pixel surface vs shape of experiment 1 at 19Hz

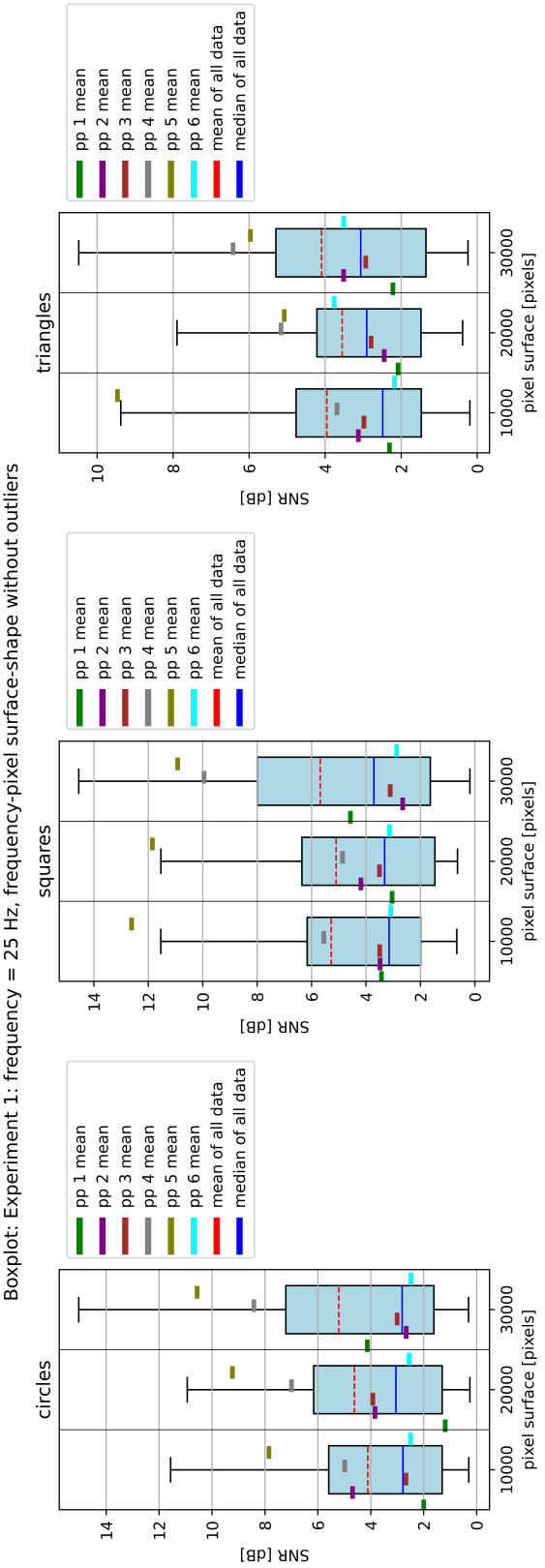


Figure 4.14: SNR boxplot of pixel surface vs shape of experiment 1 at 25Hz

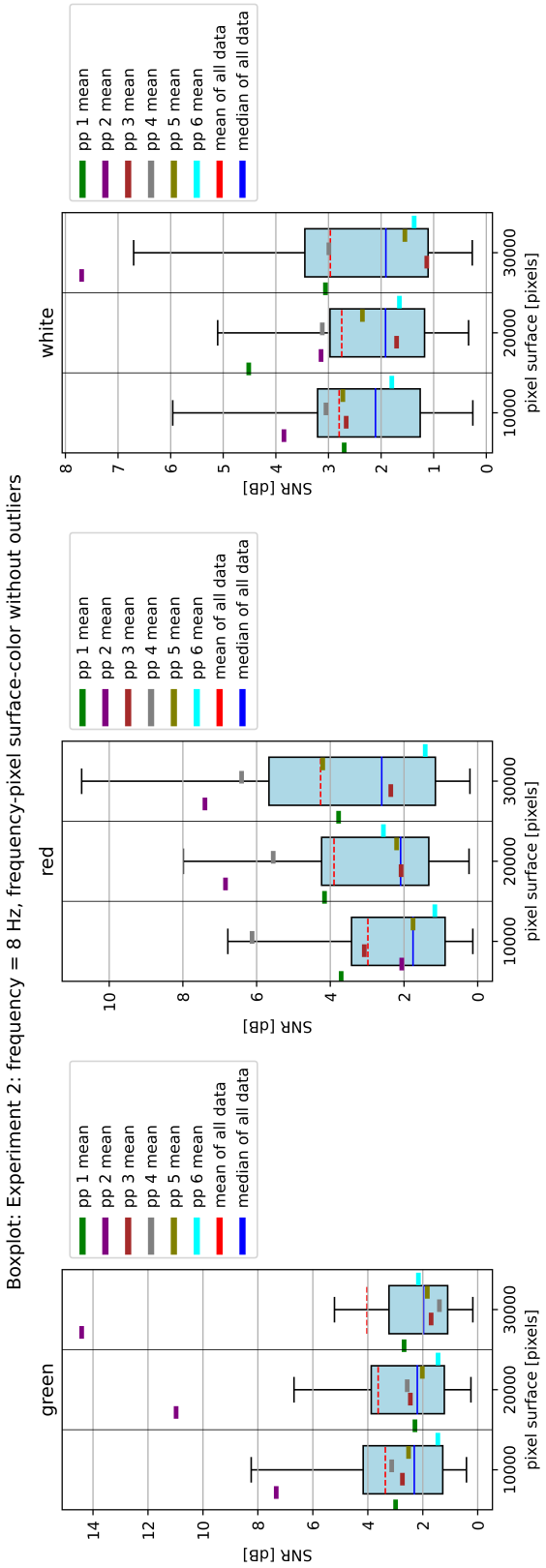


Figure 4.15: SNR boxplot of pixel surface vs color of experiment 2 at 8Hz

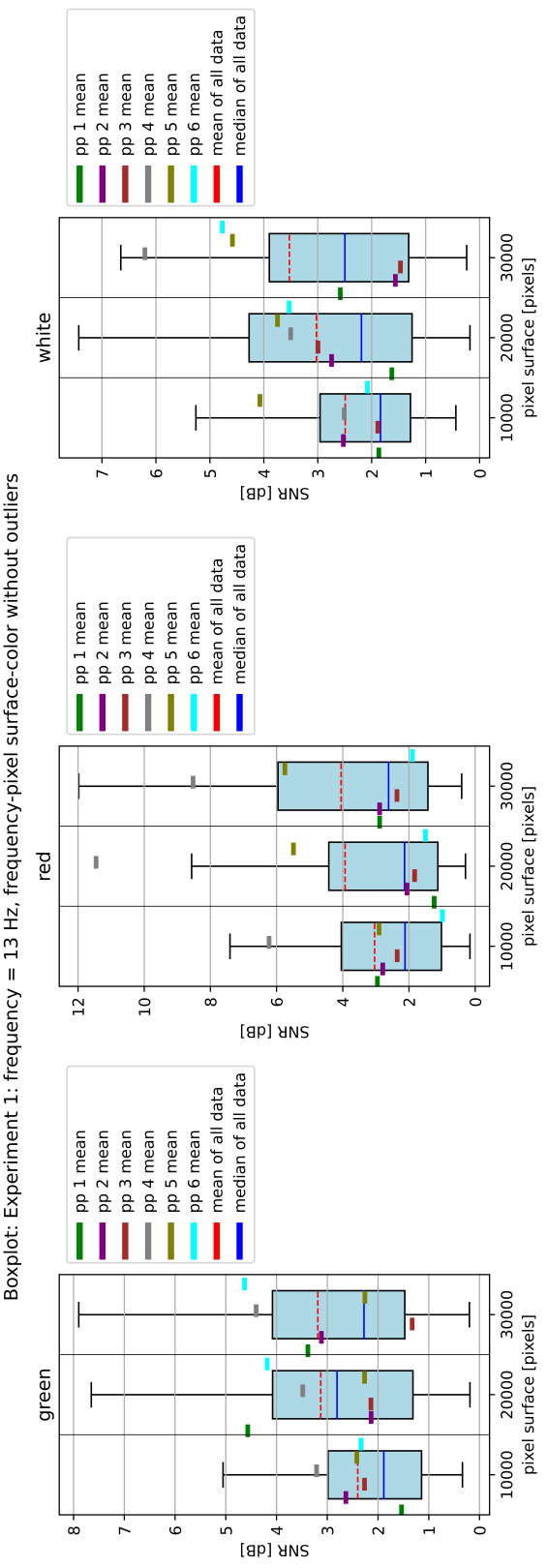


Figure 4.16: SNR boxplot of pixel surface vs color of experiment 2 at 13Hz

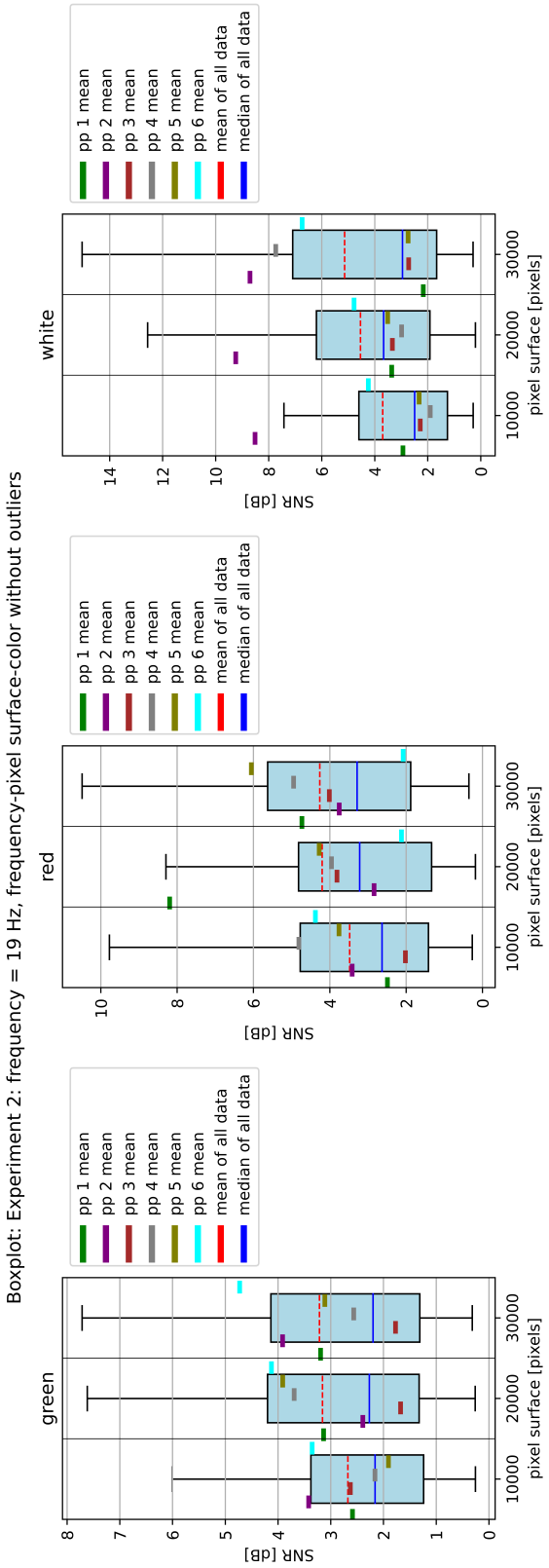


Figure 4.17: SNR boxplot of pixel surface vs color of experiment 2 at 19Hz

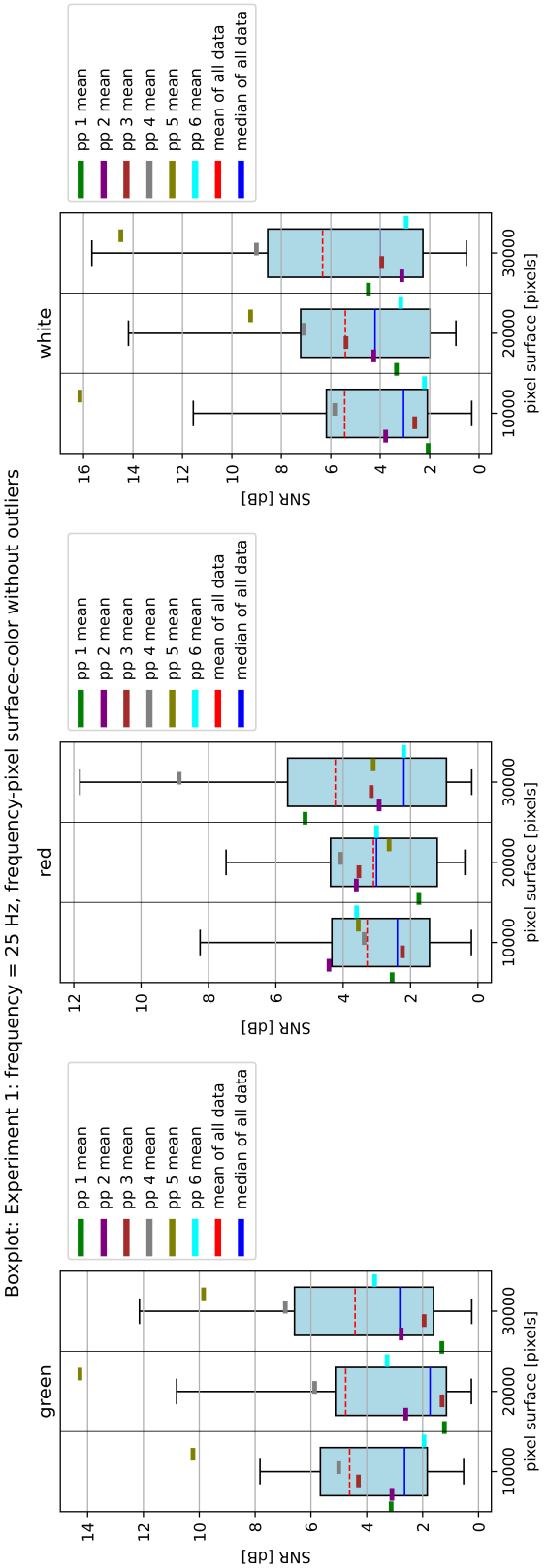


Figure 4.18: SNR boxplot of pixel surface vs color of experiment 2 at 25Hz

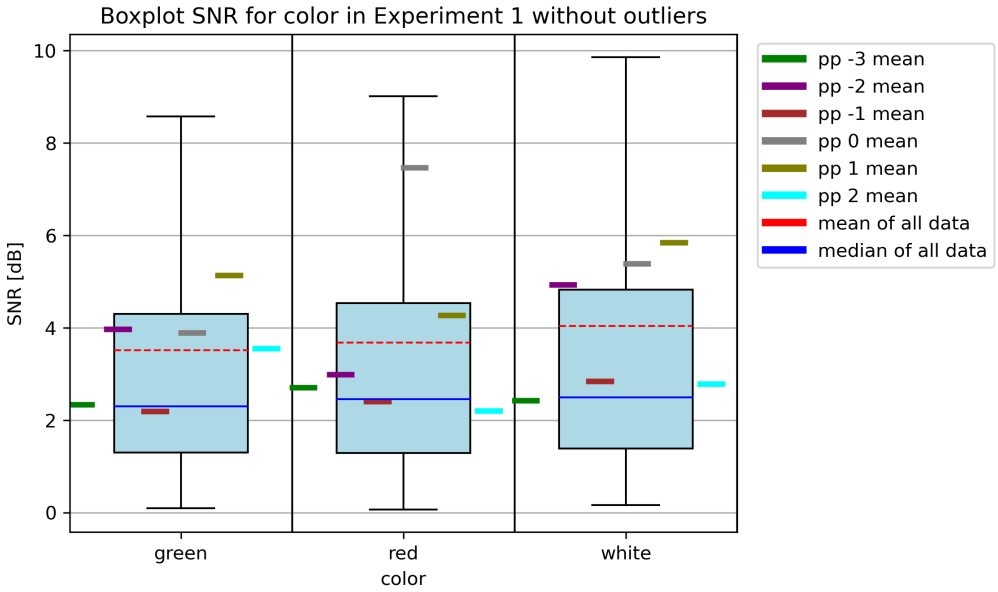


Figure 4.19: Experiment 1 boxplot of color

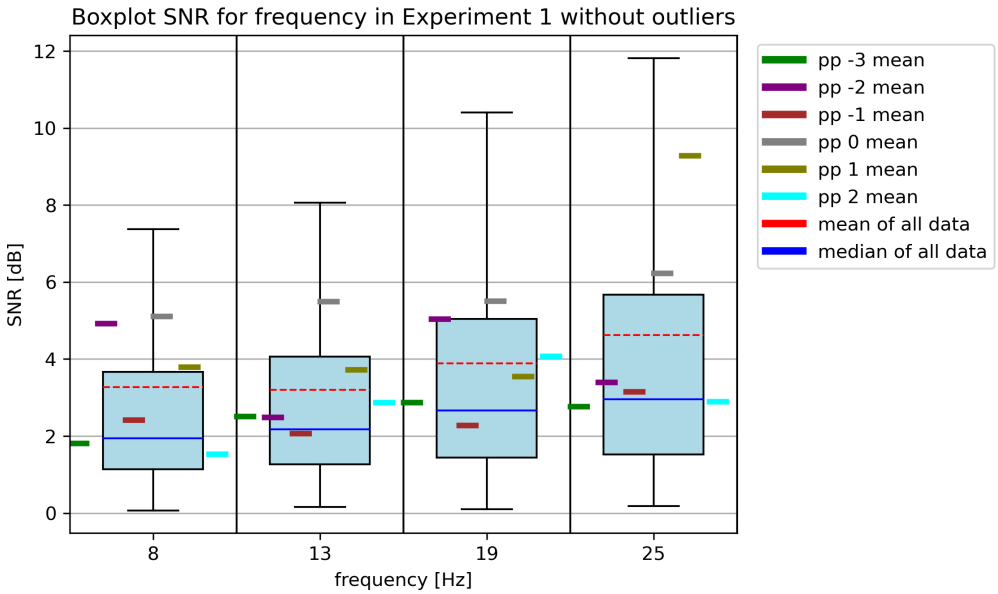


Figure 4.20: Experiment 1 boxplot of frequency

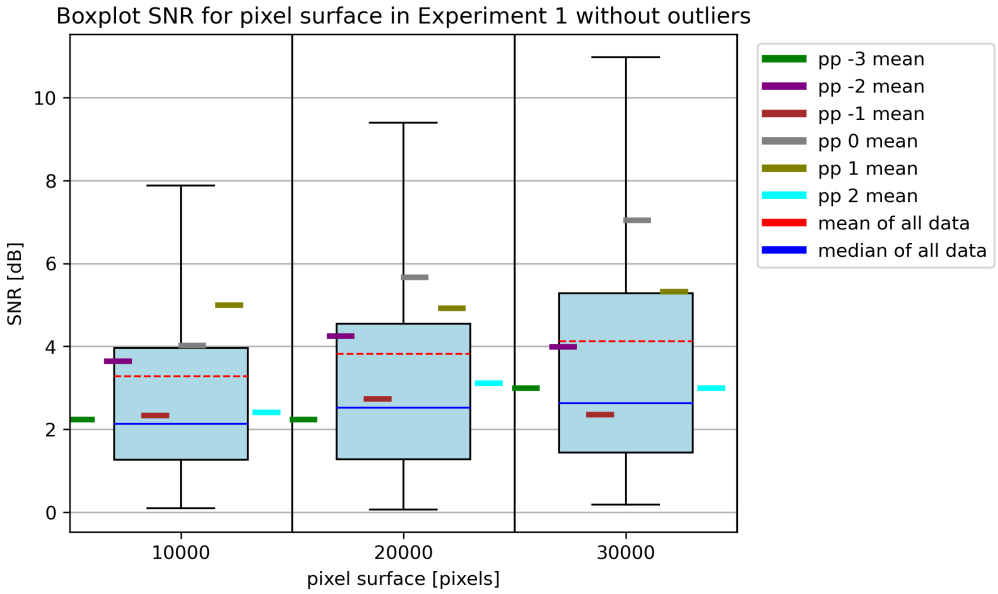


Figure 4.21: Experiment 1 boxplot of pixel surface

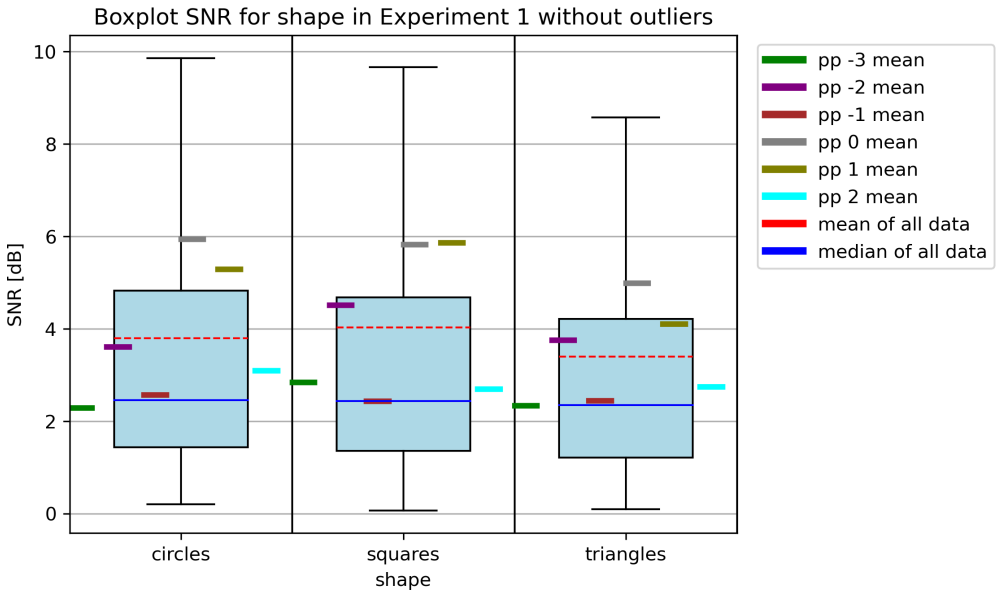


Figure 4.22: Experiment 1 boxplot of shape

4.3. Experiment 1

Experiment 1 showed a single target stimulus at a time. The stimuli vary across colors, shapes, sizes, and frequencies. It is used to investigate how different combinations of interface characteristics influence the measured SNR in an isolated environment.

Visualizations of the overall distribution of the SNR measurements for each stimulus characteristic can be found in figures 4.19 until 4.22. The visualizations also show the means of each participant and these vary a lot. Additionally, alternative visualizations of the results of experiment 1 for the interface characteristics can be found in appendix B, and the exact results are displayed in tables 4.4 until 4.11. Important to note is that the standard deviations suggest that the SNR goes below 0 dB in certain measurements. However, the boxplots that are shown in figures 4.19 until 4.22 prove otherwise. The reason is that the SNR can never go below 0 dB. An overview of the exact sample sizes used for each group can be found in table 4.4.

According to table 4.5, the analysis shows no difference between the means of the different colors (Welch's ANOVA, p=0.097). The mean SNR values for the colors green, red, and white, were 3.51, 3.68, and 4.04 respectively. However, it shows differences in the category pixel surface (Welch's ANOVA, p=0.001), shape (Welch's ANOVA, p=0.024), and frequency (Welch's ANOVA, p<0.001). The results of the Games-Howell post hoc tests for pixel surface, shape, and frequency, can be found in tables 4.10 with 4.11, 4.8 with 4.9, and 4.6 with 4.7 respectively.

The results show for the category pixel surface that 10.000 versus 30.000 pixels have a significant difference (Games-Howell, p<0.001) between the means. Looking at table 4.4 and figure B.2, a clear positive relationship can be seen between the increase of the pixel surface and the measured SNR.

For the category shape, the results show only a significant difference (Games-Howell, p=0.025) between the triangles and squares. Looking at table 4.4 and figure B.4, the triangles report an SNR mean of 3.40, followed by circles with 3.80, and squares with 4.03.

The category frequency shows multiple significant differences between the means. Between 8Hz and 25Hz (Games-Howell, p<0.001), 13Hz and 25Hz (Games-Howell, p<0.001), and 13Hz and 19Hz (Games-Howell, p=0.019). Table 4.4 and figure B.1 show for 8Hz, 13Hz, 19Hz, and 25Hz, the SNR means of 3.26, 3.20, 3.89, and 4.62 respectively.

Table 4.5: Experiment 1 analyzation workflow

category	groups	p value bartlett	Variances equal?	ANOVA method	p value ANOVA method	df between groups	df within groups	Mean different?	Type of post hoc test
pixel surface	[10000 20000 30000]	0.000	No	Welch's ANOVA	0.001	2	1280.31	Yes	Games-Howell
color	['green' 'red' 'white']	0.001	No	Welch's ANOVA	0.097	2	1288.98	No	None
shape	['circles' 'squares' 'triangles']	0.000	No	Welch's ANOVA	0.024	2	1280.03	Yes	Games-Howell
frequency	[8 13 19 25]	0.000	No	Welch's ANOVA	0.000	3	1062.11	Yes	Games-Howell

Table 4.6: Experiment 1 frequency p value Games-Howell post hoc

frequency	8	13	19	25
8		0.992	0.106	0.000
13	0.992		0.019	0.000
19	0.106	0.019		0.064
25	0.000	0.000	0.064	

Table 4.7: Experiment 1 frequency significant Games-Howell post hoc

frequency	8	13	19	25
8	X	Not significant	Not significant	Significant
13	Not significant	X	Significant	Significant
19	Not significant	Significant	X	Not significant
25	Significant	Significant	Not significant	X

Table 4.8: Experiment 1 shape p value Games-Howell post hoc

shape	circles	squares	triangles
circles		0.625	0.148
squares	0.625		0.025
triangles	0.148	0.025	

Table 4.9: Experiment 1 shape significant Games-Howell post hoc

Shape	circles	squares	triangles
circles	X	Not significant	Not significant
squares	Not significant	X	Significant
triangles	Not significant	Significant	X

Table 4.10: Experiment 1 pixel surface p value Games-Howell post hoc

Pixel surface	10000	20000	30000
10000		0.053	0.000
20000	0.053		0.458
30000	0.000	0.458	

Table 4.11: Experiment 1 pixel surface significant Games-Howell post hoc

Pixel surface	10000	20000	30000
10000	X	Not significant	Significant
20000	Not significant	X	Not significant
30000	Significant	Not significant	X

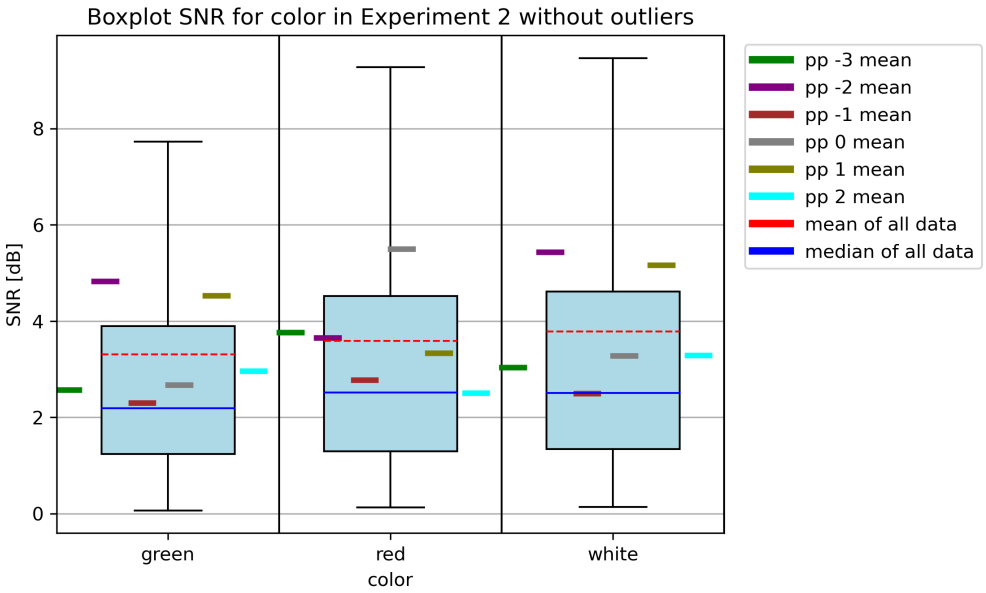


Figure 4.23: Experiment 2 boxplot of color

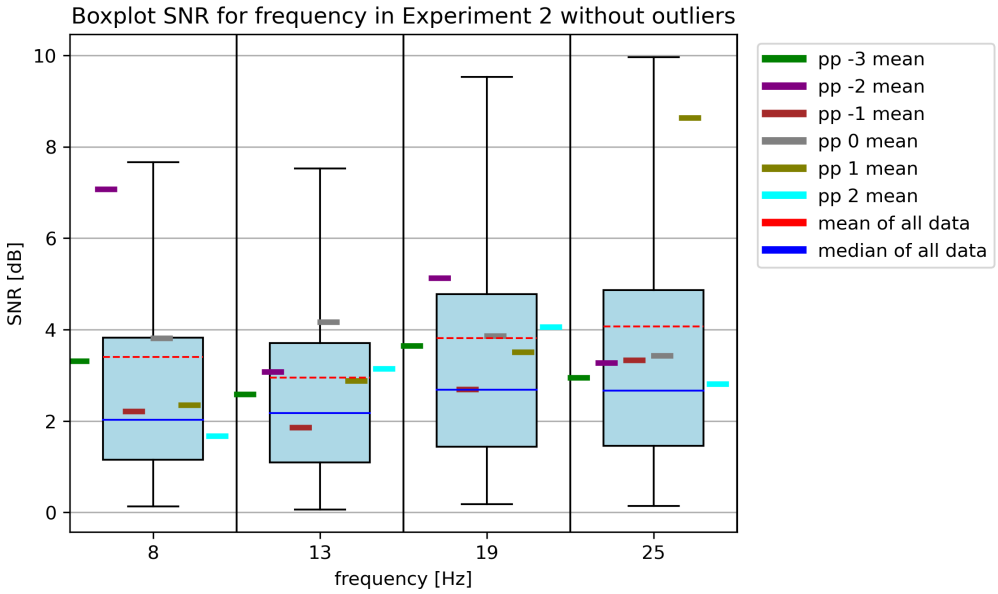


Figure 4.24: Experiment 2 boxplot of frequency

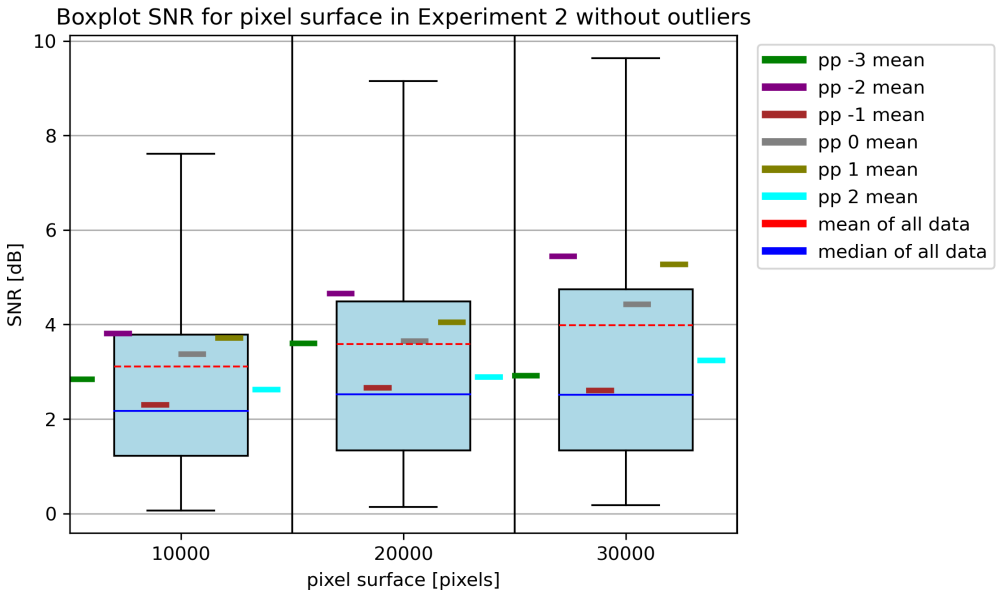


Figure 4.25: Experiment 2 boxplot of pixel surface

4.4. Experiment 2

Experiment 2 showed 4 stimuli simultaneously and one of these stimuli is our target stimulus. It is shown across the same frequencies, colors, and sizes as experiment 1. It is used to investigate how different combinations of interface characteristics influence the measured SNR in an SSVEP-speller environment. The exact sample sizes used for each group are stated in table 4.4.

Visualizations of the overall distribution of the SNR measurements can be found in figures 4.23 until 4.25. The visualizations also show the means of each participant and these vary a lot between the participants. The exact results for the interface characteristics of experiment 2 can be found in table 4.4, tables 4.12 until 4.18, and appendix B. According to table 4.12, the ANOVA tests indicate different means for all the categories. The pixel surface (Welch's ANOVA, $p<0.001$) category shows significant differences between means between various pairs of sizes. It shows a significant difference between 10.000 and 20.0000 pixels (Games-Howell, $p=0.021$), and between 10.000 and 30.000 pixels (Games-Howell, $p<0.001$) (see tables 4.17 and 4.18). The SNR means for the 10.000, 20.000, and 30.000 pixels, are 3.11, 3.59, and 3.99 respectively.

Color (One-way ANOVA, $p=0.049$) shows only a significant difference between the colors green and white (Tukey's, $p=0.039$) (see tables 4.15 and 4.16). For the colors green, red, and white, the SNR means were 3.31, 3.59, and 3.78 respectively.

The p-values in the frequency category (Welch's ANOVA, $p<0.001$) showed significant differences in means between various frequencies. This was the case between 8Hz and 25Hz (Games-Howell, $p=0.038$), 13Hz and 19Hz (Games-Howell, $p<0.001$), and between 13Hz and 25Hz (Games-Howell, $p<0.001$) (see tables 4.13 and 4.14). The SNR means for 8Hz, 13Hz, 19Hz, and 25Hz were 3.41, 2.95, 3.82, and 4.07 respectively.

Table 4.12: Experiment 2 analyzation workflow

category	groups	p value bartlett	Variances equal?	ANOVA method	p value ANOVA method	df between groups	df within groups	Mean different?	Type of post hoc test
pixel surface	[10000 20000 30000]	0.000	No	Welch's ANOVA	0.000	2	1704.70	Yes	Games-Howell
color	['green' 'red' 'white']	0.165	Yes	One-way ANOVA	0.049	2	2589.00	Yes	Tukey's test
frequency	[8 13 19 25]	0.000	No	Welch's ANOVA	0.000	2	1407.59	Yes	Games-Howell

Table 4.13: Experiment 2 frequency p value Games-Howell post hoc

Frequency	8	13	19	25
8		0.107	0.276	0.038
13	0.107		0.000	0.000
19	0.276	0.000		0.714
25	0.038	0.000	0.714	

Table 4.14: Experiment 2 frequency significant Games-Howell post hoc

Frequency	8	13	19	25
8	X	Not significant	Not significant	Significant
13	Not significant	X	Significant	Significant
19	Not significant	Significant	X	Not significant
25	Significant	Significant	Not significant	X

Table 4.15: Experiment 2 color p value Tukey's post hoc

Color	green	red	white
green		0.318	0.039
red	0.318		0.580
white	0.039	0.580	

Table 4.16: Experiment 2 color significant Tukey's post hoc

Color	green	red	white
green	X	Not significant	Significant
red	Not significant	X	Not significant
white	Significant	Not significant	X

Table 4.17: Experiment 2 pixel surface p value Games-Howell post hoc

Pixel surface	10000	20000	30000
10000		0.021	0.000
20000	0.021		0.124
30000	0.000	0.124	

Table 4.18: Experiment 2 pixel surface significant Games-Howell post hoc

Pixel surface	10000	20000	30000
10000	X	Significant	Significant
20000	Significant	X	Not significant
30000	Significant	Not significant	X

4.5. Experiment 1 vs Experiment 2

This comparison between a single stimulus versus multiple stimuli is novel to our knowledge. It answers the question if the surrounding stimuli in an SSVEP-speller affect the measured SNR of the target stimulus. To investigate if there were any differences between experiment 1 and experiment 2 across the settings a one-on-one comparison is made using the described statistical workflow. The distribution and sample sizes and means of each characteristic (group) per experiment can be found in table 4.4. The results of processing each characteristic and comparing the results between experiments 1 and 2 can be seen in table 4.19. The results show no significant differences between the means. However, when looking at table 4.4 the means show an overall slight decrease in each characteristic property of each category. The only exception is frequency 8, which increased by 0.15 dB in experiment 2. The biggest difference between experiment 1 and experiment 2 could be seen in the 25Hz characteristic, which had 0.55 dB decreased in experiment 2.

Table 4.19: One-on-one comparison of each setting between experiment 1 and experiment 2

category	groups	p value bartlett	Variances equal?	ANOVA method	p value ANOVA method	df between groups	df within groups	Mean different?
pixel surface	10000	0.070	Yes	One-way ANOVA	0.365	1	1510.00	No
pixel surface	30000	0.130	Yes	One-way ANOVA	0.556	1	1510.00	No
pixel surface	20000	0.000	No	Welch's ANOVA	0.298	1	1243.27	No
frequency	8	0.446	Yes	One-way ANOVA	0.587	1	1132.00	No
frequency	25	0.142	Yes	One-way ANOVA	0.060	1	1132.00	No
frequency	19	0.281	Yes	One-way ANOVA	0.769	1	1132.00	No
frequency	13	0.005	No	Welch's ANOVA	0.178	1	973.83	No
color	white	0.006	No	Welch's ANOVA	0.268	1	1313.47	No
color	red	0.731	Yes	One-way ANOVA	0.676	1	1510.00	No
color	green	0.110	Yes	One-way ANOVA	0.343	1	1510.00	No

4.6. Questionnaire

Each participant was given a questionnaire (see appendix C). The questionnaire is used to capture the subjective perspective of the participants. These insights could help in finding a balance between, for example, comfort levels and the measured SNR values for future research. For clarification, the items

that are named such as black/red in the color section mean black background with the red stimulus.

One important side note has to be taken for participant 3 because the participant deviated from the questionnaire. After all, this person stated that they experienced different experiences with respect to the frequency characteristic between experiment 1 and experiment 2. This relative difference was only 1 level in the high frequency and middle frequency. However, in the low-frequency range, the comfort level scored very high(5) in experiment 1 but very low(1) in experiment 2. Another big difference was seen in the participant's focus level, which was high(4) in experiment 1 and very low(1) in experiment 2. For the other metrics, the difference was non-existent or only 1. To mitigate this problem the average of the scores between the 2 experiments is taken.
The results of the questionnaire are reported in table 4.21. Pearson's correlation coefficients are calculated to see if there is a linear correlation between the metrics used in the questionnaire. This helps in capturing the level of independent metrics used in the questionnaire. This helps in seeing if a combination of certain metrics can be related to the measured SNR values of each interface characteristic. The results can be seen in table 4.20. For clarification, values above ±0.7 suggest a strong relationship, values between ±0.5 and ±0.7 a moderate relationship, and values between ±0.3 and ±0.5 a weak relationship. In short, the closer the values are to ±1 the stronger the relationship. The results show a strong relationship between eye irritation and comfort experienced by the participants. The easiness to focus on the stimulus center shows a moderate relationship with comfort. The results suggest more moderate relationships. Examples can be found between a person's focus level with respect to how easy it is to focus on the stimulus center, and eye irritation with respect to how easy it is to focus on the stimulus center.
These relationships also suggest that a statistical analysis between the metrics is not possible, as the groups for the analysis need to be independent variables. However, analysis internally within the same metric is still possible. By following the statistical workflow the results show no significant differences between the mean values of each setting (e.g., red) of each interface characteristic category (see table 4.22). To clarify the sample sizes used to calculate the coefficients, all data with respect to a specific metric such as Comfort is combined. A total of 6 participants and there are 14 aspects that are graded that range from Black/Red color to Multiple stimuli (see table 4.20). This means that 14*6=84 values of that corresponding metric are there in total. All this data represent that metric and is used to calculate the Pearson's correlation coefficient. The other metrics are also 14 times mentioned across different subjects and are graded by 6 participants, which means each metric has 84 values. This means that to calculate the Person's correlation coefficient both metrics have a sample size of 84 values.

Even though there are no significant differences that can be confirmed statistically, there are observable differences in the results of the questionnaire (see table 4.21). At low frequencies, the participants experienced the highest comfort levels, the lowest eye irritation, and had to blink less compared to the other frequencies. Additionally, the color red showed the highest comfortability score and the lowest eye irritation with respect to the other colors. Furthermore, the triangle was experienced as the least comfortable shape, and participants reported it to be the hardest to focus on the center of the stimulus. The circle, however, was the most comfortable and the easiest to focus on the center of the stimulus. Moreover, the participants stated that they had to blink the least amount at 10.000 pixels. They experienced the 20.000 pixels as stimulus size, the easiest size for focusing on the center of the stimulus.

Table 4.20: Pearsons's correlation coefficients calculated between the metrics

r-values	Comfort	Person's focus level	Eye Irritation	Easiness to focus on stimulus center	Had to blink a lot
Comfort	1.000	0.491	-0.805	0.627	-0.603
Person's focus level	0.491	1.000	-0.394	0.536	-0.368
Eye Irritation	-0.805	-0.394	1.000	-0.562	0.681
Easiness to focus on stimulus center	0.627	0.536	-0.562	1.000	-0.524
Had to blink a lot	-0.603	-0.368	-0.524	-0.524	1.000

Table 4.21: Results of Questionnaire

		Metrics	pp1	pp2	pp3	pp4	pp5	pp6	AVERAGE	STD
Colors	**Black/Red**	Comfort	5	5	2	3	4	4	3.83	1.07
		Person's focus level	5	2	4	4	3	4	3.67	0.94
		Eye Irritation	2	1	4	3	2	2	2.33	0.94
		Easiness to focus on stimulus center	5	3	2	4	4	4	3.67	0.94
		Had to blink a lot	2	3	5	4	2	4	3.33	1.11
Colors	**Black/Green**	Comfort	4	1	2	5	2	2	2.67	1.37
		Person's focus level	4	2	3	4	2	2	2.83	0.90
		Eye Irritation	3	5	2	2	4	4	3.33	1.11
		Easiness to focus on stimulus center	4	3	3	4	2	2	3.00	0.82
		Had to blink a lot	2	3	3	3	4	4	3.17	0.69
Colors	**Black/White**	Comfort	2	3	5	2	4	2	3.00	1.15
		Person's focus level	3	3	3	3	4	4	3.33	0.47
		Eye Irritation	4	3	1	4	2	4	3.00	1.15
		Easiness to focus on stimulus center	3	3	5	2	4	3	3.33	0.94
		Had to blink a lot	3	3	2	4	3	4	3.17	0.69
Shapes	**Triangle**	Comfort	2	2	1	3	3	4	2.50	0.96
		Person's focus level	2	3	4	4	4	2	3.17	0.90
		Eye Irritation	4	3	4	3	3	2	3.17	0.69
		Easiness to focus on stimulus center	2	3	2	2	3	4	2.67	0.75
		Had to blink a lot	4	3	4	2	2	4	3.17	0.90
Shapes	**Square**	Comfort	5	4	3	3	4	4	3.83	0.69
		Person's focus level	4	3	4	2	4	2	3.17	0.90
		Eye Irritation	2	3	2	3	3	2	2.50	0.50
		Easiness to focus on stimulus center	4	3	5	4	4	3	3.83	0.69
		Had to blink a lot	2	3	3	4	2	3	2.83	0.69
Shapes	**Circle**	Comfort	5	4	5	4	4	4	4.33	0.47
		Person's focus level	4	2	4	4	4	2	3.33	0.94
		Eye Irritation	2	3	1	1	3	2	2.00	0.82
		Easiness to focus on stimulus center	5	3	5	5	4	2	4.00	1.15
		Had to blink a lot	2	3	1	2	3	3	2.33	0.75
Frequency	**High frequency**	Comfort	5	4	1.5	1	2	2	2.58	1.43
		Person's focus level	5	4	1.5	1	3	2	2.75	1.41
		Eye Irritation	1	2	4	4	4	4	3.17	1.21
		Easiness to focus on stimulus center	5	3	3.5	2	3	2	3.08	1.02
		Had to blink a lot	1	3	4.5	3	4	5	3.42	1.30
Frequency	**Middle Frequency**	Comfort	2	2	3.5	5	3	2	2.92	1.10
		Person's focus level	2	2	2.5	4	4	2	2.75	0.90
		Eye Irritation	4	4	2.5	2	2	4	3.08	0.93
		Easiness to focus on stimulus center	2	2	2.5	4	4	2	2.75	0.90
		Had to blink a lot	3	3	2.5	3	3	4	3.08	0.45
Frequency	**Low Frequency**	Comfort	4	4	3	5	4	4	4.00	0.58
		Person's focus level	4	2	2.5	3	4	4	3.25	0.80
		Eye Irritation	1	2	1	2	2	2	1.67	0.47
		Easiness to focus on stimulus center	4	3	1.5	5	4	4	3.58	1.10
		Had to blink a lot	2	3	3	3	2	2	2.50	0.50
Size stimulus	**Small size**	Comfort	5	3	2	2	4	4	3.33	1.11
		Person's focus level	5	2	2	4	4	4	3.50	1.12
		Eye Irritation	1	3	2	4	2	2	2.33	0.94
		Easiness to focus on stimulus center	5	3	1	2	3	4	3.00	1.29
		Had to blink a lot	1	3	2	4	3	2	2.50	0.96
Size stimulus	**Medium size**	Comfort	4	3	3	3	4	3	3.33	0.47
		Person's focus level	4	2	3	4	3	3	3.17	0.69
		Eye Irritation	2	3	2	2	3	2	2.33	0.47
		Easiness to focus on stimulus center	4	3	3	4	4	4	3.67	0.47
		Had to blink a lot	2	3	2	3	3	3	2.67	0.47
Size stimulus	**Large Size**	Comfort	2	3	4	5	2	2	3.00	1.15
		Person's focus level	2	3	4	3	2	2	2.67	0.75
		Eye Irritation	4	3	2	2	4	4	3.17	0.90
		Easiness to focus on stimulus center	1	2	4	5	3	2	2.83	1.34
		Had to blink a lot	4	3	2	2	4	4	3.17	0.90
Stimulus mode	**Single stimulus**	Comfort	2	4	1	5	3	4	3.17	1.34
		Person's focus level	3	3	2	4	4	2	3.00	0.82
		Eye Irritation	4	2	4	2	2	2	2.67	0.94
		Easiness to focus on stimulus center	3	2	4	4	4	2	3.17	0.90
		Had to blink a lot	3	3	4	2	2	3	2.83	0.69
Stimulus mode	**Multiple stimuli**	Comfort	4	2	4	3	2	1	2.67	1.11
		Person's focus level	2	2	5	5	2	2	3.00	1.41
		Eye Irritation	2	3	3	4	4	4	3.33	0.75
		Easiness to focus on stimulus center	2	3	2	3	1	2	2.17	0.69
		Had to blink a lot	2	3	5	4	4	4	3.67	0.94

Table 4.22: Statistical workflow and results of questionnaire

category	groups	metric	p value bartlett	Variances equal?	ANOVA method	p value ANOVA method	df within groups	df between groups	Mean different?
Colors	['Black/Red' 'Black/-Green' 'Black/White']	Comfort	0.953	Yes	One-way ANOVA	0.057	2	15	No
Colors	['Black/Red' 'Black/-Green' 'Black/White']	Person's focus level	1.000	Yes	One-way ANOVA	0.063	2	15	No
Colors	['Black/Red' 'Black/-Green' 'Black/White']	Eye Irritation	0.923	Yes	One-way ANOVA	0.124	2	15	No
Colors	['Black/Red' 'Black/-Green' 'Black/White']	Focus level on stimulus center	0.718	Yes	One-way ANOVA	0.401	2	15	No
Colors	['Black/Red' 'Black/-Green' 'Black/White']	Had to blink a lot	0.662	Yes	One-way ANOVA	0.639	2	15	No
Frequency	['High frequency' 'Middle Frequency' 'Low Frequency']	Comfort	0.953	Yes	One-way ANOVA	0.057	2	15	No
Frequency	['High frequency' 'Middle Frequency' 'Low Frequency']	Person's focus level	1.000	Yes	One-way ANOVA	0.063	2	15	No
Frequency	['High frequency' 'Middle Frequency' 'Low Frequency']	Eye Irritation	0.923	Yes	One-way ANOVA	0.124	2	15	No
Frequency	['High frequency' 'Middle Frequency' 'Low Frequency']	Focus level on stimulus center	0.718	Yes	One-way ANOVA	0.401	2	15	No
Frequency	['High frequency' 'Middle Frequency' 'Low Frequency']	Had to blink a lot	0.662	Yes	One-way ANOVA	0.639	2	15	No
Shapes	['Triangle' 'Square' 'Circle']	Comfort	0.953	Yes	One-way ANOVA	0.057	2	15	No
Shapes	['Triangle' 'Square' 'Circle']	Person's focus level	1.000	Yes	One-way ANOVA	0.063	2	15	No
Shapes	['Triangle' 'Square' 'Circle']	Eye Irritation	0.923	Yes	One-way ANOVA	0.124	2	15	No
Shapes	['Triangle' 'Square' 'Circle']	Focus level on stimulus center	0.718	Yes	One-way ANOVA	0.401	2	15	No
Shapes	['Triangle' 'Square' 'Circle']	Had to blink a lot	0.662	Yes	One-way ANOVA	0.639	2	15	No
Size stimulus	['Small size' 'Medium size' 'Large Size']	Comfort	0.953	Yes	One-way ANOVA	0.057	2	15	No
Size stimulus	['Small size' 'Medium size' 'Large Size']	Person's focus level	1.000	Yes	One-way ANOVA	0.063	2	15	No
Size stimulus	['Small size' 'Medium size' 'Large Size']	Eye Irritation	0.923	Yes	One-way ANOVA	0.124	2	15	No
Size stimulus	['Small size' 'Medium size' 'Large Size']	Focus level on stimulus center	0.718	Yes	One-way ANOVA	0.401	2	15	No
Size stimulus	['Small size' 'Medium size' 'Large Size']	Had to blink a lot	0.662	Yes	One-way ANOVA	0.639	2	15	No
Stimulus mode	['Single stimulus' 'Multiple stimuli']	Comfort	0.662	Yes	One-way ANOVA	0.639	1	10	No
Stimulus mode	['Single stimulus' 'Multiple stimuli']	Person's focus level	0.662	Yes	One-way ANOVA	0.639	1	10	No
Stimulus mode	['Single stimulus' 'Multiple stimuli']	Eye Irritation	0.662	Yes	One-way ANOVA	0.639	1	10	No
Stimulus mode	['Single stimulus' 'Multiple stimuli']	Focus level on stimulus center	0.662	Yes	One-way ANOVA	0.639	1	10	No
Stimulus mode	['Single stimulus' 'Multiple stimuli']	Had to blink a lot	0.662	Yes	One-way ANOVA	0.639	1	10	No

5

Discussion, Limitations

When looking at the SNR results and their analysis, the results seem to indicate that increased pixel surface is a contributing factor to the measured SNR. Otherwise said, they have a positively correlated relationship. Figures 4.21 and 4.25 (see section 4), and figures B.2 and B.6 (see appendix B) show a positive relationship between the pixel surface and the measured SNR. This seemed to be confirmed when comparing the 10.000 with the 30.000 pixels as this seemed to be statistically significant in both experiment 1 and experiment 2. This result seems to be in line with the results reported by [9], which reported a similar relationship. An interesting result is that the participants had to blink the least amount at 10.000 pixels, and at 20.000 pixels it was the easiest center of the stimulus to focus on (see table 4.21). Thus, it can be concluded that the relationship between pixel surface and SNR can be defined as a positive relationship, meaning that an increase in pixel surface results in an increase in the SNR.

To answer the question what the relationship is between the measured SNR and shape the results showed that the triangles seem to be the least favored by the participants and resulted in the lowest amount of SNR relative to the squares (see tables 4.21 and 4.4). The difference was significant in experiment 1. Furthermore, even though it is not statistically confirmed, the results do seem to show a higher overall mean when using a square compared to a circle. This is an interesting result because Duszyk et al. [9] also reported no significant difference between the two. However, the overall results of Duszyk et al. [9] seemed to favor the circle.
The participants also experienced the circle to be the most comfortable and easiest to focus on (see table 4.21). Thus, the overall conclusion is that when using a shape a circle or square is preferred over a triangle and will likely pose better results.

To dissect the relationship between SNR and color, the SNR results might suggest that in the category color, the white color is the best suited as it reached the highest SNR mean in both experiments. However, it needs to be noted that experiment 1 did not show any significance between colors (Welch's ANOVA, p=0.097). Experiment 2, however, showed a significant difference between green and white. Additionally, both experiments showed the same sequence of colors when looking from the lowest to the highest amount of mean SNR: green, red, and white. This might carefully suggest that white shows better overall SNR across various SSVEP-based interfaces but the conclusion remains indecisive. Duszyk et al. [9] also compared the colors white and red and did also test for no significant differences between the two colors. Additionally, Duart et al. [8] suggests that the optimal color depends on the frequency and can show mixed results. Albawardi et al. [1] indicated that green would be a very comfortable color. However, even though not confirmed if there are any significant differences (One-way ANOVA, p=0.057), the questionnaire showed the lowest eye irritation and the highest comfortability score for the color red (see table 4.21).

Duart et al. [8] stated that different frequencies can result in different SNR values. That varying interface characteristics influence the SNR seems to be suggested when looking at the multi-dimensional SNR plots (see appendix A, and figures 4.3 until 4.18). It does seem to suggest that varying a combination of parameters such as pixel surface, color, shape, and frequency, might result in different SNR

values.
This is interesting as it could help explain the question surrounding the relationship between color, frequency, and SNR, as posed by Regan [35]. Moreover, this could also be participant dependent, and more characteristics could be an influential factor. Looking at these results also makes it hard to compare to other research. Additionally, other research often used different frequencies in the neighborhood of the frequencies in this research, but not exactly the same [9][8][8][6]. An interesting result is that both experiment 1 and experiment 2 showed between the same 3 pairs of frequencies a significant difference in SNR means. This was the case between 8Hz and 25HZ, 19Hz and 25Hz, and between 13Hz and 19Hz. Both experiments showed also the same sequence of frequencies when looking from the least mean SNR to the highest mean SNR: 13Hz, 8Hz, 19Hz, and 25Hz. This seems to suggest that 25Hz is favored and would result in the overall highest amount of SNR. However, as this is hard to compare with other research it is hard to confirm this using other research. Pastor et al. [33] tested a wide variety of stimulus frequencies and did not seem to favor 25Hz, but 15Hz. However, 25Hz is the only actual overlapping frequency in this research. The participants in this research experienced the low frequencies to be the most comfortable, the lowest eye irritation, and the least amount of blinks during trials. This is opposed to the 25Hz, which showest the overall highest amount of mean SNR.

Lastly, when placing a single stimulus in an SSVEP-based interface showing 4 stimuli simultaneously, it would make sense that the SNR lowers slightly as other stimuli do induce more visual noise. This should assumably transfer to the SSVEP response. Even though the mean SNR slightly diminishes in all characteristics except 1. When comparing experiment 1 to experiment 2, it does not seem to have a significant influence on the measured SNR (see table 4.19). The results would suggest that this is not something to worry about, answering the question about the influence of surrounding stimuli on the SNR of the target stimulus.

Important to recognize is that this work applies only to the use of LCD screens, as different hardware can pose different results [39][41].

One of the limitations of this work is the small number of participants to draw strong conclusions. Furthermore, blinks are not detected and filtered within the EEG recordings. This means that the segments used in the analysis for the SSVEP might contain a lot of noise. Another discussion point is that the amplitude of SSVEP response grows over time, which might show clearer differences. This can be explained by lengthier recordings having more signal while the noise is (ideally) random and not fixed to the stimulation frequency [15]. Another advantage is that this will also increase the resolution between frequency bins. However, in real-time applications, a small time window will probably be needed, which would mean the approach in this research is more realistic for such use cases.

Furthermore, using non-linear interpolation in a non-linear environment might not be the best approach to measure the maximal SNR. However, if the resolution between frequency bins is sufficiently high the actual error should be minimized. Due to the power-spectral density spectrum having a frequency resolution of 0.2857Hz the maximal interpolation distance with respect to a measured frequency bin is 0.15Hz, which mitigates this problem quite a bit. Still, non-linear interpolation would be a better approach in future research. Examples are parabolic and Gaussian interpolation [12].

Another limitation is that no phase shift is incorporated in this research. This means that these results might not transfer to an SSVEP-based interface where this is applied, such as Jingnan, He, and Gao [19].

Concerning the dataset, the lack of successful measurements with eye-tracking limits the useability and the supportive role that eye-tracking can play. The exact cause of the failure of eye-tracking in these cases is still unknown and yet to be determined.

6

Conclusion

This multivariate problem is hard to grasp entirely as a combination of many factors influences the results. However, the results do help answering the main research question about the relationship between stimuli characteristics and the measured SNR. Overall, it seems that an increased pixel surface does positively influence the SNR. Important to note is that stimuli of 20.000 pixels seem to be favored by participants (see table 4.21). Furthermore, red might be a more comfortable color. However, the results show no significant difference from other colors with respect to SNR.
An interesting finding is that there seemed not to be a significant difference between looking at a single stimulus and a single stimulus with surrounding stimuli shown at different frequencies. The biggest difference was 0.55dB, which indicates the difference is most likely negligible in most cases.

Another interesting result seems to be that when comparing triangles with squares or circles the triangles are not preferred by participants. Triangles also resulted in significantly lower SNR values compared to squares. It is recommended to use either circles or squares. However, there is probably not a lot of difference between these two as our results favored the squares but other research showed slightly higher SNR values for circles [9].

To better untangle this multivariate problem future research is necessary. Furthermore, by doing more extensive research across more people stronger conclusions can be drawn. Moreover, it should be investigated what the significance of blinks is in SSVEP measurements to determine if the trials that contain them have to be redone or filtered out. Additionally, it is still strongly suggested to attempt to incorporate eye-tracking into the SSVEP-based dataset as this gives EEG measurements more context, even though it proved to be difficult. Despite in this research, its supportive roll was relatively low in half of the participants, as in most trials the eye-tracking was not recorded. Lastly, a future recommendation is to use non-linear interpolation methods and not linear interpolation methods to interpolate the SNR spectrum.

7

Epilogue

Writing this thesis has broadened my horizon and has pushed my professional ability in writing reports to a higher level. I am glad to look back at this past period of my life. It was not the smoothest period and sometimes induced a lot of stress.

As I never had any experience with working with raw eye-tracking and EEG data before, this was a new challenge for me. Due to the unknown nature of this research, I sometimes had to rely on my supervisor. However, sometimes I was left alone for a few weeks which helped me with my confidence in navigating unknown territory.

Furthermore, I also had never worked with EEG data before. Fortunately, my supervisor had already a setup ready to use in the basement of the Amsterdam UMC. Concerning the calibration, EEG recording, recording eye-tracking data, and data logging itself, a lot was already done in a previous project executed under the supervision of my supervisor. A lot of parts could be copied, or they were built-in within Experimental Builder which allowed for creating the experiments without any coding. It took a frustrating amount of time to get the application to run without dropping too many frames.

I also learned to better summarize research reports and relate them to each other to draw conclusions. Furthermore, I have learned how brain-machine interfaces and EEG work, and how statistical analyses of variances can be applied to compare these results. This helped me in going outside of my comfort zone and let me learn something new.

Fortunately, I already got a lot of experience programming in python which helped me in generating the dataset of videos using the library OpenCV in a short period of time. However, using the python library MNE (EEG library) was something new for me, and I had to learn how to analyze EEG data and extract the SNR. Fortunately, it is very well documented.

I wrote my own analysis scripts for the EEG and partially for the eye-tracking. It is partially because my supervisor had already written a script that could be used to filter out the fixations within the eye-tracking data. However, I already had written visualizations and analyzed the data using metrics such as radius and sample points on stimulus.

Additionally, during the experiments, I learned to rely on others as a second person was needed during testing to attach the headset to my head. This needed a lot of communication and planning to allow continuous progress within the thesis.

I am happy to be able to say that I can close this chapter in my life and hope to find new challenges in the future.

S.T. van Vliet

Delft, January 2023

References

[1] Hessa Albawardi et al. “Design of Low-Cost Steady State Visually Evoked Potential-Based Brain Computer Interface Using OpenBCI and Neuromore”. In: Dec. 2021, pp. 1–4. DOI: 10.1109/BioSMART54244.2021.9677782.

[2] Dr. Hossein Arsham and Miodrag Lovric. “Bartlett’s Test”. In: *International Encyclopedia of Statistical Science* 2 (Mar. 2011), pp. 20–23. DOI: 10.1007/978-3-642-04898-2_132.

[3] Andreas Bulling. *Towards high-frequency SSVEP-based target discrimination with an extended alphanumeric keyboard: Perceptual user interfaces*. 2019. URL: https://www.perceptualui.org/publications/abdelnabi19_smc/.

[4] Teng Cao et al. “Flashing color on the performance of SSVEP-based brain-computer interfaces”. In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*. 2012, pp. 1819–1822. ISBN: 9781424441198. DOI: 10.1109/EMBC.2012.6346304.

[5] Xiaogang Chen et al. “Combination of Augmented Reality Based Brain- Computer Interface and Computer Vision for High-Level Control of a Robotic Arm”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 28.12 (Dec. 2020), pp. 3140–3147. ISSN: 15580210. DOI: 10.1109/TNSRE.2020.3038209.

[6] Anna Cysewska-Sobusiak and Marcin Jukiewicz. “Stimuli design for SSVEP-based brain computer-interface”. In: *International Journal of Electronics and Telecommunications* vol. 62.No 2 (2016). DOI: 10.1515/eletel-2016-0014. URL: http://journals.pan.pl/Content/88003/PDF/14.pdf.

[7] Stavros I. Dimitriadis and Avraam D. Marimpis. “Enhancing performance and bit rates in a brain–computer interface system with phase-to-amplitude cross-frequency coupling: Evidences from traditional c-VEP, fast c-VEP, and SSVEP designs”. In: *Frontiers in Neuroinformatics* 12 (May 2018). ISSN: 16625196. DOI: 10.3389/fninf.2018.00019.

[8] Xavier Duart et al. “Evaluating the effect of stimuli color and frequency on SSVEP”. In: *Sensors (Switzerland)* 21.1 (Jan. 2021), pp. 1–19. ISSN: 14248220. DOI: 10.3390/s21010117.

[9] Anna Duszyk et al. *Towards an optimization of stimulus parameters for brain-computer interfaces based on steady state visual evoked potentials*. 2014. URL: https://journals.plos.org/plosone/article?id=10.1371%5C%2Fjournal.pone.0112099.

[10] *EEG CAP: Standard 64ch-ACTICAP-slim with built-in electrodes*. URL: https://www.brainlatam.com/manufacturers/easycap/eeg-cap-standard-64ch-acticap-slim-with-built-in-electrodes-205.

[11] Jim Frost. *Benefits of welch’s ANOVA compared to the classic one-way ANOVA*. Sept. 2022. URL: https://statisticsbyjim.com/anova/welchs-anova-compared-to-classic-one-way-anova/.

[12] M Gasior and JL Gonzalez. *EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH CERN AB DEPARTMENT Improving FFT Frequency Measurement Resolution by Parabolic and Gaussian Spectrum Interpolation Improving FFT Frequency Measurement Resolution by Parabolic and Gaussian Spectrum Interpolation*. 2004.

[13] Felix Gembler, Piotr Stawicki, and Ivan Volosyak. “Suitable Number of Visual Stimuli for SSVEP-Based BCI Spelling Applications”. In: May 2017, pp. 441–452. ISBN: 978-3-319-59146-9. DOI: 10.1007/978-3-319-59147-6_38.

[14] GISGeography. *Inverse distance weighting (IDW) interpolation*. Nov. 2022. URL: https://gisgeography.com/inverse-distance-weighting-idw-interpolation/.

[15] Alexandre Gramfort et al. “MEG and EEG Data Analysis with MNE-Python”. In: *Frontiers in Neuroscience* 7.267 (2013), pp. 1–13. DOI: 10.3389/fnins.2013.00267.

[16] Osman Berke Guney, Muhtasham Oblokulov, and Huseyin Ozkan. “A Deep Neural Network for SSVEP-Based Brain-Computer Interfaces”. In: *IEEE Transactions on Biomedical Engineering* 69.2 (Feb. 2022), pp. 932–944. DOI: 10.1109/tbme.2021.3110440. URL: https://doi.org/10.1109%2Ftbme.2021.3110440.

[17] C S Herrmann. “Human EEG responses to 1-100 Hz flicker: resonance phenomena in visual cortex and their potential correlation to cognitive phenomena”. en. In: *Exp Brain Res* 137.3-4 (Apr. 2001), pp. 346–353.

[18] *Home en*. URL: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft.

[19] Sun Jingnan, Jing He, and Xiaorong Gao. *Neurofeedback training of control network improves SSVEP-based BCI performance in children*. Feb. 2020. DOI: 10.21203/rs.2.24387/v1.

[20] Suguru Kanoga et al. “Robustness analysis of decoding SSVEPs in humans with head movements using a moving visual flicker”. In: *Journal of Neural Engineering*. Vol. 17. 1. Institute of Physics Publishing, 2020. DOI: 10.1088/1741-2552/ab5760.

[21] Ibrahim Kaya, Jorge Bohórquez, and Özcan Özdamar. “A BCI gaze sensing method using low jitter code modulated VEP”. In: *Sensors (Switzerland)* 19 (17 Sept. 2019). ISSN: 14248220. DOI: 10.3390/s19173797.

[22] Ahmadreza Keihani et al. “Use of Sine Shaped High-Frequency Rhythmic Visual Stimuli Patterns for SSVEP Response Analysis and Fatigue Rate Evaluation in Normal Subjects”. In: *Frontiers in Human Neuroscience* 12 (May 2018). ISSN: 16625161. DOI: 10.3389/fnhum.2018.00201.

[23] Karl Kruusamäe and Ilya Kuzovkin. *Visuaalselt esilekutsutud potentsiaalidel põhinev aju-arvuti liides robootika rakendustes*. Tech. rep. 2018.

[24] Minglun Li et al. “Brain-Computer Interface Speller Based on Steady-State Visual Evoked Potential: A Review Focusing on the Stimulus Paradigm and Performance”. en. In: *Brain Sci* 11.4 (Apr. 2021).

[25] Bingchuan Liu et al. “BETA: A Large Benchmark Database Toward SSVEP-BCI Application”. In: *Frontiers in Neuroscience* 14 (June 2020). ISSN: 1662453X. DOI: 10.3389/fnins.2020.00627.

[26] Bingchuan Liu et al. “eldBETA: A Large Eldercare-oriented Benchmark Database of SSVEP-BCI for the Aging Population”. In: *Scientific Data* 9 (May 2022), p. 252. DOI: 10.1038/s41597-022-01372-9.

[27] Pengxiao Liu et al. “An SSVEP-BCI in Augmented Reality”. In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 2019, pp. 5548–5551. DOI: 10.1109/EMBC.2019.8857859.

[28] Thomas Meigen and Michael Bach. “On the statistical significance of electrophysiological steady-state responses”. In: *Documenta Ophthalmologica* 98.3 (July 1999), pp. 207–232.

[29] Masaki Nakanishi et al. “Enhancing detection of SSVEPs for a high-speed brain speller using task-related component analysis”. In: *IEEE Transactions on Biomedical Engineering* 65 (1 Jan. 2018), pp. 104–112. ISSN: 15582531. DOI: 10.1109/TBME.2017.2694818.

[30] Viswam Nathan et al. “A 16-channel bluetooth enabled wearable EEG platform with dry-contact electrodes for brain computer interface”. In: Nov. 2013. DOI: 10.1145/2534088.2534098.

[31] *One-way ANOVA (cont...)* URL: https://statistics.laerd.com/statistical-guides/one-way-anova-statistical-guide-4.php.

[32] Zeki Oralhan and Mahmut Tokmakçi. “The Effect of Duty Cycle and Brightness Variation of Visual Stimuli on SSVEP in Brain Computer Interface Systems”. In: *IETE Journal of Research* 62.6 (2016), pp. 795–803. DOI: 10.1080/03772063.2016.1176543. eprint: https://doi.org/10.1080/03772063.2016.1176543. URL: https://doi.org/10.1080/03772063.2016.1176543.

[33] Maria A. Pastor et al. “Human Cerebral Activation during Steady-State Visual-Evoked Responses”. In: *Journal of Neuroscience* 23.37 (2003), pp. 11621–11627. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.23-37-11621.2003. eprint: https://www.jneurosci.org/content/23/37/11621.full.pdf. URL: https://www.jneurosci.org/content/23/37/11621.

[34] Rabie Ramadan and Athanasios Vasilakos. “Brain Computer Interface: Control Signals Review”. In: *Neurocomputing* 223 (Oct. 2016). DOI: 10.1016/j.neucom.2016.10.024.

[35] D Regan. “An Effect of Stimulus Colour on Average Steady-state Potentials evoked in Man”. In: *Nature* 210.5040 (June 1966), pp. 1056–1057.

[36] Robert J Summers and Tim S Meese. “The influence of fixation points on contrast detection and discrimination of patches of grating: masking and facilitation”. en. In: *Vision Res* 49.14 (May 2009), pp. 1894–1900.

[37] G. Bhat Swati et al. “Steady State Visually Evoked Potential Based Brain Computer Interface for Game Control”. In: *2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. 2018, pp. 422–426. DOI: 10.1109/ICEECCOT43722.2018.9001590.

[38] Tsvetomira Tsoneva, Gary Garcia-Molina, and Peter Desain. “SSVEP phase synchronies and propagation during repetitive visual stimulation at high frequencies”. In: *Scientific Reports* 11.1 (Mar. 2021), p. 4975. ISSN: 2045-2322. DOI: 10.1038/s41598-021-83795-9. URL: https://doi.org/10.1038/s41598-021-83795-9.

[39] Ivan Volosyak, Hubert Cecotti, and Axel Graeser. “Impact of Frequency Selection on LCD Screens for SSVEP Based Brain-Computer Interfaces”. In: vol. 5517. June 2009, pp. 706–713. ISBN: 978-3-642-02477-1. DOI: 10.1007/978-3-642-02478-8_88.

[40] Ivan Volosyak, Hubert Cecotti, and Axel Graeser. “Impact of Frequency Selection on LCD Screens for SSVEP Based Brain-Computer Interfaces”. In: vol. 5517. June 2009, pp. 706–713. ISBN: 978-3-642-02477-1. DOI: 10.1007/978-3-642-02478-8_88.

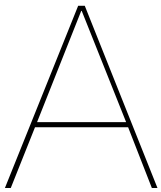
[41] Rui Zhang et al. “The effect of stimulus number on the recognition accuracy and information transfer rate of SSVEP-BCI in augmented reality”. In: *Journal of Neural Engineering* 19 (3 2022). ISSN: 17412552. DOI: 10.1088/1741-2552/ac6ae5.

[42] Shangen Zhang and Xiaogang Chen. “Effect of background luminance of visual stimulus on elicited steady-state visual evoked potentials”. In: *Brain Science Advances* 8.1 (2022), pp. 50–56. DOI: 10.26599/BSA.2022.9050006. eprint: https://doi.org/10.26599/BSA.2022.9050006. URL: https://doi.org/10.26599/BSA.2022.9050006.

[43] Shangen Zhang, Xiaorong Gao, and Xiaogang Chen. “Humanoid Robot Walking in Maze Controlled by SSVEP-BCI Based on Augmented Reality Stimulus”. In: *Frontiers in Human Neuroscience* 16 (July 2022), p. 908050. DOI: 10.3389/fnhum.2022.908050.

[44] Wenjie Zheng et al. *Effect of stimulus size and shape on steady-state ... - ESPCI Paris*. URL: https://www.neurones.espci.fr/Articles_PS/IJCCI%5C%202013.pdf.

[45] Danhua Zhu et al. “A Survey of Stimulation Methods Used in SSVEP-Based BCIs”. In: *Computational Intelligence and Neuroscience* 2010 (Mar. 2010), p. 702357. ISSN: 1687-5265. DOI: 10.1155/2010/702357. URL: https://doi.org/10.1155/2010/702357.



Multi-Dimensional SNR plots

A.1. 4D plots

In 3D, the relationship is visualized between 3 independent variables on one metric, which means the plots could be called 4-dimensional.

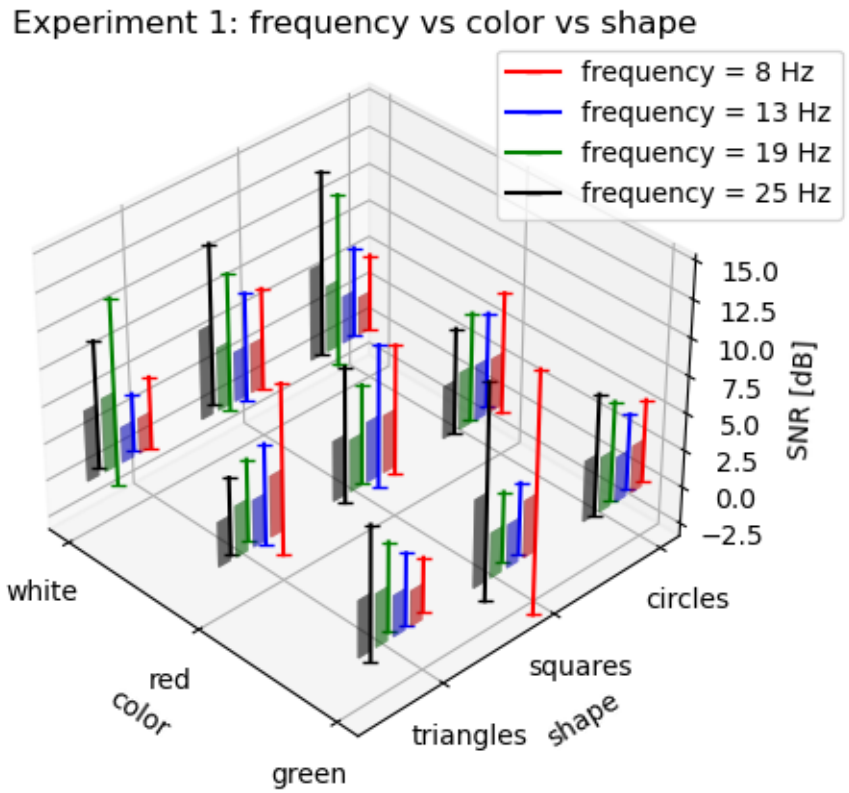


Figure A.1: Experiment 1 results of SNR between frequency, color, and shape.

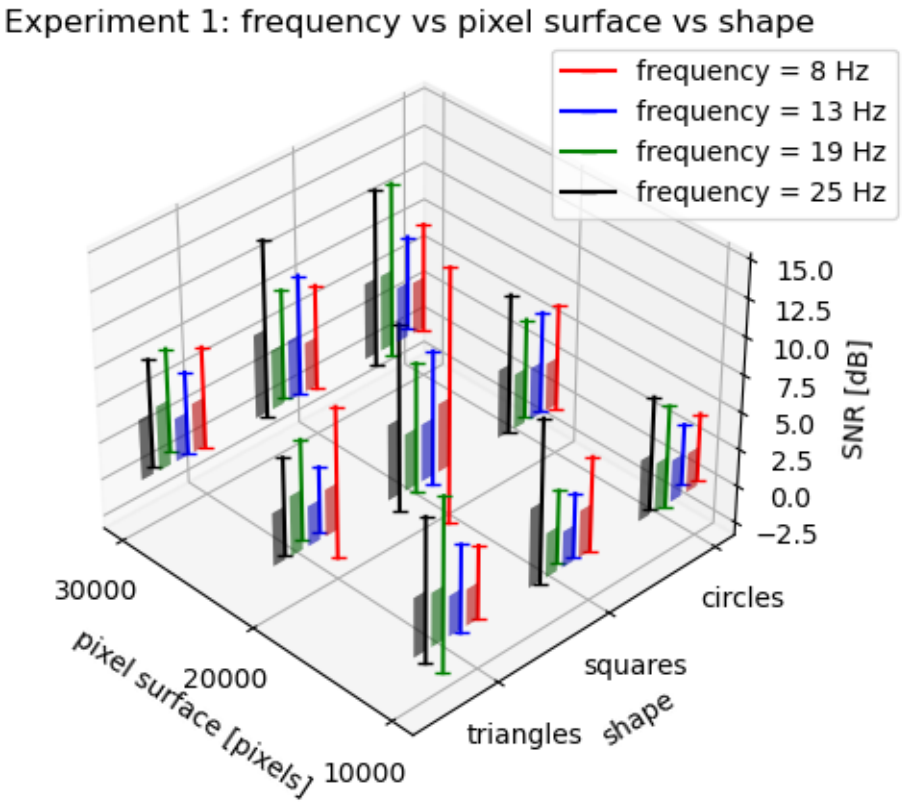


Figure A.2: Experiment 1 results of SNR between frequency, pixel surface, and shape.

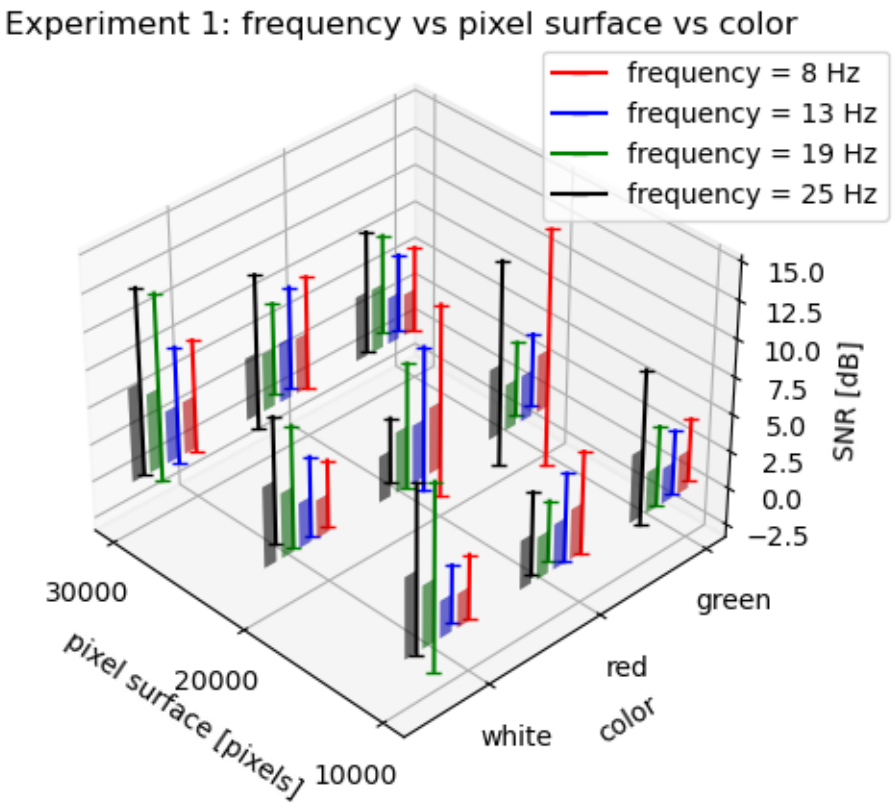


Figure A.3: Experiment 1 results of SNR between frequency, pixel surface, and color.

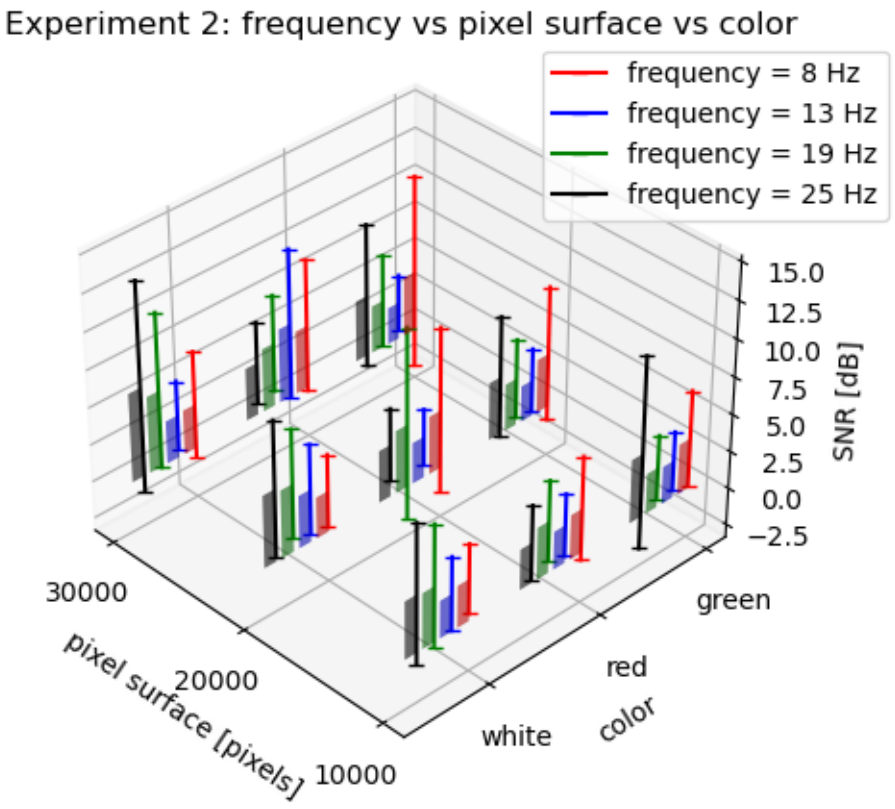


Figure A.4: Experiment 2 results of SNR between frequency, pixel surface, and color.

A.2. 3D plots

Here the 4-dimensional data is shown at only one frequency per plot. This reduces the 4-dimensional data to more 3-dimensional data. The plots here show the SNR mean, and standard deviation with respect to 2 other categories of interface characteristics at that frequency. The value distribution of the legends is uniformly fixed across the plots as this makes visual comparisons between plots easier.

A.2.1. 3D plots experiment 1

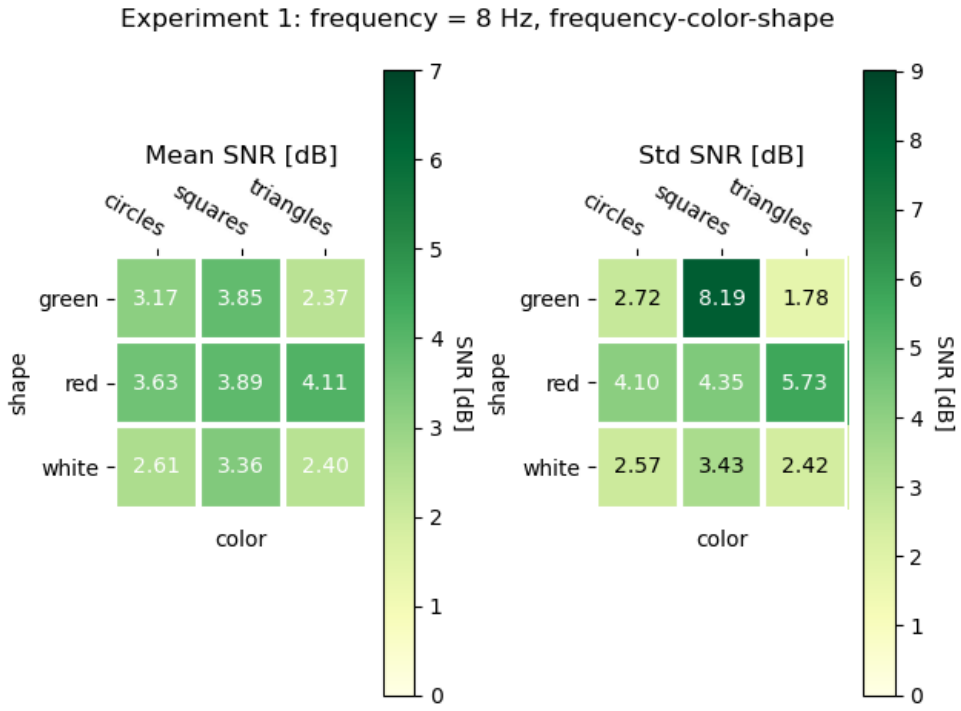


Figure A.5: SNR relationship between color and shape at 8Hz of experiment 1

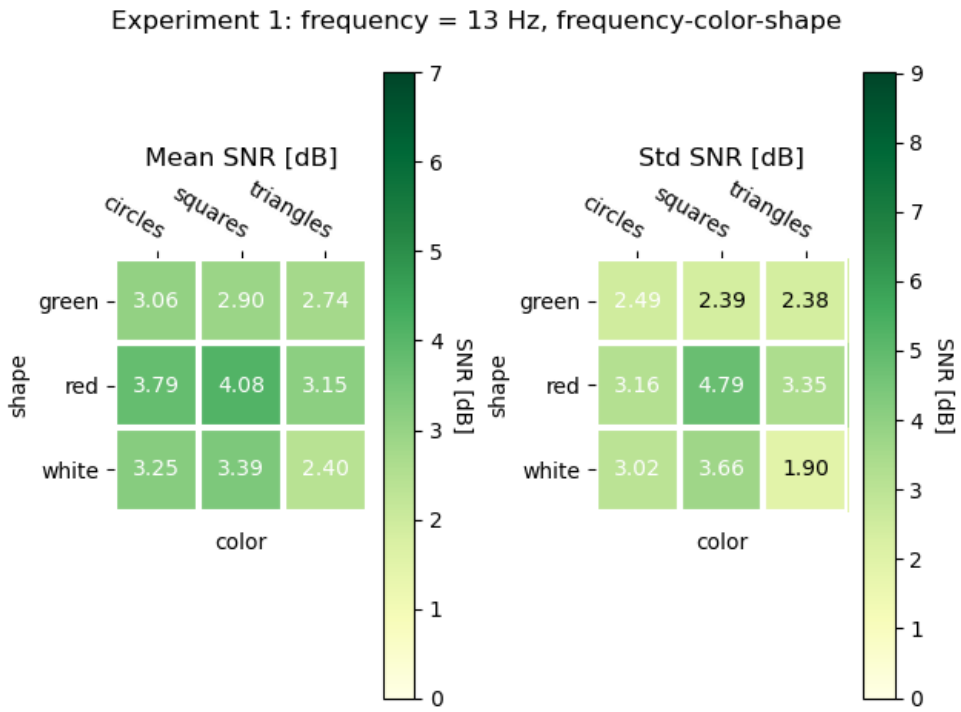


Figure A.6: SNR relationship between color and shape at 13Hz of experiment 1

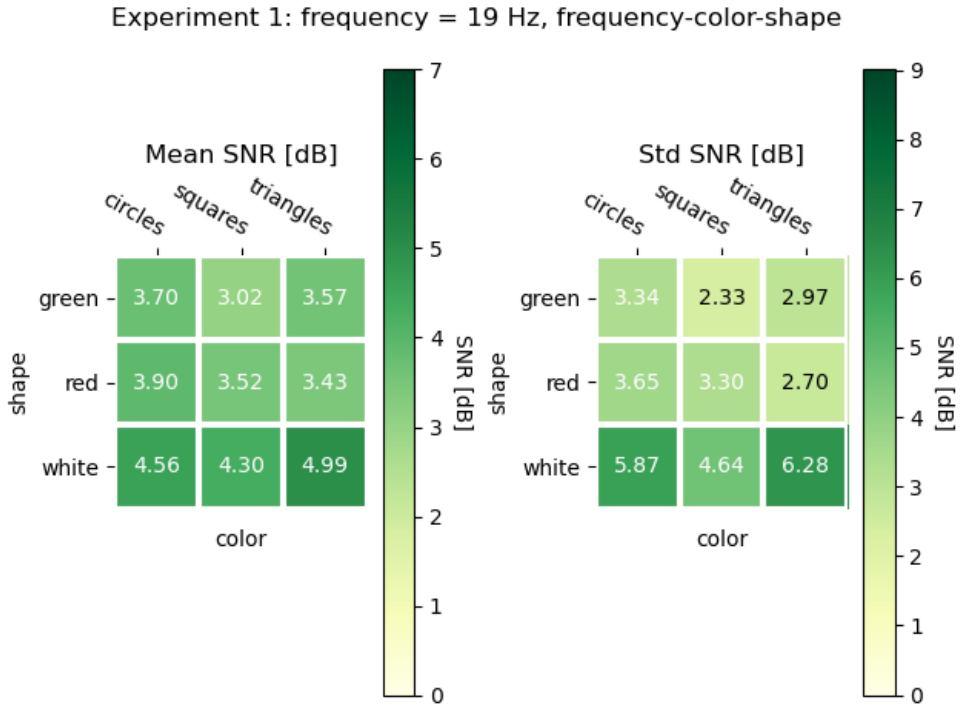


Figure A.7: SNR relationship between color and shape at 19Hz of experiment 1

A.2.2. Experiment 2

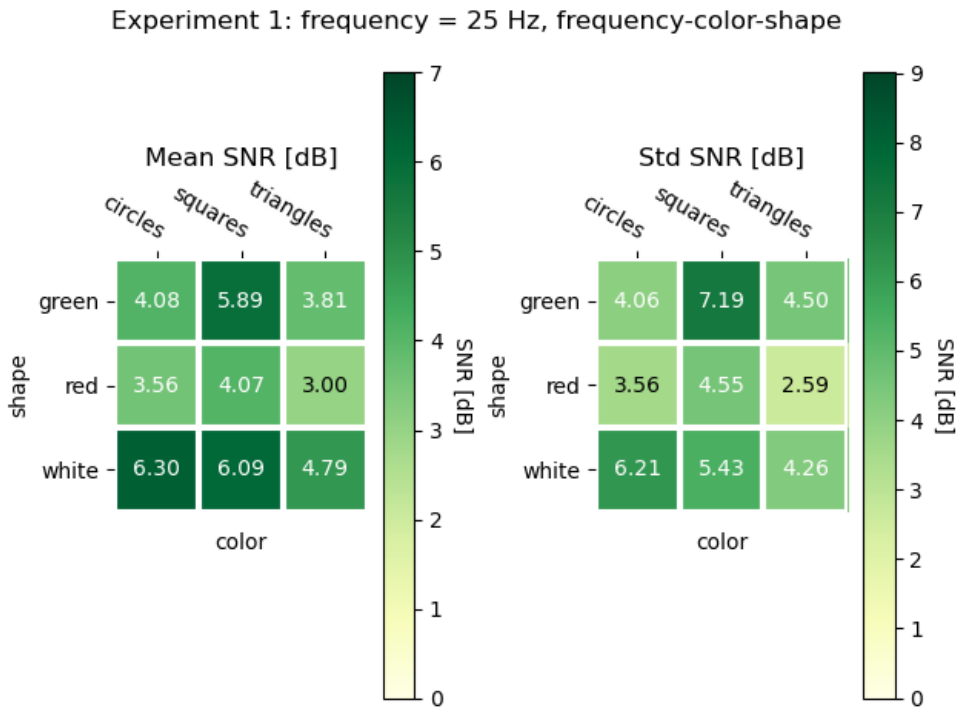


Figure A.8: SNR relationship between color and shape at 25Hz of experiment 1

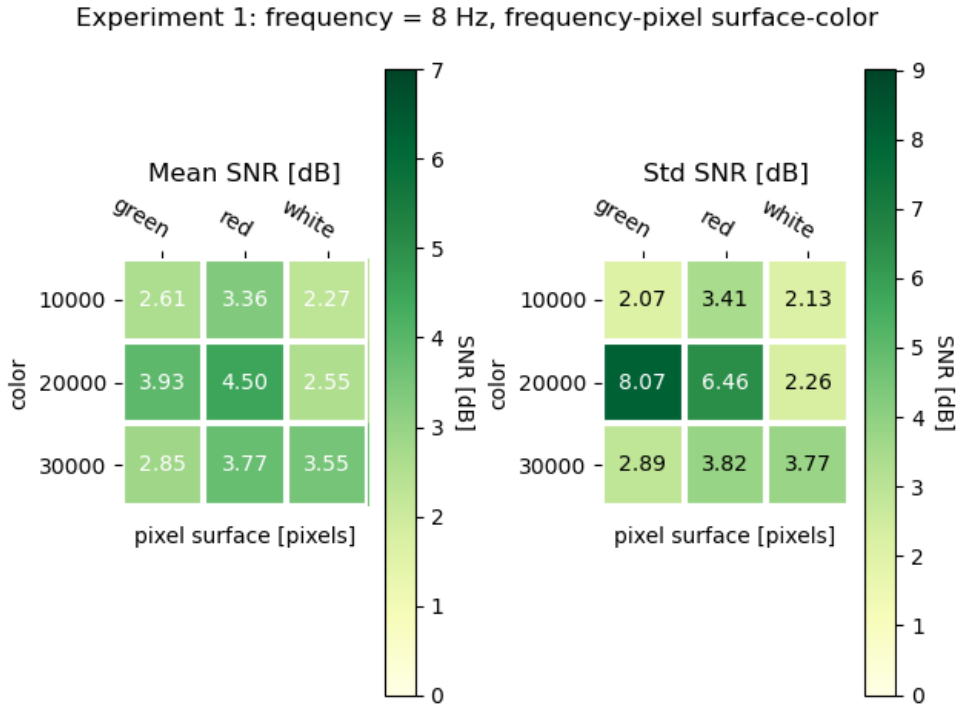


Figure A.9: SNR relationship between pixel surface and color at 8Hz of experiment 1

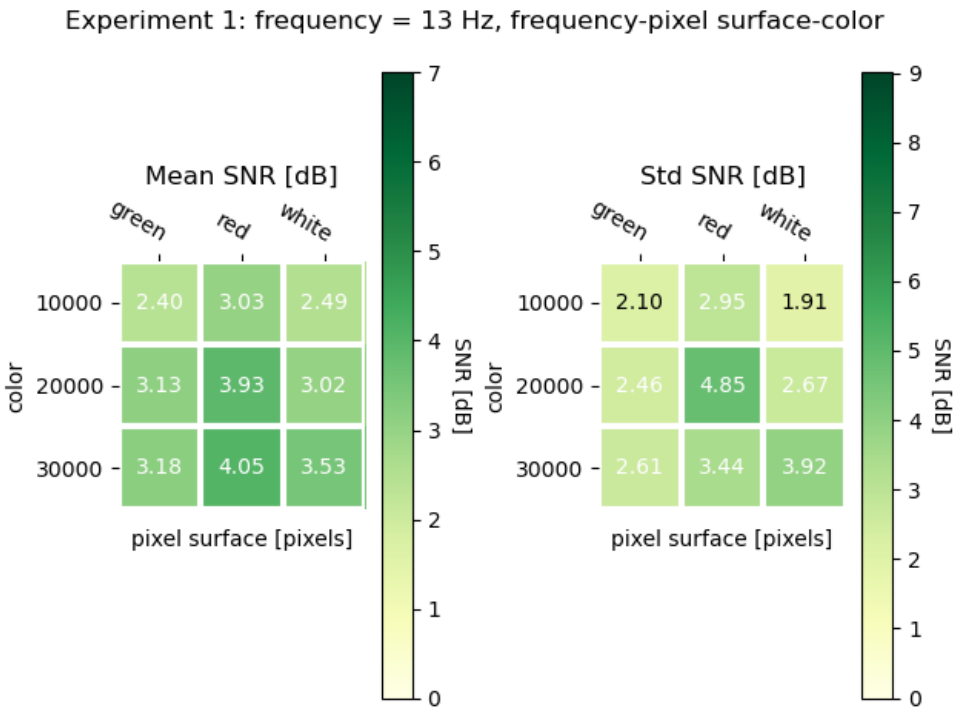


Figure A.10: SNR relationship between pixel surface and color at 13Hz of experiment 1

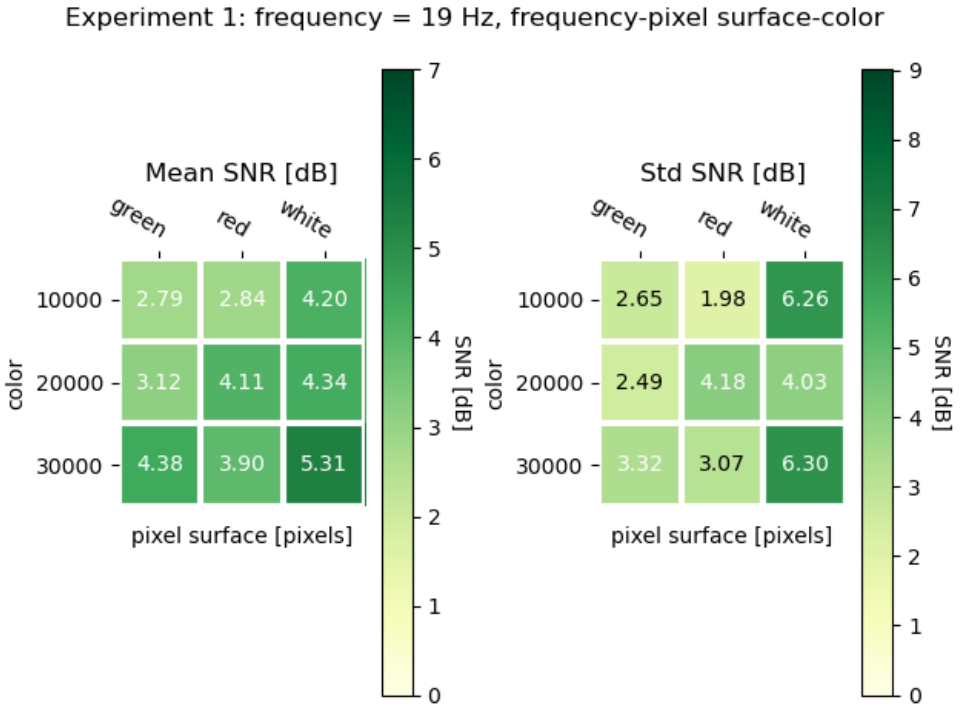


Figure A.11: SNR relationship between pixel surface and color at 19Hz of experiment 1

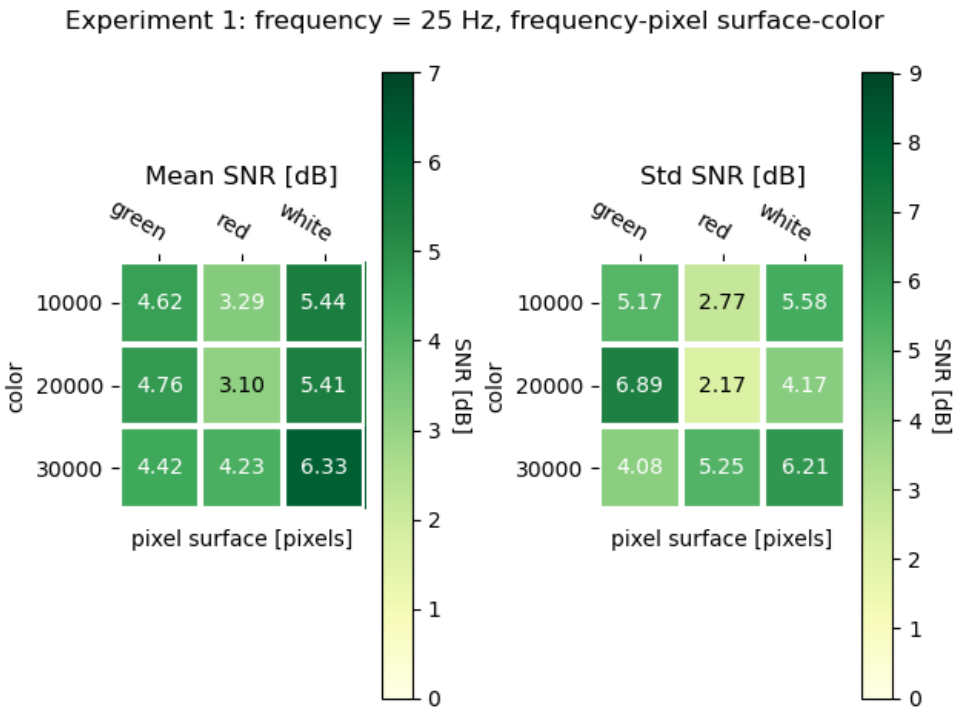


Figure A.12: SNR relationship between pixel surface and color at 25Hz of experiment 1

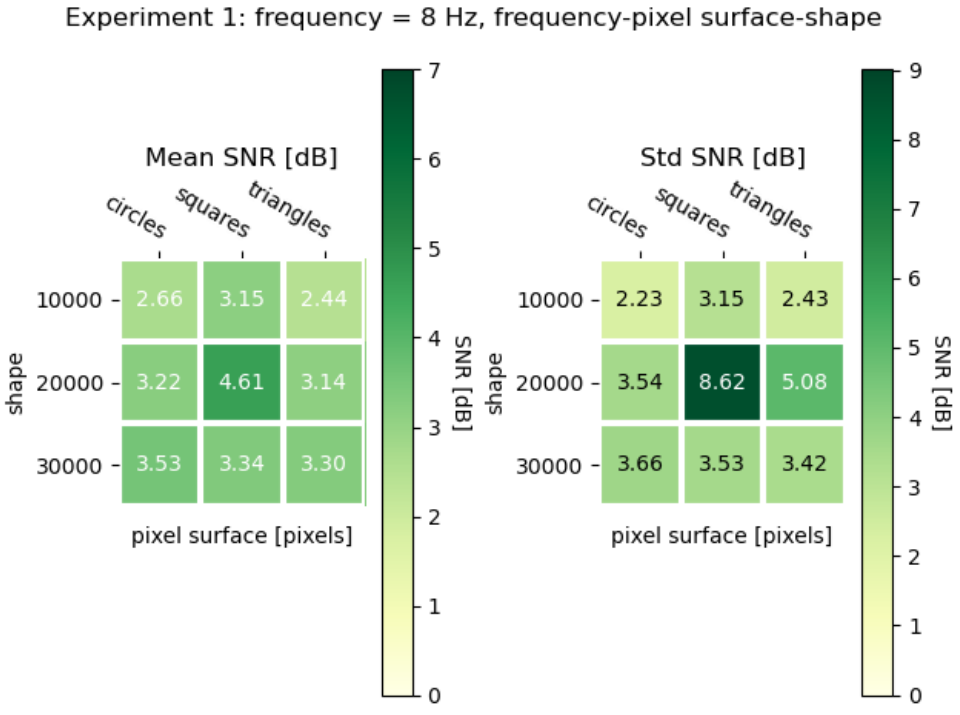


Figure A.13: SNR relationship between pixel surface and shape at 8Hz of experiment 1

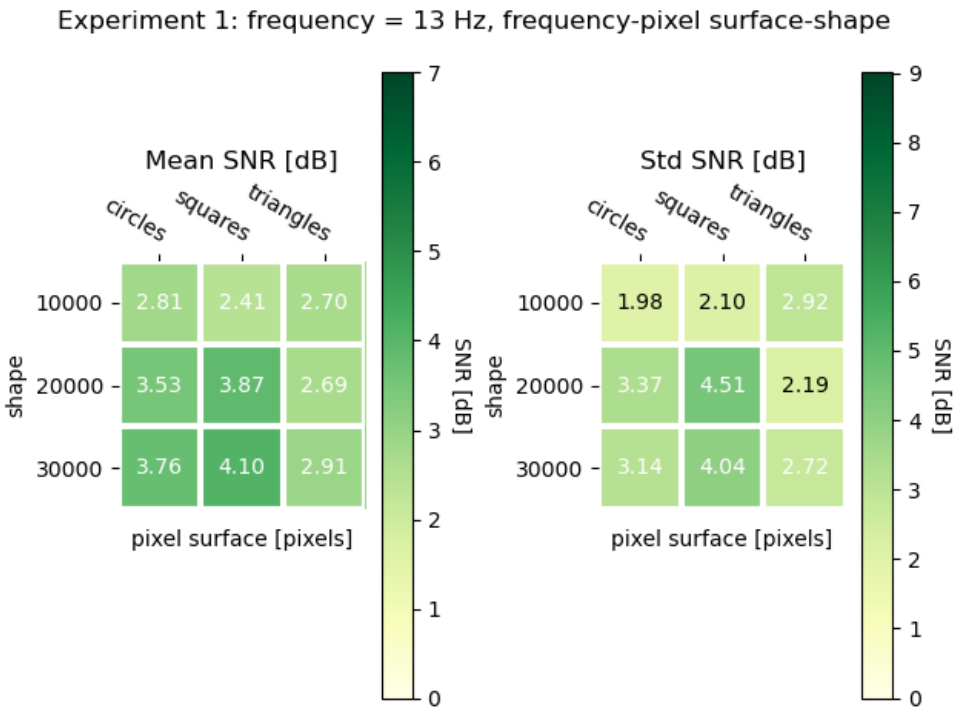


Figure A.14: SNR relationship between pixel surface and shape at 13Hz of experiment 1

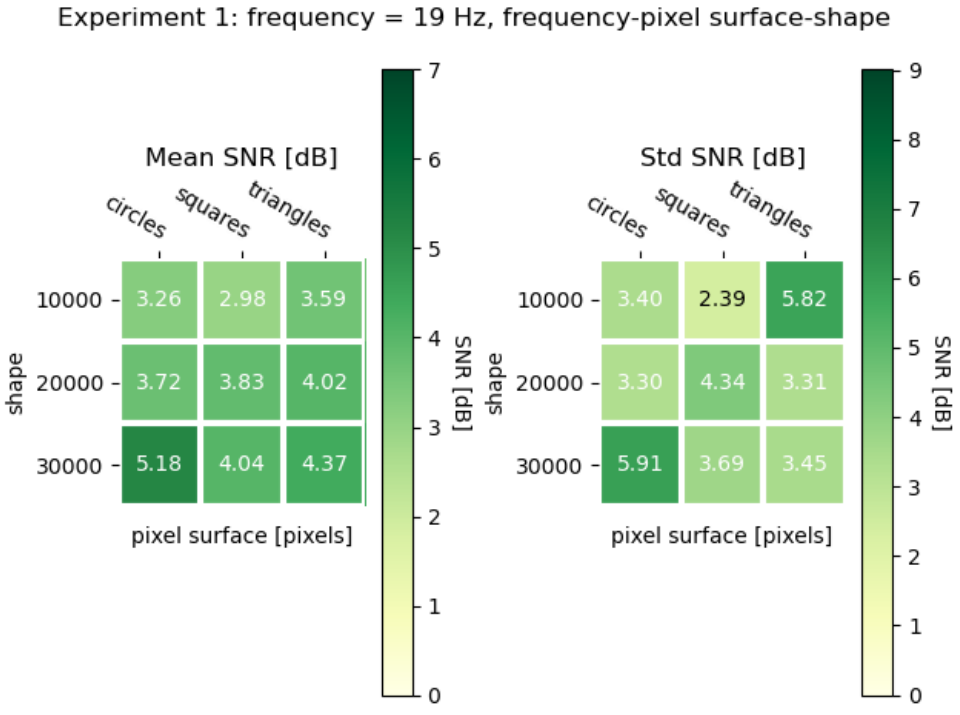


Figure A.15: SNR relationship between pixel surface and shape at 19Hz of experiment 1

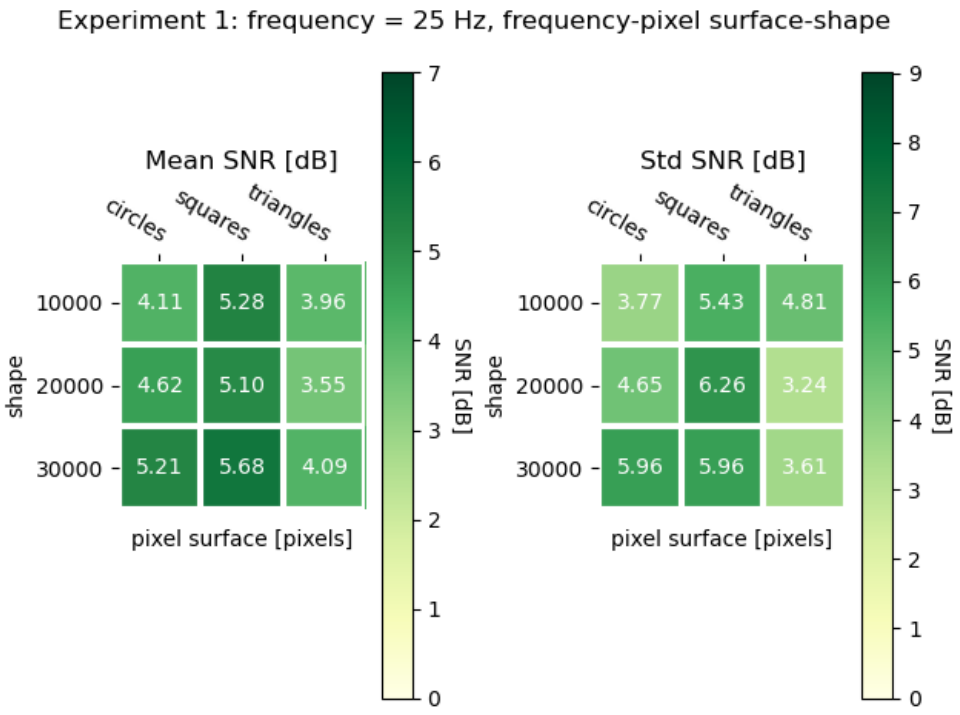


Figure A.16: SNR relationship between pixel surface and shape at 25Hz of experiment 1

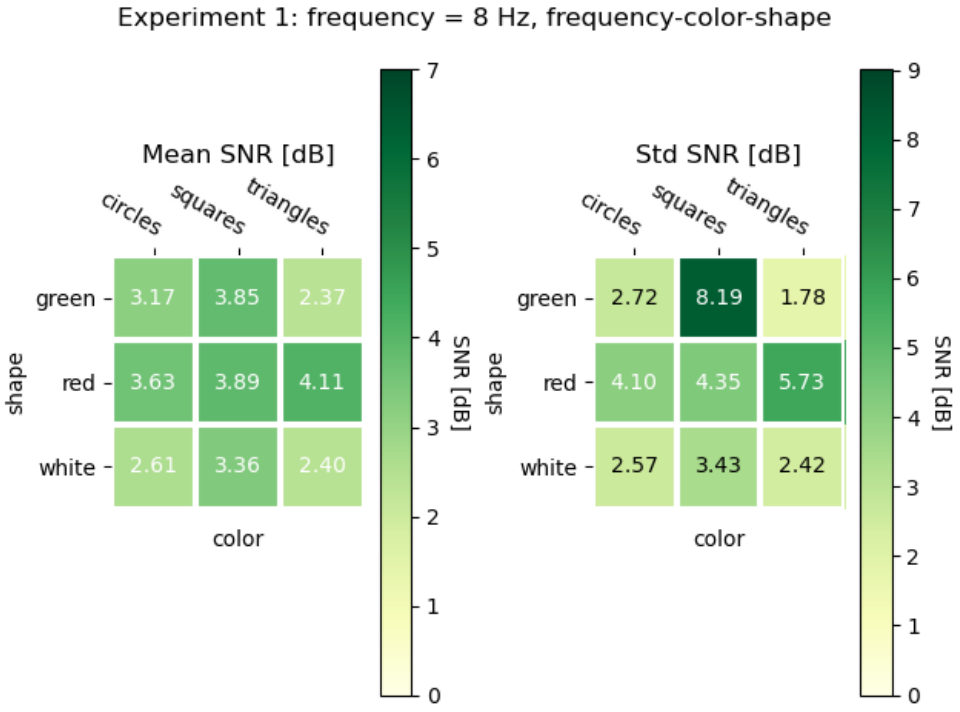


Figure A.17: SNR relationship between color and shape at 8Hz of experiment 2

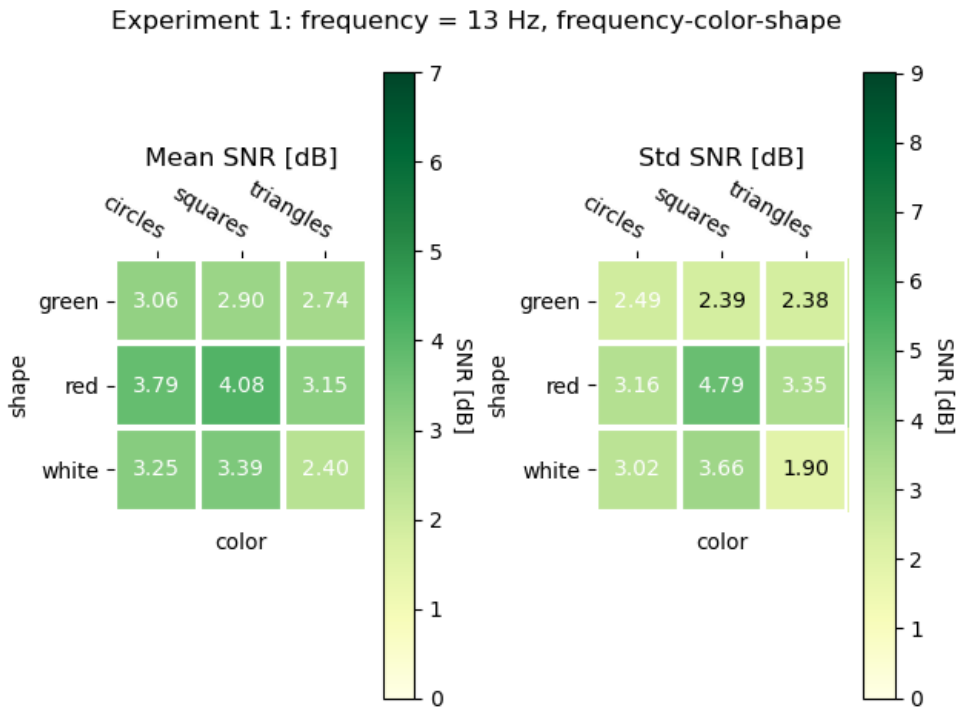


Figure A.18: SNR relationship between color and shape at 13Hz of experiment 2

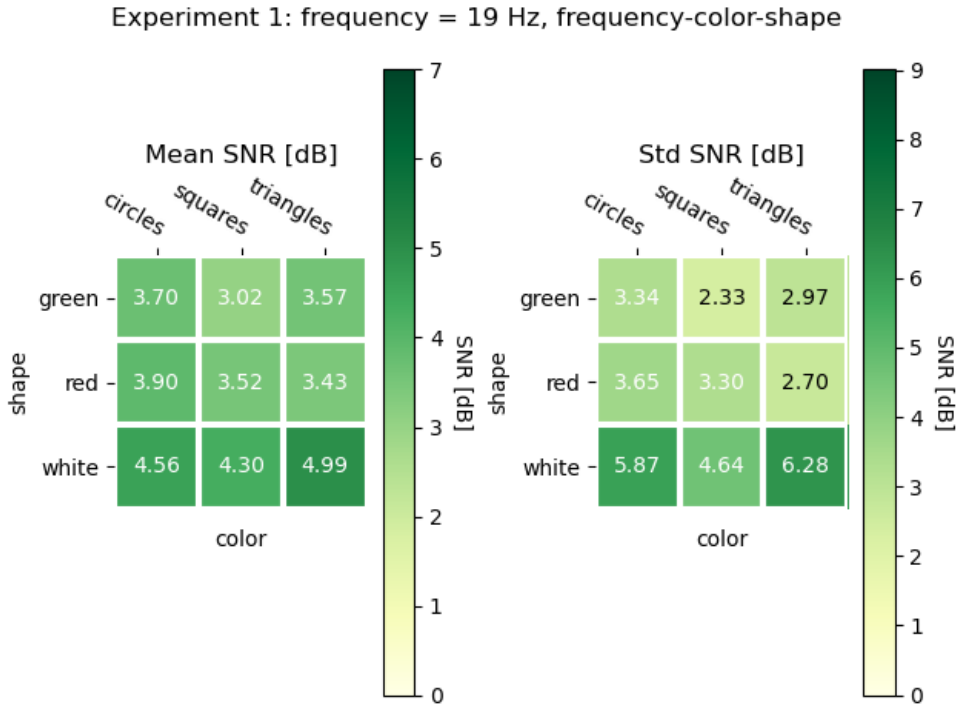


Figure A.19: SNR relationship between color and shape at 19Hz of experiment 2

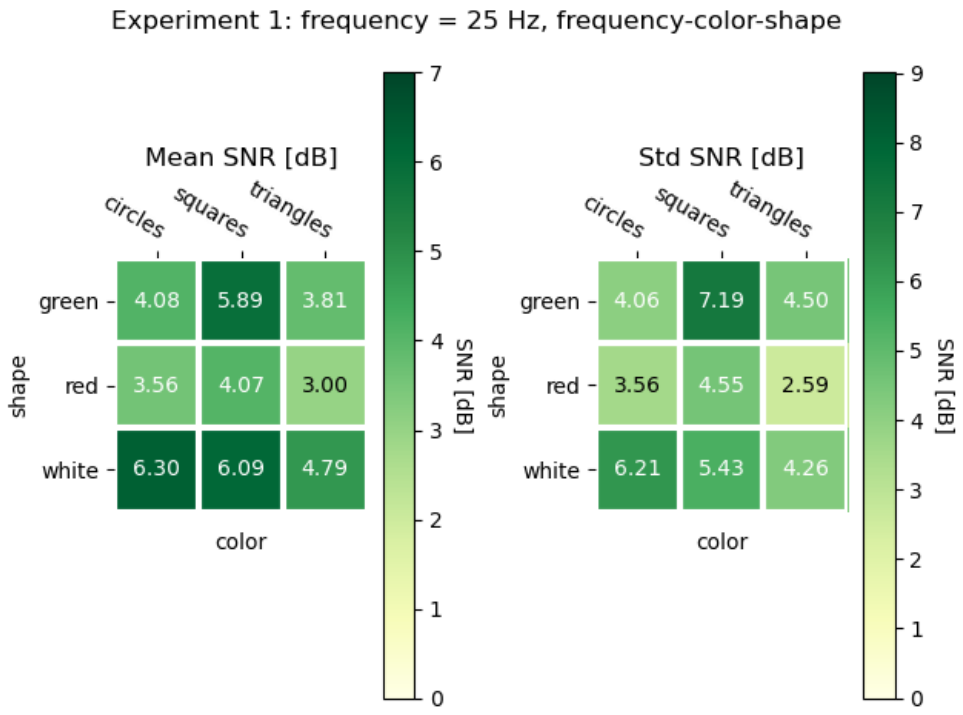


Figure A.20: SNR relationship between color and shape at 25Hz of experiment 2

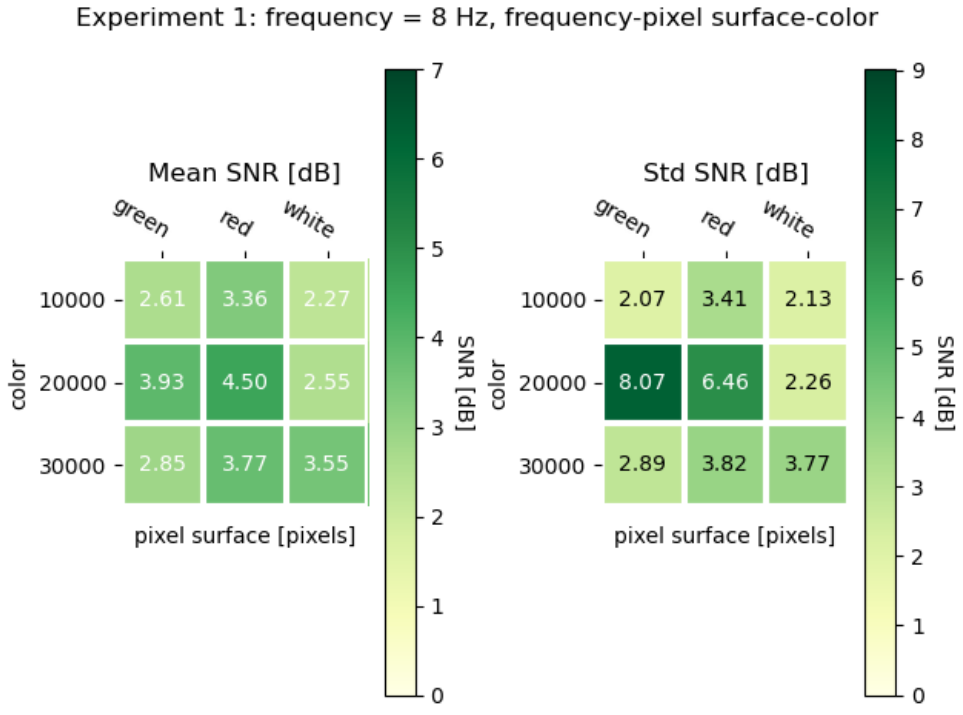


Figure A.21: SNR relationship between pixel surface and color at 8Hz of experiment 2

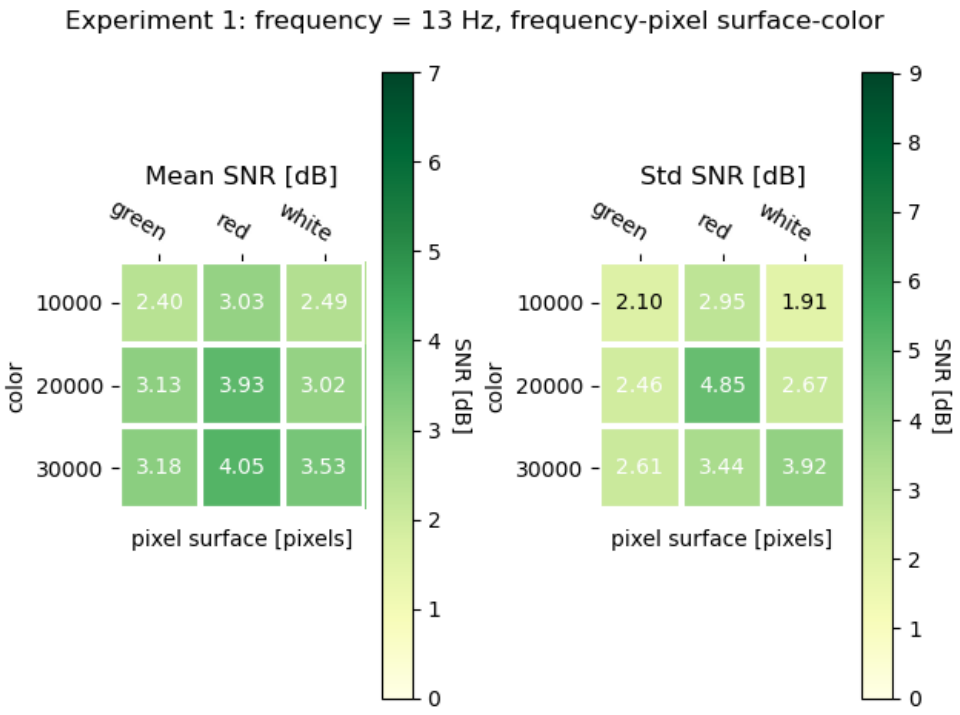


Figure A.22: SNR relationship between pixel surface and color at 13Hz of experiment 2

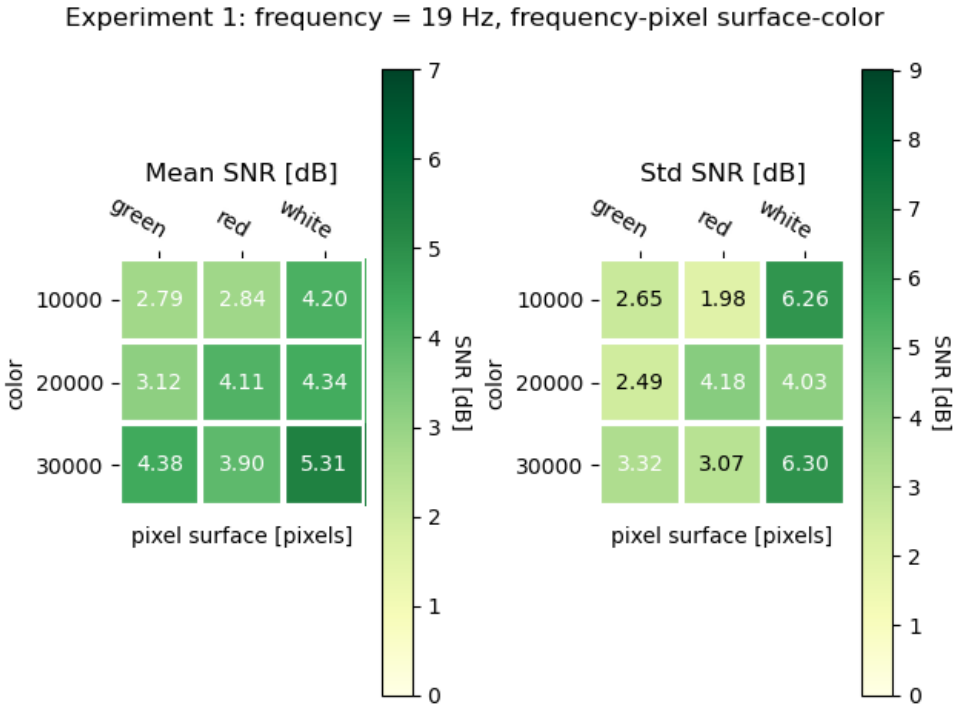


Figure A.23: SNR relationship between pixel surface and color at 19Hz of experiment 2

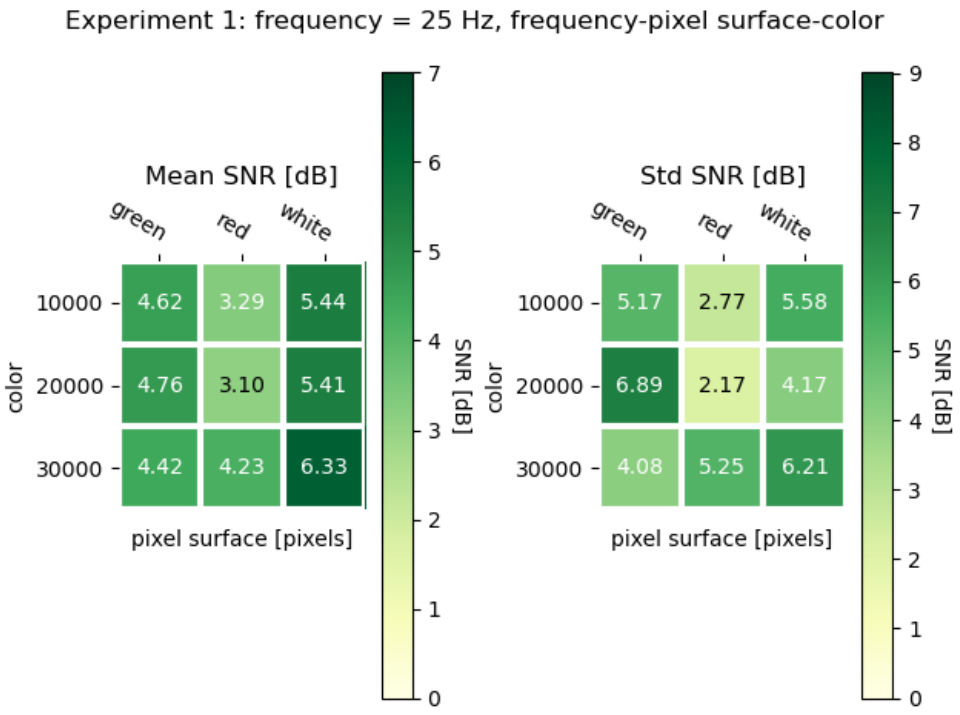


Figure A.24: SNR relationship between pixel surface and color at 25Hz of experiment 2

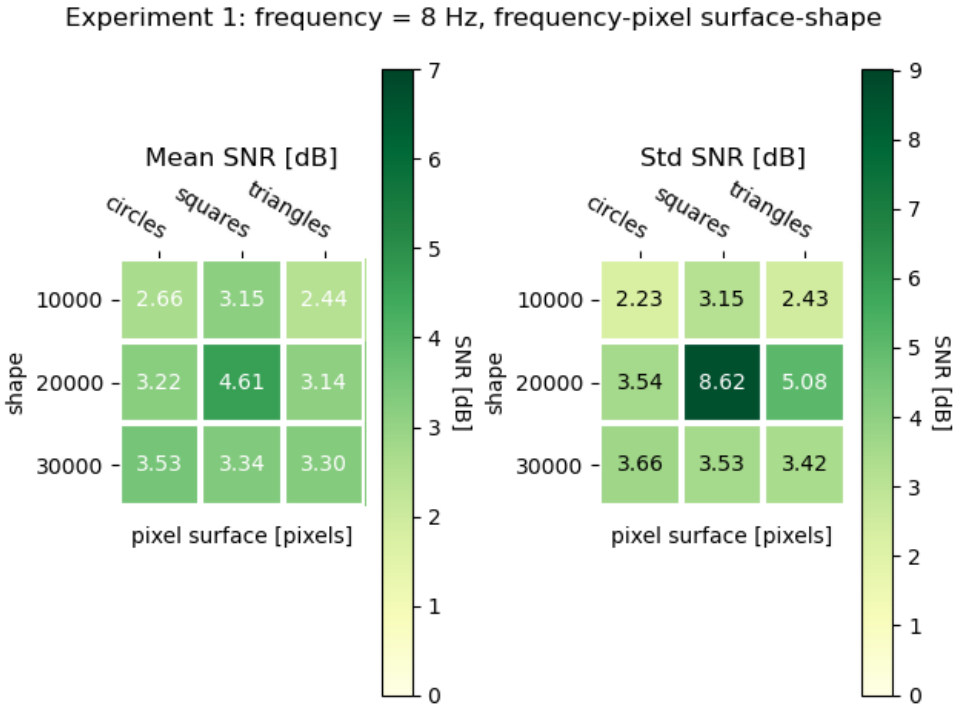


Figure A.25: SNR relationship between pixel surface and shape at 8Hz of experiment 2

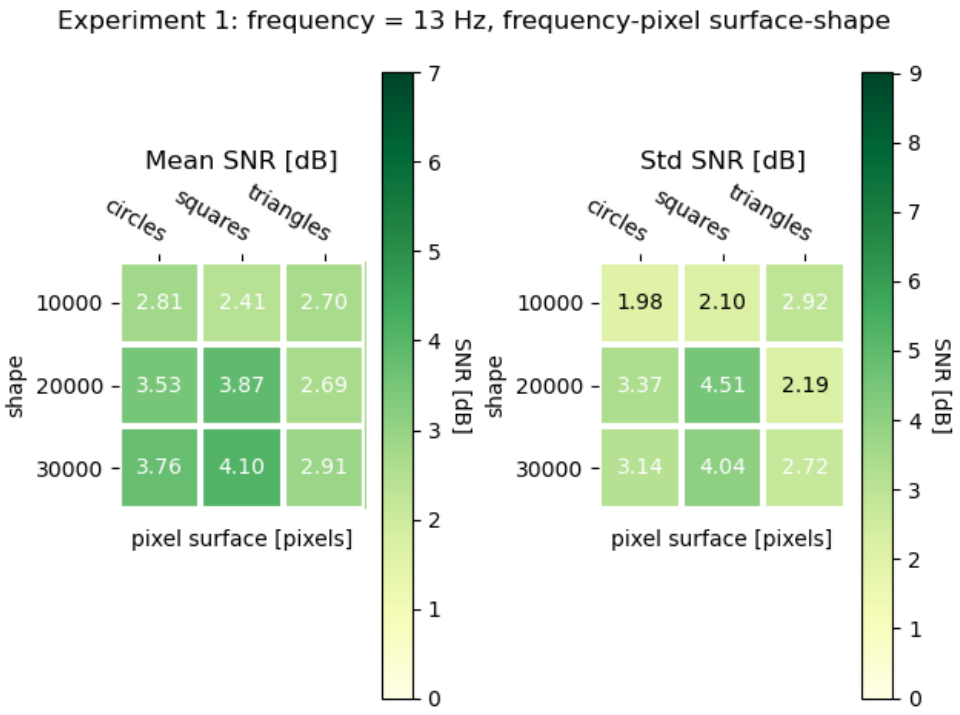


Figure A.26: SNR relationship between pixel surface and shape at 13Hz of experiment 2

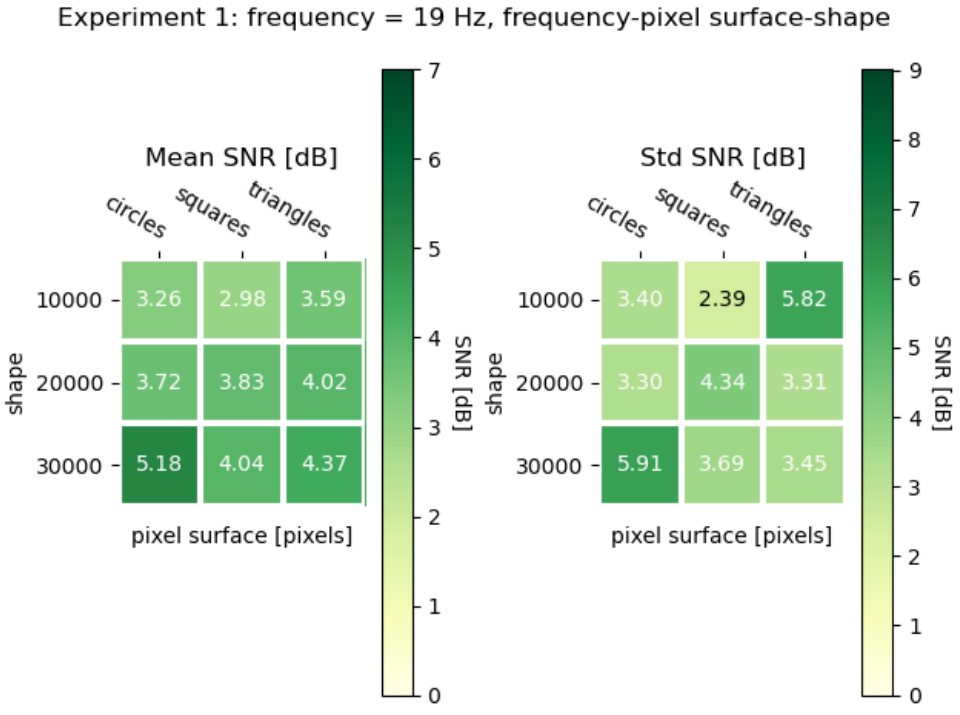


Figure A.27: SNR relationship between pixel surface and shape at 19Hz of experiment 2

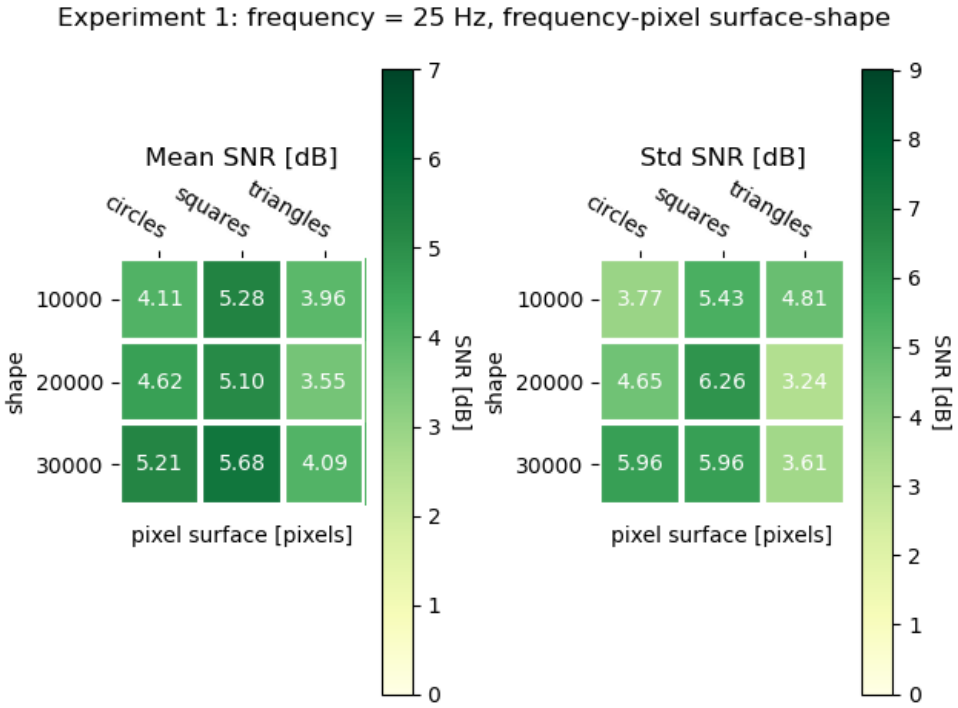


Figure A.28: SNR relationship between pixel surface and shape at 25Hz of experiment 2

A.3. Multi-dimensional SNR data

Table A.1: SNR table containing all the details of the SNR data of the multi-dimensional plots

experiment	category combination	group	SNR mean [dB]	SNR std [dB]	number of samples
Experiment 1	frequency-pixel surface-color	8-10000-green	2.61	2.07	54
Experiment 1	frequency-pixel surface-color	8-10000-red	3.36	3.41	54
Experiment 1	frequency-pixel surface-color	8-10000-white	2.27	2.13	54
Experiment 1	frequency-pixel surface-color	8-20000-green	3.93	8.07	54
Experiment 1	frequency-pixel surface-color	8-20000-red	4.50	6.46	54
Experiment 1	frequency-pixel surface-color	8-20000-white	2.55	2.26	54
Experiment 1	frequency-pixel surface-color	8-30000-green	2.85	2.89	54
Experiment 1	frequency-pixel surface-color	8-30000-red	3.77	3.82	54
Experiment 1	frequency-pixel surface-color	8-30000-white	3.55	3.77	54
Experiment 1	frequency-pixel surface-color	13-10000-green	2.40	2.10	54
Experiment 1	frequency-pixel surface-color	13-10000-red	3.03	2.95	54
Experiment 1	frequency-pixel surface-color	13-10000-white	2.49	1.91	54
Experiment 1	frequency-pixel surface-color	13-20000-green	3.13	2.46	54
Experiment 1	frequency-pixel surface-color	13-20000-red	3.93	4.85	54
Experiment 1	frequency-pixel surface-color	13-20000-white	3.02	2.67	54
Experiment 1	frequency-pixel surface-color	13-30000-green	3.18	2.61	54
Experiment 1	frequency-pixel surface-color	13-30000-red	4.05	3.44	54
Experiment 1	frequency-pixel surface-color	13-30000-white	3.53	3.92	54
Experiment 1	frequency-pixel surface-color	19-10000-green	2.79	2.65	54
Experiment 1	frequency-pixel surface-color	19-10000-red	2.84	1.98	54
Experiment 1	frequency-pixel surface-color	19-10000-white	4.20	6.26	54
Experiment 1	frequency-pixel surface-color	19-20000-green	3.12	2.49	54
Experiment 1	frequency-pixel surface-color	19-20000-red	4.11	4.18	54
Experiment 1	frequency-pixel surface-color	19-20000-white	4.34	4.03	54
Experiment 1	frequency-pixel surface-color	19-30000-green	4.38	3.32	54
Experiment 1	frequency-pixel surface-color	19-30000-red	3.90	3.07	54
Experiment 1	frequency-pixel surface-color	19-30000-white	5.31	6.30	54
Experiment 1	frequency-pixel surface-color	25-10000-green	4.62	5.17	54
Experiment 1	frequency-pixel surface-color	25-10000-red	3.29	2.77	54
Experiment 1	frequency-pixel surface-color	25-10000-white	5.44	5.58	54
Experiment 1	frequency-pixel surface-color	25-20000-green	4.76	6.89	54
Experiment 1	frequency-pixel surface-color	25-20000-red	3.10	2.17	54
Experiment 1	frequency-pixel surface-color	25-20000-white	5.41	4.17	54
Experiment 1	frequency-pixel surface-color	25-30000-green	4.42	4.08	54
Experiment 1	frequency-pixel surface-color	25-30000-red	4.23	5.25	54
Experiment 1	frequency-pixel surface-color	25-30000-white	6.33	6.21	54
Experiment 1	frequency-pixel surface-shape	8-10000-circles	2.66	2.23	54
Experiment 1	frequency-pixel surface-shape	8-10000-squares	3.15	3.15	54
Experiment 1	frequency-pixel surface-shape	8-10000-triangles	2.44	2.43	54
Experiment 1	frequency-pixel surface-shape	8-20000-circles	3.22	3.54	54
Experiment 1	frequency-pixel surface-shape	8-20000-squares	4.61	8.62	54
Experiment 1	frequency-pixel surface-shape	8-20000-triangles	3.14	5.08	54
Experiment 1	frequency-pixel surface-shape	8-30000-circles	3.53	3.66	54
Experiment 1	frequency-pixel surface-shape	8-30000-squares	3.34	3.53	54
Experiment 1	frequency-pixel surface-shape	8-30000-triangles	3.30	3.42	54
Experiment 1	frequency-pixel surface-shape	13-10000-circles	2.81	1.98	54
Experiment 1	frequency-pixel surface-shape	13-10000-squares	2.41	2.10	54
Experiment 1	frequency-pixel surface-shape	13-10000-triangles	2.70	2.92	54
Experiment 1	frequency-pixel surface-shape	13-20000-circles	3.53	3.37	54

Experiment 1	frequency-pixel surface-shape	13-20000-squares	3.87	4.51	54
Experiment 1	frequency-pixel surface-shape	13-20000-triangles	2.69	2.19	54
Experiment 1	frequency-pixel surface-shape	13-30000-circles	3.76	3.14	54
Experiment 1	frequency-pixel surface-shape	13-30000-squares	4.10	4.04	54
Experiment 1	frequency-pixel surface-shape	13-30000-triangles	2.91	2.72	54
Experiment 1	frequency-pixel surface-shape	19-10000-circles	3.26	3.40	54
Experiment 1	frequency-pixel surface-shape	19-10000-squares	2.98	2.39	54
Experiment 1	frequency-pixel surface-shape	19-10000-triangles	3.59	5.82	54
Experiment 1	frequency-pixel surface-shape	19-20000-circles	3.72	3.30	54
Experiment 1	frequency-pixel surface-shape	19-20000-squares	3.83	4.34	54
Experiment 1	frequency-pixel surface-shape	19-20000-triangles	4.02	3.31	54
Experiment 1	frequency-pixel surface-shape	19-30000-circles	5.18	5.91	54
Experiment 1	frequency-pixel surface-shape	19-30000-squares	4.04	3.69	54
Experiment 1	frequency-pixel surface-shape	19-30000-triangles	4.37	3.45	54
Experiment 1	frequency-pixel surface-shape	25-10000-circles	4.11	3.77	54
Experiment 1	frequency-pixel surface-shape	25-10000-squares	5.28	5.43	54
Experiment 1	frequency-pixel surface-shape	25-10000-triangles	3.96	4.81	54
Experiment 1	frequency-pixel surface-shape	25-20000-circles	4.62	4.65	54
Experiment 1	frequency-pixel surface-shape	25-20000-squares	5.10	6.26	54
Experiment 1	frequency-pixel surface-shape	25-20000-triangles	3.55	3.24	54
Experiment 1	frequency-pixel surface-shape	25-30000-circles	5.21	5.96	54
Experiment 1	frequency-pixel surface-shape	25-30000-squares	5.68	5.96	54
Experiment 1	frequency-pixel surface-shape	25-30000-triangles	4.09	3.61	54
Experiment 1	frequency-color-shape	8-green-circles	3.17	2.72	54
Experiment 1	frequency-color-shape	8-green-squares	3.85	8.19	54
Experiment 1	frequency-color-shape	8-green-triangles	2.37	1.78	54
Experiment 1	frequency-color-shape	8-red-circles	3.63	4.10	54
Experiment 1	frequency-color-shape	8-red-squares	3.89	4.35	54
Experiment 1	frequency-color-shape	8-red-triangles	4.11	5.73	54
Experiment 1	frequency-color-shape	8-white-circles	2.61	2.57	54
Experiment 1	frequency-color-shape	8-white-squares	3.36	3.43	54
Experiment 1	frequency-color-shape	8-white-triangles	2.40	2.42	54
Experiment 1	frequency-color-shape	13-green-circles	3.06	2.49	54
Experiment 1	frequency-color-shape	13-green-squares	2.90	2.39	54
Experiment 1	frequency-color-shape	13-green-triangles	2.74	2.38	54
Experiment 1	frequency-color-shape	13-red-circles	3.79	3.16	54
Experiment 1	frequency-color-shape	13-red-squares	4.08	4.79	54
Experiment 1	frequency-color-shape	13-red-triangles	3.15	3.35	54
Experiment 1	frequency-color-shape	13-white-circles	3.25	3.02	54
Experiment 1	frequency-color-shape	13-white-squares	3.39	3.66	54
Experiment 1	frequency-color-shape	13-white-triangles	2.40	1.90	54
Experiment 1	frequency-color-shape	19-green-circles	3.70	3.34	54
Experiment 1	frequency-color-shape	19-green-squares	3.02	2.33	54
Experiment 1	frequency-color-shape	19-green-triangles	3.57	2.97	54
Experiment 1	frequency-color-shape	19-red-circles	3.90	3.65	54
Experiment 1	frequency-color-shape	19-red-squares	3.52	3.30	54
Experiment 1	frequency-color-shape	19-red-triangles	3.43	2.70	54
Experiment 1	frequency-color-shape	19-white-circles	4.56	5.87	54
Experiment 1	frequency-color-shape	19-white-squares	4.30	4.64	54
Experiment 1	frequency-color-shape	19-white-triangles	4.99	6.28	54
Experiment 1	frequency-color-shape	25-green-circles	4.08	4.06	54
Experiment 1	frequency-color-shape	25-green-squares	5.89	7.19	54
Experiment 1	frequency-color-shape	25-green-triangles	3.81	4.50	54
Experiment 1	frequency-color-shape	25-red-circles	3.56	3.56	54
Experiment 1	frequency-color-shape	25-red-squares	4.07	4.55	54
Experiment 1	frequency-color-shape	25-red-triangles	3.00	2.59	54

Experiment 1	frequency-color-shape	25-white-circles	6.30	6.21	54
Experiment 1	frequency-color-shape	25-white-squares	6.09	5.43	54
Experiment 1	frequency-color-shape	25-white-triangles	4.79	4.26	54
Experiment 2	frequency-pixel surface-color	8-10000-green	3.36	3.20	72
Experiment 2	frequency-pixel surface-color	8-10000-red	2.98	3.47	72
Experiment 2	frequency-pixel surface-color	8-10000-white	2.80	2.31	72
Experiment 2	frequency-pixel surface-color	8-20000-green	3.62	4.54	72
Experiment 2	frequency-pixel surface-color	8-20000-red	3.90	5.54	72
Experiment 2	frequency-pixel surface-color	8-20000-white	2.75	2.44	72
Experiment 2	frequency-pixel surface-color	8-30000-green	4.03	6.56	72
Experiment 2	frequency-pixel surface-color	8-30000-red	4.26	4.47	72
Experiment 2	frequency-pixel surface-color	8-30000-white	2.96	3.57	72
Experiment 2	frequency-pixel surface-color	13-10000-green	2.50	1.94	72
Experiment 2	frequency-pixel surface-color	13-10000-red	2.53	2.08	72
Experiment 2	frequency-pixel surface-color	13-10000-white	2.50	2.45	72
Experiment 2	frequency-pixel surface-color	13-20000-green	2.42	2.15	72
Experiment 2	frequency-pixel surface-color	13-20000-red	2.75	1.85	72
Experiment 2	frequency-pixel surface-color	13-20000-white	3.51	3.00	72
Experiment 2	frequency-pixel surface-color	13-30000-green	2.49	1.89	72
Experiment 2	frequency-pixel surface-color	13-30000-red	5.01	5.04	72
Experiment 2	frequency-pixel surface-color	13-30000-white	2.85	2.32	72
Experiment 2	frequency-pixel surface-color	19-10000-green	2.67	2.15	72
Experiment 2	frequency-pixel surface-color	19-10000-red	3.48	2.68	72
Experiment 2	frequency-pixel surface-color	19-10000-white	3.70	4.05	72
Experiment 2	frequency-pixel surface-color	19-20000-green	3.16	2.65	72
Experiment 2	frequency-pixel surface-color	19-20000-red	4.20	6.44	72
Experiment 2	frequency-pixel surface-color	19-20000-white	4.53	3.61	72
Experiment 2	frequency-pixel surface-color	19-30000-green	3.21	3.11	72
Experiment 2	frequency-pixel surface-color	19-30000-red	4.26	3.26	72
Experiment 2	frequency-pixel surface-color	19-30000-white	5.13	5.20	72
Experiment 2	frequency-pixel surface-color	25-10000-green	4.25	6.45	72
Experiment 2	frequency-pixel surface-color	25-10000-red	2.70	2.48	72
Experiment 2	frequency-pixel surface-color	25-10000-white	3.85	4.64	72
Experiment 2	frequency-pixel surface-color	25-20000-green	3.88	4.08	72
Experiment 2	frequency-pixel surface-color	25-20000-red	3.48	2.43	72
Experiment 2	frequency-pixel surface-color	25-20000-white	4.84	4.52	72
Experiment 2	frequency-pixel surface-color	25-30000-green	4.13	4.87	72
Experiment 2	frequency-pixel surface-color	25-30000-red	3.54	2.76	72
Experiment 2	frequency-pixel surface-color	25-30000-white	5.97	7.05	72

B

2D SNR plots

Here the relationship is visualized between SNR and a category of interface characteristics, which is referred to as 2-dimensional.

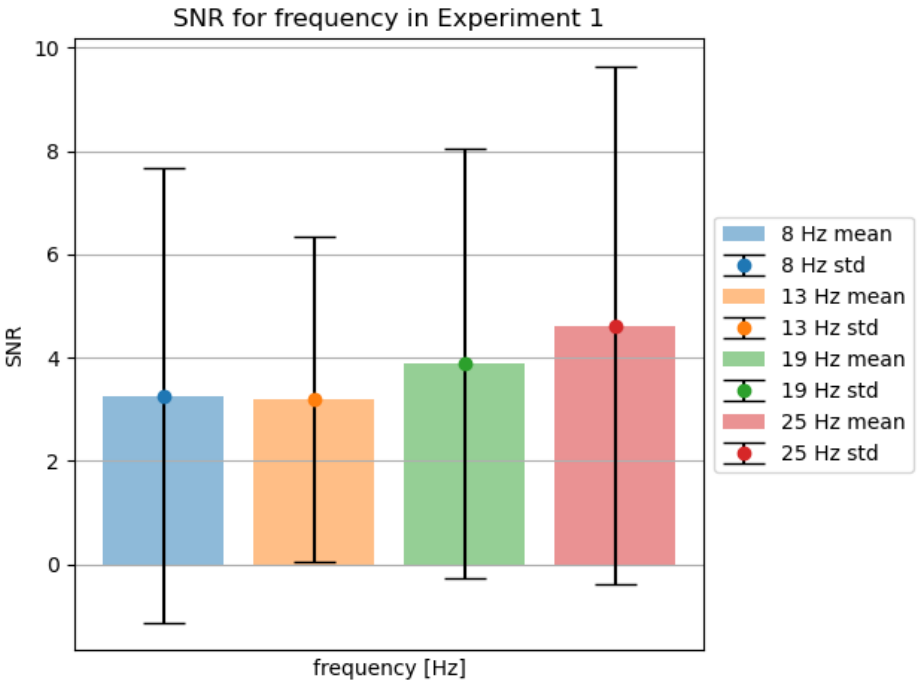


Figure B.1: Experiment 1 SNR plot for frequency

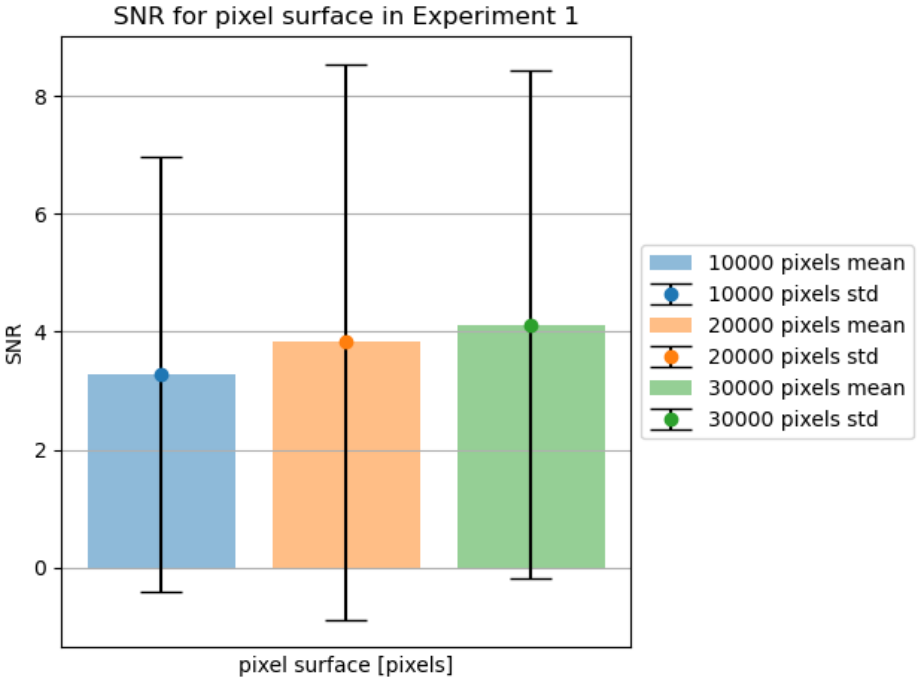


Figure B.2: Experiment 1 SNR plot for pixel surface

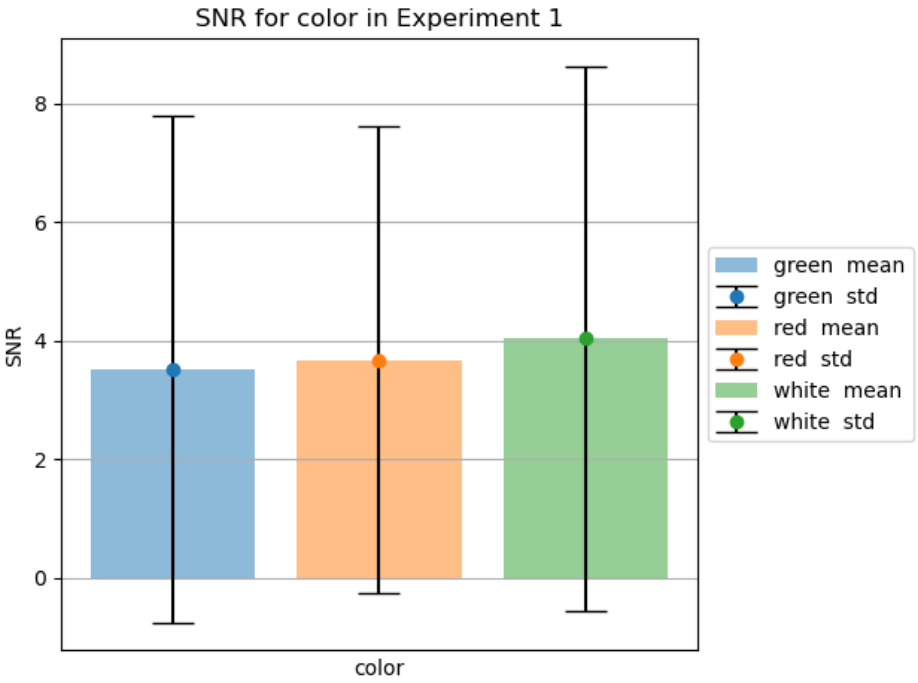


Figure B.3: Experiment 1 SNR plot for color

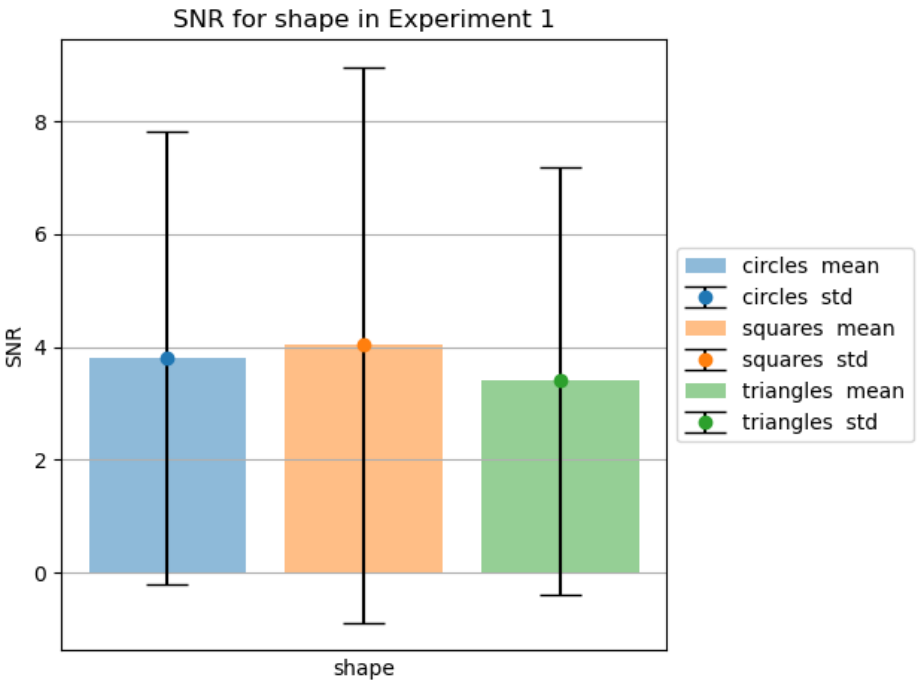


Figure B.4: Experiment 1 SNR plot for shape

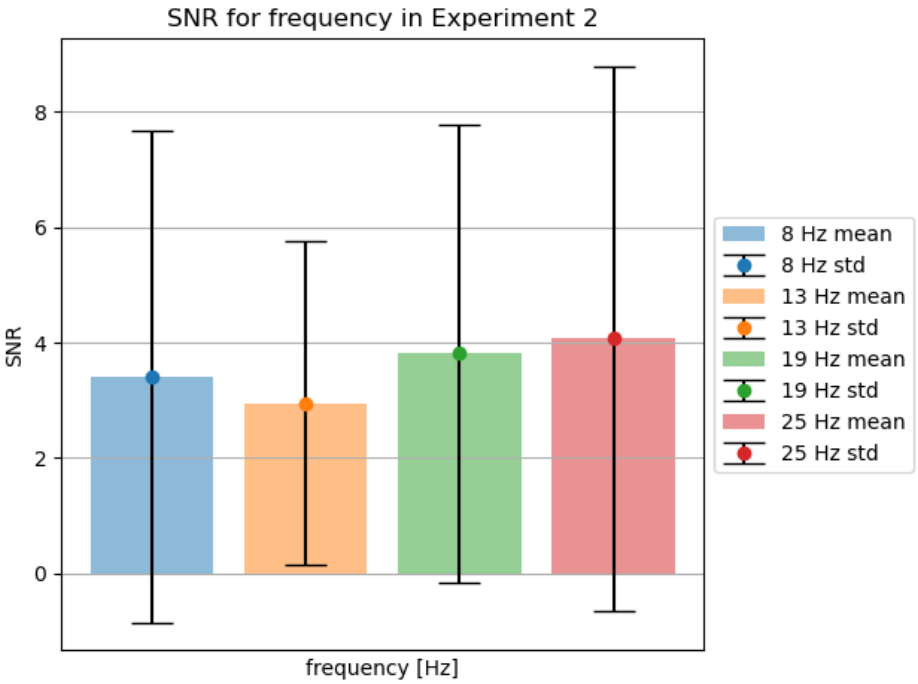


Figure B.5: Experiment 2 SNR plot for frequency

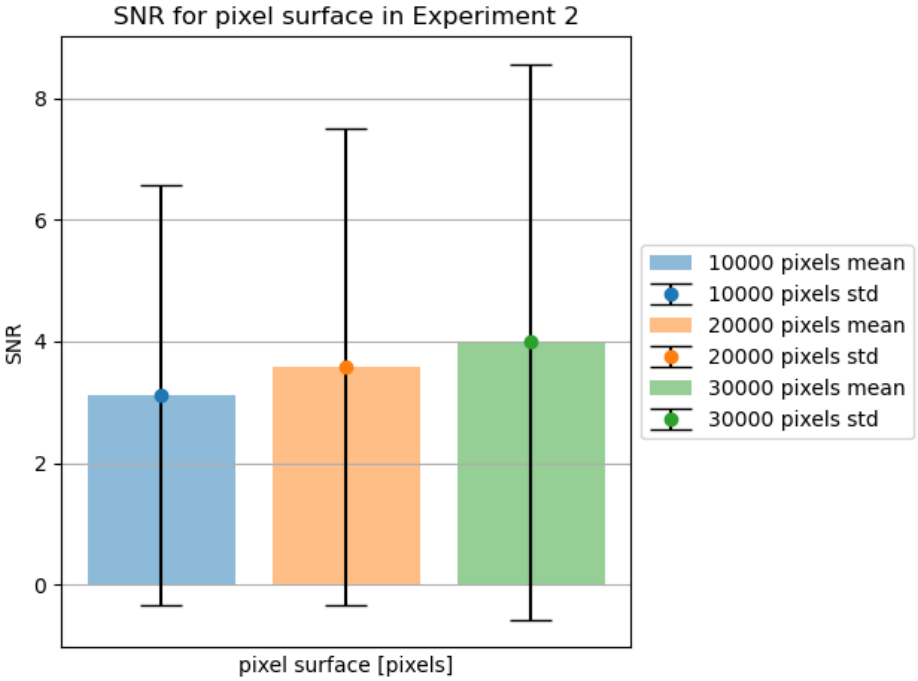


Figure B.6: Experiment 2 SNR plot for pixel surface

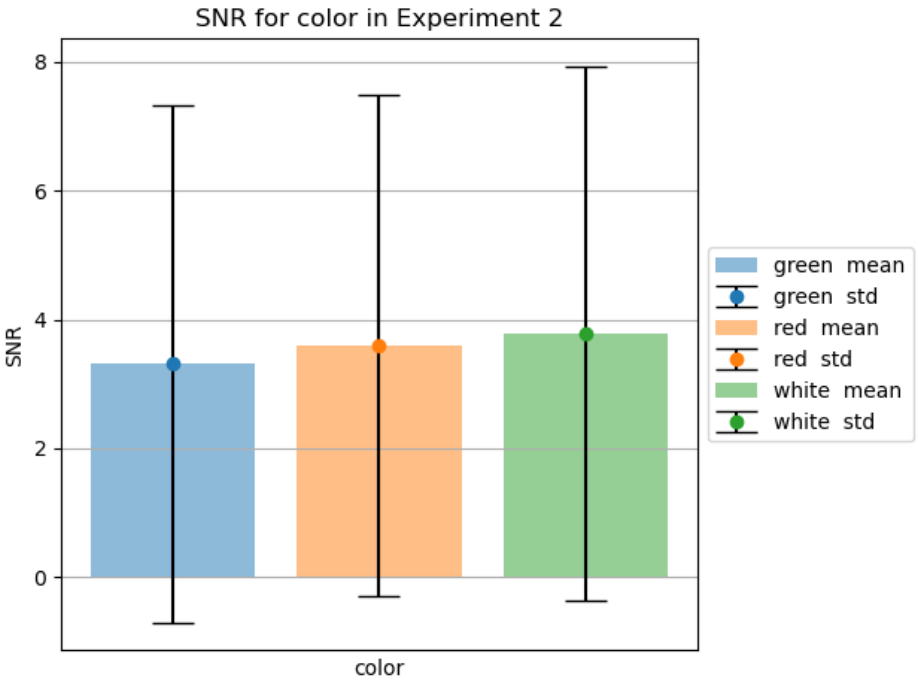


Figure B.7: Experiment 2 SNR plot for color

C

Questionnaire

SSVEP-based brain-computer interfaces

Questions form for participants

Name: …….

Date: ……..

Subject number: ……..

Colors:

Black/Red

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High
I had to blink a lot	Very Low	Low	Neutral	High	Very High

Black/green

Comfort	Very Low	Low	Neutral	High	Very High

I was very focused	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
Eye Irritation	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Black/white

Comfort	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I was very focused	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
Eye Irritation	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Shapes:

Triangle

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High
I had to blink a lot	Very Low	Low	Neutral	High	Very High

Square

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High

Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High
I had to blink a lot	Very Low	Low	Neutral	High	Very High

Circle

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High
I had to blink a lot	Very Low	Low	Neutral	High	Very High

Frequency

High frequency

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High
I had to blink a lot	Very Low	Low	Neutral	High	Very High

Middle frequency

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High

It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Low frequency

Comfort	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I was very focused	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
Eye Irritation	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Size stimulus:

Small size:

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High
I had to blink a lot	Very Low	Low	Neutral	High	Very High

Medium size:

Comfort	Very Low	Low	Neutral	High	Very High
I was very focused	Very Low	Low	Neutral	High	Very High
Eye Irritation	Very Low	Low	Neutral	High	Very High
It was easy to focus on the center of the stimulus	Very Low	Low	Neutral	High	Very High

I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Large size:

Comfort	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I was very focused	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
Eye Irritation	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Single stimulus vs multiple stimuli

Single Stimulus:

Comfort	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I was very	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

focused					
Eye Irritation	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

Multiple stimuli:

Comfort	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I was very focused	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
Eye Irritation	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
It was easy to focus on the center of the stimulus	**Very Low**	**Low**	**Neutral**	**High**	**Very High**
I had to blink a lot	**Very Low**	**Low**	**Neutral**	**High**	**Very High**

D

Code

The code is written in the programming language python. A GitHub repository has been created with a Readme for reproducibility purposes: https://github.com/SjoerdTimovanVliet/SSVEP_interface_thesis. The code has was executed on the operating system Ubuntu 20.04. The programming language of choice is python.

All the code and requirements.txt files for the conda environments in this document have been put in the sequence of the Readme of the GitHub repository.

D.1. Generate stimuli dataset

Used to generate the dataset of videos and photos that make up the trials.

D.1.1. Requirements_video_creater.txt

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: linux-64
_libgcc_mutex=0.1=main
_openmp_mutex=5.1=1_gnu
blas=1.0=openblas
bzip2=1.0.8=h7f98852_4
ca-certificates=2022.12.7=ha878542_0
cairo=1.16.0=h18b612c_1001
certifi=2022.12.7=pyhd8ed1ab_0
dbus=1.13.18=hb2f20db_0
eigen=3.4.0=h4bd325d_0
expat=2.2.10=h9c3ff4c_0
ffmpeg=4.2.2=h20bf706_0
fontconfig=2.14.1=hef1e5e3_0
freetype=2.10.4=h0708190_1
giflib=5.2.1=h36c2ea0_2
glib=2.69.1=h4ff587b_1
gmp=6.2.1=h58526e2_0
gnutls=3.6.13=h85f3911_1
graphite2=1.3.14=h295c915_1
gst-plugins-base=1.14.0=h8213a91_2
gstreamer=1.14.0=h28cd5cc_2
harfbuzz=4.3.0=hd55b92a_0
hdf5=1.10.6=h3ffc7dd_1
icu=58.2=hf484d3e_1000
```

```
jpeg=9e=h166bdaf_1
 keyutils=1.6.1=h166bdaf_0
krb5=1.19.3=h3790be6_0
lame=3.100=h7f98852_1001
ld_impl_linux-64=2.38=h1181459_1
 lerc=3.0=h295c915_0
 libblas=3.9.0=15_linux64_openblas
 libcblas=3.9.0=15_linux64_openblas
 libclang=10.0.1=default_hb85057a_2
 libdeflate=1.8=h7f8727e_5
 libedit=3.1.20191231=he28a2e2_2
libevent=2.1.12=h8f2d780_0
  libffi=3.3=he6710b0_2
libgcc-ng=11.2.0=h1234567_1
 libgfortran-ng=12.2.0=h69a702a_19
 libgfortran5=12.2.0=h337968e_19
libgomp=11.2.0=h1234567_1
liblapack=3.9.0=15_linux64_openblas
libllvm10=10.0.1=he513fc3_3
libopenblas=0.3.20=pthreads_h78a6416_0
libopus=1.3.1=h7f98852_1
libpng=1.6.37=hbc83047_0
libpq=12.9=h16c4e8d_3
 libprotobuf=3.20.1=h4ff587b_0
 libstdcxx-ng=11.2.0=h1234567_1
  libtiff=4.4.0=hecacb30_2
 libuuid=2.32.1=h7f98852_1000
libvpx=1.7.0=h439df22_0
libwebp=1.2.4=h11a3e52_0
libwebp-base=1.2.4=h5eee18b_0
libxcb=1.15=h7f8727e_0
libxkbcommon=1.0.3=he3ba5ed_0
libxml2=2.9.14=h74e7548_0
 libxslt=1.1.35=h4e12654_0
lz4-c=1.9.3=h9c3ff4c_1
ncurses=6.3=h5eee18b_3
 nettle=3.6=he412f7d_0
nspr=4.33=h295c915_0
nss=3.74=h0370c37_0
numpy=1.22.3=py39hc58783e_2
opencv=4.6.0=py39hd653453_2
openh264=2.1.1=h4ff587b_0
openjpeg=2.3.1=hf7af979_3
openssl=1.1.1s=h7f8727e_0
pcre=8.45=h9c3ff4c_0
pip=22.3.1=py39h06a4308_0
pixman=0.38.0=h516909a_1003
python=3.9.0=hdb3f193_2
python_abi=3.9=2_cp39
qt-main=5.15.2=h327a75a_7
qt-webengine=5.15.9=hd2b0992_4
qtwebkit=5.212=h4eab89a_4
readline=8.2=h5eee18b_0
setuptools=65.6.3=py39h06a4308_0
 sqlite=3.40.1=h5082296_0
tk=8.6.12=h1ccaba5_0
```

```
tzdata=2022g=h04d1e81_0
wheel=0.37.1=pyhd3eb1b0_0
x264=1!157.20191217=h7b6447c_0
xorg-kbproto=1.0.7=h7f98852_1002
xorg-libice=1.0.10=h7f98852_0
xorg-libsm=1.2.3=hd9c2040_1000
xorg-libx11=1.7.2=h7f98852_0
xorg-libxext=1.3.4=h7f98852_1
xorg-libxrender=0.9.10=h7f98852_1003
xorg-renderproto=0.11.1=h7f98852_1002
xorg-xextproto=7.3.0=h7f98852_1002
xorg-xproto=7.0.31=h7f98852_1007
xz=5.2.10=h5eee18b_1
zlib=1.2.13=h5eee18b_0
zstd=1.5.2=ha4553b6_0
```

D.1.2. ssvep_interface_video_create_v1.py

```
import cv2
import numpy as np
from math import sin, pi
from datetime import datetime
import os


class SSVEP_Interface():

    def __init__(self):
        # set the frame size
        self.frame_size = (1080, 1920)
        # set the frame rate
        self.frame_rate = 60

        self.setup_settings()
        self.create_stimuli()
        # set the extension of the video
        self.extension = '.mp4'

        if self.extension == '.mp4':
            # define the codec of the video as H264
            fourcc_type = 'avc1'
            self.fourcc = cv2.VideoWriter_fourcc(*fourcc_type)

        elif self.extension == '.avi':
            # define the codec of the video as MPEG
            self.fourcc = cv2.VideoWriter_fourcc(*'MPEG')
        # save the current date and time for folder name
        self.date_time = datetime.now().strftime("%Y%m%d_%H%M%S")

    def calculate_possible_frequencies(self) -> list:
        """ Calculates the number of possible frequencies for the stimuli that are not multiples of
            each other.
        The frequencies are alos whole number and not decimals as this would allow different
            methods to evoket the wave (square or sine)

        Returns:
```

```
            list : list of possible frequencies
        """
        # calculate possible frequencies
        possible_frequencies = []
        for i in range(1, int(self.frame_rate/2)):
            # check if the number is a whole number
            if self.frame_rate % i == 0:
                possible_frequencies.append(i)
        print(f"Possible frequencies: {possible_frequencies}")
        # find 4 frequencies in the list of possible_frequencies that are unique and not multiples
            of each other
        frequencies = []
        # loop through the list of possible frequencies and check if the number is a multiple of
            the previous number
        for i in range(len(possible_frequencies)):

            if len(frequencies) == 4:
                break
            # loop through the list of frequencies and check if the number is already in the list
            if possible_frequencies[i] not in frequencies:
                frequencies.append(possible_frequencies[i])
                for j in range(i+1, len(possible_frequencies)):
                    # check if the number is a multiple of the previous number
                    if possible_frequencies[j] % possible_frequencies[i] == 0:
                        break
                    # check if the number is the last number in the list
                    if j == len(possible_frequencies)-1:
                        frequencies.append(possible_frequencies[i])
        print("Possible frequencies after removing multiples: ", frequencies)
        return possible_frequencies

    def setup_settings(self):
        """ Sets up the settings for the stimuli
        """
        # set frequencies for the stimuli
        self.frequencies = [8, 13, 22, 29]
        # red, green, white
        self.colors = ['red', 'green', 'white']
        # pixel size of the stimuli
        self.pixel_surface = [10000, 20000, 30000]
        # types of shapes
        self.shapes = ['circles', 'squares', 'triangles']

        # draw thickness of the shapes
        self.thickness = -1 # -1 to fill circle

    def create_random_order_of_stimuli_settings_1x1(self) -> list:
        """ Creates a random order of the stimuli settings

        Returns:
            list : list of tuples with the indices of the settings
        """
        # create a list of indices for all settings
        frequency_indices = list(range(len(self.frequencies)))
        color_indices = list(range(len(self.colors)))
        pixel_surface_indices = list(range(len(self.pixel_surface)))
```

```
        shape_indices = list(range(len(self.shapes)))
        # calculate the number of stimuli
        calculate_number_of_stimuli = len(
            frequency_indices)*len(color_indices)*len(pixel_surface_indices)*len(shape_indices)
        print(f"Number of stimuli in one block: {calculate_number_of_stimuli}")
        # shuffle the indices of all settings
        np.random.shuffle(frequency_indices)
        np.random.shuffle(color_indices)
        np.random.shuffle(pixel_surface_indices)
        np.random.shuffle(shape_indices)
        # create a list of tuples with the indices of all the different combinations of settings
        indices = []
        for k in pixel_surface_indices:
            for j in color_indices:
                for i in frequency_indices:
                    for l in shape_indices:
                        indices.append((k, j, i, l))
        # shuffle the list of combinations
        np.random.shuffle(indices)
        return indices

    def create_random_order_of_stimuli_settings_2x2(self) -> list:
        """ Creates a random order of the stimuli settings

        Returns:
            list : list of tuples with the indices of the settings
        """
        # create a list of indices for all settings
        frequency_indices = list(range(len(self.frequencies)))
        color_indices = list(range(len(self.colors)))
        pixel_surface_indices = list(range(len(self.pixel_surface)))
        shape_indices = [1]  # only squares
        center_coordinates_indices = list(range(len(self.center_coordinates)))
        # calculate the number of stimuli
        calculate_number_of_stimuli = len(frequency_indices)*len(color_indices)*len(
            pixel_surface_indices)*len(shape_indices)*len(center_coordinates_indices)
        print(f"Number of stimuli in one block: {calculate_number_of_stimuli}")
        # shuffle the indices of all settings
        np.random.shuffle(frequency_indices)
        np.random.shuffle(color_indices)
        np.random.shuffle(pixel_surface_indices)
        np.random.shuffle(shape_indices)
        np.random.shuffle(center_coordinates_indices)
        # create a list of tuples with the indices of the settings
        indices = []
        for k in pixel_surface_indices:
            for j in color_indices:
                for i in frequency_indices:
                    for l in shape_indices:
                        # 1 means it is always squares
                        indices.append((k, j, i, 1))
        # shuffle the list of combinations
        np.random.shuffle(indices)
        return indices

    def create_stimuli(self):
```

```
        """Creates the stimuli by calculating the coordinates for the stimuli
        """
        # Get center coordinates of the 4 quadrants of the screen
        width_1 = int(self.frame_size[0]/4)
        height_1 = int(self.frame_size[1]/4)
        width_2 = int(self.frame_size[0]/4*3)
        height_2 = int(self.frame_size[1]/4*3)
        # create a list of tuples with the center coordinates of the 4 quadrants
        self.center_coordinates = [
            (height_1, width_1), (height_1, width_2), (height_2, width_1), (height_2, width_2)]

    def _draw_circles(self, image: np.ndarray, pixel_surface: int, center_coordinates: list,
        color_tuples: list) -> np.ndarray:
        """ Draws circles on the image
        Args:
            image (np.ndarray): The image on which the circles are drawn
            pixel_surface (int): The size of the circles in pixels
            center_coordinates (list): the coordinates of the center of the circles
            color_tuples (list): the colors of the circles

        Returns:
            np.ndarray: The image with the circles drawn on it
        """
        # calculate radius of circle
        radius = int(np.sqrt(pixel_surface/pi))
        # draw circles
        for i, center_coordinate in enumerate(center_coordinates):
            # unpack color tuple
            color_tuple = color_tuples[i]
            image = cv2.circle(image, center_coordinate,
                               radius, color_tuple, self.thickness)
        return image

    def _draw_squares(self, image: np.ndarray, pixel_surface: int, center_coordinates: list,
        color_tuples: list) -> np.ndarray:
        """ Draws squares on the image
        Args:
            image (np.ndarray): The image on which the squares are drawn
            pixel_surface (int): The size of the squares in pixels
            center_coordinates (list): the coordinates of the center of the squares
            color_tuples (list): the colors of the squares

        Returns:
            np.ndarray: The image with the squares drawn on it
        """
        # calculate side length of square
        side_length = int(np.sqrt(pixel_surface))
        # create squares
        for i, center_coordinate in enumerate(center_coordinates):
            # draw all squares. The coordinates are measured in integer values.
            # unpack color tuple
            color_tuple = color_tuples[i]
            coordinate_1 = (int(
                center_coordinate[0]-side_length//2), int(center_coordinate[1]-side_length//2))
            coordinate_2 = (int(
                center_coordinate[0]+side_length//2), int(center_coordinate[1]+side_length//2))
```

```
            image = cv2.rectangle(image, coordinate_1,
                                  coordinate_2, color_tuple, self.thickness)

        return image

    def _draw_triangles(self, image: np.ndarray, pixel_surface: int, center_coordinates: list,
                        color_tuples: list) -> np.ndarray:
        """ Draws triangles on the image
        Args:
            image (np.ndarray): The image on which the triangles are drawn
            pixel_surface (int): The size of the triangles in pixels
            center_coordinates (list): the coordinates of the center of the triangles
            color_tuples (list): the colors of the triangles

        Returns:
            np.ndarray: The image with the triangles drawn on it
        """
        # us the pixel surface to derive corner coordinates of the iscoceles triangle. All sides of
            the triangle are equal
        # calculate side length of triangle
        diagonal = np.sqrt(pixel_surface*8/np.sqrt(3))
        # half base length
        half_base_length = diagonal/2
        height = diagonal/2*np.sqrt(3)

        # create triangles
        self.triangle_center_coordinates = []
        for i, center_coordinate in enumerate(center_coordinates):
            # draw all triangles. The coordinates are measured in integer values, so the triangles
                are not perfectly centered
            color_tuple = color_tuples[i]
            coordinate_1 = (
                int(center_coordinate[0]-half_base_length), int(center_coordinate[1]+height//2))
            coordinate_2 = (
                int(center_coordinate[0]+half_base_length), int(center_coordinate[1]+height//2))
            coordinate_3 = (int(center_coordinate[0]), int(
                center_coordinate[1]-height//2))
            triangle_center_coordinate = (int(center_coordinate[0]), int(
                (coordinate_1[1]+coordinate_2[1]+coordinate_3[1])/3))

            # correct the 3 coordinates down to make center_coordinate equal to the center
                coordinate of the triangle
            if triangle_center_coordinate[1] > center_coordinate[1]:
                correction = center_coordinate[1] - \
                    triangle_center_coordinate[1]
                coordinate_1 = (coordinate_1[0], coordinate_1[1]+correction)
                coordinate_2 = (coordinate_2[0], coordinate_2[1]+correction)
                coordinate_3 = (coordinate_3[0], coordinate_3[1]+correction)

            elif triangle_center_coordinate[1] < center_coordinate[1]:
                correction = triangle_center_coordinate[1] - \
                    center_coordinate[1]
                coordinate_1 = (coordinate_1[0], coordinate_1[1]-correction)
                coordinate_2 = (coordinate_2[0], coordinate_2[1]-correction)
                coordinate_3 = (coordinate_3[0], coordinate_3[1]-correction)
```

```
            # save the center coordinates of the triangles
            self.triangle_center_coordinates.append(
                (int((coordinate_1[0]+coordinate_2[0]+coordinate_3[0])/3), int((coordinate_1[1]+
                    coordinate_2[1]+coordinate_3[1])/3)))
            # draw triangle
            triangle_cnt = np.array([coordinate_1, coordinate_2, coordinate_3])
            image = cv2.drawContours(
                image, [triangle_cnt], 0, color_tuple, self.thickness)

        return image

    def _calculate_screen_color_sinusoidal(self, frequency: float, frame_number: int) -> int:
        """ Calculates the color of the screen for a given frequency and frame number
        Args:
            frequency (float): The frequency of the sinusoidal color change
            frame_number (int): The frame number

        Returns:
            int: The color of the screen
        """
        tmp = 1/2*(1+sin(2*pi*frequency*(frame_number/self.frame_rate)))
        color = int(round(255*tmp))
        return color

    def create_video_1x1(self):
        """ Creates a video with 1 stimulus on the screen
        """
        # create a video with 1 stimulus on the screen
        length_of_video = 4  # seconds
        inter_trial_time = 1  # seconds
        # calculate number of frames for the video
        number_of_frames = length_of_video * self.frame_rate
        # The inter trial interval which is replace for a png but can be replaced by a video (
            OPTIONAL)
        number_of_frames_inter_trial = inter_trial_time * self.frame_rate
        # calculate center coordinate of the screen
        center_coordinate = (
            int(self.frame_size[1]/2), int(self.frame_size[0]/2))
        # create folder name and location
        folder = f"stimuli_videos/1x1_stimuli_{self.frequencies[0]}_{self.frequencies[1]}_{self.
            frequencies[2]}_{self.frequencies[3]}_{self.extension[1:]}_{self.date_time}"
        # create folder if it does not exist
        if not os.path.exists(folder):
            os.makedirs(folder)

        # create the order of trials
        order_settings = self.create_random_order_of_stimuli_settings_1x1()
        # for each combination of settings
        for combination in order_settings:
            # get the indices of the settings
            pixel_surface_index, color_mode_index, frequency_index, shape_index = combination
            # get the values of the settings
            pixel_surface = self.pixel_surface[pixel_surface_index]
            color_mode = self.colors[color_mode_index]
            frequency = self.frequencies[frequency_index]
            shape = self.shapes[shape_index]
```

```
# create the video name for the trial and photo name for inter trial
video_name_trial = f'{folder}/1x1_pixel_surface_' + str(pixel_surface) + "
    _color_mode_" + str(
    color_mode) + "_frequency_" + str(frequency) + "_shape_" + str(shape) + self.
        extension
video_name_inter_trial = f"{folder}/1x1_pixel_surface_" + str(pixel_surface) + "
    _color_mode_" + str(
    color_mode) + "_frequency_" + str(frequency) + "_shape_" + str(shape) + "
        _inter_trial" + self.extension
# get the values of the settings
pixel_surface = self.pixel_surface[pixel_surface_index]
color_mode = self.colors[color_mode_index]
frequency = self.frequencies[frequency_index]
shape = self.shapes[shape_index]

# create a list of frames for the trial and inter trial to save the frames
frame_list_trial = []
frame_list_inter_trial = []
# create the trial video
for frame_number in range(number_of_frames):
    # create a black image
    image = np.zeros(self.frame_size, np.uint8)
    # convert color to from BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # calculate the color of the screen
    color = self._calculate_screen_color_sinusoidal(
        frequency, frame_number)
    # create a color tuple
    if color_mode == 'red':
        # in BGR space
        color_tuple = (0, 0, color)
    elif color_mode == 'green':
        color_tuple = (0, color, 0)
    elif color_mode == 'white':
        color_tuple = (color, color, color)

    # draw the stimulus
    if shape == "circles":
        image = self._draw_circles(image, pixel_surface, [
                                   center_coordinate], [color_tuple])
    elif shape == "squares":
        image = self._draw_squares(image, pixel_surface, [
                                   center_coordinate], [color_tuple])
    elif shape == "triangles":
        image = self._draw_triangles(image, pixel_surface, [
                                     center_coordinate], [color_tuple])
    # append the frame to the list
    frame_list_trial.append(image)

# create the inter trial video (OPTIONAL). currently it is a png
for frame_number in range(1):
    # create a black image
    image = np.zeros(self.frame_size, np.uint8)
    # convert color to from BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
                # calculate the color of the screen
                color = self._calculate_screen_color_sinusoidal(frequency, 0)
                # create a color tuple
                if color_mode == 'red':
                    # in BGR space
                    color_tuple = (0, 0, color)
                elif color_mode == 'green':
                    color_tuple = (0, color, 0)
                elif color_mode == 'white':
                    color_tuple = (color, color, color)

                # draw the stimulus
                if shape == "circles":
                    image = self._draw_circles(image, pixel_surface, [
                                                center_coordinate], [color_tuple])
                elif shape == "squares":
                    image = self._draw_squares(image, pixel_surface, [
                                                center_coordinate], [color_tuple])
                elif shape == "triangles":
                    image = self._draw_triangles(image, pixel_surface, [
                                                  center_coordinate], [color_tuple])
                # append the frame to the list of inter trial
                frame_list_inter_trial.append(image)

            # create video writer for the trial
            self.video_writer = cv2.VideoWriter(
                video_name_trial, self.fourcc, self.frame_rate, (self.frame_size[1], self.
                    frame_size[0]))
            # write the frames to the video
            for frame in frame_list_trial:
                self.video_writer.write(frame)
            # close video writer
            self.video_writer.release()

            # replace video_name_inter_trial by .png to save the images
            photo_name_inter_trial = video_name_inter_trial.replace(
                self.extension, ".png")
            for frame in frame_list_inter_trial:
                # save frame as png
                cv2.imwrite(photo_name_inter_trial, frame)

    def create_random_order_3_neighbouring_frequencies(self, frequency: float, delta_frequency:
        float) -> list:
        """ create a list of 3 frequencies with 1 frequency being the same as the frequency
            argument and the other 2 frequencies being the frequency argument
        plus or minus delta_frequency. The list is shuffled.
        Args:
            frequency (float): frequency of the stimulus
            delta_frequency (float): difference between the frequencies

        Returns:
            list: list of 3 frequencies
        """
        # create a list of 3 frequencies
        frequency_list = [frequency-delta_frequency,
```

```
                frequency+delta_frequency, frequency+delta_frequency*2]
        # shuffle the list
        np.random.shuffle(frequency_list)
        return frequency_list

    def update_screen_by_frequency(self, image: np.ndarray, frequency: float, center_coordinate:
        tuple, color_mode: str, shape: str, pixel_surface: int, frame_number=0) -> np.ndarray:
        """ update the screen by the frequency. The frequency is used to calculate the color of the
            screen. The color of the screen is used to draw the stimulus on the screen.

        Args:
            image (np.ndarray): image to draw on
            frequency (float): frequency of the stimulus
            center_coordinate (tuple): center coordinate of the stimulus
            color_mode (str): color mode of the stimulus
            shape (str): shape of the stimulus
            pixel_surface (int): pixel surface of the stimulus
            frame_number (int, optional): frame number. Defaults to 0.

        Returns:
            np.ndarray: updated image
        """
        # calculate the color of the screen
        color = self._calculate_screen_color_sinusoidal(
            frequency, frame_number)

        # create a color tuple
        if color_mode == 'red':
            # in BGR space
            color_tuple = (0, 0, color)
        elif color_mode == 'green':
            color_tuple = (0, color, 0)
        elif color_mode == 'white':
            color_tuple = (color, color, color)

        # draw the stimulus
        if shape == 'circles':
            image = self._draw_circles(image, pixel_surface, [
                                       center_coordinate], [color_tuple])
        elif shape == 'squares':
            image = self._draw_squares(image, pixel_surface, [
                                       center_coordinate], [color_tuple])
        elif shape == 'triangles':
            image = self._draw_triangles(image, pixel_surface, [
                                         center_coordinate], [color_tuple])

        return image

    def create_video_2x2(self):
        """ create a video with 4 stimuli on the screen
        """
        # create a video with 4 stimuli on the screen
        length_of_video = 4  # seconds
        inter_trial_time = 2  # seconds
        # calculate the number of frames for the trial video and the inter trial video
        number_of_frames = length_of_video * self.frame_rate
```

```
number_of_frames_inter_trial = inter_trial_time * self.frame_rate # OPTIONAL
# define the radius of the fixation circle
self. fixation_circle_radius = 10
# define the delta frequency with which the frequencies are changed
delta_frequency = 0.3 # Hz
# create a older for the experiment videos
folder = f"stimuli_videos/2x2_stimuli_{self.frequencies[0]}_{self.frequencies[1]}_{self.
    frequencies[2]}_{self.frequencies[3]}_delta_{delta_frequency}_{self.extension[1:]}_{
    self.date_time}"
# create the folder if it does not exist
if not os.path.exists(folder):
    os.makedirs(folder)

# create a list of all possible combinations of the stimuli in a random order
order_settings = self.create_random_order_of_stimuli_settings_2x2()

# for each stimuli location
for stimuli_center_coordinate in self.center_coordinates:
    # for each combination of the stimuli settings
    for combination in order_settings:
        # extract the stimuli settings
        pixel_surface_index = combination[0]
        color_mode_index = combination[1]
        frequency_index = combination[2]
        shape_index = combination[3]

        # get the stimuli settings
        pixel_surface = self.pixel_surface[pixel_surface_index]
        color_mode = self.colors[color_mode_index]
        frequency = self.frequencies[frequency_index]
        shape = self.shapes[shape_index]

        # create a list of 3 frequencies
        frequency_list_neigbours = self.create_random_order_3_neighbouring_frequencies(
            frequency, delta_frequency)
        # create a list of all frequencies
        all_frequency_list = [frequency, frequency_list_neigbours[0],
                              frequency_list_neigbours[1], frequency_list_neigbours[2]]

        # get the other 3 stimuli center coordinates that are not the target stimulus
            center coordinate
        other_stimuli_center_coordinates = self.center_coordinates.copy()
        other_stimuli_center_coordinates.remove(
            stimuli_center_coordinate)

        # create video name for the trial video and the inter trial video
        video_name_trial = f"{folder}/2x2_pixel_surface_" + str(pixel_surface) + "
            _color_mode_" + str(color_mode) + "_frequency_" + str(
            frequency) + "_shape_" + str(shape) + "_coordinate_" + str(
                stimuli_center_coordinate) + self.extension
        video_name_inter_trial = f"{folder}/2x2_pixel_surface_" + str(pixel_surface) + "
            _color_mode_" + str(color_mode) + "_frequency_" + str(
            frequency) + "_shape_" + str(shape) + "_coordinate_" + str(
                stimuli_center_coordinate) + " _inter_trial " + self.extension

        # create a list of frames for the trial video and the inter trial video
```

```
 list_of_frames_inter_trial  = []
list_of_frames = []

# for the number of frames in the inter trial video. OPTIONAL. Currently it is a
    png image
for frame_i in range(1):
    # create a black rgb image
    image = np.zeros(
        (self.frame_size[0], self.frame_size[1], 3), np.uint8)
    # convert image from BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # add stimuli to image
    for i, frequency_i in enumerate(frequency_list_neigbours):
        # get the center coordinate of the stimulus
        center_coordinate = other_stimuli_center_coordinates[i]
        # add the stimulus to the image
        image = self.update_screen_by_frequency(
            image, frequency_i, center_coordinate, color_mode, shape,
                pixel_surface, 0)

    # add the target stimulus to the image
    image = self.update_screen_by_frequency(
        image, frequency, stimuli_center_coordinate, color_mode, shape,
            pixel_surface, 0)
    # add fixation circle
    image = cv2.circle(image, stimuli_center_coordinate,
                       self.fixation_circle_radius, (255, 0, 0), self.thickness)
    # add the image to the list of frames
    list_of_frames_inter_trial.append(image)

# for the number of frames in the trial video
for frame_number in range(number_of_frames):
    # create a black rgb image
    image = np.zeros(
        (self.frame_size[0], self.frame_size[1], 3), np.uint8)
    # convert image from BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # add neigbouring stimuli to image
    for i, frequency_i in enumerate(frequency_list_neigbours):
        # get the center coordinate of the stimulus
        center_coordinate = other_stimuli_center_coordinates[i]
        # add the stimulus to the image
        image = self.update_screen_by_frequency(
            image, frequency_i, center_coordinate, color_mode, shape,
                pixel_surface, frame_number)

    # add the target stimulus to the image
    image = self.update_screen_by_frequency(
        image, frequency, stimuli_center_coordinate, color_mode, shape,
            pixel_surface, frame_number)

    # save frames for video writer
    list_of_frames.append(image)

print(
```

```
            f"length of list_of_frames: {len(list_of_frames)} and length of
                list_of_frames_inter_trial : {len( list_of_frames_inter_trial )}")

        # create inter trial video. OPTIONAL. Currently, it is a png image
        photo_name_inter_trial = video_name_inter_trial.replace(
            self.extension, ".png")
        for frame in list_of_frames_inter_trial :
            # save frame as png
            cv2.imwrite(photo_name_inter_trial, frame)

        # create video for trial
        self.video_writer = cv2.VideoWriter(
            video_name_trial, self.fourcc, self.frame_rate, (self.frame_size[1], self.
                frame_size[0]))
        for frame in list_of_frames:
            self.video_writer.write(frame)
        # close video writer
        self.video_writer.release()


if __name__ == '__main__':
    # create interface
    interface = SSVEP_Interface()
    # create videos
    interface.create_video_1x1()
    interface.create_video_2x2()
```

D.2. Analyses

D.2.1. Requirements_analysis.txt

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: linux-64
_libgcc_mutex=0.1=main
_openmp_mutex=5.1=1_gnu
appdirs=1.4.4=pypi_0
bottleneck=1.3.4=py39hd257fcd_1
ca-certificates=2022.12.7=ha878542_0
certifi=2022.12.7=pyhd8ed1ab_0
charset-normalizer=3.0.1=pypi_0
contourpy=1.0.7=pypi_0
cycler=0.11.0=pypi_0
decorator=5.1.1=pypi_0
fonttools=4.38.0=pypi_0
idna=3.4=pypi_0
jinja2=3.1.2=pypi_0
kiwisolver=1.4.4=pypi_0
ld_impl_linux-64=2.38=h1181459_1
libblas=3.9.0=15_linux64_openblas
libcblas=3.9.0=15_linux64_openblas
libffi=3.3=he6710b0_2
libgcc-ng=11.2.0=h1234567_1
libgfortran-ng=12.2.0=h69a702a_19
libgfortran5=12.2.0=h337968e_19
libgomp=11.2.0=h1234567_1
```

```
liblapack=3.9.0=15_linux64_openblas
libopenblas=0.3.20=pthreads_h78a6416_0
libstdcxx-ng=11.2.0=h1234567_1
markupsafe=2.1.2=pypi_0
matplotlib=3.6.3=pypi_0
mne=1.2.3=pypi_0
ncurses=6.3=h5eee18b_3
nomkl=1.0=h5ca1d4c_0
numexpr=2.8.0=py39h194a79d_102
numpy=1.24.1=pypi_0
openssl=1.1.1o=h166bdaf_0
packaging=23.0=pypi_0
pandas=1.5.1=py39h417a72b_0
pillow=9.4.0=pypi_0
pip=22.3.1=py39h06a4308_0
pooch=1.6.0=pypi_0
pyparsing=3.0.9=pypi_0
python=3.9.0=hdb3f193_2
python-dateutil=2.8.2=pyhd8ed1ab_0
python_abi=3.9=2_cp39
pytz=2022.7.1=pyhd8ed1ab_0
readline=8.2=h5eee18b_0
requests=2.28.2=pypi_0
scipy=1.10.0=pypi_0
setuptools=65.6.3=py39h06a4308_0
six=1.16.0=pyh6c4a22f_0
sqlite=3.40.1=h5082296_0
tk=8.6.12=h1ccaba5_0
tqdm=4.64.1=pypi_0
tzdata=2022g=h04d1e81_0
urllib3=1.26.14=pypi_0
wheel=0.37.1=pyhd3eb1b0_0
xz=5.2.10=h5eee18b_1
zlib=1.2.13=h5eee18b_0
```

D.2.2. process_eeg_and_eye_tracking_v1.py

Used to process the EEG and eye-tracking data before the statistical analyses.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import cv2
import os
from math import pi
import mne
from proces_eyes import process_eyes
import copy
from typing import Union


def weighted_average_interpolation_snr(desired_freq: float, freqs: np.ndarray, snrs: np.ndarray) ->
    np.ndarray:
    """ Calculate the weighted average of the SNR values of the desired frequency because it is
        between two frequencies in the spectrum.

    Args:
```

```
        desired_freq (float): the desired frequency
        freqs (np.ndarray): arrayt with the frequencies
        snrs (np.ndarray): array of the same dimensions as freqs with the SNR values across all
            the channels

    Returns:
        np.ndarray: the weighted average of the SNR values of the desired frequency across all the
            channels
    """
    # find bin with closest frequency to desired frequency
    i_bin = np.argmin(np.abs(freqs - desired_freq))
    # calculate the error between the desired frequency and the frequency in the bin
    error = desired_freq - freqs[i_bin]
    # if the desired frequency is in the bin, return the SNR value and perform no interpolation
    if error == 0:
        # if the desired frequency is in the bin, return the SNR value
        print(
            f" No interpolation needed, desired frequency {desired_freq} is in bin {i_bin} with
                frequency {freqs[i_bin]}")
        return snrs[0, :, i_bin]
    else:
        # if the desired frequency is not in the bin, calculate the weighted average of the SNR
            values of the two frequencies in the bin
        print(
            f" Interpolation needed, desired frequency {desired_freq} is not in bin {i_bin} with
                frequency {freqs[i_bin]}")

        # calculate the weight of the two frequencies in the bin
        if error > 0:
            # eror is positive, desired frequency is higher than the frequency in the bin
            print(
                f" Error is positive, desired frequency {desired_freq} is higher than frequency {
                    freqs[i_bin]}")
            # upper_bin
            upper_bin = i_bin + 1
            # lower_bin
            lower_bin = i_bin
            # measure the difference between the desired frequency and the two frequencies in the
                bin
            difference_with_lower_bin = freqs[lower_bin] - desired_freq
            difference_with_upper_bin = desired_freq - freqs[upper_bin]
            # calculate the weight of the two frequencies in the bin
            weight_upper_bin = abs(difference_with_lower_bin) / (
                abs(difference_with_lower_bin) + abs(difference_with_upper_bin))
            weight_lower_bin = abs(difference_with_upper_bin) / (
                abs(difference_with_lower_bin) + abs(difference_with_upper_bin))
            print(
                f" Weight of frequency {freqs[upper_bin]} is {weight_upper_bin} and weight of
                    frequency {freqs[lower_bin]} is {weight_lower_bin}")
            # check if the sum of the weights is 1
            assert (weight_lower_bin+weight_upper_bin == 1)

        else:
            print(
                f"Error is negative, desired frequency {desired_freq} is lower than frequency {
                    freqs[i_bin]}")
```

```
            # upper_bin
            upper_bin = i_bin
            # lower_bin
            lower_bin = i_bin - 1
            # measure the difference between the desired frequency and the two frequencies in the
                bin
            difference_with_lower_bin = freqs[lower_bin] - desired_freq
            difference_with_upper_bin = desired_freq - freqs[upper_bin]
            # calculate the weight of the two frequencies in the bin
            weight_upper_bin = abs(difference_with_lower_bin) / (
                abs(difference_with_lower_bin) + abs(difference_with_upper_bin))
            weight_lower_bin = abs(difference_with_upper_bin) / (
                abs(difference_with_lower_bin) + abs(difference_with_upper_bin))
            print(
                f" Weight of frequency {freqs[upper_bin]} is {weight_upper_bin} and weight of
                    frequency {freqs[lower_bin]} is {weight_lower_bin}")
            # check if the sum of the weights is 1
            assert (weight_lower_bin+weight_upper_bin == 1)

        # calculate new SNR value
        new_snr = (snrs[0, :, upper_bin] * weight_upper_bin) + \
            (snrs[0, :, lower_bin] * weight_lower_bin)
        print(f" New SNR value is {new_snr}")

        return new_snr


def calculate_statistics(data: np.ndarray) -> Union[float, float, float, float, float, float]:
    """ Calculate the mean, average, standard deviation, variance, minimum and maximum of the
        data.

    Args:
        data (np.ndarray): The data to calculate the statistics for.

    Returns:
        tuple: The mean, average, standard deviation, variance, minimum and maximum of the data.

    """
    # calculate the mean, average, standard deviation, variance, minimum and maximum of the
        data
    data_mean = np.mean(data)
    data_average = np.average(data)
    data_std = np.std(data)
    data_variance = np.var(data)
    data_min = np.min(data)
    data_max = np.max(data)

    return data_mean, data_average, data_std, data_variance, data_min, data_max

# ------------------------- EEG data ------------------------- #


def generate_paths(path: str) -> Union[str, str, str, str]:
    """ Generate the paths to the .eeg, .vhdr, .vmrk and .txt files.
```

```
    Args:
        path (str): The path to the directory containing the files.

    Returns:
        tuple: The paths to the .eeg, .vhdr, .vmrk and .txt files.
    """
    eeg_path, vhdr_path, vmrk_path, txt_path = None, None, None, None
    # list all files in the directory
    files = os.listdir(path)
    # find paths to the .eeg, .vhdr, .vmrk and .txt files
    for file in files:
        if file.endswith('.eeg'):
            eeg_path = os.path.join(path, file)
        elif file.endswith('.vhdr'):
            vhdr_path = os.path.join(path, file)
        elif file.endswith('.vmrk'):
            vmrk_path = os.path.join(path, file)
        elif file.endswith('.txt'):
            txt_path = os.path.join(path, file)

    return eeg_path, vhdr_path, vmrk_path, txt_path


def snr_spectrum(psd: np.ndarray, noise_n_neighbor_freqs=1, noise_skip_neighbor_freqs=1) -> np.
    ndarray:
    """Compute SNR spectrum from PSD spectrum using convolution.

    Parameters
    ----------
    psd : ndarray, shape ([n_trials, n_channels,] n_frequency_bins)
        Data object containing PSD values. Works with arrays as produced by
        MNE's PSD functions or channel/trial subsets.
    noise_n_neighbor_freqs : int
        Number of neighboring frequencies used to compute noise level.
        increment by one to add one frequency bin ON BOTH SIDES
    noise_skip_neighbor_freqs : int
        set this >=1 if you want to exclude the immediately neighboring
        frequency bins in noise level calculation

    Returns
    -------
    snr : ndarray, shape ([n_trials, n_channels,] n_frequency_bins)
        Array containing SNR for all epochs, channels, frequency bins.
        NaN for frequencies on the edges, that do not have enough neighbors on
        one side to calculate SNR.
    """
    # Construct a kernel that calculates the mean of the neighboring
    # frequencies
    averaging_kernel = np.concatenate((
        np.ones(noise_n_neighbor_freqs),
        np.zeros(2 * noise_skip_neighbor_freqs + 1),
        np.ones(noise_n_neighbor_freqs)))
    averaging_kernel /= averaging_kernel.sum()

    # Calculate the mean of the neighboring frequencies by convolving with the
    # averaging kernel.
```

```
    mean_noise = np.apply_along_axis(
        lambda psd_: np.convolve(psd_, averaging_kernel, mode='valid'),
        axis=-1, arr=psd
    )

    # The mean is not defined on the edges so we will pad it with nas. The
    # padding needs to be done for the last dimension only so we set it to
    # (0, 0) for the other ones.
    edge_width = noise_n_neighbor_freqs + noise_skip_neighbor_freqs
    pad_width = [(0, 0)] * (mean_noise.ndim - 1) + [(edge_width, edge_width)]
    mean_noise = np.pad(
        mean_noise, pad_width=pad_width, constant_values=np.nan
    )

    return psd / mean_noise


def load_and_setup_eeg_data(eeg_path: str, vhdr_path: str, vmrk_path: str, txt_path: str) -> Union
    [pd.DataFrame, np.ndarray, np.ndarray, int, str,
                                                                                    mne
                                                                                    .
                                                                                    io
                                                                                    .
                                                                                    brainvision
                                                                                    .
                                                                                    brainvision
                                                                                    .
                                                                                    RawBrainVis
                                                                                    ,
                                                                                    np
                                                                                    .
                                                                                    ndarray
                                                                                    ,
                                                                                    float
                                                                                    ,
                                                                                    list
                                                                                    ,
                                                                                    float
                                                                                    ,
                                                                                    float
                                                                                    ]:
    """Load and setup the EEG data.

    eeg_path (str): The path to the .eeg file.
    vhdr_path (str): The path to the .vhdr file.
    vmrk_path (str): The path to the .vmrk file.
    txt_path (str): The path to the .txt file.

    Returns:
        tuple: The data, sampling frequency, channel names, raw data, data mean, data average,
```

```
            data standard deviation, data variance, data minimum and data maximum.

    """
    # generate paths to the eeg, vhdr, vmrk and txt file
    df_txt = pd.read_csv(txt_path, sep='\t')

    # read the data using mne
    raw = mne.io.read_raw_brainvision(vhdr_path, preload=True)
    # get the data
    data = raw.get_data()
    # get the sampling frequency
    sfreq = raw.info['sfreq']
    # get the channel names
    ch_names = raw.info['ch_names']

    channels_to_safe = ['O1', 'O2']
    # drop all channels except O1, 02
    channels_to_drop = [ch for ch in ch_names if ch not in channels_to_safe]

    # get the index of the channels to drop
    channels_to_drop_idx = [ch_names.index(ch) for ch in channels_to_drop]

    # drop the channels
    raw.drop_channels(channels_to_drop)
    # drop the channels from the channel names
    ch_names = [ch for ch in ch_names if ch not in channels_to_drop]

    # convert the list of trials to a mne epochs object
    all_events, _ = mne.events_from_annotations(raw, verbose=False)

    # create a mne epochs object from the data and the event
    tmin = 0.5  # start of each epoch (500ms after the trigger)
    tmax = 4    # end of each epoch (4000ms after the trigger)

    # number of events
    if '1X1' in txt_path:
        print(f"experiment 1X1")
        trials_events = np.where(all_events[:, 2] == 2)[0]
    else:
        print(f"experiment 2X2")
        trials_events = np.where(all_events[:, 2] == 4)[0]
    # number of trials
    number_of_trials = len(trials_events)

    return df_txt, all_events, trials_events, number_of_trials, txt_path, raw, data, sfreq,
        ch_names, tmin, tmax


def process_eeg_trial_data(trial_index: int, df_txt: pd.DataFrame, all_events: np.ndarray,
    trials_events: np.ndarray, txt_path: str,
                               raw: mne.io.brainvision.brainvision.RawBrainVision, sfre: float,
                                   ch_names: list, tmin: float, tmax: float) -> \
        Union[np.ndarray, float]:
    """ Process the EEG trial data.

    trial_index (int): The index of the trial.
```

```
    df_txt (pd.DataFrame): The dataframe containing the txt file.
    all_events (np.ndarray): The array containing all events.
    trials_events (np.ndarray): The array containing the trial events.
    txt_path (str): The path to the txt file.
    raw (mne.io.brainvision.brainvision.RawBrainVision): The raw data.
    sfre (float): The sampling frequency.
    ch_names (list): The list of channel names.
    tmin (float): The start of each epoch (500ms after the trigger).
    tmax (float): The end of each epoch (4000ms after the trigger).

    Returns:
        tuple:
        max_snr_interval (np.ndarray): The max snr interval.
        max_snr_interval_frequeny (float): The max snr interval frequency.
    """

    # extract the trial events
    trial_events = [all_events[trials_events[trial_index]]]
    # extract the ith row of the txt_df
    txt_row = df_txt.iloc[trial_index]

    # check if the experiment is 1X1 or 2X2
    if '1X1' in txt_path:
        # get the stimulus frequency
        stimulus_freq = txt_row['frequency']
        # pixel surface
        pixel_surface = txt_row['pixel_surface']
        # shape of the stimulus
        shape = txt_row['shape']
        # color of the stimulus
        color = txt_row['color_mode']
        # name video clip
        video_clip = txt_row['video_clip']
        # block number
        block_number = txt_row['Block_variable']
        # trial number
        trial_number = trial_index
    else:
        # get the stimulus frequency
        stimulus_freq = txt_row['frequency_2']
        # pixel surface
        pixel_surface = txt_row['pixel_surface_2']
        # shape of the stimulus
        shape = txt_row['shape_2']
        # color of the stimulus
        color = txt_row['color_mode_2']
        # name video clip
        video_clip = txt_row['video_clip_2']
        # block number
        block_number = txt_row['Block_variable']
        # trial number
        trial_number = trial_index

    # check if the experiment is 1X1 or 2X2
    if '1X1' in txt_path:
        print(f" experiment 1x1")
```

```
        # Construct epochs
        event_id = {
            'Stimulus/S  2': 2
        }
        baseline = None
        epochs = mne.Epochs(
            raw, events=trial_events,
            event_id=[event_id['Stimulus/S  2']],  tmin=tmin,
            tmax=tmax, baseline=baseline, verbose=False)
    else:
        print(f" experiment 2x2")
        # Construct epochs
        event_id = {
            'Stimulus/S  4': 4
        }
        baseline = None

        epochs = mne.Epochs(
            raw, events=trial_events,
            event_id=[event_id['Stimulus/S  4']],  tmin=tmin,
            tmax=tmax, baseline=baseline, verbose=False)

    # get the data from the epochs from time tmin to tmax and frequency fmin to fmax
    tmin = 0.5
    tmax = 4.
    fmin = 1.
    fmax = 90.

    # get the sampling frequency
    sfreq = epochs.info['sfreq']
    # calculate teh EpochsSpectrum output using the welch method to calculate the psd across the
        entire epochs. Zero overlap between segments.
    spectrum = epochs.compute_psd('welch', n_fft=int(sfreq * (tmax - tmin)), n_overlap=0,
                                  n_per_seg=None, tmin=tmin, tmax=tmax, fmin=fmin, fmax=fmax,
                                      window='boxcar', verbose=False)

    # calculate teh power spectrum density (psd) spectrum
    spectrum = epochs.compute_psd(
        'welch',
        n_fft=int(sfreq * (tmax - tmin)),
        n_overlap=0, n_per_seg=None,
        tmin=tmin, tmax=tmax,
        fmin=fmin, fmax=fmax,
        window='boxcar',
        verbose=False)

    # extract the psd and the frequencies
    psds, freqs = spectrum.get_data(return_freqs=True)

    # calculate the snr spectrum
    snrs = snr_spectrum(psds, noise_n_neighbor_freqs=3,
                        noise_skip_neighbor_freqs=1)

    # Plot the SNR spectrum
    fig, axes = plt.subplots(2, 1, sharex='all', sharey='none', figsize=(8, 5))
    freq_range = range(np.where(np.floor(freqs) == 1.) [0][0],
```

```
                    np.where(np.ceil(freqs) == fmax - 1)[0][0])

psds_plot = 10 * np.log10(psds)
psds_mean = psds_plot.mean(axis=(0, 1))[freq_range]
psds_std = psds_plot.std(axis=(0, 1))[freq_range]

# set the main tile of the figure
fig.suptitle(f'PSD and SNR spectrum for {video_clip}', fontsize=16)
# draw vertical line at stimulus frequency
axes[0].axvline(stimulus_freq, color='k', linestyle='--',
                label='stimulus frequency')
axes[1].axvline(stimulus_freq, color='k', linestyle='--',
                label='stimulus frequency')

axes[0].plot(freqs[freq_range], psds_mean, color='b')
axes[0].fill_between(
    freqs[freq_range], psds_mean - psds_std, psds_mean + psds_std,
    color='b', alpha=.2)
axes[0].set(title="PSD spectrum", ylabel='Power Spectral Density [dB]')

# SNR spectrum
snr_mean = snrs.mean(axis=(0, 1))[freq_range]
snr_std = snrs.std(axis=(0, 1))[freq_range]

axes[1].plot(freqs[freq_range], snr_mean, color='r')
axes[1].fill_between(
    freqs[freq_range], snr_mean - snr_std, snr_mean + snr_std,
    color='r', alpha=.2)
axes[1].set(
    title="SNR spectrum", xlabel='Frequency [Hz]',
    ylabel='SNR', ylim=[-2, 30], xlim=[fmin, fmax])

# save fig
video_clip = video_clip.replace('.mp4', '')
# create a directory to save the figures in the same directory as the txt file
directory = os.path.dirname(txt_path)
# create a directory to save the figures
if not os.path.exists(os.path.join(directory, 'figures')):
    os.makedirs(os.path.join(directory, 'figures'), exist_ok=True)
# save the figure
fig.savefig(os.path.join(directory, 'figures',
            f'{video_clip}_{block_number}_{trial_number}_dual_channel.png'))
print(f'Figure saved for {video_clip}')

# calculate the snr at the stimulus frequency
snr_at_stimulus_freq = weighted_average_interpolation_snr(
    stimulus_freq, freqs, snrs)
print(f'SNR at stimulus frequency: {snr_at_stimulus_freq}')

# set up the frequency interval to search for the snr value at the stimulus frequency
upper_limit_freq = stimulus_freq + 0.15
lower_limit_freq = stimulus_freq - 0.15
# calculate the snr at the upper and lower limit frequencies of the interval with interpolation
snr_at_upper_limit_freq = weighted_average_interpolation_snr(
    upper_limit_freq, freqs, snrs)
snr_at_lower_limit_freq = weighted_average_interpolation_snr(
```

```
        lower_limit_freq, freqs, snrs)
    # combine the snr values at the upper and lower limit frequencies with the snr value at the
        stimulus frequency
    snr_values_interval = np.array(
        [snr_at_lower_limit_freq, snr_at_stimulus_freq, snr_at_upper_limit_freq])
    frequency_interval = np.array(
        [lower_limit_freq, stimulus_freq, upper_limit_freq])

    # calculate the mean snr value for each channel
    mean_snr_values_channels = snr_values_interval.mean(axis=1)
    # get the maximal snr value
    max_snr_interval = mean_snr_values_channels.max()
    # get index of the channel with the maximal snr
    index_maximal_snr = np.argmax(mean_snr_values_channels)
    # get the frequency with the maximal snr
    max_snr_interval_frequency = frequency_interval[index_maximal_snr]

    print(
        f'Maximal SNR: {max_snr_interval} at frequency: {max_snr_interval_frequency}')

    return max_snr_interval, max_snr_interval_frequency


# ------------------------- Eye tracking data ------------------------- #
def process_eye_tracking_data(eye_tracking_path: str):
    """ Process the eye tracking data from the eye tracking software

    Args:
        eye_tracking_path (str): path to the eye tracking data
    """
    # load the data with utf-8 encoding
    data = pd.read_csv(eye_tracking_path, encoding='utf-16', delimiter='\t')

    # rename the columns
    # TIMESTAMP = time
    # LEFT_GAZE_X = left_gaze_x
    # LEFT_GAZE_Y = left_gaze_y
    # RIGHT_GAZE_X = right_gaze_x
    # RIGHT_GAZE_Y = right_gaze_y
    # LEFT_PUPIL_SIZE = left_p
    # RIGHT_PUPIL_SIZE = right_p
    # renamce the columns
    data.rename(columns={'TIMESTAMP': 'time', 'LEFT_GAZE_X': 'left_x', 'LEFT_GAZE_Y': 'left_y',
        'RIGHT_GAZE_X': 'right_x',
                'RIGHT_GAZE_Y': 'right_y', 'LEFT_PUPIL_SIZE': 'left_p', 'RIGHT_PUPIL_SIZE': '
                    right_p'}, inplace=True)
    # seperaate the data into trials by the trial number so that the trial_index can be used to
        access the data under the trial
    # get the trial numbers
    trial_numbers = data['TRIAL_INDEX'].unique()
    # get the data for each trial
    trial_data = []
    for trial in trial_numbers:
        data_per_trial = data[data['TRIAL_INDEX'] == trial]
        # remove data where the VIDEO_NAME is '.'
        data_per_trial = data_per_trial[data_per_trial['VIDEO_NAME'] != '.']
```

```
            # add empty columns for the average gaze and pupil size
            data_per_trial['average_p'] = np.nan
            # make a deep copy of the data
            data_per_trial = copy.deepcopy(data_per_trial)
            trial_data.append(data_per_trial)
        try:
            t, xf, yf, pf, fixations, saccades, fd, sl, ff, sf, sa, fa = process_eyes(
                trial_data)
        except:
            print('Error processing eye tracking data')
            t, xf, yf, pf, fixations, saccades, fd, sl, ff, sf, sa, fa = np.nan, np.nan, np.nan, np.
                nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan
        # create a new dataframe using the same columns as the variables t, xf, yf, pf, fixations,
            saccades, fd, sl, ff, sf, sa, fa
        new_data = pd.DataFrame(columns=[
                                    't', 'xf', 'yf', 'pf', 'fixations', 'saccades', 'fd', 'sl', 'ff', 'sf',
                                        'sa', 'fa'])
        # add the data to the new dataframe
        new_data['t'] = t
        new_data['xf'] = xf
        new_data['yf'] = yf
        new_data['pf'] = pf
        new_data['fixations'] = fixations
        new_data['saccades'] = saccades
        new_data['fd'] = fd
        new_data['sl'] = sl
        new_data['ff'] = ff
        new_data['sf'] = sf
        new_data['sa'] = sa
        new_data['fa'] = fa

        # save the data at the same path folder as the csv file
        new_data.to_csv(eye_tracking_path[:-4] +
                        '_processed_fixations.csv', index=False)
        print(
            f"Saved the processed data to {eye_tracking_path[:-4]}_processed_fixations.csv")


def _draw_circles(image: np.ndarray, pixel_surface: int, center_coordinates: np.ndarray,
        color_tuples: list) -> np.ndarray:
    """ Draw circles on the image

    Args:
        image (np.ndarray): the image on which the circles are drawn
        pixel_surface (int): the surface of the circles in pixels
        center_coordinates (np.ndarray): the center coordinates of the circles to be drawn in
            pixels
        color_tuples (list): the color of the circles to be drawn

    Returns:
        np.ndarray: the image with the circles drawn on it
    """
    # calculate radius of circle
    radius = int(np.sqrt(pixel_surface/pi))
    # draw circles
    for i, center_coordinate in enumerate(center_coordinates):
```

```
            # draw all circles.
            color_tuple = color_tuples[i]
            # draw the circle
            image = cv2.circle(image, center_coordinate, radius, color_tuple, -1)

    return image


def _draw_squares(image: np.ndarray, pixel_surface: int, center_coordinates: np.ndarray,
    color_tuples: list) -> np.ndarray:
    """ Draw squares on the image

    Args:
        image (np.ndarray): the image on which the squares are drawn
        pixel_surface (int): the surface of the squares in pixels
        center_coordinates (np.ndarray): the center coordinates of the squares to be drawn in
            pixels
        color_tuples (list): the color of the squares to be drawn

    Returns:
        np.ndarray: the image with the squares drawn on it
    """

    # calculate side length of square
    side_length = int(np.sqrt(pixel_surface))
    # create squares
    for i, center_coordinate in enumerate(center_coordinates):
        # draw all squares.
        color_tuple = color_tuples[i]
        # calculate the coordinates of the top left and bottom right corners of the square
        coordinate_1 = (int(
            center_coordinate[0]-side_length//2), int(center_coordinate[1]-side_length//2))
        coordinate_2 = (int(
            center_coordinate[0]+side_length//2), int(center_coordinate[1]+side_length//2))
        # draw the square
        image = cv2.rectangle(image, coordinate_1,
                              coordinate_2, color_tuple, -1)

    return image


def _draw_triangles(image: np.ndarray, pixel_surface: int, center_coordinates: list, color_tuples:
    list) -> np.ndarray:
    """ Draw triangles on the image

    Args:
        image (np.ndarray): the image on which the triangles are drawn
        pixel_surface (int): the surface of the triangles in pixels
        center_coordinates (np.ndarray): the center coordinates of the triangles to be drawn in
            pixels
        color_tuples (list): the color of the triangles to be drawn

    Returns:
        np.ndarray: the image with the triangles drawn on it
    """
    # us the pixel surface to derive corner coordinates of the iscoceles triangle. All sides of the
```

```
            triangle are equal
        # calculate side length of triangle
        diagonal = np.sqrt(pixel_surface*4/np.sqrt(3))
        # half base length
        half_base_length = diagonal/2
        height = diagonal/2*np.sqrt(3)

        # create triangles
        for i, center_coordinate in enumerate(center_coordinates):
            # draw all triangles. The coordinates are measured in integer values, so the triangles are
                not perfectly centered
            color_tuple = color_tuples[i]
            coordinate_1 = (
                int(center_coordinate[0]-half_base_length), int(center_coordinate[1]+height//2))
            coordinate_2 = (
                int(center_coordinate[0]+half_base_length), int(center_coordinate[1]+height//2))
            coordinate_3 = (int(center_coordinate[0]), int(
                center_coordinate[1]-height//2))
            triangle_center_coordinate = (int(center_coordinate[0]), int(
                (coordinate_1[1]+coordinate_2[1]+coordinate_3[1])/3))
            # correct the 3 coordinates down to make center_coordinate equal to the center coordinate
                of the triangle
            if triangle_center_coordinate[1] > center_coordinate[1]:
                correction = center_coordinate[1] - triangle_center_coordinate[1]
                coordinate_1 = (coordinate_1[0], coordinate_1[1]+correction)
                coordinate_2 = (coordinate_2[0], coordinate_2[1]+correction)
                coordinate_3 = (coordinate_3[0], coordinate_3[1]+correction)

            elif triangle_center_coordinate[1] < center_coordinate[1]:
                correction = triangle_center_coordinate[1] - center_coordinate[1]
                coordinate_1 = (coordinate_1[0], coordinate_1[1]-correction)
                coordinate_2 = (coordinate_2[0], coordinate_2[1]-correction)
                coordinate_3 = (coordinate_3[0], coordinate_3[1]-correction)

            # draw the triangle
            triangle_cnt = np.array([coordinate_1, coordinate_2, coordinate_3])
            image = cv2.drawContours(image, [triangle_cnt], 0, color_tuple, -1)
        return image


def draw_shape(shape: str, pixel_surface: int, center_coordinates: np.ndarray, color_tuples: list)
    -> np.ndarray:
    """ Draw shapes on the image

    Args:
        shape (str): the shape to be drawn. Must be 'circle', 'square' or 'triangle'
        pixel_surface (int): the surface of the shapes in pixels
        center_coordinates (np.ndarray): the center coordinates of the shapes to be drawn in pixels
        color_tuples (list): the color of the shapes to be drawn

    Returns:
        np.ndarray: the image with the shapes drawn on it
    """
    # create empty image
    image = np.zeros((1080, 1920, 3), np.uint8)
    # draw the shapes
```

```
    if '.mp4' in shape:
        shape = shape[:-4]
    if shape == "circles":
        image = _draw_circles(image, pixel_surface,
                              center_coordinates, color_tuples)
    elif shape == "squares":
        image = _draw_squares(image, pixel_surface,
                              center_coordinates, color_tuples)
    elif shape == "triangles":
        image = _draw_triangles(image, pixel_surface,
                                center_coordinates, color_tuples)
    else:
        raise ValueError("shape must be 'circle', 'square' or 'triangle'")

    return image


def draw_points(img: np.ndarray, x: int, y: int, gaze_x: np.ndarray, gaze_y: np.ndarray) -> Union[
    np.ndarray, list]:
    """ Draw the gaze points on the image

    Args:
        img (np.ndarray): the image on which the gaze points are drawn
        x (int): the x coordinate of the center of the shape
        y (int): the y coordinate of the center of the shape
        gaze_x (np.ndarray): the x coordinates of the gaze points
        gaze_y (np.ndarray): the y coordinates of the gaze points

    Returns:
        Union[np.ndarray, list]: the image with the gaze points drawn on it and the points on the
            shape
    """
    # create a mask with the shape
    mask = cv2.inRange(img, (0, 0, 0), (0, 0, 0))
    # flip the mask to find the shape
    mask = cv2.bitwise_not(mask)
    # find the shape
    points_shape_mask = np.where(mask == 255)

    # draw a circle at x, y with radius 5 and color red
    cv2.circle(img, (x, y), 5, (0, 0, 0), -1)
    # draw all the gaze points

    points_on_target = []
    for index in range(len(gaze_x)):

        int_x = int(round(gaze_x[index]))
        int_y = int(round(gaze_y[index]))
        # if mask is 255, the point is on the shape
        try:
            if mask[int_y, int_x] == 255:
                points_on_target.append((int_x, int_y))
        except IndexError:
            print(f"Point {int_x, int_y} is not in the image")
        cv2.circle(img, (int_x, int_y), 1, (255, 0, 0), -1)
```

```
    return img, points_on_target


def create_heatmap(x: int, y: int, gaze_x: np.ndarray, gaze_y: np.ndarray) -> np.ndarray:
    """ Create a heatmap of the gaze points

    Args:
        x (int): the x coordinate of the center of the shape
        y (int): the y coordinate of the center of the shape
        gaze_x (np.ndarray): the x coordinates of the gaze points
        gaze_y (np.ndarray): the y coordinates of the gaze points

    Returns:
        np.ndarray: the heatmap of the gaze points
    """
    # create empty image of size 1920 x 1080
    img = np.zeros((1080, 1920, 3), np.uint8)
    # draw a circle at x, y with radius 5 and color red
    cv2.circle(img, (x, y), 5, (0, 0, 255), -1)
    # generate set of points convering the drawn circle at coordinate x, y
    # create a mask of the circle
    mask = np.zeros((1080, 1920), np.uint8)
    cv2.circle(mask, (x, y), 5, (255, 255, 255), -1)
    # find the coordinates of the points in the mask
    points_circle = np.where(mask == 255)
    # convert the points to a list of tuples
    points_circle = list(zip(points_circle[0], points_circle[1]))

    # convert gaze_x and gaze_y to one dimensional arrays instead of a list of arrays
    gaze_x = np.concatenate(gaze_x)
    gaze_y = np.concatenate(gaze_y)

    print(f"Creating int points")
    # draw all the gaze points
    points = []
    for index in range(len(gaze_x)):
        x_coordinate = float(gaze_x[index])
        y_coordinate = float(gaze_y[index])
        int_x = int(round(x_coordinate))
        int_y = int(round(y_coordinate))
        points.append((int_y, int_x))

    print(f"Done creating int points")
    point = np.array(points)
    # find number of occurences of each point
    unique, counts = np.unique(point, axis=0, return_counts=True)

    maximal_value = np.max(counts)
    # normalize the counts to be between 0 and 255
    counts = (counts/maximal_value)*255

    # draw the points with the number of occurences as the color
    for index in range(len(unique)):
        y_coordinate, x_coordinate = unique[index]

        if tuple(unique[index]) in points_circle:
```

```
            color = (0, int(counts[index]), 0)
        else:
            color = (int(counts[index]), int(
                counts[index]), int(counts[index]))
        cv2.circle(img, (x_coordinate, y_coordinate), 1, color, -1)

    return img


def process_2x2_data(data_eye_tracking: pd.DataFrame, path: str, path_image_dir: str,
    data_trial_sequence: pd.DataFrame, folder_path: str):
    """ Process the data from the 2x2 experiment

    Args:
        data_eye_tracking (pd.DataFrame): data from the eye tracker
        path (str): path to the eye tracking data
        path_image_dir (str): path to the directory with the images
        data_trial_sequence (pd.DataFrame): data from the trial sequence (Expereriment X.txt/csv)
        folder_path (str): path to the main folder of the experiment
    """
    # generate paths to the files
    eeg_path, vhdr_path, vmrk_path, txt_path = generate_paths(folder_path)
    # load the eeg data
    df_txt, all_events, trials_events, num_of_trials, txt_path, raw, data, sfreq, ch_names, tmin,
        tmax = load_and_setup_eeg_data(
        txt_path, vhdr_path, vmrk_path, txt_path)
    # get and process the data from the eye tracker
    process_eye_tracking_data(path)
    # extract the headers of the data_eye_tracking
    headers = data_eye_tracking.columns.values

    # correct the trial index
    if data_eye_tracking["TRIAL_INDEX"].iloc[0] > 1:
        difference_trial_index = data_eye_tracking["TRIAL_INDEX"].iloc[0] - 1
        # correct the trial index in the data_eye_tracking
        data_eye_tracking["TRIAL_INDEX"] = data_eye_tracking["TRIAL_INDEX"] - \
            difference_trial_index
    # get the unique trial indices
    trial_indices = data_eye_tracking['TRIAL_INDEX']
    # get the unique trial indices
    trial_indices_unique = np.unique(trial_indices)
    # add trial_index column to data_trial_sequence
    data_trial_sequence['TRIAL_INDEX'] = data_trial_sequence.index+1
    # extract the headers of the data_trial_sequence
    headers_trial_sequence = data_trial_sequence.columns.values

    # get name of header with video_clip in it
    video_clip_header = [
        header for header in headers_trial_sequence if 'video_clip' in header][0]

    # create empty data frame to store the processed data
    headers_processed_data = ["TRIAL_INDEX", 'VIDEO_NAME', "BLOCK", 'PIXEL_SURFACE', "
        COLOR", "SHAPE", "FREQUENCY", "DISPLAYED_FRAMES", "DROPPED_FRAMES", "
        MAX_SNR", 'FREQUENCY_SAMPLED_AT', 'MEAN_GAZE_DISTANCE', '
        AVERAGE_GAZE_DISTANCE',
                                  'VARIANCE_GAZE_DISTANCE', 'MEAN_GAZE_ANGLE', '
```

```
                                AVERAGE_GAZE_ANGLE', 'VARIANCE_GAZE_ANGLE', "
                                MEAN_X", "X_AVERAGE", "X", "X_VARIANCE", "Y_MEAN", "
                                Y_AVERAGE", "Y", "Y_VARIANCE", "
                                NUMBER_OF_GAZE_POINTS", "POINTS_ON_TARGET"]
processed_data = pd.DataFrame(columns=headers_processed_data)
# create empty lists to store the gaze data
all_gaze_x = []
all_gaze_y = []

# get rid of the data_eye_tracking['VIDEO_NAME'] is '.'
data_eye_tracking = data_eye_tracking[data_eye_tracking['VIDEO_NAME'] != '.']

# loop through all the videos
experimental_counter = 0
counter = 0
for trial_index in trial_indices_unique:
    # Extract max SNR
     trial_index_for_list = trial_index - 1
    # block variable
    block = data_trial_sequence.iloc[ trial_index_for_list ][ 'Block_variable']
    # get the number of displayed frames
    displayed_frames = data_trial_sequence.iloc[ trial_index_for_list ][ 'displayed_frame_count']
    # get the number of dropped frames
    dropped_frames = data_trial_sequence.iloc[trial_index_for_list ][ 'dropped_frame_count']
    max_snr, frequency_sampled_at = process_eeg_trial_data(
          trial_index_for_list , df_txt, all_events, trials_events, txt_path, raw, sfreq,
              ch_names, tmin, tmax)

    # extract the video name from the data_trial_sequence
    print(f"extracting video name for trial index {trial_index}")
    # check if name available in data_eye_tracking
    if '2x2' in data_eye_tracking['VIDEO_NAME'].iloc[trial_index-1]:
        # extract the video name of the trial index
        first_index_video_name = np.where(
            data_eye_tracking['TRIAL_INDEX'] == trial_index)[0][0]
        # extract the video name using the first index
        video_name_1 = data_eye_tracking['VIDEO_NAME'].iloc[first_index_video_name]
        # extract the video name from the data_trial_sequence
        video_name_2 = data_trial_sequence.loc[data_trial_sequence['TRIAL_INDEX']
                                               == trial_index ][ video_clip_header].values[0]
        # check if the video names are the same
        assert (video_name_1 == video_name_2)
        # set the video name
        video_name = video_name_1

    # extract the # extract the coordinates of the video from the video name
    parsed_video_name = video_name.split('_')
    # extract the shape, color, and frequency of the shape
    pixel_surface = parsed_video_name[3]
    color = parsed_video_name[6]
    frequency = parsed_video_name[8]
    shape = parsed_video_name[10]
    x = int(parsed_video_name[12][1:])
    y = int(parsed_video_name[13].split(')') [0])

    # check if the video has gaze data
```

```
gaze_data = True

print(f"extracting data for video {video_name}")

# extract all the data for the current video
video_data = data_eye_tracking.loc[data_eye_tracking['TRIAL_INDEX'] == trial_index]
# extract the gaze data for the left and right eye

gaze_x, gaze_y = video_data['AVERAGE_GAZE_X'], video_data['AVERAGE_GAZE_Y']
try:
    gaze_x = np.array([float(x) for x in gaze_x])
    gaze_y = np.array([float(y) for y in gaze_y])
except ValueError:
    # if the video has no gaze data
    print(f"video {video_name} has no gaze data")
    gaze_data = False

print(f"calculating the metrics for video {video_name}")
if gaze_data:
    # convert the data to a numpy array
    gaze_x = np.array(gaze_x)
    gaze_y = np.array(gaze_y)

    # calcualte the between the gaze and the center of the shape
    gaze_x_diff = gaze_x - x
    gaze_y_diff = gaze_y - y

    # calculate the distance between the gaze and the center of the shape
    gaze_distance = np.zeros(len(gaze_x_diff))
    gaze_angle = np.zeros(len(gaze_x_diff))
    for index in range(len(gaze_x_diff)):
        gaze_distance[index] = np.sqrt(
            gaze_x_diff[index] ** 2 + gaze_y_diff[index] ** 2)
        gaze_angle[index] = np.arctan2(
            gaze_y_diff[index], gaze_x_diff[index])
    # convert to numpy array
    gaze_distance = np.array(gaze_distance)
    gaze_angle = np.array(gaze_angle)
    # calculate the number of gaze measurements
    number_of_gaze_measurements_trial = len(gaze_x)

    # calculate the STATISTICS
    #data_mean, data_average, data_std, data_variance, data_min, data_max
    gaze_x_mean, gaze_x_average, gaze_x_std, gaze_x_variance, gaze_x_min,
        gaze_x_max = calculate_statistics(
        gaze_x)
    gaze_y_mean, gaze_y_average, gaze_y_std, gaze_y_variance, gaze_y_min,
        gaze_y_max = calculate_statistics(
        gaze_y)
    gaze_distance_mean, gaze_distance_average, gaze_distance_std,
        gaze_distance_variance, gaze_distance_min, gaze_distance_max =
        calculate_statistics(
        gaze_distance)
    gaze_angle_mean, gaze_angle_average, gaze_angle_std, gaze_angle_variance,
        gaze_angle_min, gaze_angle_max = calculate_statistics(
        gaze_angle)
```

```
        print(f"video name: {video_name}| mean gaze distance: {gaze_distance_mean} |
            average gaze distance: {gaze_distance_average} | variance gaze distance: {
            gaze_distance_variance} | mean gaze angle: {gaze_angle_mean} | average gaze
            angle: {gaze_angle_average} | variance gaze angle: {gaze_angle_variance}, max
            snr: {max_snr}")
    else:
        if trial_index > 4:
            counter += 1
        else:
            experimental_counter += 1

    # create a figure
    # select the color of the shape
    if color == 'red':
        color_tuple = (0, 0, 255)
    elif color == "green":
        color_tuple = (0, 255, 0)
    elif color == "blue":
        color_tuple = (255, 0, 0)
    elif color == "white":
        color_tuple = (255, 255, 255)
    # draw the shape
    img = draw_shape(shape, int(pixel_surface), [(x, y)], [color_tuple])

    if gaze_data:
        # draw the points on the target
        img, points_on_target = draw_points(img, x, y, gaze_x, gaze_y)
        # calculate the number of points on the target
        number_of_points_on_target = len(points_on_target)
        # calculate the number of points on the target
        row = [trial_index, video_name, block, pixel_surface, color, shape, frequency,
            displayed_frames, dropped_frames, max_snr, frequency_sampled_at,
            gaze_distance_mean, gaze_distance_average, gaze_distance_variance,
                gaze_angle_mean, gaze_angle_average, gaze_angle_variance, gaze_x_mean,
                    gaze_x_average, x, gaze_x_variance, gaze_y_mean, gaze_y_average, y,
                    gaze_y_variance, number_of_gaze_measurements_trial,
                    number_of_points_on_target]
    else:
        # when there is no gaze data
        row = [trial_index, video_name, block, pixel_surface, color, shape, frequency,
            displayed_frames, dropped_frames, max_snr, frequency_sampled_at,
                np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan,
                    np.nan, np.nan, np.nan, np.nan, np.nan, np.nan]

    # append the row to the processed data
    processed_data.loc[len(processed_data)] = row
    # save the figure
    video_name_to_jpg = str(trial_index)+"_" + \
        video_name.split('.')[0] + '.jpg'

    # save the image
    cv2.imwrite(os.path.join(path_image_dir, video_name_to_jpg), img)
    # save the processed data
    processed_data.to_csv(os.path.join(
        path_image_dir, 'processed_data.csv'), index=False)
```

```
            if gaze_data:
                all_gaze_x.append(gaze_x)
                all_gaze_y.append(gaze_y)

        # get the dropped frames across all the trials
        dropped_frames = processed_data['DROPPED_FRAMES']
        # plot the dropped frames across all the trials with matplotlib
        # create new figure
        plt.figure(figsize=(20, 10))
        # plot the dropped frames
        plt.plot(dropped_frames)
        # set the x and y labels
        plt.ylabel('dropped frames')
        plt.xlabel('trial index')
        # set x limits to the number of trials
        plt.xlim(0, len(dropped_frames))
        # save the figure
        plt.savefig(os.path.join(path_image_dir, 'dropped_frames.png'))
        plt.close()
        # create a heatmap of the gaze data
        heatmap = create_heatmap(x, y, all_gaze_x, all_gaze_y)
        # save the heatmap
        cv2.imwrite(os.path.join(path_image_dir, 'heatmap.jpg'), heatmap)
        print(
            f"Of all the videos {counter} videos had no gaze data, and {len(trial_indices_unique)-4-
                counter} videos had gaze data")
        print(
            f"Of all the Experimental videos {experimental_counter} videos had no gaze data, and {4-
                experimental_counter} videos had gaze data")
        # skip the first 4 trials because they are not relevant
        experimental_processed_data = processed_data.iloc[4:]
        # extract the video names
        video_names = experimental_processed_data['VIDEO_NAME']
        # count the unique video names and number of occurences
        unique_video_names, counts = np.unique(video_names, return_counts=True)
        # check if the number of occurences is the same for all the video names is 3
        if np.all(counts == 3):
            print("All the video names have 3 occurences")


def process_1x1_data(data_eye_tracking: pd.DataFrame, path: str, path_image_dir: str,
        data_trial_sequence: pd.DataFrame, folder_path: str):
    """ Process the data from the 1x1 experiment

    Args:
        data_eye_tracking (pd.DataFrame): data from the eye tracker
        path (str): path to the eye tracking data
        path_image_dir (str): path to the directory with the images
        data_trial_sequence (pd.DataFrame): data from the trial sequence (Expereriment X.txt/csv)
        folder_path (str): path to the main folder of the experiment
    """
    # generate paths to the files
    eeg_path, vhdr_path, vmrk_path, txt_path = generate_paths(folder_path)
    # load the eeg data
    df_txt, all_events, trials_events, num_of_trials, txt_path, raw, data, sfreq, ch_names, tmin,
        tmax = load_and_setup_eeg_data(
```

```
        txt_path, vhdr_path, vmrk_path, txt_path)
    # get and process the data from the eye tracker
    process_eye_tracking_data(path)
    # extract the headers of the data_eye_tracking
    headers = data_eye_tracking.columns.values
    # correct the trial index
    if data_eye_tracking["TRIAL_INDEX"].iloc[0] > 1:
        difference_trial_index = data_eye_tracking["TRIAL_INDEX"].iloc[0] - 1
        # correct the trial index in the data_eye_tracking
        data_eye_tracking["TRIAL_INDEX"] = data_eye_tracking["TRIAL_INDEX"] - \
            difference_trial_index
    # get the trial indices
    trial_indices = data_eye_tracking['TRIAL_INDEX']
    # get the unique trial indices
    trial_indices_unique = np.unique( trial_indices )
    # add trial_index column to data_trial_sequence
    data_trial_sequence['TRIAL_INDEX'] = data_trial_sequence.index+1
    # extract the headers of the data_trial_sequence
    headers_trial_sequence = data_trial_sequence.columns.values

    # get name of header with video_clip in it
    video_clip_header = [
        header for header in headers_trial_sequence if 'video_clip' in header][0]

    # create empty data frame to store the processed data
    headers_processed_data = ["TRIAL_INDEX", 'VIDEO_NAME', "BLOCK", 'PIXEL_SURFACE', "
        COLOR", "SHAPE", "FREQUENCY", "DISPLAYED_FRAMES", "DROPPED_FRAMES", "
        MAX_SNR", 'FREQUENCY_SAMPLED_AT', 'MEAN_GAZE_DISTANCE', '
        AVERAGE_GAZE_DISTANCE',
                                'VARIANCE_GAZE_DISTANCE', 'MEAN_GAZE_ANGLE', '
                                    AVERAGE_GAZE_ANGLE', 'VARIANCE_GAZE_ANGLE', "
                                    MEAN_X", "X_AVERAGE", "X", "X_VARIANCE", "Y_MEAN", "
                                    Y_AVERAGE", "Y", "Y_VARIANCE", "
                                    NUMBER_OF_GAZE_POINTS", "POINTS_ON_TARGET"]
    processed_data = pd.DataFrame(columns=headers_processed_data)

    # create empty lists to store the gaze data
    all_gaze_x = []
    all_gaze_y = []

    # get rid of the data_eye_tracking['VIDEO_NAME'] is '.'
    data_eye_tracking = data_eye_tracking[data_eye_tracking['VIDEO_NAME'] != '.']

    experimental_counter = 0
    counter = 0
    # loop through all the videos
    for trial_index in trial_indices_unique:
        # Extract max SNR
        trial_index_for_list = trial_index - 1
        # block variable
        block = data_trial_sequence.iloc[ trial_index_for_list ][ 'Block_variable']
        # get the number of displayed frames
        displayed_frames = data_trial_sequence.iloc[ trial_index_for_list ][ 'displayed_frame_count']
        # get the number of dropped frames
        dropped_frames = data_trial_sequence.iloc[ trial_index_for_list ][ 'dropped_frame_count']
        max_snr, frequency_sampled_at = process_eeg_trial_data(
```

```
        trial_index_for_list , df_txt, all_events, trials_events, txt_path, raw, sfreq,
            ch_names, tmin, tmax)

    # extract the video name from the data_trial_sequence
    print(f"extracting video name for trial index {trial_index}")
    if '1x1' in data_eye_tracking['VIDEO_NAME'].iloc[trial_index-1]:
        # extract the video name of the trial index by finding the first index of the trial
            index in the data_eye_tracking
        first_index_video_name = np.where(
            data_eye_tracking['TRIAL_INDEX'] == trial_index)[0][40]
        # extract the video name using the first index
        video_name_1 = data_eye_tracking['VIDEO_NAME'].iloc[first_index_video_name]
        # extract the video name from the data_trial_sequence
        video_name_2 = data_trial_sequence.loc[data_trial_sequence['TRIAL_INDEX']
                                               == trial_index ][video_clip_header].values[0]
        # check if the video names are the same
        assert (video_name_1 == video_name_2)
        # set the video name
        video_name = video_name_1

    # extract the # extract the coordinates of the video from the video name
    parsed_video_name = video_name.split('_')
    # extract the shape, color, and frequency of the shape
    pixel_surface = parsed_video_name[3]
    color = parsed_video_name[6]
    frequency = parsed_video_name[8]
    shape = parsed_video_name[10][:-4]

    # check if the video has gaze data
    gaze_data = True

    print(f"extracting data for video {video_name}")
    # set the x and y coordinates of the center of the shape
    x = 960
    y = 540
    # extract all the data for the current video
    video_data = data_eye_tracking.loc[data_eye_tracking['TRIAL_INDEX'] == trial_index]
    # extract the average gaze data for the left and right eye
    gaze_x, gaze_y = video_data['AVERAGE_GAZE_X'], video_data['AVERAGE_GAZE_Y']
    try:
        gaze_x = np.array([float(x) for x in gaze_x])
        gaze_y = np.array([float(y) for y in gaze_y])
    except ValueError:
        # if the video has no gaze data
        print(f"video {video_name} has no gaze data")
        gaze_data = False

    print(f"calculating the metrics for video {video_name}")
    if gaze_data:
        # convert the data to a numpy array
        gaze_x = np.array(gaze_x)
        gaze_y = np.array(gaze_y)

        # calcualte the between the gaze and the center of the shape
        gaze_x_diff = gaze_x - x
        gaze_y_diff = gaze_y - y
```

```
            # calculate the distance between the gaze and the center of the shape
            gaze_distance = np.zeros(len(gaze_x_diff))
            gaze_angle = np.zeros(len(gaze_x_diff))
            for index in range(len(gaze_x_diff)):
                gaze_distance[index] = np.sqrt(
                    gaze_x_diff[index] ** 2 + gaze_y_diff[index] ** 2)
                gaze_angle[index] = np.arctan2(
                    gaze_y_diff[index], gaze_x_diff[index])
            # convert to numpy array
            gaze_distance = np.array(gaze_distance)
            gaze_angle = np.array(gaze_angle)
            # calculate the number of gaze measurements
            number_of_gaze_measurements_trial = len(gaze_x)

            #data_mean, data_average, data_std, data_variance, data_min, data_max
            gaze_x_mean, gaze_x_average, gaze_x_std, gaze_x_variance, gaze_x_min,
                gaze_x_max = calculate_statistics(
                gaze_x)
            gaze_y_mean, gaze_y_average, gaze_y_std, gaze_y_variance, gaze_y_min,
                gaze_y_max = calculate_statistics(
                gaze_y)
            gaze_distance_mean, gaze_distance_average, gaze_distance_std,
                gaze_distance_variance, gaze_distance_min, gaze_distance_max =
                calculate_statistics(
                gaze_distance)
            gaze_angle_mean, gaze_angle_average, gaze_angle_std, gaze_angle_variance,
                gaze_angle_min, gaze_angle_max = calculate_statistics(
                gaze_angle)
            print(f"video name: {video_name}| mean gaze distance: {gaze_distance_mean} |
                average gaze distance: {gaze_distance_average} | variance gaze distance: {
                gaze_distance_variance} | mean gaze angle: {gaze_angle_mean} | average gaze
                angle: {gaze_angle_average} | variance gaze angle: {gaze_angle_variance}")

        else:
            # if the no practice trials anymore
            if trial_index > 4:
                counter += 1
            else:
                experimental_counter += 1

        # create a figure and start with choosing the color of the shape
        if color == 'red':
            color_tuple = (0, 0, 255)
        elif color == "green":
            color_tuple = (0, 255, 0)
        elif color == "blue":
            color_tuple = (255, 0, 0)
        elif color == "white":
            color_tuple = (255, 255, 255)

        # draw the shape
        img = draw_shape(shape, int(pixel_surface), [(x, y)], [color_tuple])

        # draw the gaze points
        if gaze_data:
```

```
            # draw the gaze points and extract the number of points on the target
            img, points_on_target = draw_points(img, x, y, gaze_x, gaze_y)
            number_of_points_on_target = len(points_on_target)
            # calculate the number of points on the target
            row = [trial_index, video_name, block, pixel_surface, color, shape, frequency,
                displayed_frames, dropped_frames, max_snr, frequency_sampled_at,
                gaze_distance_mean, gaze_distance_average, gaze_distance_variance,
                    gaze_angle_mean, gaze_angle_average, gaze_angle_variance, gaze_x_mean,
                        gaze_x_average, x, gaze_x_variance, gaze_y_mean, gaze_y_average, y,
                        gaze_y_variance, number_of_gaze_measurements_trial,
                        number_of_points_on_target]
        else:
            # when there is no gaze data
            row = [trial_index, video_name, block, pixel_surface, color, shape, frequency,
                displayed_frames, dropped_frames, max_snr, frequency_sampled_at,
                    np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan,
                        np.nan, np.nan, np.nan, np.nan, np.nan, np.nan]
        # append the row to the processed data
        processed_data.loc[len(processed_data)] = row
        # save the figure
        video_name_to_jpg = str(trial_index)+"_" + \
            video_name.split('.')[0] + '.jpg'

        # save the image
        cv2.imwrite(os.path.join(path_image_dir, video_name_to_jpg), img)
        # save the processed data
        processed_data.to_csv(os.path.join(
            path_image_dir, 'processed_data.csv'), index=False)
        # save the gaze data
        if gaze_data:
            all_gaze_x.append(gaze_x)
            all_gaze_y.append(gaze_y)

# get the dropped frames across all the trials
dropped_frames = processed_data['DROPPED_FRAMES']
# plot the dropped frames across all the trials with matplotlib
# create new figure
plt.figure(figsize=(20, 10))
# plot the dropped frames
plt.plot(dropped_frames)
# set the x and y labels
plt.xlabel('trial index')
plt.ylabel('dropped frames')
# set x limits to the number of trials
plt.xlim(0, len(dropped_frames))
# save the figure
plt.savefig(os.path.join(path_image_dir, 'dropped_frames.png'))
plt.close()

# create a heatmap of the gaze data
heatmap = create_heatmap(x, y, all_gaze_x, all_gaze_y)
# save the heatmap
cv2.imwrite(os.path.join(path_image_dir, 'heatmap.jpg'), heatmap)
print(
    f"Of all the videos {counter} videos had no gaze data, and {len(trial_indices_unique)-4-
        counter} videos had gaze data")
```

```
    print(
        f" Of all the Experimental videos {experimental_counter} videos had no gaze data, and {4-
            experimental_counter} videos had gaze data")
    # skip the first 4 trials because they are not relevant
    experimental_processed_data = processed_data.iloc[4:]
    # extract the video names
    video_names = experimental_processed_data['VIDEO_NAME']
    # count the unique video names and number of occurences
    unique_video_names, counts = np.unique(video_names, return_counts=True)
    # check if the number of occurences is the same for all the video names is 3
    if np.all(counts == 3):
        print("All the video names have 3 occurences")


def load_csv_data_utf8(path: str) -> pd.DataFrame:
    """ Load the data from the csv file with UTF-8 encoding

    Args:
        path (str): Path to the csv file

    Returns:
        pd.DataFrame: Dataframe with the data from the csv file
    """
    # convert path to raw string
    new_path = r'{}'.format(path)
    with open(new_path, encoding='UTF-8') as f:
        # read csv file and store in dataframe with the correct headers
        data = pd.read_csv(f, sep='\t')
    return data


def main():
    # Load the eye tracking data
    folder_path = r'/media/sjoerd/BackUp Drive/Thesis_project/participant data/raw data/pp1/
        Experiment 1'

    if 'Experiment 2' in folder_path:
        path_eye_tracking = folder_path + '/Experiment_2x2_eye_tracking.csv'
        path_trial_sequence = folder_path + '/EXPERIMENT_2X2.csv'
    elif 'Experiment 1' in folder_path:
        path_eye_tracking = folder_path + '/Experiment_1x1_eye_tracking.csv'
        path_trial_sequence = folder_path + '/EXPERIMENT_1X1.csv'
    else:
        print("Please select the correct folder")
        return

    # create folder for images
    path_img_folder = path_eye_tracking.split('.csv')[0]+'/images'
    print(f" The folder to save the images to is {path_img_folder}")

    # create folder for images
    if not os.path.exists(path_img_folder):
        os.makedirs(path_img_folder)
    # convert path to raw string
    new_path = r'{}'.format(path_eye_tracking)
    with open(new_path, encoding='UTF-16') as f:
```

```
        data_eye_tracking = pd.read_csv(f, sep='\t')
    # extract the headers
    headers = data_eye_tracking.columns.values
    video_names = data_eye_tracking['VIDEO_NAME']
    # remove all the video names that are '.'
    video_names = np.array(video_names[video_names != '.'])
    # pdb.set_trace()

    data_trial_sequence = load_csv_data_utf8(path_trial_sequence)
    if '2x2' in video_names[20]:
        print('2x2 found')
        process_2x2_data(data_eye_tracking, path_eye_tracking,
                         path_img_folder, data_trial_sequence, folder_path)
    else:
        print('2x2 not found')
        if '1x1' in video_names[20]:
            print('1x1 found')
            process_1x1_data(data_eye_tracking, path_eye_tracking,
                             path_img_folder, data_trial_sequence, folder_path)
        else:
            print("no valid video name found")
            # stop the program
            return
    print(f"finished processing data for {path_eye_tracking}")


main()
```

D.2.3. process_SNR.py

Combines all the SNR data of all the participants from process_eeg_and_eye_tracking_v1.py.

```
import numpy as np
import pandas as pd
import glob
import sys


def filter_dataframe_exp(df_pp: pd.DataFrame, df_snr: pd.DataFrame) -> pd.DataFrame:
    """
    Filter the dataframe per participant and per video and put it in another format for the
        analysis
    :param df_pp: dataframe with the snr data per participant
    :param df_snr: dataframe with the snr data per video
    :return: dataframe with the filtered data
    """

    pp_numbers = df_snr['pp'].unique()
    pp_numbers.sort()
    # loop over the files
    for folder in df_snr['Folder'].unique():
        # extract unique video names of the folder
        unique_video_names = df_snr[df_snr['Folder']
                                    == folder]['Video'].unique()
        # loop over the unique video names and extract the data
        for video in unique_video_names:
            # extract the data for the video
```

```
            video_data = df_snr[(df_snr['Folder'] == folder)
                                & (df_snr['Video'] == video)]
            # extract the settings of the video such as pixel surface, color, shape, frequency
            pixel_surface = video_data['pixel_surface'].unique()[0]
            color = video_data['color'].unique()[0]
            shape = video_data['shape'].unique()[0]
            frequency = video_data['frequency'].unique()[0]
            row = [folder, video, pixel_surface, color, shape, frequency]
            # loop over the pp numbers
            for pp_number in pp_numbers:
                # extract the data for the pp
                pp_data = video_data[video_data['pp'] == pp_number]
                # extract the snr data for the pp
                snr_data = np.array(pp_data['MAX_SNR'])[0]
                # append the snr data to the row
                row.append(snr_data)

            # append the row to the dataframe
            df_pp.loc[len(df_pp)] = row

    return df_pp


# main folder path
print("main folder path")
path = r"/media/sjoerd/BackUp Drive/Thesis_project/Data_SNR"
# find child folders
folders = glob.glob(path + "/*")
print(f" folders: {folders}")
# find all files in child folders
files = {}
# loop over the folders
for folder in folders:
    # remove the path from the folder name
    folder_name = folder.replace(path + "/", "")
    files[folder_name] = glob.glob(folder + "/*")

# create a dictionary to store the data
files_data = {}
# create a dataframe to store the data
headers_dataframe = ['Folder', 'File', 'pp', 'Video',
                     'pixel_surface', 'color', 'shape', 'frequency', 'MAX_SNR']
# create empty dataframe
df_snr = pd.DataFrame(columns=headers_dataframe)

for folder in files:
    print(f"folder: {folder}")
    print(f" files : {files[folder]}")
    # create a dictionary for the folder
    files_data[folder] = {}
    for file in files[folder]:
        # remove the path from the file name
        file_name = file.replace(path + "/" + folder + "/", "")
        # read the data
        data = pd.read_csv(file, header=None)
        # get headers from the first row of the data
```

```
        headers = data.iloc[0]
        # set the headers
        data.columns = headers
        # make tthe first row the headers
        data = data[1:]
        # remove the first 4 trials
        data = data[4:]
        if 'Experiment_1' == folder:
            assert (len(data['TRIAL_INDEX']) == 324)
        elif 'Experiment_2' == folder:
            assert (len(data['TRIAL_INDEX']) == 432)
        else:
            print(f"Error: folder {folder} not recognized")
            sys.exit()

        # add the data to the dictionary
        files_data[folder][file_name] = {}
        files_data[folder][file_name]['data'] = data
        # get the VIDEO_NAME column
        video_name = data['VIDEO_NAME']
        # find the unique video names
        unique_video_names = video_name.unique()

        if 'Experiment_1' == folder:
            unique_video_names_experiment_1 = unique_video_names
        elif 'Experiment_2' == folder:
            unique_video_names_experiment_2 = unique_video_names

        # pdb.set_trace()
        files_data[folder][file_name]['unique_video_names'] = np.array(
            unique_video_names)
        # extract for each video_name the data
        for video in unique_video_names:
            # get the data for the video
            video_data = data[data['VIDEO_NAME'] == video]
            # extract the SNR data
            video_data_snr = video_data['MAX_SNR']
            # add the data to the dictionary
            files_data[folder][file_name][video] = video_data
            files_data[folder][file_name]['SNR'] = video_data_snr
            # extract the settings of each video
            pixel_surface = int(video_data['PIXEL_SURFACE'].unique()[0])
            color = video_data['COLOR'].unique()[0]
            shape = video_data['SHAPE'].unique()[0]
            frequency = int(video_data['FREQUENCY'].unique()[0])
            # extract the pp number from the file name
            pp_number = int(file_name.split(
                "_")[-1].split(".")[0].replace("pp", ""))
            # create the row for the dataframe
            row_dataframe = [folder, file_name, pp_number, video, pixel_surface,
                             color, shape, frequency, np.array(video_data_snr).astype(float)]

            # add the row to the dataframe
            df_snr.loc[len(df_snr)] = row_dataframe

# save the dataframe
```

```
df_snr.to_csv(path + "/SNR.csv", index=False)

# create empty dataframe
# generate the dataframe in the format 'Folder', 'Video'. 'pixel_surface', 'color', 'shape', '
    frequency' followed by the SNR data for each pp
headers_dataframe = ['Folder', 'Video',
                     'pixel_surface', 'color', 'shape', 'frequency']
# add the pp numbers to the headers
pp_number = df_snr['pp'].unique()

for pp in pp_number:
    headers_dataframe.append(f'pp{pp}')
# create empty dataframe
df_sorted_by_participant = pd.DataFrame(columns=headers_dataframe)

df_pp = filter_dataframe_exp(df_sorted_by_participant, df_snr)
# save the dataframe to a csv file named 'SNR_sorted_by_participant.csv'
df_pp.to_csv(path + "/SNR_sorted_by_participant.csv", index=False)
```

D.2.4. Requirements_snr_analysis.txt

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: linux-64
_libgcc_mutex=0.1=main
_openmp_mutex=5.1=1_gnu
appdirs=1.4.4=pypi_0
bottleneck=1.3.4=py39hd257fcd_1
ca-certificates=2022.12.7=ha878542_0
certifi=2022.12.7=pyhd8ed1ab_0
charset-normalizer=3.0.1=pypi_0
contourpy=1.0.7=pypi_0
cycler=0.11.0=pypi_0
decorator=5.1.1=pypi_0
fonttools=4.38.0=pypi_0
idna=3.4=pypi_0
jinja2=3.1.2=pypi_0
kiwisolver=1.4.4=pypi_0
ld_impl_linux-64=2.38=h1181459_1
libblas=3.9.0=15_linux64_openblas
libcblas=3.9.0=15_linux64_openblas
libffi=3.3=he6710b0_2
libgcc-ng=11.2.0=h1234567_1
libgfortran-ng=12.2.0=h69a702a_19
libgfortran5=12.2.0=h337968e_19
libgomp=11.2.0=h1234567_1
liblapack=3.9.0=15_linux64_openblas
libopenblas=0.3.20=pthreads_h78a6416_0
libstdcxx-ng=11.2.0=h1234567_1
markupsafe=2.1.2=pypi_0
matplotlib=3.6.3=pypi_0
mne=1.2.3=pypi_0
ncurses=6.3=h5eee18b_3
nomkl=1.0=h5ca1d4c_0
numexpr=2.8.0=py39h194a79d_102
numpy=1.24.1=pypi_0
```

```
openssl=1.1.1o=h166bdaf_0
packaging=23.0=pypi_0
pandas=1.5.1=py39h417a72b_0
pillow=9.4.0=pypi_0
pip=22.3.1=py39h06a4308_0
pooch=1.6.0=pypi_0
pyparsing=3.0.9=pypi_0
python=3.9.0=hdb3f193_2
python-dateutil=2.8.2=pyhd8ed1ab_0
python_abi=3.9=2_cp39
pytz=2022.7.1=pyhd8ed1ab_0
readline=8.2=h5eee18b_0
requests=2.28.2=pypi_0
scipy=1.10.0=pypi_0
setuptools=65.6.3=py39h06a4308_0
six=1.16.0=pyh6c4a22f_0
sqlite=3.40.1=h5082296_0
tk=8.6.12=h1ccaba5_0
tqdm=4.64.1=pypi_0
tzdata=2022g=h04d1e81_0
urllib3=1.26.14=pypi_0
wheel=0.37.1=pyhd3eb1b0_0
xz=5.2.10=h5eee18b_1
zlib=1.2.13=h5eee18b_0
```

D.2.5. SNR_statistic_analysis_v1.py

This script is used to perform the statistical analyses with respect to the SNR.

```
import numpy as np
import scipy.stats as stats
import sys
import os
import pingouin as pg
import pandas as pd
import matplotlib.pyplot as plt
from typing import Union


def create_boxplot(row: np.ndarray, path_to_save: str, data_per_participant_dict: dict):
    """ Create a boxplot for the data

    Args:
        data (np.ndarray): Data to plot in the boxplot
        path_to_save (str): Path to save the figure
    """
    experiment = row[0]
    category = row[1]
    groups_in_category_array = row[2]
    # create a figure
    fig, ax = plt.subplots()

    # create a list of colors for the legend
    colors = ['green', 'purple', 'brown', 'grey', 'olive', 'cyan']
    # create a list for the labels for the legend
    labels = []
    # create a list for the handles for the legend
```

```
handles = []
# the sequence consits of 4 elements: snr, mean, data_length, data
for i in range(len(groups_in_category_array)):
    group = groups_in_category_array[i]
    average = row[3+i*4]
    std = row[4+i*4]
    data_length = row[5+i*4]
    data = row[6+i*4]

    # boxplot with median and mean and using different colors for the mean and median also do
        not show the outliers
    bplot = ax.boxplot(data, positions=[i], widths=0.6, showmeans=True, meanline=True,
        showfliers=False,
                       patch_artist=True, medianprops=dict(color='blue'), meanprops=dict(
                           color='red'))
    # set the color of the boxplot to be filled the color light blue
    for patch in bplot['boxes']:
        patch.set_facecolor('lightblue')

    # extract the keys of the dictionary which are the participant numbers
    participant_keys = data_per_participant_dict[group].keys()

    # put the keys in numerical order
    participant_keys = sorted(
        participant_keys, key=lambda x: int(x.split('pp')[1]))

    # loop over the participants
    for j, participant in enumerate(participant_keys):
        # calculate the stepsize for the mean lines
        stepsize = 1 / len(data_per_participant_dict[group].keys())
        # extract the data for the participant
        data_participant = data_per_participant_dict[group][participant]
        # combine the data of the participant to a list
        data_participant = list(data_participant)
        # convert the data to a numpy array and flatten it
        data_participant = np.array(data_participant).flatten()
        # calculate the mean of the data for the participant
        participant_mean = np.mean(data_participant)

        # extract the participant number
        # pp5 turns into pp1, etc.
        participant_number = int(participant.split('pp')[1]) - 4

        # show the lines of the mean of each participant in the boxplot within each box with
            an alpha of 0.5 at the correct position with respect to the x axis
        ax.plot([i-0.5 + j*stepsize, i - 0.4 + j*stepsize],
                [participant_mean, participant_mean], color=colors[j], lw=3)
        if i == 0:
            # create the labels and handles for the legend
            label = f'pp {participant_number} mean'
            # add the label to the list of labels
            labels.append(label)
            handles.append(plt.Line2D([0], [0], color=colors[j], lw=4))

    # put vline after each boxplot
    ax.axvline(x=i+0.5, color='black', lw=1)
```

```
    if category == 'pixel_surface':
        ax.set_xlabel('pixel surface [pixels]')
        str_add = 'pixels'
    elif category == 'frequency':
        ax.set_xlabel('frequency [Hz]')
        str_add = 'Hz'
    else:
        ax.set_xlabel(category)
        str_add = ''

    ax.set_ylabel('SNR [dB]')
    # replace the "_" with a space in the category name and the experiment name
    category = category.replace('_', ' ')
    experiment = experiment.replace('_', ' ')

    ax.set_title('Boxplot SNR for ' + category + ' in ' +
                 experiment + ' without outliers')
    # set the x axis ticks to the groups in the category
    ax.set_xticks(np.arange(len(groups_in_category_array)))
    ax.set_xticklabels(groups_in_category_array)
    ax.yaxis.grid(True)

    # create a legend for the mean which is color red and the median which is color blue
    labels.append('mean of all data')
    labels.append('median of all data')

    # create a red line for the mean
    handles.append(plt.Line2D([0], [0], color='red', lw=4))
    # create a blue line for the median
    handles.append(plt.Line2D([0], [0], color='blue', lw=4))

    # create a custom legend with the labels and the colors
    plt.legend(handles, labels, loc='upper right',
               bbox_to_anchor=(1.4, 1.0), ncol=1)

    # save the figure
    plt.savefig(path_to_save + '/boxplot_' + experiment +
                '_' + category + '.png', dpi=300, bbox_inches='tight')


def create_figure(row: list, path_to_save: str):
    """ Create a figure with the snr and mean for each group in the category

    Args:
        row (list): Contains the experiment name, the category, the groups in the category, the snr
            and mean for each group in the category
        path_to_save (str): Path to save the figure
    """
    experiment = row[0]
    category = row[1]
    groups_in_category_array = row[2]

    snr_array = row[3:3+len(groups_in_category_array)*3]
    # the sequence consits of 3 elements: snr, mean, data_length
    # filter the snr and mean by removing the data_length, every 3rd element
```

```
    snr_array = [snr_array[i] for i in range(len(snr_array)) if i % 3 != 2]

    # for each group in the category extract the snr and mean
    snr_array = np.array(snr_array).reshape(len(groups_in_category_array), 2)
    # create a figure
    fig, ax = plt.subplots()
    # plot the snr and mean using a bar plot
    for i in range(len(groups_in_category_array)):
        # plot the snr with on the x axis the category and on the y axis the snr
        ax.bar(i, snr_array[i, 0], align='center',
               alpha=0.5, ecolor='black', capsize=10)
        # plot the std on the bars with respect to the snr mean
        ax.errorbar(i, snr_array[i, 0], yerr=snr_array[i, 1],
                    fmt='o', ecolor='black', capsize=10)

    ax.set_ylabel('SNR')
    if category == 'pixel_surface':
        ax.set_xlabel('pixel surface [pixels]')
        str_add = 'pixels'
    elif category == 'frequency':
        ax.set_xlabel('frequency [Hz]')
        str_add = 'Hz'
    else:
        ax.set_xlabel(category)
        str_add = ''

    # replace the "_" with a space in the category name and the experiment name
    category = category.replace('_', ' ')
    experiment = experiment.replace('_', ' ')

    ax.set_title('SNR for ' + category + ' in ' + experiment)
    # hide the x axis ticks
    ax.set_xticks([])
    ax.yaxis.grid(True)
    # show legend with the bars labeled with the groups in the category and the error bars labeled
        with the std
    # create the labels for the legend
    labels = []

    for i in range(len(groups_in_category_array)):

        labels.append(str(groups_in_category_array[i]) + f' {str_add} mean')
        labels.append(str(groups_in_category_array[i]) + f' {str_add} std')
    # create the legend and place it outside the figure showing the plots
    ax.legend(labels, loc='center left', bbox_to_anchor=(1, 0.5))
    # make sure the legend is not cut off
    plt.tight_layout()

    # save the figure in the path_to_save
    plt.savefig(path_to_save + '/' + experiment + '_' + category + '.png')


def convert_dataframe_strings_to_list_SNR(df: pd.DataFrame) -> pd.DataFrame:
    """ Convert the list of strings in the dataframe to list of floats

    Args:
```

```
        df (pd.DataFrame): Dataframe with the data as list of strings

    Returns:
        pd.DataFrame: Dataframe with the data as list of floats
    """
    # make an empty dataframe that will be filled with the data
    df_new = pd.DataFrame(columns=df.columns)

    # loop over columns
    for j in range(df.shape[0]):
        row = []
        # copy of row j the first 6 columns
        row = list(df.iloc[j, 0:6].copy())

        for i in range(6, df.shape[1]):
            # remove first and last character of the string
            string = df.iloc[j, i][1:-1]
            # separate the string by space and convert to list
            list_of_strings = string.split(' ')
            # remove '' from the list
            list_of_strings = [x for x in list_of_strings if x != '']
            # convert the list to float
            array_of_floats = [float(x) for x in list_of_strings]
            # save the array in the dataframe

            row.append(array_of_floats)
        # append the row to the dataframe
        df_new.loc[len(df_new)] = row

    return df_new


def create_table_with_significance_using_pvalue(pvalue_matrix: np.ndarray, groups: np.ndarray,
    alpha_threshold=0.05) -> pd.DataFrame:
    """ Create a table with the significance of the pairwise comparisons

    Args:
        pvalue_matrix (np.ndarray): Pvalue matrix with the pvalues of the pairwise comparisons
        groups (np.ndarray): Array with the groups
        alpha_threshold (float, optional): Threshold for the significance. Defaults to 0.05.

    Returns:
        pd.DataFrame: Table with the significance of the pairwise comparisons
    """
    # create a table with the pairwise comparisons with the rows one group tested against the
        other group mentioned in the columns
    pairwise_comparisons_table_significance = pd.DataFrame(
        index=groups, columns=groups)

    # loop through the unique groups and check if the pairwise comparison is in the pairwise
        comparisons table
    for i in range(groups.shape[0]):
        for j in range(groups.shape[0]):

            # extract labels of the groups to be compared
            group_A = groups[i]
```

```
            group_B = groups[j]

            # save the significance in the table pairwise_comparisons_table_significance
            if pvalue_matrix[i, j] <= alpha_threshold:
                pairwise_comparisons_table_significance.loc[group_A,
                                                            group_B] = 'Significant'
            else:
                pairwise_comparisons_table_significance.loc[group_A,
                                                            group_B] = 'Not significant'

    return pairwise_comparisons_table_significance


def create_table_with_pairwise_comparisons(pairwise_comparisons: pd.DataFrame,
    alpha_pairwise_comparisons=0.05) -> Union[pd.DataFrame, pd.DataFrame]:
    """ Create a table with the pairwise comparisons with the rows one group tested against the
        other group mentioned in the columns

    Args:
        pairwise_comparisons (pd.DataFrame): Table with the pairwise comparisons
        alpha_pairwise_comparisons (float, optional): Alpha value for the pairwise comparisons.
            Defaults to 0.05.

    Returns:
        Union[pd.DataFrame, pd.DataFrame]: Table with the p-values and table with the significance
            of the pairwise comparisons
    """
    # extract the unique group labels from the pairwise comparisons table
    unique_groups_A = pairwise_comparisons['A'].unique()
    unique_groups_B = pairwise_comparisons['B'].unique()
    unique_groups = np.unique(np.concatenate(
        (unique_groups_A, unique_groups_B)))

    # extract the p-values from the pairwise comparisons table
    p_values_pairwise_comparisons = pairwise_comparisons['pval']

    # create a table with the pairwise comparisons with the rows one group tested against the
        other group mentioned in the columns
    pairwise_comparisons_table_p_values = pd.DataFrame(
        index=unique_groups, columns=unique_groups)
    pairwise_comparisons_table_significance = pd.DataFrame(
        index=unique_groups, columns=unique_groups)

    # loop through the unique groups and check if the pairwise comparison is in the pairwise
        comparisons table
    for i in range(unique_groups.shape[0]):
        for j in range(unique_groups.shape[0]):

            # extract labels of the groups to be compared
            group_A = unique_groups[i]
            group_B = unique_groups[j]

            # check where the group labels are in the pairwise comparisons table
            Check_if_pair_A_B_in_pairwise_comparisons = (
                pairwise_comparisons['A'] == group_A) & (pairwise_comparisons['B'] == group_B)
```

```
# check if any of them are true
if np.any(Check_if_pair_A_B_in_pairwise_comparisons):
    print("Pairwise comparison between group {} and group {} is in the pairwise
        comparisons table".format(
        group_A, group_B))
    # save the p-value in the table
    p_value = float(p_values_pairwise_comparisons[(
        pairwise_comparisons['A'] == group_A) & (pairwise_comparisons['B'] ==
            group_B)])
    pairwise_comparisons_table_p_values.loc[group_A,
                                            group_B] = p_value

    # save the significance in the table pairwise_comparisons_table_significance
    if p_value <= alpha_pairwise_comparisons:
        pairwise_comparisons_table_significance.loc[group_A,
                                                    group_B] = ' Significant '
    else:
        pairwise_comparisons_table_significance.loc[group_A,
                                                    group_B] = 'Not significant '

else:
    Check_if_pair_B_A_in_pairwise_comparisons = (
        pairwise_comparisons['A'] == group_B) & (pairwise_comparisons['B'] ==
            group_A)

    if np.any(Check_if_pair_B_A_in_pairwise_comparisons):
        print("Pairwise comparison between group {} and group {} is in the pairwise
            comparisons table".format(
            group_B, group_A))
        # save the p-value in the table
        p_value = float(p_values_pairwise_comparisons[(
            pairwise_comparisons['A'] == group_B) & (pairwise_comparisons['B'] ==
                group_A)])
        # save the p-value in the table pairwise_comparisons_table_p_values
        pairwise_comparisons_table_p_values.loc[group_A,
                                                group_B] = p_value

        # save the significance in the table pairwise_comparisons_table_significance
        if p_value <= alpha_pairwise_comparisons:
            pairwise_comparisons_table_significance.loc[group_A,
                                                        group_B] = ' Significant '
        else:
            pairwise_comparisons_table_significance.loc[group_A,
                                                        group_B] = 'Not significant '

    # if the group labels are not in the pairwise comparisons table, put a NaN in the
        table
    else:
        print("Pairwise comparison between group {} and group {} is not in the pairwise
            comparisons table".format(
            group_A, group_B))
        # put a NaN in the table
        pairwise_comparisons_table_p_values.loc[group_A,
                                                group_B] = np.nan
        # put an X in the table to indicate that the comparison is not in the pairwise
            comparisons table
```

```
                    pairwise_comparisons_table_significance.loc[group_A,
                                                                group_B] = "X"

    return pairwise_comparisons_table_p_values, pairwise_comparisons_table_significance


path = r"/media/sjoerd/BackUp Drive/Thesis_project/Data_SNR/SNR_sorted_by_participant.csv"
# get path of folder where the results are saved
path_folder = os.path.dirname(path)
# create new folder named "SNR results" in the folder where the results are saved
path_folder_results = os.path.join(path_folder, "SNR results")
# check if the folder already exists
if not os.path.exists(path_folder_results):
    # if not, create the folder
    os.makedirs(path_folder_results)

# read the data
data = pd.read_csv(path)
# convert the strings in the data to lists
data = convert_dataframe_strings_to_list_SNR(data)

# get the headers of the data
headers = data.columns.values
# unique experiments (folder)
unique_experiments = data['Folder'].unique()
# unique categories
unique_categories_all = headers[2:6]

# dictionary with the unique groups per experiment per category
groups_per_experiment_dict = {}
for experiment in unique_experiments:
    groups_per_experiment_dict[experiment] = {}
    for category in unique_categories_all:
        # extract unique metrics per experiment per category
        unique_groups = data[data['Folder'] == experiment][category].unique()
        # sort the unique groups
        unique_groups.sort()
        groups_per_experiment_dict[experiment][category] = unique_groups

# dictionary to store the results of the analysis
data_logger = {}

# loop over the experiments
for experiment in unique_experiments:
    # create a dictionary for the experiment
    data_logger[experiment] = {}

    # headers with average and std of the groups in the experiment
    headers = ['category', 'groups', 'p_value_bartlett', 'Variances equal?', 'ANOVA method', 'df
        within groups', 'df between groups',
            'p_value ANOVA method', 'Mean different?', 'Type of post hoc test', 'p_value_matrix'
                , 'Significance_matrix']
    # create an empty dataframe to store the results of each step of the analysis
    results_logger = pd.DataFrame(columns=headers)

    # snr_metrics dataframe of the experiment and groups from A to B
```

```
headers_snr = ['experiment', 'category', 'groups', 'SNR group A average', 'SNR group A std', '
    group A number of samples', 'SNR group B average', 'SNR group B std',
                'group B number of samples', 'SNR group C average', 'SNR group C std', 'group
                    C number of samples', 'SNR group D average', 'SNR group D std', 'group D
                    number of samples', ]
# create an empty dataframe to store the results of each step of the analysis
snr_metrics = pd.DataFrame(columns=headers_snr)

# snr_metrics dataframe of the experiment and groups from A to B
if 'Experiment_2' == experiment:
    # remove 'pixel_surface' from the unique categories
    unique_categories = [x for x in unique_categories_all if x != 'shape']
else:
    unique_categories = unique_categories_all

# create a dictionary for the categories
data_logger[experiment]['category'] = {}
headers_snr_extensive = ['experiment', 'category', 'groups', 'SNR group A average', 'SNR
    group A std', 'group A number of samples', 'Group A data', 'SNR group B average', 'SNR
    group B std', 'group B number of samples', 'Group B data',
                         'SNR group C average', 'SNR group C std', 'group C number of
                             samples', 'Group C data', 'SNR group D average', 'SNR group D
                             std', 'group D number of samples', 'Group D data']
# create an empty dataframe to store the results of each step of the analysis
snr_metrics_extensive = pd.DataFrame(columns=headers_snr_extensive)
# loop over the categories
for category in unique_categories:
    # create a dictionary for the category
    data_logger[experiment]['category'][category] = {}
    # create a row for the results_logger dataframe
    row = []

    # get the groups in the category
    groups_in_category_array = groups_per_experiment_dict[experiment][category]
    # create a dictionary for the groups
    data_analysis = {}
    # store the experiment, category and groups in the row
    row_snr = [experiment, category, groups_in_category_array]
    row_snr_extensive = [experiment, category, groups_in_category_array]
    # create a dictionary for the data of the participants
    data_per_participant_dict = {}
    # loop over the groups
    for group in groups_in_category_array:
        # create an empty dictionary for the group in the data_per_participant_dict
        data_per_participant_dict[group] = {}
        # create a dictionary for the group
        data_logger[experiment]['category'][category][group] = {}
        # extract snr data of participants in the group
        data_experiment = data[(data['Folder'] == experiment)]
        # extract all the snr data of the group from the participants
        data_group = data_experiment[data_experiment[category]
                                     == group].iloc[:, 6:]

        # extract keys of the data_group dataframe
        participant_keys = data_group.keys()
        # loop over the keys
```

```
            for key in participant_keys:
                # extract the data of the participant
                data_participant = data_group[key]
                # put the data in a dictionary
                data_per_participant_dict[group][key] = data_participant

            # convert the dataframe to a list
            data_group_list = data_group.values.tolist()
            # convert the list to a 1d array
            data_group_1d_array = np.array(data_group_list).flatten()

            # log the data
            data_logger[experiment]['category'][category][group]['data'] = data_group_1d_array
            # store the data in a dictionary
            data_analysis[group] = data_group_1d_array

            # calculate the average and std of the group
            average = np.mean(data_group_1d_array)
            std = np.std(data_group_1d_array)

            # log the average and std
            data_logger[experiment]['category'][category][group]['average'] = average
            data_logger[experiment]['category'][category][group]['std'] = std
            data_logger[experiment]['category'][category][group]['number of samples'] = len(
                data_group_1d_array)

            # append the average and std to the row
            row_snr.append(average)
            row_snr.append(std)
            row_snr.append(len(data_group_1d_array))
            row_snr_extensive.append(average)
            row_snr_extensive.append(std)
            row_snr_extensive.append(len(data_group_1d_array))
            row_snr_extensive.append(data_group_1d_array)

        if len(groups_in_category_array) == 3:
            # add the nan for the group D which is not present in the experiment
            row_snr.append(np.nan)
            row_snr.append(np.nan)
            row_snr.append(np.nan)
            row_snr_extensive.append(np.nan)
            row_snr_extensive.append(np.nan)
            row_snr_extensive.append(np.nan)
            row_snr_extensive.append(np.nan)

        try:
            # append the row to the snr_metrics dataframe
            print(
                f"Appending row to snr_metrics dataframe for experiment: {experiment} and
                    category: {category}")
            snr_metrics.loc[len(snr_metrics)] = row_snr
            snr_metrics_extensive.loc[len(
                snr_metrics_extensive)] = row_snr_extensive
            # create the figures
            print("Creating figures for experiment: ", experiment)
            create_figure(row_snr, path_folder_results)
```

```
            create_boxplot(row_snr_extensive, path_folder_results,
                           data_per_participant_dict)

        except:
            print(
                'Error in the creation of the figure or the append of the row to the snr_metrics
                    dataframe')
            #print ('row_snr: ', row_snr)
            print('path_folder_results: ', path_folder_results)
            sys.exit()

        # Start the statistical analysis
        # perform bartlett's test
        # check number of groups
        if len(groups_in_category_array) == 3:
            # perform Bartlett's test
            print(stats.bartlett(data_analysis[groups_in_category_array[0]],
                  data_analysis[groups_in_category_array[1]], data_analysis[
                      groups_in_category_array[2]]))
            # save the p-value from the test
            p_value_bartlett = stats.bartlett(
                data_analysis[groups_in_category_array[0]], data_analysis[
                    groups_in_category_array[1]], data_analysis[groups_in_category_array[2]])[1]

        elif len(groups_in_category_array) == 4:
            # perform Bartlett's test
            print(stats.bartlett(data_analysis[groups_in_category_array[0]], data_analysis[
                groups_in_category_array[1]],
                  data_analysis[groups_in_category_array[2]], data_analysis[
                      groups_in_category_array[3]]))
            # save the p-value from the test
            p_value_bartlett = stats.bartlett(data_analysis[groups_in_category_array[0]],
                data_analysis[groups_in_category_array[1]],
                                              data_analysis[groups_in_category_array[2]],
                                                  data_analysis[groups_in_category_array[3]])[1]
        # perform bartlett's test
        alpha_bartlett = 0.05  # 95% confidence
        # compare p value with alpha (0.05). If the p value is lower than alpha, the variances are
            not equal
        if p_value_bartlett > alpha_bartlett:
            print('The variances are equal')
            variances_equal = 'Yes'
            # if the variances are equal, perform one-way ANOVA
            print("Performing one-way ANOVA")
            anova_method = 'One-way ANOVA'

            # perform one-way ANOVA
            alpha_one_way_anova = 0.05 # 95% confidence

            if len(groups_in_category_array) == 3:
                # perform one-way ANOVA
                print(stats.f_oneway(data_analysis[groups_in_category_array[0]],
                      data_analysis[groups_in_category_array[1]], data_analysis[
                          groups_in_category_array[2]]))
                # save the p-value from the test
                pvalue_f_oneway = stats.f_oneway(
```

```
            data_analysis[groups_in_category_array[0]], data_analysis[
                groups_in_category_array[1]], data_analysis[groups_in_category_array[2]])
                [1]

    elif len(groups_in_category_array) == 4:
        # perform one-way ANOVA
        print(stats.f_oneway(data_analysis[groups_in_category_array[0]], data_analysis[
            groups_in_category_array[1]],
              data_analysis[groups_in_category_array[2]], data_analysis[
                  groups_in_category_array[3]]))
        # save the p-value from the test
        pvalue_f_oneway = stats.f_oneway(data_analysis[groups_in_category_array[0]],
            data_analysis[groups_in_category_array[1]],
                                         data_analysis[groups_in_category_array[2]],
                                             data_analysis[groups_in_category_array[3]])

    # # calculate the total degrees of freedom of the f one-way anova
    # # calculate the total number of observations over all groups
    total_number_of_observations = 0
    for group_i in groups_in_category_array:
        total_number_of_observations += len(data_analysis[group_i])
    # calculate all the degrees of freedom of the f one-way anova
    df_between_groups = len(groups_in_category_array) - 1
    df_within_groups = total_number_of_observations - \
        len(groups_in_category_array)
    df_total = total_number_of_observations - 1

    # save the p-value from the test
    p_value_anova_method = pvalue_f_oneway

    # compare p value with alpha (0.05). If the p value is lower than alpha, the means are
        not equal
    if pvalue_f_oneway > alpha_one_way_anova:
        print('The means are equal')
        mean_different = 'No'
        type_of_post_hoc_test = 'None'
        p_value_matrix = 'None'
        significance_matrix = 'None'

    elif pvalue_f_oneway <= alpha_one_way_anova:
        print('The means are not equal')
        mean_different = 'Yes'
        print("Performing Tukey's test")
        type_of_post_hoc_test = "Tukey's test"

        # perform Tukey's test
        # check number of groups
        if len(groups_in_category_array) == 3:
            # perform Tukey's post-hoc test
            print(stats.tukey_hsd(data_analysis[groups_in_category_array[0]],
                    data_analysis[groups_in_category_array[1]], data_analysis[
                        groups_in_category_array[2]]))
            # save the p-value from the test
            results = stats.tukey_hsd(
                data_analysis[groups_in_category_array[0]], data_analysis[
                    groups_in_category_array[1]], data_analysis[groups_in_category_array
```

```
                    [2]])

            elif len(groups_in_category_array) == 4:
                # perform Tukey's post-hoc test
                print(stats.tukey_hsd(data_analysis[groups_in_category_array[0]],
                    data_analysis[groups_in_category_array[1]],
                      data_analysis[groups_in_category_array[2]], data_analysis[
                          groups_in_category_array[3]]))
                # save the p-value from the test
                results = stats.tukey_hsd(data_analysis[groups_in_category_array[0]],
                    data_analysis[groups_in_category_array[1]],
                                          data_analysis[groups_in_category_array[2]],
                                              data_analysis[groups_in_category_array[3]])

            # save the p-value from the test
            pvalue_matrix = results.pvalue

            alpha_tukey = 0.05 # 95% confidence
            tukey_table_significance = create_table_with_significance_using_pvalue(
                pvalue_matrix, groups_in_category_array, alpha_tukey)
            # save the p-value matrix and significance matrix
            p_value_matrix = pvalue_matrix
            significance_matrix = tukey_table_significance

            # convert the p-value matrix to a dataframe with the same index and columns as
                the significance matrix
            p_value_matrix_df = pd.DataFrame(
                p_value_matrix, index=significance_matrix.index, columns=significance_matrix.
                    columns)

            # save the p-value matrix and significance matrix to a csv file
            p_value_matrix_df.to_csv(
                f'{path_folder_results}/Tukey_p_value_matrix_{experiment}_{category}.csv',
                    index=True)
            significance_matrix.to_csv(
                f'{path_folder_results}/Tukey_significance_matrix_{experiment}_{category}.csv'
                    , index=True)

    elif p_value_bartlett <= alpha_bartlett:
        variances_equal = 'No'
        print('The variances are not equal')
        # if the variances are not equal, perform Welch's ANOVA
        print("Performing Welch's ANOVA")
        anova_method = "Welch's ANOVA"

        # check number of groups
        if len(groups_in_category_array) == 3:
            # create a dataframe with the values of the groups
            df = pd.DataFrame({'value': np.concatenate((data_analysis[
                groups_in_category_array[0]], data_analysis[groups_in_category_array[1]],
                data_analysis[groups_in_category_array[2]])), 'group': np.concatenate((np.
                repeat(groups_in_category_array[0], len(
                data_analysis[groups_in_category_array[0]])), np.repeat(
                    groups_in_category_array[1], len(data_analysis[groups_in_category_array
                    [1]])), np.repeat(groups_in_category_array[2], len(data_analysis[
                    groups_in_category_array[2]])))})
```

```
elif len(groups_in_category_array) == 4:
    # create a dataframe with the values of the groups
    df = pd.DataFrame({'value': np.concatenate((data_analysis[
        groups_in_category_array[0]], data_analysis[groups_in_category_array[1]],
        data_analysis[groups_in_category_array[2]], data_analysis[
        groups_in_category_array[3]])), 'group': np.concatenate((np.repeat(
        groups_in_category_array[0], len(
        data_analysis[groups_in_category_array[0]])), np.repeat(
            groups_in_category_array[1], len(data_analysis[groups_in_category_array
            [1]])), np.repeat(groups_in_category_array[2], len(data_analysis[
            groups_in_category_array[2]])), np.repeat(groups_in_category_array[3], len
            (data_analysis[groups_in_category_array[3]])))})

# perform Welch's ANOVA
print(pg.welch_anova(data=df, dv='value', between='group'))
# PERFORM WELCH'S ANOVA USING PINGOUIN and save the results in a variable
results = pg.welch_anova(data=df, dv='value', between='group')
# extract df between groups from the results
df_between_groups = results['ddof1'][0]
# extract the degrees of freedom within groups from the results
df_within_groups = results['ddof2'][0]
df_total = df_between_groups + df_within_groups

# extract the p-value from the results
p_value_welch_anova = results['p-unc'][0]

alpha_welch_anova = 0.05 # 95% confidence
# check if the p-value is smaller than the alpha. If it is, the mean values are
    different
if p_value_welch_anova > alpha_welch_anova:
    mean_different = 'No'
    print('The mean values are equal')
    type_of_post_hoc_test = 'None'
    p_value_matrix = 'None'
    Significance_matrix = 'None'

# if the mean values are different, perform pairwise comparisons
elif p_value_welch_anova <= alpha_welch_anova:
    mean_different = 'Yes'
    print('The mean values are not equal')
    type_of_post_hoc_test = 'Games-Howell'
    # perform THE pairwise games-howell post hoc test
    print(pg.pairwise_gameshowell(
        data=df, dv='value', between='group'))

    # create a table with the pairwise comparisons
    pairwise_comparisons = pg.pairwise_gameshowell(
        data=df, dv='value', between='group')
    alpha_pairwise_comparisons = 0.05 # 95% confidence
    table_with_p_values, p_values_with_significance =
        create_table_with_pairwise_comparisons(
        pairwise_comparisons, alpha_pairwise_comparisons)

    # save the table with the p values in a csv file data frame
    table_with_p_values.to_csv(
        f'{path_folder_results}/GamesHowell_table_with_p_values_{experiment}_{
```

```
                    category}.csv', index=True)
                # save the table with the p values with significance in a csv file data frame
                p_values_with_significance.to_csv(
                    f'{path_folder_results}/GamesHowell_p_values_with_significance_{experiment}
                        _{category}.csv', index=True)

                # save the data in the logger
                p_value_matrix = table_with_p_values
                significance_matrix = p_values_with_significance

            p_value_anova_method = p_value_welch_anova
        try:
            # round the total degrees of freedom to 2 decimal places
            df_between_groups = round(df_between_groups, 2)
            df_within_groups = round(df_within_groups, 2)
            df_total = round(df_total, 2)

            # create a row with the results
            row = [category, np.array(groups_in_category_array), p_value_bartlett, variances_equal,
                anova_method, df_between_groups, df_within_groups,
                    p_value_anova_method, mean_different, type_of_post_hoc_test, p_value_matrix,
                        significance_matrix]

            # add the row to results_logger dataframe
            results_logger.loc[len(results_logger)] = row
        except:
            print(
                f"Error with appending the row to the results_logger dataframe for the category {
                    category}")
            # stop the execution of the code
            sys.exit()

    # save the results in a csv file
    results_logger.to_csv(
        f'{path_folder_results}/results_logger_{experiment}.csv', index=False)
    # save the snr metrics in a csv file
    snr_metrics.to_csv(
        f'{path_folder_results}/snr_metrics_{experiment}.csv', index=False)


def process_data_logger(data_logger: dict):
    """ Processes the data_logger and performs the statistical analysis.
    Does the pairwise comparisons between the groups of the overlapping categories of the two
        experiments.

    Args:
        data_logger (dict): dictionary with the data of the two experiments
    """
    headers = ['category', 'groups', 'p_value_bartlett', 'Variances equal?', 'ANOVA method', 'df
        within groups', 'df between groups',
            'p_value ANOVA method', 'Mean different?', 'Type of post hoc test', '
                p_value_matrix', 'Significance_matrix']
    # create an empty dataframe to store the results of each step of the analysis
    results_logger = pd.DataFrame(columns=headers)

    # extract the unique categories from the data_logger
```

```
unique_categories_experiment_1 = list(
    data_logger['Experiment_1']['category'].keys())
unique_categories_experiment_2 = list(
    data_logger['Experiment_2']['category'].keys())
# find the overlapping categories between the two experiments
unique_categories_all = list(set(unique_categories_experiment_1).intersection(
    unique_categories_experiment_2))
n_number_of_experiments = 2
for category in unique_categories_all:
    # extract the groups from the data_logger
    groups_experiment_1 = list(
        data_logger['Experiment_1']['category'][category].keys())
    groups_experiment_2 = list(
        data_logger['Experiment_2']['category'][category].keys())
    # find the overlapping groups between the two experiments
    groups_all = list(
        set(groups_experiment_1).intersection(groups_experiment_2))

    # create a dictionary containing the data of the groups from A to B
    data_experiment_1 = {}
    data_experiment_2 = {}
    for group in groups_all:
        data_experiment_1[group] = data_logger['Experiment_1']['category'][category][group]
        data_experiment_2[group] = data_logger['Experiment_2']['category'][category][group]
        print('
            ----------------------------------------------------------------------
            ')
        print(f'Category: {category}')
        print('
            ----------------------------------------------------------------------
            ')
        print(f"groups all: {groups_all}")

        # perform bartlett's test
        alpha_bartlett = 0.05  # 95% confidence
        # check number of groups

        # perform Bartlett's test for the group between the two experiments
        print(stats.bartlett(
            data_experiment_1[group]['data'], data_experiment_2[group]['data']))

        p_value_bartlett = stats.bartlett(
            data_experiment_1[group]['data'], data_experiment_2[group]['data'])[1]

        # if the p value is less than the alpha, the variances are not equal
        if p_value_bartlett > alpha_bartlett:
            print('The variances are equal')
            variances_equal = 'Yes'
            # if the variances are equal, perform one-way ANOVA
            print("Performing one-way ANOVA")
            anova_method = 'One-way ANOVA'

            # perform one-way ANOVA
            print(stats.f_oneway(
                data_experiment_1[group]['data'], data_experiment_2[group]['data']))
            pvalue_f_oneway = stats.f_oneway(
```

```
        data_experiment_1[group]['data'], data_experiment_2[group]['data'])[1]

alpha_one_way_anova = 0.05 # 95% confidence

# calculate the total number of observations over all groups
total_number_of_observations = 0
total_number_of_observations += len(
    data_experiment_1[group]['data'])
total_number_of_observations += len(
    data_experiment_2[group]['data'])

# calculate all the degrees of freedom of the f one-way anova
df_between_groups = n_number_of_experiments - 1
df_within_groups = total_number_of_observations - n_number_of_experiments
df_total = total_number_of_observations - 1

# save the p value of the one-way ANOVA
p_value_anova_method = pvalue_f_oneway
# if the p value is less than the alpha, the means are not equal
if pvalue_f_oneway > alpha_one_way_anova:
    print('The means are equal')
    mean_different = 'No'
    type_of_post_hoc_test = 'None'
    p_value_matrix = 'None'
    significance_matrix = 'None'

elif pvalue_f_oneway <= alpha_one_way_anova:
    print('The means are not equal')
    mean_different = 'Yes'
    print("Performing Tukey's test")
    anova_method = 'Tukey\'s test'

    # perform Tukey's post-hoc test
    print(stats.tukey_hsd(
        data_experiment_1[group]['data'], data_experiment_2[group]['data']))
    # save the p-value from the test
    results = stats.tukey_hsd(
        data_experiment_1[group]['data'], data_experiment_2[group]['data'])

    pvalue_matrix = results.pvalue
    # Alpha is the significance level at which we reject the null hypothesis
    alpha_tukey = 0.05 # 95% confidence
    label_groups = [
        f"Experiment 1: {group}", f"Experiment 2: {group}"]
    # create a dataframe determining the significance of the pairwise comparisons
    tukey_table_significance = create_table_with_significance_using_pvalue(
        pvalue_matrix, label_groups, alpha_tukey)

    # log the results of the analysis
    p_value_matrix = pvalue_matrix
    significance_matrix = tukey_table_significance
    # save the matrices of the analysis
    p_value_matrix_df.to_csv(
        f'{path_folder_results}/Tukey_p_value_matrix_{category}_{group}.csv',
            index=True)
    significance_matrix.to_csv(
```

```
                f'{path_folder_results}/Tukey_significance_matrix_{category}_{group}.csv',
                    index=True)

        # if the variances are not equal, perform Welch's ANOVA
        elif p_value_bartlett <= alpha_bartlett:
            variances_equal = 'No'
            print('The variances are not equal')
            # if the variances are not equal, perform Welch's ANOVA
            print("Performing Welch's ANOVA")
            anova_method = "Welch's ANOVA"
            # create a dataframe containing the data from the two experiments and the group
                name for welch's ANOVA
            df = pd.DataFrame({'value': np.concatenate((data_experiment_1[group]['data'],
                data_experiment_2[group]['data'])),
                                'group': np.concatenate((np.repeat(f"Experiment 1: {group}", len
                                    (data_experiment_1[group]['data'])),
                                                         np.repeat(f"Experiment 2: {group}", len
                                                             (data_experiment_2[group]['data']))
                                                             ))})

            # perform Welch's ANOVA
            print(pg.welch_anova(data=df, dv='value', between='group'))
            # PERFORM WELCH'S ANOVA USING PINGOUIN and save the results in a
                variable
            results = pg.welch_anova(data=df, dv='value', between='group')
            # extract df between groups from the results
            df_between_groups = results['ddof1'][0]
            # extract the degrees of freedom within groups from the results
            df_within_groups = results['ddof2'][0]
            df_total = df_between_groups + df_within_groups

            # extract the degrees of freedom from the results which
            p_value_welch_anova = results['p-unc'][0]
            alpha_welch_anova = 0.05 # 95% confidence

            if p_value_welch_anova > alpha_welch_anova:
                mean_different = 'No'
                print('The mean values are equal')
                type_of_post_hoc_test = 'None'
                p_value_matrix = 'None'
                significance_matrix = 'None'

            elif p_value_welch_anova <= alpha_welch_anova:
                mean_different = 'Yes'
                print('The mean values are not equal')
                type_of_post_hoc_test = 'Games-Howell'
                # PERFROM THE pairwise games-howell post hoc test
                print(pg.pairwise_gameshowell(
                    data=df, dv='value', between='group'))

                # create a table with the pairwise comparisons
                pairwise_comparisons = pg.pairwise_gameshowell(
                    data=df, dv='value', between='group')
                alpha_pairwise_comparisons = 0.05 # 95% confidence
                table_with_p_values, p_values_with_significance =
                    create_table_with_pairwise_comparisons(
```

```
                    pairwise_comparisons, alpha_pairwise_comparisons)

                # save the table with the p values in a csv file data frame
                table_with_p_values.to_csv(
                    f'{path_folder_results}/GamesHowell_table_with_p_values_{category}_{
                        group}.csv', index=True)
                # save the table with the p values with significance in a csv file data frame
                p_values_with_significance.to_csv(
                    f'{path_folder_results}/GamesHowell_p_values_with_significance_{
                        category}_{group}.csv', index=True)
                p_value_matrix = table_with_p_values
                significance_matrix = p_values_with_significance

            # save the p-value from the test
            p_value_anova_method = p_value_welch_anova

        try:
            # round the total degrees of freedom to 2 decimal places
            df_between_groups = round(df_between_groups, 2)
            df_within_groups = round(df_within_groups, 2)
            df_total = round(df_total, 2)

            # create a row with the results of the analysis
            row = [category, group, p_value_bartlett, variances_equal, anova_method,
                df_between_groups, df_within_groups, p_value_anova_method,
                    mean_different, type_of_post_hoc_test, p_value_matrix, significance_matrix]

            # add the row to results_logger dataframe
            results_logger.loc[len(results_logger)] = row
        except:
            print(
                f" Error in {category} {group} with {anova_method} and trying to save the
                    results in the results_logger dataframe")
            sys.exit()

    # save the results in a csv file
    results_logger.to_csv(
        f'{path_folder_results}/results_logger_exp1_vs_exp2.csv', index=False)


process_data_logger(data_logger)
```

D.2.6. SNR_statistic_4d_plot_v1.py

Creates the 4D plots.

```
import numpy as np
import os
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from collections import OrderedDict


def create_combinations_of_categories(categories: np.ndarray) -> list:
    """ Create all possible combinations of the categories
```

```
    Args:
        categories (np.ndarray): The categories to be combined

    Returns:
        list: All possible combinations of the categories
    """
    # create all possible combinations of the categories with the main category being frequency
        and two subcategories
    main_category = 'frequency'
    # filter out the categories that or not the main category
    subcategories = [
        category for category in categories if category != main_category]
    # find all possbile combinations between the categories
    combinations_of_categories = []
    for i in range(len(subcategories)):
        for j in range(i+1, len(subcategories)):
            combinations_of_categories.append(
                [main_category, subcategories[i], subcategories[j]])

    return combinations_of_categories


def create_combinations_of_groups(dictionary_of_categories_with_groups: dict) -> list:
    """ Create all possible combinations of the groups

    Args:
        dictionary_of_categories_with_groups (dict): The dictionary of categories with groups

    Returns:
        list: All possible combinations of the groups
    """
    # unpack the categories keys from the dictionary
    categories = list(
        dictionary_of_categories_with_groups['categories'].keys())
    # create all possible combinations of the groups between the categories
    group_settings_of_categories = dictionary_of_categories_with_groups['categories']
    # category 1 is the main category and category 2 and 3 are the subcategories
    # find all possbile combinations between the groups of the categories
    combinations_of_groups = []
    for group_1 in group_settings_of_categories[categories[0]]:
        for group_2 in group_settings_of_categories[categories[1]]:
            for group_3 in group_settings_of_categories[categories[2]]:
                combinations_of_groups.append([group_1, group_2, group_3])

    return combinations_of_groups


def create_4d_plot(dict_of_combinations_with_data: dict, groups_per_experiment_dict: dict,
    unique_experiments: np.ndarray, path_folder_results: str):
    """ Create 3d plots for the SNR statistic

    Args:
        dict_of_combinations_with_data (dict): Dictionary of combinations with data
        groups_per_experiment_dict (dict): Dictionary of groups per experiment
        unique_experiments (np.ndarray): Unique experiments
        path_folder_results (str): Path to the folder where the results are saved
```

```
    """
    # get the name of the experiments
    experiments = list(dict_of_combinations_with_data.keys())
    for experiment in experiments:
        # unpack the dict of combinations with data for the experiment
        experiment_data = dict_of_combinations_with_data[experiment]
        # find the keys in the dict of combinations with data
        combinations_categories = list(experiment_data.keys())
        for combination_of_categories in combinations_categories:
            # split the combination of categiries by '-'
            combination_of_categories_seperated = combination_of_categories.split(
                '-')
            # unpack the dict of combinations with data for the combination of categories except
                the key categories
            combination_of_categories_data = experiment_data[combination_of_categories]
            # unpacking values for x axis from the categories dict
            three_categories = combination_of_categories_data['categories']
            three_categories_keys = list(three_categories.keys())
            # list all the keys in the dict of combinations with data for the combination of
                categories
            combinations_groups = list(combination_of_categories_data.keys())
            # filter out the key categories
            combinations_groups = [
                combination for combination in combinations_groups if combination != 'categories']
            # types of frequencies
            frequencies = three_categories[three_categories_keys[0]]
            # create a figure
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            # label the x axis SNR
            ax.set_zlabel('SNR [dB]')

            # set the x axis ticks
            if three_categories_keys[1] == 'pixel_surface':
                ax.set_xlabel('pixel surface [pixels]')
                x_values_axis = [10000, 20000, 30000]
                # set the x axis ticks
                ax.set_xticks(x_values_axis)
                str_add = 'pixels'

            elif three_categories_keys[1] == 'color':
                ax.set_xlabel('color')
                # count the number of colors
                number_of_colors = len(
                    three_categories[three_categories_keys[1]])
                xticks = np.arange(number_of_colors)
                # set the x axis ticks
                x_values_axis = three_categories[three_categories_keys[1]]
                ax.set_xticks(xticks)
                ax.set_xticklabels(x_values_axis)
                str_add = ''

            elif three_categories_keys[1] == 'shape':
                ax.set_xlabel('shape')
                # count the number of shapes
                number_of_shapes = len(
```

```
            three_categories[three_categories_keys[1]])
        xticks  = np.arange(number_of_shapes)
        # set the x axis ticks
        x_values_axis = three_categories[three_categories_keys[1]]
        ax.set_xticks(xticks)
        ax.set_xticklabels(x_values_axis)
        str_add = ''

    # label the y axis
    if three_categories_keys[2] == 'pixel_surface':
        # label the y axis
        ax.set_ylabel('pixel surface [pixels]')
        y_values_axis = [10000, 20000, 30000]
        # set the y axis ticks
        ax.set_yticks(y_values_axis)
        str_add = 'pixels'

    elif three_categories_keys[2] == 'color':
        # label the y axis
        ax.set_ylabel('color')
        # count the number of colors
        number_of_colors = len(
            three_categories[three_categories_keys[2]])
        yticks  = np.arange(number_of_colors)
        y_values_axis = three_categories[three_categories_keys[2]]
        # set the y axis ticks
        ax.set_yticks(yticks)
        ax.set_yticklabels(y_values_axis)
        str_add = ''

    elif three_categories_keys[2] == 'shape':
        # label the y axis
        ax.set_ylabel('shape')
        # count the number of shapes
        number_of_shapes = len(
            three_categories[three_categories_keys[2]])
        yticks  = np.arange(number_of_shapes)
        y_values_axis = three_categories[three_categories_keys[2]]
        # set the y axis ticks
        ax.set_yticks(yticks)
        ax.set_yticklabels(y_values_axis)
        str_add = ''

    # create a look up dict for the x axis
    if three_categories_keys[2] != 'pixel_surface':
        look_up_dict_y_axis = {}
        for i in range(len(y_values_axis)):
            look_up_dict_y_axis[y_values_axis[i]] = i
    # create a look up dict for the y axis
    if three_categories_keys[1] != 'pixel_surface':
        look_up_dict_x_axis = {}
        for i in range(len(x_values_axis)):
            look_up_dict_x_axis[x_values_axis[i]] = i

    # for each combination of settings in the combination of categories
    for combination_of_settings in combinations_groups:
```

```
            # split the combination of settings by '-'
            combination_of_settings_seperated = combination_of_settings.split(
                '-')
            # unpack the data for the combination of settings
            x_value = combination_of_settings_seperated[1]
            y_value = combination_of_settings_seperated[2]
            # try to convert the x value to an int
            try:
                x_value = int(x_value)
            except ValueError:
                # look up the x value in the look up dict
                x_value = look_up_dict_x_axis[x_value]
            # try to convert the y value to an int
            try:
                y_value = int(y_value)
            except ValueError:
                # look up the y value in the look up dict
                y_value = look_up_dict_y_axis[y_value]

            # for each frequency use a different color and label
            if int(combination_of_settings_seperated[0]) == frequencies[0]:
                color = 'red'
                label = 'frequency = ' + str(frequencies[0]) + ' Hz'
            elif int(combination_of_settings_seperated[0]) == frequencies[1]:
                color = 'blue'
                label = 'frequency = ' + str(frequencies[1]) + ' Hz'
                y_value = y_value + 0.15
            elif int(combination_of_settings_seperated[0]) == frequencies[2]:
                color = 'green'
                label = 'frequency = ' + str(frequencies[2]) + ' Hz'
                y_value = y_value + 0.30
            elif int(combination_of_settings_seperated[0]) == frequencies[3]:
                color = 'black'
                label = 'frequency = ' + str(frequencies[3]) + ' Hz'
                y_value = y_value + 0.45

            # unpack the dict of combinations with data for the combination of settings
            combination_of_settings_data = combination_of_categories_data[
                combination_of_settings]
            # unpack the mean snr and std from the dict of combinations with data for the
                combination of settings
            snr = combination_of_settings_data['mean']
            std = combination_of_settings_data['std']
            z_value = snr
            # show mean snr as bar
            ax.bar3d(x_value, y_value, 0, 0.01, 0.1, snr,
                     color=color, alpha=0.35, label=label)
            # show std error bars
            ax.plot([x_value, x_value], [y_value, y_value], [
                    z_value-std, z_value+std], color=color, alpha=1.0, marker='_', label=label)

    # set zlim between -3 and 15
    ax.set_zlim(-3, 15)
    ax.view_init(35, 136)
    # get the legend handles and labels
    handles, labels = ax.get_legend_handles_labels()
```

```
            # create a dict of the legend handles and labels
            by_label = OrderedDict(zip(labels, handles))
            # set the legend
            plt.legend(by_label.values(), by_label.keys(
            ), loc='upper left', bbox_to_anchor=(0.7, 1.0), borderaxespad=0.)
            experiment = experiment.replace('_', ' ')
            category_1 = three_categories_keys[0].replace('_', ' ')
            category_2 = three_categories_keys[1].replace('_', ' ')
            category_3 = three_categories_keys[2].replace('_', ' ')
            # set the title of the plot
            plt.title(experiment + ': ' + category_1 +
                      ' vs ' + category_2 + ' vs ' + category_3)
            # make sure legend fits on tight layout
            plt.tight_layout()
            # save the plot
            plt.savefig(path_folder_results + '/' + experiment + '_' +
                        three_categories_keys[0] + '_' + three_categories_keys[1] + '_' +
                            three_categories_keys[2] + str_add + '.png')


def heatmap(data, row_labels, col_labels, ax=None,
            cbar_kw=None, cbarlabel="", **kwargs):
    """
    Create a heatmap from a numpy array and two lists of labels.

    Parameters
    ----------
    data
        A 2D numpy array of shape (M, N).
    row_labels
        A list or array of length M with the labels for the rows.
    col_labels
        A list or array of length N with the labels for the columns.
    ax
        A `matplotlib.axes.Axes` instance to which the heatmap is plotted. If
        not provided, use current axes or create a new one. Optional.
    cbar_kw
        A dictionary with arguments to `matplotlib.Figure.colorbar`. Optional.
    cbarlabel
        The label for the colorbar. Optional.
    **kwargs
        All other arguments are forwarded to `imshow`.
    """

    if ax is None:
        ax = plt.gca()

    if cbar_kw is None:
        cbar_kw = {}

    # Plot the heatmap
    im = ax.imshow(data, **kwargs)
    # check if in kwargs is a max value for the colorbar
    if 'vmax' in kwargs:
        # if so, set the colorbar to that value
        im.set_clim(0, kwargs['vmax'])
```

```
    # Create colorbar
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

    # Show all ticks and label them with the respective list entries.
    ax.set_xticks(np.arange(data.shape[1]), labels=col_labels)
    ax.set_yticks(np.arange(data.shape[0]), labels=row_labels)

    # Let the horizontal axes labeling appear on top.
    ax.tick_params(top=True, bottom=False,
                   labeltop=True, labelbottom=False)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=-30, ha="right",
             rotation_mode="anchor")

    # Turn spines off and create white grid.
    ax.spines[:].set_visible(False)

    ax.set_xticks(np.arange(data.shape[1]+1)-.5, minor=True)
    ax.set_yticks(np.arange(data.shape[0]+1)-.5, minor=True)
    ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
    ax.tick_params(which="minor", bottom=False, left=False)

    return im, cbar


def annotate_heatmap(im, data=None, valfmt="{x:.2f}",
                     textcolors=("black", "white"),
                     threshold=None, **textkw):
    """
    A function to annotate a heatmap.

    Parameters
    ----------
    im
        The AxesImage to be labeled.
    data
        Data used to annotate. If None, the image's data is used. Optional.
    valfmt
        The format of the annotations inside the heatmap. This should either
        use the string format method, e.g. "$ {x:.2f}", or be a
        `matplotlib.ticker.Formatter`. Optional.
    textcolors
        A pair of colors. The first is used for values below a threshold,
        the second for those above. Optional.
    threshold
        Value in data units according to which the colors from textcolors are
        applied. If None (the default) uses the middle of the colormap as
        separation. Optional.
    **kwargs
        All other arguments are forwarded to each call to `text` used to create
        the text labels.
    """

    if not isinstance(data, (list, np.ndarray)):
```

```
            data = im.get_array()

        # Normalize the threshold to the images color range.
        if threshold is not None:
            threshold = im.norm(threshold)
        else:
            threshold = im.norm(data.max())/2.

        # Set default alignment to center, but allow it to be
        # overwritten by textkw.
        kw = dict(horizontalalignment="center",
                  verticalalignment="center")
        kw.update(textkw)

        # Get the formatter in case a string is supplied
        if isinstance(valfmt, str):
            valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)

        # Loop over the data and create a 'Text' for each "pixel".
        # Change the text's color depending on the data.
        texts = []
        for i in range(data.shape[0]):
            for j in range(data.shape[1]):
                kw.update(color=textcolors[int(im.norm(data[i, j]) > threshold)])
                text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
                texts.append(text)

        return texts


def create_boxplot(dict_of_combinations_with_data: dict, groups_per_experiment_dict: dict,
                   unique_experiments: np.ndarray, path_folder_results: str):
    """ Create 3d plots for the SNR statistic

    Args:
        dict_of_combinations_with_data (dict): Dictionary of combinations with data
        groups_per_experiment_dict (dict): Dictionary of groups per experiment
        unique_experiments (np.ndarray): Unique experiments
        path_folder_results (str): Path to the folder where the results are saved
    """
    # get the name of the experiments
    experiments = list(dict_of_combinations_with_data.keys())

    for experiment in experiments:
        # unpack the dict of combinations with data for the experiment
        experiment_data = dict_of_combinations_with_data[experiment]
        # find the keys in the dict of combinations with data
        combinations_categories = list(experiment_data.keys())

        for combination_of_categories in combinations_categories:
            # split the combination of categiries by '-'
            combination_of_categories_seperated = combination_of_categories.split(
                '-')
            # unpack the dict of combinations with data for the combination of categories except
                the key categories
            combination_of_categories_data = experiment_data[combination_of_categories]
```

```
# unpacking values for x axis from the categories dict
three_categories = combination_of_categories_data['categories']
three_categories_keys = list(three_categories.keys())
# list all the keys in the dict of combinations with data for the combination of
    categories
combinations_groups = list(combination_of_categories_data.keys())
# filter out the key categories
combinations_groups = [
    combination for combination in combinations_groups if combination != 'categories']

# types of frequencies
frequencies = three_categories[three_categories_keys[0]]
for frequency in frequencies:
    # create a figure with 3 subplots
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

    # extract the groups of the last category
    groups = groups_per_experiment_dict[experiment][three_categories_keys[2]]

    # for each subplot set the groups variables as title
    ax1.set_title(str(groups[0]))
    ax2.set_title(str(groups[1]))
    ax3.set_title(str(groups[2]))

    # set the title of the plot
    # remove the "_" from experiment name
    experiment_title = experiment.replace('_', ' ')
    combination_of_categories_title = combination_of_categories.replace(
        '_', ' ')

    title = 'Boxplot: ' + experiment_title + ': ' + three_categories_keys[0] + ' = ' +
        str(
        frequency) + ' Hz, ' + combination_of_categories_title + ' without outliers'
    fig.suptitle(title)
    print("Creating boxplot for: ", title)

    # set the x axis ticks
    if three_categories_keys[1] == 'pixel_surface':
        ax1.set_xlabel('pixel surface [pixels]')
        ax2.set_xlabel("pixel surface [pixels]")
        ax3.set_xlabel("pixel surface [pixels]")
        # count the number of pixel surfaces
        number_of_sizes = len(
            three_categories[three_categories_keys[1]])
        xticks = np.arange(number_of_sizes)
        # set the x axis ticks labels
        x_values_axis = three_categories[three_categories_keys[1]]
        x_values_axis = [str(x) for x in x_values_axis]
        str_add = 'pixels'

    elif three_categories_keys[1] == 'color':
        ax1.set_xlabel('color')
        ax2.set_xlabel('color')
        ax3.set_xlabel('color')
        # count the number of colors
```

```
            number_of_colors = len(
                three_categories[three_categories_keys[1]])
            xticks = np.arange(number_of_colors)
            # set the x axis ticks labels
            x_values_axis = three_categories[three_categories_keys[1]]
            x_values_axis = [str(x) for x in x_values_axis]
            str_add = ' '

        elif three_categories_keys[1] == 'shape':
            ax1.set_xlabel('shape')
            ax2.set_xlabel('shape')
            ax3.set_xlabel('shape')
            # count the number of shapes
            number_of_shapes = len(
                three_categories[three_categories_keys[1]])
            xticks = np.arange(number_of_shapes)
            # set the x axis ticks labels
            x_values_axis = three_categories[three_categories_keys[1]]
            x_values_axis = [str(x) for x in x_values_axis]
            str_add = ' '

        # filter the combinations groups for the target frequency
        filtered_combinations_groups_target_frequency = [
            combination for combination in combinations_groups if combination.split('-')[0]
                == str(frequency)]
        for combination_of_settings in filtered_combinations_groups_target_frequency:
            # split the combination of settings by '-'
            combination_of_settings_seperated = combination_of_settings.split(
                '-')
            x_value_lookup = combination_of_settings_seperated[1]
            # find the index of the x value in the x values axis
            x_values_axis = np.array(x_values_axis)
            x_value_index = np.where(
                x_value_lookup == x_values_axis)[0][0]
            x_values_axis = list(x_values_axis)
            # get the look up key for the current combination of settings to determine the
                correct subplot
            look_up_key = combination_of_settings_seperated[2]
            plot_number = np.where(look_up_key == groups)[0][0]

            # set the correct subplot
            if plot_number == 0:
                ax = ax1
            elif plot_number == 1:
                ax = ax2
            elif plot_number == 2:
                ax = ax3

            # get the data for the current combination of settings
            dict_of_combination = dict_of_combinations_with_data[experiment][
                combination_of_categories][combination_of_settings]

            # get the data
            data = dict_of_combination['data']
            mean = dict_of_combination['mean']
            std = dict_of_combination['std']
```

```
# create the boxplot
bplot = ax.boxplot(data, positions=[x_value_index], widths=0.6, showmeans=
    True, meanline=True,
                   showfliers=False, patch_artist=True, medianprops=dict(
                       color='blue'), meanprops=dict(color='red'), zorder=1)
# set the color of the boxplot to be filled the color light blue
for patch in bplot['boxes']:
    patch.set_facecolor('lightblue')
# extract the participant specific data
participant_specific_data = dict_of_combination['participant_specific_data']
# extract the participant specific keys
participant_specific_keys = list(
    participant_specific_data.keys())

# create a list of colors for the legend
colors = ['green', 'purple',
          'brown', 'grey', 'olive', 'cyan']
# create a list of labels for the legend
labels = []
# create a list of handles for the legend
handles = []

# measure the number of participants and divide 1 by the number of participants
    to get the stepsize in the x axis
stepsize = 1 / len(participant_specific_keys)

# sort the participant specific keys in ascending order
participant_specific_keys = sorted(
    participant_specific_keys, key=lambda x: int(x.split('pp')[1]))
for i, participant in enumerate(participant_specific_keys):
    # remove the pp from the participant key and correct it. pp5 is now pp1
    participant_number = int(participant[2:])
    # get the participant specific data
    participant_data = participant_specific_data[participant]
    # get the mean and std of the participant
    participant_mean = np.mean(participant_data)
    participant_std = np.std(participant_data)

    # create a label for the legend
    label = f'pp {participant_number} mean'
    # add the label to the list of labels
    labels.append(label)
    # create a handle for the legend
    handles.append(plt.Line2D(
        [0], [0], color=colors[i], lw=4))

    # show the lines of the mean of each participant in the boxplot within
        each box with an alpha of 0.5 at the correct position with respect to
        the x axis
    ax.plot([x_value_index-0.5 + i*stepsize, x_value_index - 0.4 + i*stepsize],
        [
            participant_mean, participant_mean], color=colors[i], lw=3, zorder
            =2)

# set the x axis ticks
```

```
                ax.set_ylabel('SNR [dB]')
                ax.yaxis.grid(True)

                # create vertical lines at the x axis ticks positions of the x axis
                for x in xticks:
                    ax.axvline(x-0.5, color='black',
                               linestyle ='-', linewidth=0.5)

                # create a legend for the mean which is color red and the median which is color
                    blue
                labels.append(f'mean of all data')
                labels.append(f'median of all data')

                # create handles for the legend
                # create a red line for the mean
                handles.append(plt.Line2D([0], [0], color='red', lw=4))
                # create a blue line for the median
                handles.append(plt.Line2D([0], [0], color='blue', lw=4))

                # use subplots adjust to set the space between the subplots
                plt.subplots_adjust(wspace=1.2)
                # create a custom legend with the labels and the colors for each subplot
                    located exactly in the upper right corner
                ax.legend(handles, labels, loc='upper right',
                          bbox_to_anchor=(1.9, 1.0))

            # set the x axis ticks and labels
            plt.setp([ax1, ax2, ax3], xticks=xticks,
                     xticklabels=x_values_axis)

            # save the plt image
            plt.savefig(path_folder_results + '/boxplot_' + experiment + '_' +
                        combination_of_categories + '_' + str(frequency) + 'Hz' + '.png', dpi
                            =1000, bbox_inches='tight')


def create_heatmap_plot(dict_of_combinations_with_data: dict, groups_per_experiment_dict: dict,
    unique_experiments: np.ndarray, path_folder_results: str):
    """ Create 3d plots for the SNR statistic

    Args:
        dict_of_combinations_with_data (dict): Dictionary of combinations with data
        groups_per_experiment_dict (dict): Dictionary of groups per experiment
        unique_experiments (np.ndarray): Unique experiments
        path_folder_results (str): Path to the folder where the results are saved
    """
    # get the name of the experiments
    experiments = list(dict_of_combinations_with_data.keys())

    max_value_mean_snr = 0
    max_value_std_snr = 0

    for experiment in experiments:
        # unpack the dict of combinations with data for the experiment
        experiment_data = dict_of_combinations_with_data[experiment]
        # find the keys in the dict of combinations with data
```

```
combinations_categories = list(experiment_data.keys())

for combination_of_categories in combinations_categories:
    # split the combination of categiries by '-'
    combination_of_categories_seperated = combination_of_categories.split(
        '-')
    # unpack the dict of combinations with data for the combination of categories except
        the key categories
    combination_of_categories_data = experiment_data[combination_of_categories]

    # unpacking values for x axis from the categories dict
    three_categories = combination_of_categories_data['categories']
    three_categories_keys = list(three_categories.keys())
    # list all the keys in the dict of combinations with data for the combination of
        categories
    combinations_groups = list(combination_of_categories_data.keys())
    # filter out the key categories
    combinations_groups = [
        combination for combination in combinations_groups if combination != 'categories']
    # types of frequencies

    frequencies = three_categories[three_categories_keys[0]]
    for frequency in frequencies:
        # create two heatmaps for each frequency in the experiment. One for the mean and
            one for the std

        # create a figures
        fig = plt.figure()
        # create a subplot for the mean
        ax1 = fig.add_subplot(121)
        # create a subplot for the std
        ax2 = fig.add_subplot(122)
        # set the title of the plot
        # remove the "_" from experiment name
        experiment_title = experiment.replace('_', ' ')
        combination_of_categories_title = combination_of_categories.replace(
            '_', ' ')
        title = experiment_title + ': ' + three_categories_keys[0] + ' = ' + str(
            frequency) + ' Hz, ' + combination_of_categories_title
        plt.suptitle(title)
        # set the title of the mean plot
        ax1.set_title('Mean SNR [dB]')
        # set the title of the std plot
        ax2.set_title('Std SNR [dB]')
        # set the x axis ticks
        if three_categories_keys[1] == 'pixel_surface':
            ax1.set_xlabel('pixel surface [pixels]')
            ax2.set_xlabel("pixel surface [pixels]")
            x_values_axis = [10000, 20000, 30000]
            # set the x axis ticks
            ax1.set_xticks(x_values_axis)
            ax2.set_xticks(x_values_axis)
            str_add = 'pixels'

        elif three_categories_keys[1] == 'color':
            ax1.set_xlabel('color')
```

```
            ax2.set_xlabel('color')
            # count the number of colors
            number_of_colors = len(
                three_categories[three_categories_keys[1]])
            xticks = np.arange(number_of_colors)
            # set the x axis ticks
            x_values_axis = three_categories[three_categories_keys[1]]
            ax1.set_xticks(xticks)
            ax2.set_xticks(xticks)
            ax1.set_xticklabels(x_values_axis)
            ax2.set_xticklabels(x_values_axis)
            str_add = ''

        elif three_categories_keys[1] == 'shape':
            ax1.set_xlabel('shape')
            ax2.set_xlabel('shape')
            # count the number of shapes
            number_of_shapes = len(
                three_categories[three_categories_keys[1]])
            xticks = np.arange(number_of_shapes)
            # set the x axis ticks
            x_values_axis = three_categories[three_categories_keys[1]]
            ax1.set_xticks(xticks)
            ax2.set_xticks(xticks)
            ax1.set_xticklabels(x_values_axis)
            ax2.set_xticklabels(x_values_axis)
            str_add = ''

        # label the y axis
        if three_categories_keys[2] == 'pixel_surface':
            # label the y axis
            ax1.set_ylabel('pixel surface [pixels]')
            ax2.set_ylabel('pixel surface [pixels]')
            y_values_axis = [10000, 20000, 30000]
            # set the y axis ticks
            ax1.set_yticks(y_values_axis)
            ax2.set_yticks(y_values_axis)
            str_add = 'pixels'

        elif three_categories_keys[2] == 'color':
            # label the y axis
            ax1.set_ylabel('color')
            ax2.set_ylabel('color')
            # count the number of colors
            number_of_colors = len(
                three_categories[three_categories_keys[2]])
            yticks = np.arange(number_of_colors)
            y_values_axis = three_categories[three_categories_keys[2]]
            # set the y axis ticks
            ax1.set_yticks(yticks)
            ax2.set_yticks(yticks)
            ax1.set_yticklabels(y_values_axis)
            ax2.set_yticklabels(y_values_axis)
            str_add = ''

        elif three_categories_keys[2] == 'shape':
```

```
        # label the y axis
        ax1.set_ylabel('shape')
        ax2.set_ylabel('shape')
        # count the number of shapes
        number_of_shapes = len(
            three_categories[three_categories_keys[2]])
        yticks = np.arange(number_of_shapes)
        y_values_axis = three_categories[three_categories_keys[2]]
        # set the y axis ticks
        ax1.set_yticks(yticks)
        ax2.set_yticks(yticks)
        ax1.set_yticklabels(y_values_axis)
        ax2.set_yticklabels(y_values_axis)
        str_add = ''

    # create a look up dict for the x axis
    look_up_dict_x_axis = {}
    for i in range(len(x_values_axis)):
        look_up_dict_x_axis[str(x_values_axis[i])] = i
    # create a look up dict for the y axis
    look_up_dict_y_axis = {}
    for i in range(len(y_values_axis)):
        look_up_dict_y_axis[str(y_values_axis[i])] = i

    # create an empty array for the mean values
    mean_values = np.zeros(
        (len(x_values_axis), len(y_values_axis)))
    # create an empty array for the std values
    std_values = np.zeros((len(x_values_axis), len(y_values_axis)))

    # filter the combinations groups for the target frequency
    filtered_combinations_groups_target_frequency = [
        combination for combination in combinations_groups if combination.split('-')[0]
            == str(frequency)]
    for combination_of_settings in filtered_combinations_groups_target_frequency:
        # split the combination of settings by '-'
        combination_of_settings_seperated = combination_of_settings.split(
            '-')
        # unpack the data for the combination of settings
        x_value_index = combination_of_settings_seperated[1]
        y_value_index = combination_of_settings_seperated[2]

        # look up the x value in the look up dict
        x_value = look_up_dict_x_axis[x_value_index]
        # look up the y value in the look up dict
        y_value = look_up_dict_y_axis[y_value_index]

        # unpack the dict of combinations with data for the combination of settings
        combination_of_settings_data = combination_of_categories_data[
            combination_of_settings]
        # unpack the mean snr and std from the dict of combinations with data for the
            combination of settings
        snr_mean = combination_of_settings_data['mean']
        snr_std = combination_of_settings_data['std']
        # add the mean and std to the arrays
        mean_values[x_value, y_value] = snr_mean
```

```
                    std_values[x_value, y_value] = snr_std
                    if snr_mean > max_value_mean_snr:
                        max_value_mean_snr = snr_mean
                    if snr_std > max_value_std_snr:
                        max_value_std_snr = snr_std

                # plot the mean values using the heatmap function
                image_mean, cbar_mean = heatmap(mean_values, x_values_axis, y_values_axis,
                    ax=ax1,
                                                cmap="YlGn", cbarlabel="SNR [dB]", vmax=7)
                # annotate the heatmap with the mean values
                annotate_heatmap(image_mean, valfmt="{x:.2f}")

                # plot the std values using the heatmap function
                image_std, cbar_mean = heatmap(std_values, x_values_axis, y_values_axis, ax=
                    ax2,
                                               cmap="YlGn", cbarlabel="SNR [dB]", vmax=9)
                # annotate the heatmap with the std values
                annotate_heatmap(image_std, valfmt="{x:.2f}")

                plt.tight_layout()
                # save the plt image
                plt.savefig(path_folder_results + '/' + 'heatmap_{experiment}_{frequency}_{
                    category1}_{category2}.png'.format(
                    experiment=experiment, frequency=frequency, category1=
                        three_categories_keys[1], category2=three_categories_keys[2]))
    print(f'finished plotting the heatmaps for the experiment {experiment}')
    print(f'max_value_mean_snr: {max_value_mean_snr}')
    print(f'max_value_std_snr: {max_value_std_snr}')


def convert_dataframe_strings_to_list_SNR(df: pd.DataFrame) -> pd.DataFrame:
    """ Convert the strings in the dataframe to list of floats

    Args:
        df (pd.DataFrame): dataframe with the strings to convert

    returns:
        df_new (pd.DataFrame): dataframe with the converted strings
    """
    # make an empty dataframe that will be filled with the data
    df_new = pd.DataFrame(columns=df.columns)
    # loop over columns

    for j in range(df.shape[0]):
        row = []
        # copy of row j the first 6 columns
        row = list(df.iloc[j, 0:6].copy())

        for i in range(6, df.shape[1]):
            # remove first and last character of the string
            string = df.iloc[j, i][1:-1]
            # separate the string by space and convert to list
            list_of_strings = string.split(' ')
            # remove '' from the list
            list_of_strings = [x for x in list_of_strings if x != '']
```

```
            # convert the list to float
            array_of_floats = [float(x) for x in list_of_strings ]
            # save the array in the dataframe

            row.append(array_of_floats)
        # append the row to the dataframe
        df_new.loc[len(df_new)] = row

    return df_new


def preprocess_analysis(data: pd.DataFrame, combinations_categories: list,
    groups_per_experiment_dict: dict, unique_experiments: np.ndarray, path_folder_results: str) ->
     dict:
    """ Preprocess the data for the analysis by creating a dict with the combinations of categories
        and the groups of each experiment. Data is also saved, the mean snr and std snr of each
        group is calculated and saved in a table.

    Args:
        data (pd.DataFrame): The data to be preprocessed of the experiments.
        combinations_categories (list): The combinations of categories to be analyzed.
        groups_per_experiment_dict (dict): Dictionary with the groups of each experiment with
            respect to each category.
        unique_experiments (np.ndarray): The unique experiments.
        path_folder_results (str): The path to the folder where the results will be saved.

    Returns:
        dict: A dict with the combinations of categories and the groups of each experiment
            according with the data extracted from the data.
    """
    # create a dict to store the results
    combination_dict = {}
    # create a list to store the headers of the table
    headers = ['experiment', 'combination', 'group',
               'SNR mean', 'SNR std', 'number of samples']
    row = []
    # iterate over the unique experiments (folder)
    for experiment in unique_experiments:
        variables_dict = {}

        # remove the combinations with 'shape' if the experiment is 'Experiment_2'
        if experiment == 'Experiment_2':
            combinations_categories_filtered = [
                combination for combination in combinations_categories if 'shape' not in
                    combination]
        else:
            combinations_categories_filtered = combinations_categories

        for combination in combinations_categories_filtered:
            # create a unique string for the combination of categories
            combination_string = '-'.join(combination)
            # create an empty dict for the combination of categories
            variables_dict[combination_string] = {}
            # iterate over the groups in each category
            variables_dict[combination_string]['categories'] = {}
            for category in combination:
```

```
            variables_dict[combination_string]['categories'][category] =
                groups_per_experiment_dict[experiment][category]
        # create all combinations of groups in the combination of categories
        combinations_groups = create_combinations_of_groups(
            variables_dict[combination_string])

        # iterate over the combinations of groups
        for combination_group_setting_loaded in combinations_groups:
            # create a unique string for the combination of groups
            combination_group_setting_loaded_string = str(combination_group_setting_loaded
                [0]) + '-' + str(
                combination_group_setting_loaded[1]) + '-' + str(
                    combination_group_setting_loaded[2])
            # create an empty dict for the combination of groups
            variables_dict[combination_string][combination_group_setting_loaded_string] = {
            }

            # extract the data of the combination of groups
            data_combination_group_setting_loaded = data[(data['Folder'] == experiment) & (
                data[combination[0]] == combination_group_setting_loaded[0]) & (
                data[combination[1]] == combination_group_setting_loaded[1]) & (data[
                    combination[2]] == combination_group_setting_loaded[2])]
            # extract only the SNR metrics
            data_combination_group_setting_loaded_SNR_df =
                data_combination_group_setting_loaded.iloc[
                :, 6:]

            # convert the dataframe to a 1D array
            data_combination_group_setting_loaded_SNR = np.array(
                data_combination_group_setting_loaded_SNR_df.values.tolist()).flatten()

            # save the data of the combination of groups in the dict under the scope data
            variables_dict[combination_string][combination_group_setting_loaded_string]['data']
                = data_combination_group_setting_loaded_SNR
            # save the number of measurements in the dict under the scope data_length
            variables_dict[combination_string][combination_group_setting_loaded_string][
                'data_length'] = len(
                data_combination_group_setting_loaded_SNR)

            # measure the mean and standard deviation of the data
            mean = np.mean(data_combination_group_setting_loaded_SNR)
            std = np.std(data_combination_group_setting_loaded_SNR)

            # save the mean and standard deviation in the dict under the scope mean and std
            variables_dict[combination_string][combination_group_setting_loaded_string]['mean
                '] = mean
            variables_dict[combination_string][combination_group_setting_loaded_string]['std']
                = std

            # create a dict for participant specific data
            variables_dict[combination_string][combination_group_setting_loaded_string][
                'participant_specific_data'] = {
            }
            # get all headers of data_combination_group_setting_loaded_SNR_df (participant
                specific data)
            headers_participants = data_combination_group_setting_loaded_SNR_df.columns
```

```
                for participant in headers_participants:
                    # extract the data of teh specific participant from
                        data_combination_group_setting_loaded_SNR_df
                    data_combination_group_setting_loaded_SNR_df_participant =
                        data_combination_group_setting_loaded_SNR_df[
                        participant ]
                    # convert the dataframe to a 1D array
                    data_combination_group_setting_loaded_SNR_participant = np.array(
                        data_combination_group_setting_loaded_SNR_df_participant.values.tolist()).
                            flatten()
                    # save the data of the specific participant in the dict under the scope
                        participant_specific
                    variables_dict[combination_string][combination_group_setting_loaded_string][
                        'participant_specific_data'][participant] =
                            data_combination_group_setting_loaded_SNR_participant

                # create a row for the table with the results
                row.append([experiment, combination_string,
                    combination_group_setting_loaded_string, mean, std, len(
                    data_combination_group_setting_loaded_SNR)])
                # add the row to the table with the results
            # save the dict of the combination of categories in the dict of the experiment
            combination_dict[experiment] = variables_dict
        # create a dataframe with the results
        results_df = pd.DataFrame(row, columns=headers)
        # save the dataframe with the results
        results_df.to_csv(os.path.join(path_folder_results,
                        'Combination_analysis_results.csv'), index=False)

        return combination_dict


def main():
    path = r"/media/sjoerd/BackUp Drive/Thesis_project/Data_SNR/SNR_sorted_by_participant.csv"

    # get path of folder where the results are saved
    path_folder = os.path.dirname(path)
    # create new folder named "SNR results" in the folder where the results are saved
    path_folder_results = os.path.join(path_folder, "SNR results")
    # check if the folder already exists
    if not os.path.exists(path_folder_results):
        # if not, create the folder
        os.makedirs(path_folder_results)

    # read the csv file
    data = pd.read_csv(path)

    # convert the strings in the dataframe to a list
    data = convert_dataframe_strings_to_list_SNR(data)

    # get the headers of the dataframe
    headers = data.columns.values

    # unique experiments (folder)
    unique_experiments = data['Folder'].unique()
    # unique categories
```

```
    unique_categories_all = headers[2:6]

    # create a dict with the unique groups per experiment per category
    groups_per_experiment_dict = {}
    for experiment in unique_experiments:
        groups_per_experiment_dict[experiment] = {}
        for category in unique_categories_all:
            # extract unique metrics per experiment per category
            unique_groups = data[data['Folder']
                                 == experiment][category].unique()
            # sort the unique groups
            unique_groups.sort()
            groups_per_experiment_dict[experiment][category] = unique_groups

    # create combinations of categories
    combinations_categories = create_combinations_of_categories(
        unique_categories_all)
    # create a dict with the combinations of categories and the data of the combinations of groups
    dict_of_combinations_with_data = preprocess_analysis(
        data, combinations_categories, groups_per_experiment_dict, unique_experiments,
            path_folder_results)
    # create a 3D and 4D plot for each combination of categories
    create_boxplot(dict_of_combinations_with_data,
                   groups_per_experiment_dict, unique_experiments, path_folder_results)
    create_heatmap_plot(dict_of_combinations_with_data,
                        groups_per_experiment_dict, unique_experiments, path_folder_results)
    create_4d_plot(dict_of_combinations_with_data,
                   groups_per_experiment_dict, unique_experiments, path_folder_results)


main()
```

D.2.7. analysis_of_questionnaire_v1.py

This script performs the statistical analysis of the questionnaire.

```
import numpy as np
import scipy.stats as stats
import sys
import pingouin as pg
import pandas as pd
from typing import Union


def create_table_with_significance_using_pvalue(pvalue_matrix: np.ndarray, groups: np.ndarray,
    alpha_threshold=0.05) -> pd.DataFrame:
    """ Create a table with the significance of the pairwise comparisons

    Args:
        pvalue_matrix (np.ndarray): Pvalue matrix with the pvalues of the pairwise comparisons
        groups (np.ndarray): Array with the groups
        alpha_threshold (float, optional): Threshold for the significance. Defaults to 0.05.

    Returns:
        pd.DataFrame: Table with the significance of the pairwise comparisons
    """
    # create a table with the pairwise comparisons with the rows one group tested against the
```

```
        other group mentioned in the columns
    pairwise_comparisons_table_significance = pd.DataFrame(
        index=groups, columns=groups)

    # loop through the unique groups and check if the pairwise comparison is in the pairwise
        comparisons table
    for i in range(groups.shape[0]):
        for j in range(groups.shape[0]):

            # extract labels of the groups to be compared
            group_A = groups[i]
            group_B = groups[j]

            # save the significance in the table pairwise_comparisons_table_significance
            if pvalue_matrix[i, j] <= alpha_threshold:
                pairwise_comparisons_table_significance.loc[group_A,
                                                            group_B] = 'Significant'
            else:
                pairwise_comparisons_table_significance.loc[group_A,
                                                            group_B] = 'Not significant'

    return pairwise_comparisons_table_significance


def create_table_with_pairwise_comparisons(pairwise_comparisons: pd.DataFrame,
    alpha_pairwise_comparisons=0.05) -> Union[pd.DataFrame, pd.DataFrame]:
    """ Create a table with the pairwise comparisons with the rows one group tested against the
        other group mentioned in the columns

    Args:
        pairwise_comparisons (pd.DataFrame): Table with the pairwise comparisons
        alpha_pairwise_comparisons (float, optional): Alpha value for the pairwise comparisons.
            Defaults to 0.05.

    Returns:
        Union[pd.DataFrame, pd.DataFrame]: Table with the p-values and table with the significance
            of the pairwise comparisons
    """
    # extract the unique group labels from the pairwise comparisons table
    unique_groups_A = pairwise_comparisons['A'].unique()
    unique_groups_B = pairwise_comparisons['B'].unique()
    unique_groups = np.unique(np.concatenate(
        (unique_groups_A, unique_groups_B)))

    # extract the p-values from the pairwise comparisons table
    p_values_pairwise_comparisons = pairwise_comparisons['pval']

    # create a table with the pairwise comparisons with the rows one group tested against the
        other group mentioned in the columns
    pairwise_comparisons_table_p_values = pd.DataFrame(
        index=unique_groups, columns=unique_groups)
    pairwise_comparisons_table_significance = pd.DataFrame(
        index=unique_groups, columns=unique_groups)

    # loop through the unique groups and check if the pairwise comparison is in the pairwise
        comparisons table
```

```
for i in range(unique_groups.shape[0]):
    for j in range(unique_groups.shape[0]):

        # extract labels of the groups to be compared
        group_A = unique_groups[i]
        group_B = unique_groups[j]

        # check where the group labels are in the pairwise comparisons table
        Check_if_pair_A_B_in_pairwise_comparisons = (
            pairwise_comparisons['A'] == group_A) & (pairwise_comparisons['B'] == group_B)

        # check if any of them are true
        if np.any(Check_if_pair_A_B_in_pairwise_comparisons):
            print("Pairwise comparison between group {} and group {} is in the pairwise
                comparisons table".format(
                group_A, group_B))
            # save the p-value in the table
            p_value = float(p_values_pairwise_comparisons[(
                pairwise_comparisons['A'] == group_A) & (pairwise_comparisons['B'] ==
                    group_B)])
            pairwise_comparisons_table_p_values.loc[group_A,
                                                    group_B] = p_value

            # save the significance in the table pairwise_comparisons_table_significance
            if p_value <= alpha_pairwise_comparisons:
                pairwise_comparisons_table_significance.loc[group_A,
                                                            group_B] = ' Significant '
            else:
                pairwise_comparisons_table_significance.loc[group_A,
                                                            group_B] = 'Not significant '

        else:
            Check_if_pair_B_A_in_pairwise_comparisons = (
                pairwise_comparisons['A'] == group_B) & (pairwise_comparisons['B'] ==
                    group_A)

            if np.any(Check_if_pair_B_A_in_pairwise_comparisons):
                print("Pairwise comparison between group {} and group {} is in the pairwise
                    comparisons table".format(
                    group_B, group_A))
                # save the p-value in the table
                p_value = float(p_values_pairwise_comparisons[(
                    pairwise_comparisons['A'] == group_B) & (pairwise_comparisons['B'] ==
                        group_A)])
                # save the p-value in the table pairwise_comparisons_table_p_values
                pairwise_comparisons_table_p_values.loc[group_A,
                                                        group_B] = p_value

                # save the significance in the table pairwise_comparisons_table_significance
                if p_value <= alpha_pairwise_comparisons:
                    pairwise_comparisons_table_significance.loc[group_A,
                                                                group_B] = ' Significant '
                else:
                    pairwise_comparisons_table_significance.loc[group_A,
                                                                group_B] = 'Not significant '
```

```
                # if the group labels are not in the pairwise comparisons table, put a NaN in the
                    table
                else:
                    print("Pairwise comparison between group {} and group {} is not in the pairwise
                        comparisons table".format(
                        group_A, group_B))
                    # put a NaN in the table
                    pairwise_comparisons_table_p_values.loc[group_A,
                                                            group_B] = np.nan
                    # put an X in the table to indicate that the comparison is not in the pairwise
                        comparisons table
                    pairwise_comparisons_table_significance.loc[group_A,
                                                                group_B] = "X"
                    continue

    return pairwise_comparisons_table_p_values, pairwise_comparisons_table_significance


# load the data
path = '/media/sjoerd/BackUp Drive/Thesis_project/Questionairre results/Questionairre_answers.csv
    '
data_csv = pd.read_csv(path)

# extract the categories from the data table which is the first column except the values that are
    NaN
categories = data_csv.iloc[:, 0].dropna()
# unique categories
unique_categories = np.unique(categories)

# extract the groups from the data table which is the second column except the values that are
    NaN
groups = data_csv.iloc[:, 1].dropna()

# extract the metrics from the data table which is the third column except the values that are NaN
metrics = data_csv.iloc[0:5, 2]

# headers
headers = ['category', 'groups', 'metric', 'p_value_bartlett', 'Variances equal?', 'ANOVA
    method', 'df within groups', 'df between groups',
           'p_value ANOVA method', 'Mean different?', 'Type of post hoc test', 'p_value_matrix', '
               Significance_matrix']


# create an empty dataframe to store the results of each step of the analysis
results_logger = pd.DataFrame(columns=headers)
for category in unique_categories:
    # create a dictionary to store the results of each step of the analysis
    results_back_log = {}
    # create a list to store the results of each step of the analysis
    row = []
    # save the category in the dictionary
    results_back_log['category'] = category
    # filter the groups that belong to the category
    groups_in_category = groups[categories == category]

    # perform for each metric a comparison between the groups in the category
```

```
for metric in metrics:

    # extract the values of the metric for each group in the category
    data = {}
    # for each group in the category
    for group in groups_in_category:
        print(f'category: {category}, group: {group}, metric: {metric}')
        # get the index of the group in the groups_in_category
        index_group = np.where(groups_in_category == group)[0][0]
        index_metric = np.where(metrics == metric)[0][0]
        actual_index_in_data_csv = index_group + index_metric
        # extract data of the group in the category for the metric
        data[group] = np.array(
            data_csv.iloc[actual_index_in_data_csv, 3:9]).astype(np.float64)
        print(
            f" The variance of the group {group} in metric {metric} is {np.var(data[group],
                ddof=1)}")

    # convert the list of groups in the category to an array
    groups_in_category_array = np.array(groups_in_category)

    # perform bartlett's test
    alpha_bartlett = 0.05  # 95% confidence
    # check number of groups
    if len(groups_in_category) == 3:
        # perform Bartlett's test
        print(stats.bartlett(data[groups_in_category_array[0]],
                data[groups_in_category_array[1]], data[groups_in_category_array[2]]))
        # save the p-value from the test
        p_value_bartlett = stats.bartlett(
            data[groups_in_category_array[0]], data[groups_in_category_array[1]], data[
                groups_in_category_array[2]])[1]
        # Alpha is the significance level at which we reject the null hypothesis
    elif len(groups_in_category) == 4:
        # perform Bartlett's test
        print(stats.bartlett(data[groups_in_category_array[0]], data[groups_in_category_array
            [1]],
                data[groups_in_category_array[2]], data[groups_in_category_array[3]]))
        # save the p-value from the test
        p_value_bartlett = stats.bartlett(data[groups_in_category_array[0]], data[
            groups_in_category_array[1]],
                                          data[groups_in_category_array[2]], data[
                                              groups_in_category_array[3]])[1]

    # if true variances are not equal
    if p_value_bartlett > alpha_bartlett:
        print('The variances are equal')
        variances_equal = 'Yes'
        # if the variances are equal, perform one-way ANOVA
        print("Performing one-way ANOVA")
        anova_method = 'One-way ANOVA'
        # perform one-way ANOVA
        alpha_one_way_anova = 0.05 # 95% confidence
        if len(groups_in_category) == 3:
            # perform one-way ANOVA
            print(stats.f_oneway(data[groups_in_category_array[0]],
```

```
            data[groups_in_category_array[1]], data[groups_in_category_array[2]]))
        # save the p-value from the test
        pvalue_f_oneway = stats.f_oneway(
            data[groups_in_category_array[0]], data[groups_in_category_array[1]], data[
                groups_in_category_array[2]])[1]
        # Alpha is the significance level at which we reject the null hypothesis
    elif len(groups_in_category) == 4:
        # perform one-way ANOVA
        print(stats.f_oneway(data[groups_in_category_array[0]], data[
            groups_in_category_array[1]],
            data[groups_in_category_array[2]], data[groups_in_category_array[3]]))
        # save the p-value from the test
        pvalue_f_oneway = stats.f_oneway(data[groups_in_category_array[0]], data[
            groups_in_category_array[1]],
                                         data[groups_in_category_array[2]], data[
                                             groups_in_category_array[3]])

    # save the p-value from the test
    p_value_anova_method = pvalue_f_oneway

    # calculate the degrees of freedom of the f one-way anova
    total_number_of_observations = 0
    for group_i in groups_in_category_array:
        total_number_of_observations += len(data[group_i])
    # calculate all the degrees of freedom of the f one-way anova
    df_between_groups = len(groups_in_category_array) - 1
    df_within_groups = total_number_of_observations - \
        len(groups_in_category_array)
    df_total = total_number_of_observations - 1

    # Alpha is the significance level at which we reject the null hypothesis
    # if p-value is greater than alpha, accept null hypothesis (the means are equal)
    if pvalue_f_oneway > alpha_one_way_anova:
        print('The means are equal')
        mean_different = 'No'
        type_of_post_hoc_test = 'None'
        p_value_matrix = 'None'
        significance_matrix = 'None'

    # if p-value is less than alpha, reject null hypothesis (the means are not equal)
    elif pvalue_f_oneway <= alpha_one_way_anova:
        print('The means are not equal')
        mean_different = 'Yes'
        print("Performing Tukey's test")
        anova_method = 'Tukey\'s test'

        # perform Tukey's test
        if len(groups_in_category) == 3:
            # perform Tukey's post-hoc test
            print(stats.tukey_hsd(
                data[groups_in_category_array[0]], data[groups_in_category_array[1]], data[
                    groups_in_category_array[2]]))
            # save the p-value from the test
            results = stats.tukey_hsd(
                data[groups_in_category_array[0]], data[groups_in_category_array[1]], data[
                    groups_in_category_array[2]])
```

```
            # Alpha is the significance level at which we reject the null hypothesis
        elif len(groups_in_category) == 4:
            # perform Tukey's post-hoc test
            print(stats.tukey_hsd(data[groups_in_category_array[0]], data[
                groups_in_category_array[1]],
                  data[groups_in_category_array[2]], data[groups_in_category_array[3]]))
            # save the p-value from the test
            results = stats.tukey_hsd(data[groups_in_category_array[0]], data[
                groups_in_category_array[1]],
                                      data[groups_in_category_array[2]], data[
                                          groups_in_category_array[3]])

        # save the p-value from the test
        pvalue_matrix = results.pvalue
        alpha_tukey = 0.05 # 95% confidence
        tukey_table_significance = create_table_with_significance_using_pvalue(
            pvalue_matrix, groups_in_category_array, alpha_tukey)
        # save the p-value from the test
        p_value_matrix = pvalue_matrix
        # save the significance matrix
        significance_matrix = tukey_table_significance

# if the variances are not equal, perform Welch's ANOVA
elif p_value_bartlett <= alpha_bartlett:
    # log that variances are not equal
    variances_equal = 'No'
    print('The variances are not equal')
    # if the variances are not equal, perform Welch's ANOVA
    print("Performing Welch's ANOVA")
    anova_method = "Welch's ANOVA"

    # check number of groups
    if len(groups_in_category) == 3:
        # create a dataframe with the values of the groups
        df = pd.DataFrame({'value': np.concatenate((data[groups_in_category_array[0]],
            data[groups_in_category_array[1]], data[groups_in_category_array[2]])), 'group':
              np.concatenate((np.repeat(
            groups_in_category[0], len(data[groups_in_category[0]])), np.repeat(
                groups_in_category[1], len(data[groups_in_category[1]])), np.repeat(
                groups_in_category_array[2], len(data[groups_in_category_array[2]])))})
    elif len(groups_in_category) == 4:
        # create a dataframe with the values of the groups
        df = pd.DataFrame({'value': np.concatenate((data[groups_in_category_array[0]],
            data[groups_in_category_array[1]], data[groups_in_category_array[2]], data[
            groups_in_category_array[3]])), 'group': np.concatenate((np.repeat(
            groups_in_category_array[0], len(
            data[groups_in_category_array[0]])), np.repeat(groups_in_category_array[1],
                len(data[groups_in_category_array[1]])), np.repeat(
                groups_in_category_array[2], len(data[groups_in_category_array[2]])), np.
                repeat(groups_in_category_array[3], len(data[groups_in_category_array[3]])
                ))))})

    # perform Welch's ANOVA
    print(pg.welch_anova(data=df, dv='value', between='group'))
    # PERFORM WELCH'S ANOVA USING PINGOUIN and save the results in a variable
    results = pg.welch_anova(data=df, dv='value', between='group')
```

```
            # extract df between groups from the results
            df_between_groups = results['ddof1'][0]
            # extract the degrees of freedom within groups from the results
            df_within_groups = results['ddof2'][0]
            df_total = df_between_groups + df_within_groups

            # extract the p-value from the results
            p_value_welch_anova = results['p-unc'][0]
            alpha_welch_anova = 0.05 # 95% confidence

            # if the p-value is greater than alpha, accept null hypothesis (the means are equal)
            if p_value_welch_anova > alpha_welch_anova:
                mean_different = 'No'
                print('The mean values are equal')
                type_of_post_hoc_test = 'None'
                p_value_matrix = 'None'
                Significance_matrix = 'None'

            # if the p-value is less than alpha, reject null hypothesis (the means are not equal)
            elif p_value_welch_anova <= alpha_welch_anova:
                mean_different = 'Yes'
                print('The mean values are not equal')
                type_of_post_hoc_test = 'Games-Howell'
                # PERFROM THE pairwise games-howell post hoc test
                print(pg.pairwise_gameshowell(
                    data=df, dv='value', between='group'))

                # create a table with the pairwise comparisons
                pairwise_comparisons = pg.pairwise_gameshowell(
                    data=df, dv='value', between='group')
                alpha_pairwise_comparisons = 0.05 # 95% confidence
                table_with_p_values, p_values_with_significance =
                    create_table_with_pairwise_comparisons(
                    pairwise_comparisons, alpha_pairwise_comparisons)

                # save the data in the logger
                p_value_matrix = table_with_p_values
                significance_matrix = p_values_with_significance

            # save the p-value from the variance test
            p_value_anova_method = p_value_welch_anova

        try: # round the total degrees of freedom to 2 decimal places
            df_between_groups = round(df_between_groups, 2)
            df_within_groups = round(df_within_groups, 2)
            df_total = round(df_total, 2)

            # create a row with the results
            row = [category, np.array(groups_in_category), metric, p_value_bartlett,
                variances_equal, anova_method, df_between_groups,
                    df_within_groups, p_value_anova_method, mean_different,
                        type_of_post_hoc_test, p_value_matrix, significance_matrix]

            # add the row to results_logger dataframe
            results_logger.loc[len(results_logger)] = row
```

```
        except:
            print("Error in the data logger")
            # quit the program
            sys.exit()

# save the results in a csv file
results_logger.to_csv('results_logger_questionnaire.csv', index=False)
```

E

Consent Form

SSVEP-based brain-computer interfaces

Informed consent form for participants

Researchers

Sjoerd van Vliet, MSc student
E-mail: s.t.vanvliet@student.tudelft.nl
Tel: +31(0)6 83390874

Dr.ir. Yke Bauke Eisma
E-mail: y.b.eisma@tudelft.nl

Location

Amsterdam UMC, locatie AMC
Meibergdreef 9, 1105 AZ Amsterdam

This document describes the purpose of this study, the experimental procedure, the right to withdraw and data handling. Read all sections carefully and answer the questions on page 2.

Research purpose

Brain-computer interfaces do not require conventional input (e.g., via a keyboard) but are controlled through electrical brainwaves measured with electroencephalography (EEG). For visual brain-computer interfaces, a phenomenon called steady-state visually evoked potentials (SSVEP) is often used. SSVEP are electrical potential differences that occur in the brain as a response to high-frequency visual stimuli (e.g., a flashlight).

Detecting SSVEP reliably and accurately is critical for the successful functioning of SSVEP-based brain-machine interfaces. The goal of this research is to investigate the effects of various aspects of visual stimuli on the signal-to-noise ratio in SSVEP-based interfaces.

Research purpose and experiment procedure

You will be sitting in front of a computer screen. You will be wearing an EEG headset recording the electrical activity in your brain (Figure 1, left). An eye-tracking camera will be recording your eye movements (Figure 1, right). The computer screen will be showing flickering stimuli with various colours, shapes, sizes, and frequencies. Your only task is to look at the stimuli.

Before the experiment, an experimenter will place the EEG cap and, based on the visual output on the computer screen, inspect whether the signal quality of each electrode is satisfactory. If not, electrodes will be slightly repositioned, and a few drops of a saline EEG gel will be added under the electrodes to reduce their impedance. The preparation of the cap will take about 30 minutes.

The experiment will consist of two blocks, each lasting about 45 and 52 minutes. After each block, you can take a 10-min break. The total duration of the experiment will be about 2 hours.

Figure 1. Left: EEG cap. Right: Laptop with eye-tracking camera mounted on top of the keyboard.

Risk of participating

This experiment could induce epileptic seizures. If you have epilepsy, do not participate in this experiment. If you experience any discomfort, please inform the experiment supervisor so that the experiment can be stopped and restarted later.

Right to withdraw

Your participation is completely voluntary, and you may stop at any time during the experiment for any reason. You have the right to refuse or withdraw from the experiment at any time, without negative consequences and without having to provide any explanation.

Data handling

All data will be collected anonymously and used for scientific research only. The eye-tracker only records eye movements and no images of your eyes or face. You will not be personally identifiable in future publications based on this work, in data files shared with other researchers, or in data repositories. This signed consent form will be kept in a dedicated locker.

Prevention of the spread of COVID-19

You may not participate if you show any symptoms indicative of COVID-19. You will be asked to disinfect your hands before touching any equipment. All equipment used in the experiment will be disinfected before participation.

Please respond to the following statements

Statement	**Yes**	**No**
I consent to participate voluntarily in this study.	○	○
I have read and understood the information provided in this document.	○	○
I adhere to the preventative measures with regard to COVID-19 explained above.	○	○
I understand that I can withdraw from the study at any time without any negative consequences.	○	○
I agree that the data collected during the experiment will be used for scientific research and may be presented in a publication and public data repository.	○	○

Signature

Name:

Date:

Signature: _____